User Manual for the BCM1250, BCM1125, and BCM1125H
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1250-UM100-R</td>
<td>06/25/01</td>
<td>Initial release.</td>
</tr>
<tr>
<td>1250-UM101-R</td>
<td>01/02/02</td>
<td>New Sections:</td>
</tr>
</tbody>
</table>

  - Section: “Interrupts” on page 47
  - Section: “ZBbus Cycle Count and Compare” on page 58.
  - Section: “Reduced Cache Size” on page 94.
  - Section: “DDR FCRAMs” on page 123.
  - Section: “HyperTransport Read Restrictions” on page 200.
  - Section: “HyperTransport Bounce Space” on page 213.
  - Section: “HyperTransport Differences from Revision 1.03 Specification” on page 224.
  - Section: “HyperTransport Differences from Revision 0.17 Specification” on page 222.
  - Section: “HyperTransport Target Done Counter” on page 236.
  - Section: “TCP Checksum Checking” on page 282.
  - Section: “Flow Control In Encoded Packet FIFO Modes” on page 293.
  - Section: “Restrictions When Resetting the Interface” on page 300.
  - Section: “Burst Mode” on page 370.
  - Section: “Early Chip Select” on page 372.
  - Section: “BSRMODE - Holding Boundary Scan Active” on page 434.

**New Tables:**

- Table 15, Table 48, Table 126, Table 127, Table 138, Table 139, Table 157, Table 203, Table 212, and Table 214.
This list summarizes the major changes between 1250-UM101 and 1250_1125-UM100. The 1250_1125-UM100CB version of this manual has change bars indicating all changes between the older and newer versions.

- Section 1: Updates to describe BCM1125/H
- Section 2: Updates to describe BCM1125/H.
- Section 3: Additional Clarifications and BCM1125/H descriptions
  - New section: Error Conditions
- Section 4: Additional Clarifications and BCM1125/H descriptions
  - Expanded section: Bus Watcher
  - New Section: Magic Decoder Ring for Using The Trace Buffer
- Section 5: Updates for BCM1125/H
  - New register: Level 2 Cache Settings Register
- Section 6: Additional Clarifications and BCM1125/H descriptions
  - New Section: Memory Access Sequencing
  - New Section: Example CHannel and Chip Select Configurations
  - Updated guidelines: Timing Parameter Guidelines
- Section 7: Additional Clarifications and BCM1125/H descriptions
  - New Section: Unaligned Buffer Descriptor Format for Ethernet DMA
  - New Section: CRC and Checksum Generators
- Section 8: Additional Clarifications and BCM1125/H descriptions
- Section 9: Additional Clarifications and BCM1125/H descriptions
  - New Section: Prepended Header Frame Format
  - New Functionality: Destination Address Filtering
  - New Functionality: Receive DMA Channel Selection
  - New Functionality: Flow Control
- Section 16: Cross reference links added from register to defining table
- Index: Expanded

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1250_1125-UM100-R</td>
<td>10/21/02</td>
<td>This list summarizes the major changes between 1250-UM101 and 1250_1125-UM100. The 1250_1125-UM100CB version of this manual has change bars indicating all changes between the older and newer versions.</td>
</tr>
</tbody>
</table>

- Section 1: Updates to describe BCM1125/H
- Section 2: Updates to describe BCM1125/H.
- Section 3: Additional Clarifications and BCM1125/H descriptions
  - New section: Error Conditions
- Section 4: Additional Clarifications and BCM1125/H descriptions
  - Expanded section: Bus Watcher
  - New Section: Magic Decoder Ring for Using The Trace Buffer
- Section 5: Updates for BCM1125/H
  - New register: Level 2 Cache Settings Register
- Section 6: Additional Clarifications and BCM1125/H descriptions
  - New Section: Memory Access Sequencing
  - New Section: Example CHannel and Chip Select Configurations
  - Updated guidelines: Timing Parameter Guidelines
- Section 7: Additional Clarifications and BCM1125/H descriptions
  - New Section: Unaligned Buffer Descriptor Format for Ethernet DMA
  - New Section: CRC and Checksum Generators
- Section 8: Additional Clarifications and BCM1125/H descriptions
- Section 9: Additional Clarifications and BCM1125/H descriptions
  - New Section: Prepended Header Frame Format
  - New Functionality: Destination Address Filtering
  - New Functionality: Receive DMA Channel Selection
  - New Functionality: Flow Control
- Section 16: Cross reference links added from register to defining table
- Index: Expanded
# Table of Contents

## Section 1: Introduction

The SiByte Broadband Processor Family ................................................................. 1
The BCM1250 ........................................................................................................... 2
The BCM1125 and BCM1125H ............................................................................. 3
Audience .................................................................................................................. 3
Other Documentation ............................................................................................... 4
Terminology .............................................................................................................. 5

## Section 2: Signal Overview

BCM1250 Signal Groups .......................................................................................... 7
BCM1125/H Signal Groups ....................................................................................... 8

## Section 3: System Overview

Introduction .............................................................................................................. 9
Internal Registers .................................................................................................. 11
Coherence ................................................................................................................. 12
Ordering Rules and Device Drivers ....................................................................... 14
CPU Speculative Execution .................................................................................... 16
Error Conditions ..................................................................................................... 17
  Cache Error Exceptions ....................................................................................... 17
  Bus Error Exceptions ......................................................................................... 18
CPU to CPU Communication (BCM1250 Only) .................................................... 19
External Interrupts ................................................................................................. 19
Overview of the ZBbus Protocol ......................................................................... 20
  Arbitration ........................................................................................................... 21
  Address Phase ................................................................................................... 22
  Response Phase ................................................................................................. 23
  Data Phase ......................................................................................................... 24
Reset ....................................................................................................................... 26
Clocks ...................................................................................................................... 31
Memory Map .......................................................................................................... 34

## Section 4: System Control and Debug Unit

Introduction ............................................................................................................ 41
## System Control

- Mailbox Registers ................................................................. 46
- Interrupts .................................................................................. 47
  - HyperTransport Interrupts ..................................................... 48
  - The Full Interrupt Mapper ................................................... 50
- Timers ....................................................................................... 57
  - Watchdog Timers ................................................................. 57
  - General Timers ................................................................. 58
  - Timer Special Cases .......................................................... 58
  - ZBbus Cycle Count and Compare ...................................... 58
  - Timer Registers ................................................................. 59
- System Performance Counters ................................................. 61
- Bus Watcher ........................................................................... 64
- Address Trapping..................................................................... 67
- Trace Unit ................................................................................ 70
  - Trigger Events .................................................................. 70
  - Trigger Sequences ........................................................... 73
  - Using the Trace Buffer ....................................................... 76
  - Reading the Trace Buffer .................................................. 79
  - Magic Decoder Ring For Using The Trace Buffer ................ 81
  - Connections to the Trace Logic ......................................... 83
  - Trace Example 1: All CPU0 Activity ................................. 84
  - Trace Example 2: Network Packet Headers ....................... 85

## Section 5: L2 Cache

- Introduction ............................................................................ 89
- Normal Operation .................................................................... 89
- Using the L2 Cache as Memory ............................................. 91
  - Standard RAM .................................................................. 92
  - Memory Locked in the L2 Cache ....................................... 92
  - Comments on Using the L2 as Memory ............................ 93
  - Reduced Cache Size ......................................................... 94
- Cache Management Access ................................................... 94
  - Standard Management Mode Accesses (both ECC_diag address bits zero) ................................................................. 97
  - ECC Diagnostic Management Accesses (ECC_diag bits nonzero) .............................................................................. 98
Cache Configuration Register ........................................................................................................ 99
Example Startup Code to clear the L2 Cache .................................................................................. 99

Registers .......................................................................................................................................... 100

Section 6: DRAM ............................................................................................................................. 103

Introduction ...................................................................................................................................... 103
A Comment on the term Bank .......................................................................................................... 103
Memory Controller Architecture ..................................................................................................... 104
Memory Access Sequencing ............................................................................................................. 107
Clock Ratios and Clocking Scheme ................................................................................................. 107
Memory Configurations .................................................................................................................... 109
Mapping .......................................................................................................................................... 109
Channel Select ................................................................................................................................. 109
Chip Select ....................................................................................................................................... 110
Example Channel and Chip Select Configurations ......................................................................... 112
Row, Column and Bank Configuration ............................................................................................. 117
Choosing Interleave Parameters ..................................................................................................... 120
Page Policy ....................................................................................................................................... 122
Supported DRAMs and DIMMs ....................................................................................................... 123
DDR SDRAMs .................................................................................................................................. 123
DDR FCRAMs ................................................................................................................................... 123
DIMMs ............................................................................................................................................ 124
Larger Memory Systems ................................................................................................................... 124
ECC .................................................................................................................................................... 125
SDRAM Timing ................................................................................................................................. 125
SDRAM Refresh ................................................................................................................................. 126
SDRAM Initialization and Commands ............................................................................................... 126
I/O Control ...................................................................................................................................... 128
Timing Parameter Guidelines ............................................................................................................ 131
Performance Monitoring Features .................................................................................................... 134
ZBBus Monitoring ............................................................................................................................. 134
Configuration Registers ................................................................................................................... 135

Section 7: DMA .................................................................................................................................. 147
DMA Controllers ............................................................................................................................... 147
Data Buffers and Descriptors ............................................................................................................ 147
Unaligned Buffer Descriptor format for Ethernet DMA ................................................................. 154

DMA Coherence and Cache Options ................................................................................................ 155
DMA Configurations .......................................................................................................................... 156
Ethernet and Serial DMA Engines .................................................................................................. 157
  Descriptor Count Watermarks ........................................................................................................ 157
  Completion Interrupts ..................................................................................................................... 158
  Explicit Descriptor Interrupts ........................................................................................................ 158
  ASIC Mode Transfers .................................................................................................................... 159
Option and Flag Bits for Ethernet MACs .......................................................................................... 171
Control and Flag Bits for Synchronous Serial Interface ................................................................. 174

Data Mover .................................................................................................................................... 176
  Data Mover Operation .................................................................................................................... 176
  CRC and Checksum Generators .................................................................................................... 178
    Checksum Generation .................................................................................................................. 178
    CRC Generation .......................................................................................................................... 179
    Computation Sizes and Bandwidth .............................................................................................. 180
    Examples ....................................................................................................................................... 181
  Data Mover Control Registers ....................................................................................................... 184
  Data Mover Descriptors ................................................................................................................ 187

Section 8: PCI Bus and HyperTransport Fabric .......................................................................... 190
  Introduction .................................................................................................................................... 190
  PCI and HyperTransport Address Range .................................................................................... 192
    Memory Mapped Devices ............................................................................................................ 194
    HyperTransport Expansion Space .............................................................................................. 194
    Configuration Space .................................................................................................................. 194
    PCI I/O Space ............................................................................................................................. 195
    The SouthBridge, VGA and Subtractive Decode ........................................................................ 195
    HyperTransport End Of Interrupt (EOI) Signaling Space ......................................................... 198
    Legacy Interrupt Acknowledge (IACK) Space ............................................................................. 199
    PCI Full Access Space ................................................................................................................ 199
    Special HyperTransport Space .................................................................................................. 199
    HyperTransport Read Restrictions ............................................................................................. 200
Endian Policies ............................................................................................................................... 201
  Little Endian System: No Swaps .................................................................................................... 201
Buffer Control: TxBufCountMax and DataBufAlloc ................................................................. 260
HyperTransport Resets ............................................................................................................. 260

Section 9: Ethernet MACs ........................................................................................................... 264

Introduction ............................................................................................................................... 264
Interface Overview .................................................................................................................... 265
Protocol Engine and GMII/MII .................................................................................................. 267
  Ethernet Frame Format ............................................................................................................ 268
  Prepended Header Frame Format ........................................................................................... 270
  Protocol Engine Configuration ............................................................................................... 271
  Interface to PHY ..................................................................................................................... 271

Transmitter Operation .................................................................................................................. 272
  Transmitter Configuration ........................................................................................................ 272
  Transmit Path .......................................................................................................................... 274

Receiver Operation .................................................................................................................... 275
  Receiver Configuration .......................................................................................................... 275
  Receive Path ........................................................................................................................... 277
  Destination Address Filtering ................................................................................................. 278
  Receive DMA Channel Selection ........................................................................................... 281
  Packet Type Identification ...................................................................................................... 282
  IPv4 Header Checksum ........................................................................................................... 283
  TCP Checksum Checking ........................................................................................................ 283
  Packets Dropped by the DMA Channel .................................................................................. 283

Flow Control ............................................................................................................................... 284

Interrupts ..................................................................................................................................... 286
  Standard Interrupt Signaling .................................................................................................. 286
  Split Interrupt Signaling ......................................................................................................... 286

Management Interface to PHY .................................................................................................. 287

RMON Counters ......................................................................................................................... 289

Packet FIFO Interfaces .............................................................................................................. 292
  Flow Control In Encoded Packet FIFO Modes ...................................................................... 294

8-Bit Packet FIFO Operation .................................................................................................... 294
  8-Bit GMII Style Packet FIFO ................................................................................................. 295
  8-Bit Encoded Packet FIFO ................................................................................................... 296
  8-Bit SOP Flagged Packet FIFO ............................................................................................ 297
8-Bit EOP Flagged Packet FIFO ................................................................. 298

16-Bit Packet FIFO Operation ................................................................. 299
  16-Bit GMII Style Packet FIFO ............................................................. 299
  16-Bit Encoded Packet FIFO ............................................................... 300

Restrictions When Resetting the Interface ........................................... 301

MAC Registers ....................................................................................... 302

Section 10: Serial Interfaces ................................................................. 321

Introduction .......................................................................................... 321

Asynchronous Mode ............................................................................ 322

Baud Rate Generators .......................................................................... 322

Operation ............................................................................................... 323

Interrupts ............................................................................................... 325

Loopback ............................................................................................... 326

DUART Registers ............................................................................... 327

Synchronous Mode .............................................................................. 337

  Functional Overview ........................................................................... 337

  Input Line Interface ........................................................................... 340

    Input Using an External Enable ....................................................... 340

    Input Using the Internal Sequencer ............................................... 341

  Output Line Interface ....................................................................... 342

    Output Using an External Enable .................................................. 342

    Output Using the Internal Sequencer ............................................. 343

Synchronous Serial Protocol Engine ................................................... 344

  Operation in HDLC Mode ................................................................. 344

    Framing Parameters ....................................................................... 345

    HDLC Transmitter ......................................................................... 345

    HDLC Receiver .............................................................................. 347

  Operation in Transparent Mode ........................................................ 349

    Transmitter in Transparent Mode .................................................. 350

    Receiver in Transparent Mode ...................................................... 350

Synchronous Interface Configuration .................................................. 351

  DMA Configuration ........................................................................... 351

  FIFO Configuration .......................................................................... 351

  Protocol Engine Configuration ....................................................... 351
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line Interface Configuration</td>
<td></td>
<td>352</td>
</tr>
<tr>
<td>Synchronous Serial Interrupts</td>
<td></td>
<td>352</td>
</tr>
<tr>
<td>Synchronous Serial Loopback</td>
<td></td>
<td>352</td>
</tr>
<tr>
<td>RMON Counters</td>
<td></td>
<td>353</td>
</tr>
<tr>
<td>Synchronous Serial Register Summary</td>
<td></td>
<td>354</td>
</tr>
<tr>
<td>Section 11: Generic/Boot Bus</td>
<td></td>
<td>362</td>
</tr>
<tr>
<td>Introduction</td>
<td></td>
<td>362</td>
</tr>
<tr>
<td>Overview</td>
<td></td>
<td>362</td>
</tr>
<tr>
<td>Configuring a Chip Select Region</td>
<td></td>
<td>363</td>
</tr>
<tr>
<td>Address Range</td>
<td></td>
<td>363</td>
</tr>
<tr>
<td>Cacheable Access Blocking</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>Generic Bus Parity</td>
<td></td>
<td>364</td>
</tr>
<tr>
<td>Bus Width</td>
<td></td>
<td>365</td>
</tr>
<tr>
<td>Generic Bus Timing</td>
<td></td>
<td>365</td>
</tr>
<tr>
<td>Fixed Cycle Read Access</td>
<td></td>
<td>367</td>
</tr>
<tr>
<td>Fixed Cycle Write Access</td>
<td></td>
<td>368</td>
</tr>
<tr>
<td>Acknowledgement Read Access</td>
<td></td>
<td>369</td>
</tr>
<tr>
<td>Acknowledgement Write Access</td>
<td></td>
<td>370</td>
</tr>
<tr>
<td>Burst Mode</td>
<td></td>
<td>371</td>
</tr>
<tr>
<td>Early Chip Select</td>
<td></td>
<td>373</td>
</tr>
<tr>
<td>Boot ROM Support</td>
<td></td>
<td>373</td>
</tr>
<tr>
<td>Generic Bus Errors</td>
<td></td>
<td>374</td>
</tr>
<tr>
<td>Drive Strength Control</td>
<td></td>
<td>374</td>
</tr>
<tr>
<td>Generic Bus Registers</td>
<td></td>
<td>375</td>
</tr>
<tr>
<td>Section 12: PCMCIA Control Interface</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>Introduction</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>Connecting a PCMCIA Slot</td>
<td></td>
<td>384</td>
</tr>
<tr>
<td>Direct Connection of a Memory Only Card</td>
<td></td>
<td>385</td>
</tr>
<tr>
<td>Other PCMCIA Signals</td>
<td></td>
<td>389</td>
</tr>
<tr>
<td>Using The PCMCIA Card</td>
<td></td>
<td>390</td>
</tr>
<tr>
<td>Example PCMCIA Timings</td>
<td></td>
<td>391</td>
</tr>
<tr>
<td>Using the Power Outputs</td>
<td></td>
<td>393</td>
</tr>
</tbody>
</table>
Section 13: GPIO

Introduction .............................................................................................................................................. 397
The GPIO Pins ............................................................................................................................................. 397
GPIO Registers .......................................................................................................................................... 399
Other Pins That Can Be Used ................................................................................................................. 401
  Serial Ports ............................................................................................................................................. 401
  PCI ......................................................................................................................................................... 401
  MACs ..................................................................................................................................................... 401
  PCMCIA Power Control Pins .............................................................................................................. 401

Section 14: Serial Configuration Interface .............................................................................................. 404

Introduction .............................................................................................................................................. 404
SMBus Overview ...................................................................................................................................... 404
  Transport Protocol .............................................................................................................................. 404
  Transport Protocol Reset .................................................................................................................... 406
  SMBus Protocol .................................................................................................................................. 406
  Extended Protocol .............................................................................................................................. 408
Programming Model ................................................................................................................................ 410
  Using SMBus Protocols ...................................................................................................................... 410
  Using Extended Protocols .................................................................................................................. 412
Direct Access ........................................................................................................................................... 413
Booting Using an SMBus EEPROM ...................................................................................................... 413
  Switching from SMBus Mode ........................................................................................................... 414
SMBus Registers ...................................................................................................................................... 416

Section 15: JTAG and Debug .................................................................................................................. 422

Introduction .............................................................................................................................................. 422
TAP Controller .......................................................................................................................................... 422
  BYPASS Instruction ............................................................................................................................ 425
  IDCODE Instruction ............................................................................................................................. 425
  WAFFERID Instruction .......................................................................................................................... 425
  IMPCODE Instruction ........................................................................................................................... 426
  ADDRESS Instruction .......................................................................................................................... 426
  DATA Instruction ................................................................................................................................. 426
  CONTROL Instruction .......................................................................................................................... 426
EJTAGALL Instruction ........................................................................................................... 426
EJTAGBOOT Instruction ........................................................................................................ 426
NORMALBOOT Instruction .................................................................................................... 427
SCAN Instructions (0x26 - 0x38) .......................................................................................... 427
SYSCTRL Instruction .......................................................................................................... 427
TRACE Instruction ............................................................................................................... 430
PERF Instruction ................................................................................................................ 430
TRACECTRL and TRACECURRENT Instructions .................................................................. 431
PROCESSMON Instruction .................................................................................................... 432
Boundary Scan Register ...................................................................................................... 432
BSRMODE - Holding Boundary Scan Active ........................................................................... 435
Processor and Probe Access ............................................................................................... 436
Processor Accesses to the JTAG Space ................................................................................ 438
Probe Accesses to the ZBbus ................................................................................................. 438
Address Register ................................................................................................................. 439
Data Register ...................................................................................................................... 439
EJTAG Control Register ....................................................................................................... 440
Differences from EJTAG 2.5 (Feb. 22, 2000) Specification ..................................................... 442

Section 16: Reference ......................................................................................................... 446
Internal Register Addresses by Function ............................................................................... 446
BCM1250/BCM1125/H Internal Registers Ordered by Address ................................................ 464
LIST OF FIGURES

Figure 1: BCM1250 Block Diagram .................................................................................................................... 2
Figure 2: BCM1125/H Block Diagram .................................................................................................................. 3
Figure 3: BCM1250 Signals ................................................................................................................................. 7
Figure 4: BCM1125/H Signals ............................................................................................................................... 8
Figure 5: Logical Block Diagram of BCM1250 and BCM1125/H ........................................................................ 9
Figure 6: Internal Control and Status Register Alignment ................................................................................ 11
Figure 7: Decision Tree for Memory Space Address Accesses ........................................................................ 26
Figure 8: Clock Distribution Overview .............................................................................................................. 33
Figure 9: Memory Map ..................................................................................................................................... 35
Figure 10: Per-CPU Interrupt Mapper (replicated for each CPU; x = 0 or 1) .......................................................... 51
Figure 11: Connections to Trace Logic ................................................................................................................ 83
Figure 12: Level 2 Cache Way Disable Access Address .................................................................................... 91
Figure 13: Cache Management Address ........................................................................................................... 95
Figure 14: Memory Controller Block Diagram ................................................................................................ 105
Figure 15: Chip Select Options .......................................................................................................................... 110
Figure 16: Example Single Channel 128MB ....................................................................................................... 112
Figure 17: Example 1GB with two chip selects on one channel ........................................................................ 113
Figure 18: Example 1GB with two chip selects interleaved on one channel ...................................................... 114
Figure 19: Example 1GB with two chip selects interleaved across both channels ........................................... 115
Figure 20: Example 2GB with two chip selects interleaved on one channel ...................................................... 116
Figure 21: Timing Relationships Set by DLLs .................................................................................................. 130
Figure 22: Nominal Windows at 133MHz for First Edge of DQS for Various Settings of [tCrD, tCrDh, tFIFO] .... 131
Figure 23: DMA Buffer ...................................................................................................................................... 148
Figure 24: DMA Descriptor ................................................................................................................................. 149
Figure 25: Packet Spanning Three Buffers ......................................................................................................... 150
Figure 26: DMA Descriptor Ring ....................................................................................................................... 151
Figure 27: DMA Descriptor Chain ..................................................................................................................... 153
Figure 28: Standard and Unaligned Buffer DMA Descriptors ........................................................................... 154
Figure 29: Packet Reception Flow using DMA ASIC Mode ............................................................................... 159
Figure 30: ASIC Mode Address Generation .................................................................................................. 160
Figure 31: Sending the Whole Packet in ASIC Mode ....................................................................................... 161
Figure 32: Sending a Packet Header in ASIC Mode .......................................................................................... 162
Figure 33: Example 1 - TCP checksum a packet...................................................................................181
Figure 34: Example 2 - Preparing an iSCSI packet ........................................................................182
Figure 35: Example 3 - Fragmenting an iSCSI packet........................................................................183
Figure 36: PCI and HyperTransport Organization .............................................................................191
Figure 37: Address Ranges for CPU Access to PCI and HyperTransport ...........................................193
Figure 38: Little Endian System .............................................................................................................201
Figure 39: Match Byte Lane Endian Policy ..........................................................................................202
Figure 40: Match Bit Lane Endian Policy ...............................................................................................203
Figure 41: PCI BAR0 Address Mapping Table ......................................................................................206
Figure 42: Default Host Mode Memory Map from PCI Bus Master ..........................................................209
Figure 43: Memory Map From HyperTransport Device ......................................................................210
Figure 44: Buffers Used for Accesses from the ZBbus to PCI and HyperTransport ................................214
Figure 45: Buffers Used for DMA Accesses from the PCI and HyperTransport ....................................216
Figure 46: PCI Adaptive Retry Parameters ...........................................................................................218
Figure 47: Buffers Used for PCI to HyperTransport Peer-to-Peer Accesses .........................................220
Figure 48: Buffers Used for HyperTransport to PCI Peer-to-Peer Accesses ........................................221
Figure 49: Configuration Space Address ............................................................................................234
Figure 50: HyperTransport Interface Clocks and FIFOs ....................................................................256
Figure 51: Ethernet Interface Block Diagram ......................................................................................265
Figure 52: Ethernet Frame Format .........................................................................................................268
Figure 53: Prepended Header Format ...................................................................................................270
Figure 54: Transmit FIFO Thresholds ....................................................................................................272
Figure 55: Receive FIFO Thresholds .......................................................................................................275
Figure 56: Receive Address Filter ...........................................................................................................278
Figure 57: Receive Channel Selection .....................................................................................................281
Figure 58: Selecting the Channel Offset ...............................................................................................281
Figure 59: MDIO Flows ..........................................................................................................................288
Figure 60: 8-bit Packet FIFO GMII Style ...............................................................................................295
Figure 61: 8-Bit Packet FIFO Encoded Style ..........................................................................................296
Figure 62: 8-Bit Packet FIFO SOP Style .................................................................................................297
Figure 63: 8-Bit Packet FIFO EOP Style .................................................................................................298
Figure 64: 16-Bit GMII Style Packet FIFO ............................................................................................299
Figure 65: 16-Bit Encoded Packet FIFO ...............................................................................................300
Figure 66: UART Interrupt Generation ...............................................................................................325
Figure 67: Synchronous Interface Block Diagram ................................................................................338
LIST OF TABLES

Table 1: ZBbus Agent IDs ............................................................................................................................... 20
Table 2: ZBbus Signals ................................................................................................................................... 20
Table 3: ZBbus Commands ............................................................................................................................. 22
Table 4: ZBbus Level 1 Cache Attributes ........................................................................................................ 23
Table 5: ZBbus Byte Lane Assignments ........................................................................................................ 24
Table 6: ZBbus Data Status Codes ................................................................................................................ 24
Table 7: Operation of Different Reset Sources ............................................................................................. 27
Table 8: Static Configuration Options .......................................................................................................... 27
Table 9: Core and HyperTransport Clock Settings ........................................................................................ 31
Table 10: Overview of BCM1250 Physical Address Map ............................................................................ 34
Table 11: Address Map Details .................................................................................................................... 36
Table 12: System Identification and Revision Register .................................................................................. 42
Table 13: Part Revisions ............................................................................................................................... 42
Table 14: Manufacturing Information Register ............................................................................................ 43
Table 15: System Configuration Register .................................................................................................... 43
Table 16: Scratch Register ............................................................................................................................ 45
Table 17: Mailbox Registers .......................................................................................................................... 46
Table 18: Interrupt Mappings ....................................................................................................................... 47
Table 19: Interrupt Message Format for Writes to interrupt_ldt_set Register .............................................. 49
Table 20: Delivery of HyperTransport Interrupts ........................................................................................ 49
Table 21: Interrupt Registers ....................................................................................................................... 52
Table 22: Interrupt Sources .......................................................................................................................... 52
Table 23: Watchdog Timer Initial Count Registers ....................................................................................... 59
Table 24: Watchdog Timer Current Count Registers .................................................................................... 59
Table 25: Watchdog Timer Configuration Registers ...................................................................................... 59
Table 26: General Timer Initial Count Registers .......................................................................................... 60
Table 27: General Timer Current Count Registers ....................................................................................... 60
Table 28: General Timer Configuration Registers ........................................................................................ 60
Table 29: ZBbus Count Register .................................................................................................................. 61
Table 30: ZBbus Count Compare Registers ................................................................................................ 61
Table 31: System Performance Counter Configuration Registers .................................................................. 61
Table 32: System Performance Counters ..................................................................................................... 62
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>System Performance Counter Sources</td>
<td>62</td>
</tr>
<tr>
<td>34</td>
<td>Bus Watcher Counters</td>
<td>64</td>
</tr>
<tr>
<td>35</td>
<td>Bus Watcher Error Status Register</td>
<td>65</td>
</tr>
<tr>
<td>36</td>
<td>Bus Watcher Error Status Debug Register</td>
<td>65</td>
</tr>
<tr>
<td>37</td>
<td>Bus Watcher Error Data Registers</td>
<td>65</td>
</tr>
<tr>
<td>38</td>
<td>Bus Watcher L2 ECC Counter Register</td>
<td>66</td>
</tr>
<tr>
<td>39</td>
<td>Bus Watcher Memory and I/O Error Counter Register</td>
<td>66</td>
</tr>
<tr>
<td>40</td>
<td>Address Trap Trigger Index Register</td>
<td>68</td>
</tr>
<tr>
<td>41</td>
<td>Address Trap Trigger Debug Register</td>
<td>68</td>
</tr>
<tr>
<td>42</td>
<td>Address Trap Trigger Address Register</td>
<td>68</td>
</tr>
<tr>
<td>43</td>
<td>Address Trap Range Top Address Registers</td>
<td>68</td>
</tr>
<tr>
<td>44</td>
<td>Address Trap Range Base Address Registers</td>
<td>69</td>
</tr>
<tr>
<td>45</td>
<td>Address Trap Configuration Registers</td>
<td>69</td>
</tr>
<tr>
<td>46</td>
<td>Trace Event Register</td>
<td>72</td>
</tr>
<tr>
<td>47</td>
<td>Trace Sequence Control Registers</td>
<td>74</td>
</tr>
<tr>
<td>48</td>
<td>Trace Control Register</td>
<td>76</td>
</tr>
<tr>
<td>49</td>
<td>Trace Buffer Address/Control Bundle</td>
<td>77</td>
</tr>
<tr>
<td>50</td>
<td>Trace Entry Format and Read Order</td>
<td>79</td>
</tr>
<tr>
<td>51</td>
<td>Decode of some TIDs for system revision PERIPH_REV3</td>
<td>81</td>
</tr>
<tr>
<td>52</td>
<td>Encoded Byte Enables for CPU Transactions</td>
<td>82</td>
</tr>
<tr>
<td>53</td>
<td>Addresses for Memory Banks</td>
<td>92</td>
</tr>
<tr>
<td>54</td>
<td>Management Address</td>
<td>95</td>
</tr>
<tr>
<td>55</td>
<td>ECC Diagnostic Operations</td>
<td>98</td>
</tr>
<tr>
<td>56</td>
<td>Level 2 Cache Tag Register</td>
<td>100</td>
</tr>
<tr>
<td>57</td>
<td>Level 2 Cache Settings Register</td>
<td>100</td>
</tr>
<tr>
<td>58</td>
<td>Clock Speed</td>
<td>107</td>
</tr>
<tr>
<td>59</td>
<td>Percent Deltas from Popular DIMM Frequencies</td>
<td>108</td>
</tr>
<tr>
<td>60</td>
<td>Mapping Physical Address To Memory Controller Address</td>
<td>109</td>
</tr>
<tr>
<td>61</td>
<td>Address Bits Used by a Memory Channel</td>
<td>117</td>
</tr>
<tr>
<td>62</td>
<td>Example for 128 MByte CS Region with 4K Rows, 1K Columns</td>
<td>118</td>
</tr>
<tr>
<td>63</td>
<td>Example for 128 MByte CS Region with 4K Rows, 1K Columns, 64 Byte Interleave</td>
<td>118</td>
</tr>
<tr>
<td>64</td>
<td>Example for 128 MByte CS Region with 4K Rows, 1K Columns, 128 Byte Interleave</td>
<td>119</td>
</tr>
<tr>
<td>65</td>
<td>Example for 256 MByte Region with 4K Rows, 1K Columns, two CS, and 128 Byte Interleave</td>
<td>119</td>
</tr>
<tr>
<td>66</td>
<td>Example for 512 MByte Region with 4K Rows, 1K Columns, four CS, and 128 Byte Interleave</td>
<td>119</td>
</tr>
<tr>
<td>67</td>
<td>Example for 128 MByte + 64 MB + 64 MB Mixed_CS Mode</td>
<td>120</td>
</tr>
</tbody>
</table>
Table 68: Supported SDRAMs ...................................................................................................................... 123
Table 69: Commands that can be Issued Through mc_dramcmd Register .................................................. 126
Table 70: Adjustment Percentages and Multiplier for Values of DLL M ........................................................ 128
Table 71: First DQS Window Opening and Closing (Typical)........................................................................ 132
Table 72: Memory Channel Configuration Register on BCM1250 ........................................................... 135
Table 73: Memory Channel Configuration Register on BCM1125/H ............................................................. 136
Table 74: Memory Clock Configuration Register ....................................................................................... 138
Table 75: DRAM Command Register ............................................................................................................ 139
Table 76: DRAM Mode Register ................................................................................................................ 139
Table 77: SDRAM Timing Register ............................................................................................................ 140
Table 78: SDRAM Timing Register 2 ....................................................................................................... 141
Table 79: Chip Select Start Address Register .......................................................................................... 141
Table 80: Chip Select End Address Register .......................................................................................... 141
Table 81: Chip Select Interleave Register ................................................................................................. 142
Table 82: Row Address Bits Select Register .............................................................................................. 142
Table 83: Column Address Bits Select Register .......................................................................................... 143
Table 84: Bank Address Bits Select Register ........................................................................................ 143
Table 85: Chip Select Attribute Register ............................................................................................... 143
Table 86: ECC Test Data Register .............................................................................................................. 144
Table 87: ECC Test ECC Register .............................................................................................................. 144
Table 88: Data Buffer Parameters ........................................................................................................... 148
Table 89: Data Parameters ...................................................................................................................... 148
Table 90: Address Used for ASIC Mode Transfers .................................................................................. 160
Table 91: Ethernet and Serial DMA Configuration Register 0 ................................................................ 163
Table 92: Ethernet and Serial DMA Configuration Register 1 ................................................................ 164
Table 93: Ethernet and Serial DMA Descriptor Base Address Register ................................................... 166
Table 94: ASIC Mode Base Address ......................................................................................................... 166
Table 95: Descriptor Count Register ........................................................................................................ 166
Table 96: Current Descriptor A Debug Register ...................................................................................... 167
Table 97: Current Descriptor B Debug Register ...................................................................................... 167
Table 98: Current Descriptor Address Register ...................................................................................... 167
Table 99: Ethernet Receive Packet Drop Registers (Only if System Revision >= PERIPH_REV3) .............. 168
Table 100: DMA Descriptor First Doubleword ......................................................................................... 169
Table 101: DMA Descriptor Second Doubleword .................................................................................... 169
Table 102: Unaligned Buffer Format DMA Descriptor First Doubleword .................................................. 170
Table 103: Unaligned Buffer Format DMA Descriptor Second Doubleword ..................................................170
Table 104: Status Flags for Ethernet Receive Channel .............................................................171
Table 105: Option Flags for Ethernet Receive Channel ...........................................................172
Table 106: Status Flags for Ethernet Transmit Channel ............................................................172
Table 107: Option Flags for Ethernet Transmit Channel ..............................................................173
Table 108: Status Flags for Synchronous Serial Receive Channel ....................................................174
Table 109: Option Flags for Synchronous Serial Receive Channel ....................................................174
Table 110: Status Flags for Synchronous Serial Transmit Channel ...................................................174
Table 111: Option Flags for Synchronous Serial Transmit Channel ...................................................174
Table 112: Result in memory of appending the result CRC[31:0] ...........................................................179
Table 113: Example CRC configurations .........................................................................................180
Table 114: Data Mover Descriptor Base Address Register ................................................................184
Table 115: Debug Data Mover Descriptor Base Address Register .......................................................184
Table 116: Data Mover Descriptor Count Register .............................................................................185
Table 117: Data Mover Current Descriptor Address .............................................................................185
Table 118: Data Mover CRC Definition Registers (Only if System Revision >= PERIPH_REV3) ..............185
Table 119: Data Mover CRC/Checksum Definition Registers (Only if System Revision >= PERIPH_REV3) 186
Table 120: Data Mover Channel Partial Result Registers (Only if System Revision >= PERIPH_REV3) ....186
Table 121: Data Mover Descriptor First Doubleword ..............................................................................187
Table 122: Data Mover Descriptor Second Doubleword ...........................................................................188
Table 123: PCI Base Address Register Use ..........................................................................................205
Table 124: Adaptive Retry Delay ...........................................................................................................218
Table 125: Error Routing Registers .....................................................................................................229
Table 126: PCI CSR Access Rules .........................................................................................................233
Table 127: PCI Interface Configuration Header (Type 0) .........................................................................236
Table 128: PCI Command Register - Offset 4 Bits [15:0] ........................................................................240
Table 129: PCI Status Register - Offset 4 Bits [31:16] ............................................................................240
Table 130: PCI Latency Timer - Offset 0C Bits [15:8] .............................................................................241
Table 131: PCI Cache Line Size - Offset 0C Bits [7:0] ............................................................................241
Table 132: PCI Timeout Register - Offset 40 Bits [15:0] ....................................................................241
Table 133: PCI Feature Control Register - Offset 40 Bits [31:16] ............................................................242
Table 134: PCI BAR0 Map Table Entry - Offset 44 – 80 .................................................................242
Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0] .....................................242
Table 136: PCI INTA Control Register - Offset 90 Bits [31:0] ...............................................................243
Table 137: PCI Read Host Register - Offset 94 Bits [31:0] .................................................................243
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>138</td>
<td>PCI Adaptive Extend Register - Offset 98 Bits [31:0]</td>
<td>243</td>
</tr>
<tr>
<td>139</td>
<td>PCI Bypass Control Register - RevId &gt;= 3 Offset A8 Bits [31:0]</td>
<td>244</td>
</tr>
<tr>
<td>140</td>
<td>HyperTransport Configuration Header (Type 1)</td>
<td>245</td>
</tr>
<tr>
<td>141</td>
<td>HyperTransport Bridge Command Register - Offset 4 Bits [15:0]</td>
<td>247</td>
</tr>
<tr>
<td>142</td>
<td>HyperTransport Bridge Primary (ZBbus) Status Register - Offset 4 Bits [31:16]</td>
<td>248</td>
</tr>
<tr>
<td>143</td>
<td>HyperTransport Bridge Secondary (HT) Status Register - Offset 1C Bits [31:16]</td>
<td>248</td>
</tr>
<tr>
<td>144</td>
<td>HyperTransport Bridge Control Register - Offset 3C Bits [31:16]</td>
<td>249</td>
</tr>
<tr>
<td>145</td>
<td>HyperTransport Command Register - Offset 40 Bits [31:16]</td>
<td>249</td>
</tr>
<tr>
<td>146</td>
<td>HyperTransport Link Control Register - Offset 44 Bits [15:0]</td>
<td>250</td>
</tr>
<tr>
<td>147</td>
<td>HyperTransport Link Configuration Register - Offset 44 Bits [31:16]</td>
<td>251</td>
</tr>
<tr>
<td>148</td>
<td>HyperTransport Link Frequency Register - Offset 48 Bits [15:8]</td>
<td>251</td>
</tr>
<tr>
<td>149</td>
<td>HyperTransport SRI Command Register - Offset 50 Bits [31:16]</td>
<td>252</td>
</tr>
<tr>
<td>150</td>
<td>HyperTransport Isochronous BAR - Offset 5C Bits [31:0]</td>
<td>252</td>
</tr>
<tr>
<td>151</td>
<td>HyperTransport Isochronous Ignore Mask - Offset 60Bits [31:0]</td>
<td>253</td>
</tr>
<tr>
<td>152</td>
<td>HyperTransport Error Control Register - Offset 68 Bits [23:0]</td>
<td>253</td>
</tr>
<tr>
<td>153</td>
<td>HyperTransport Error Status Register - Offset 68 Bits [31:24]</td>
<td>254</td>
</tr>
<tr>
<td>154</td>
<td>HyperTransport SRI Transmit Control Register - Offset 6C Bits [23:16]</td>
<td>254</td>
</tr>
<tr>
<td>155</td>
<td>HyperTransport SRI Data Buffer Allocation Register - Offset 6C Bits [15:0]</td>
<td>254</td>
</tr>
<tr>
<td>156</td>
<td>HyperTransport Additional Status Register - Offset 70</td>
<td>255</td>
</tr>
<tr>
<td>157</td>
<td>HyperTransport SRI Transmit Buffer Count Max Register - Offset C8 Bits [31:0]</td>
<td>255</td>
</tr>
<tr>
<td>158</td>
<td>HyperTransport Diagnostic Receive CRC Expected - Offset DC</td>
<td>255</td>
</tr>
<tr>
<td>159</td>
<td>HyperTransport Diagnostic Receive CRC Received - Offset F0</td>
<td>255</td>
</tr>
<tr>
<td>160</td>
<td>Ethernet Frame Fields</td>
<td>269</td>
</tr>
<tr>
<td>161</td>
<td>Transmission Error Conditions</td>
<td>273</td>
</tr>
<tr>
<td>162</td>
<td>Receiver Error Conditions</td>
<td>276</td>
</tr>
<tr>
<td>163</td>
<td>Ethernet Type Mappings</td>
<td>282</td>
</tr>
<tr>
<td>164</td>
<td>Back Pressure Methods in Half-Duplex Operation</td>
<td>284</td>
</tr>
<tr>
<td>165</td>
<td>Pause Frame Options</td>
<td>285</td>
</tr>
<tr>
<td>166</td>
<td>MAC to PHY Management Protocol</td>
<td>289</td>
</tr>
<tr>
<td>167</td>
<td>RMON Counters</td>
<td>289</td>
</tr>
<tr>
<td>168</td>
<td>BCM1125 Ethernet/Fifo Pin Usage</td>
<td>292</td>
</tr>
<tr>
<td>169</td>
<td>BCM1250 Ethernet/Fifo Pin Usage</td>
<td>293</td>
</tr>
<tr>
<td>170</td>
<td>Codes for GMII Packet FIFO Mode</td>
<td>295</td>
</tr>
<tr>
<td>171</td>
<td>Codes for 8-Bit Encoded Bypass Mode</td>
<td>296</td>
</tr>
<tr>
<td>172</td>
<td>Codes for 8-Bit SOP Packet FIFO</td>
<td>297</td>
</tr>
</tbody>
</table>
Table 208: DUART Input Port Change Status Register for Channel A ......................................................... 331
Table 209: DUART Input Port Change Status Register for Channel B ......................................................... 331
Table 210: DUART Output Port Control Register ...................................................................................... 331
Table 211: DUART Per Channel Output Control Registers .......................................................................... 332
Table 212: DUART Aux Control Register ...................................................................................................... 332
Table 213: DUART Per Channel Aux Control Registers ............................................................................... 332
Table 214: DUART Interrupt Status Register ............................................................................................. 333
Table 215: DUART Channel A Only Interrupt Status Register ...................................................................... 333
Table 216: DUART Channel B Only Interrupt Status Register ...................................................................... 333
Table 217: DUART Interrupt Mask Register ................................................................................................. 334
Table 218: DUART Channel A Only Interrupt Mask Register ....................................................................... 334
Table 219: DUART Channel B Only Interrupt Mask Register ....................................................................... 334
Table 220: DUART Output Port Set Register ................................................................................................. 335
Table 221: DUART Output Port Clear Register ............................................................................................. 335
Table 222: DUART Output Port RTS Register ............................................................................................... 335
Table 223: Synchronous Serial Interface Signal Names ............................................................................... 339
Table 224: Synchronous Serial Interface GPIO Pins ..................................................................................... 339
Table 225: Sequencer Table Entries .............................................................................................................. 341
Table 226: HDLC Frame Structure .................................................................................................................. 344
Table 227: Option Flags for Synchronous Serial Transmit Channel ............................................................. 345
Table 228: Status Flags for Synchronous Serial Receive Channel ............................................................... 349
Table 229: Recommended Line Interface Settings for Loopback Mode 2 ..................................................... 353
Table 230: RMON Counters .......................................................................................................................... 353
Table 231: Serial Mode Configuration Register ............................................................................................ 354
Table 232: Synchronous Serial Clock Source and Line Interface Mode Register ......................................... 354
Table 233: Synchronous Serial Command Register (Write-only) ................................................................. 355
Table 234: Serial Write Threshold Register ................................................................................................. 356
Table 235: Serial Transmit Read Threshold Register .................................................................................... 356
Table 236: Serial Receive Read Threshold Register ..................................................................................... 356
Table 237: Serial Minimum Frame Size Register .......................................................................................... 356
Table 238: Serial Maximum Frame Size Register .......................................................................................... 357
Table 239: Serial DMA Enable Registers ..................................................................................................... 357
Table 240: Synchronous Serial Status Register ........................................................................................... 357
Table 241: Serial Status Debug Register ...................................................................................................... 358
Table 242: Serial Interrupt Mask Register ................................................................................................... 358
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>243</td>
<td>Serial Address Mask Register</td>
<td>359</td>
</tr>
<tr>
<td>244</td>
<td>Serial Address Match Register</td>
<td>359</td>
</tr>
<tr>
<td>245</td>
<td>Sequencer Table Entries</td>
<td>359</td>
</tr>
<tr>
<td>246</td>
<td>Serial RMON Counters</td>
<td>360</td>
</tr>
<tr>
<td>247</td>
<td>Byte Lanes for the Generic Bus</td>
<td>363</td>
</tr>
<tr>
<td>248</td>
<td>Generic Bus Timing Parameters</td>
<td>366</td>
</tr>
<tr>
<td>249</td>
<td>Burst Cycle Summary</td>
<td>372</td>
</tr>
<tr>
<td>250</td>
<td>Generic Bus Configuration for Each Boot Mode</td>
<td>373</td>
</tr>
<tr>
<td>251</td>
<td>Generic Bus Region Configuration Registers</td>
<td>375</td>
</tr>
<tr>
<td>252</td>
<td>Generic Bus Region Size Registers</td>
<td>375</td>
</tr>
<tr>
<td>253</td>
<td>Generic Bus Region Base Address Registers</td>
<td>376</td>
</tr>
<tr>
<td>254</td>
<td>Generic Bus Region Timing 0 Registers</td>
<td>376</td>
</tr>
<tr>
<td>255</td>
<td>Generic Bus Region Timing 1 Registers</td>
<td>377</td>
</tr>
<tr>
<td>256</td>
<td>Generic Bus Interrupt Status Register</td>
<td>377</td>
</tr>
<tr>
<td>257</td>
<td>Generic Bus Error Data Register 0</td>
<td>378</td>
</tr>
<tr>
<td>258</td>
<td>Generic Bus Error Data Register 1</td>
<td>378</td>
</tr>
<tr>
<td>259</td>
<td>Generic Bus Error Data Register 2</td>
<td>378</td>
</tr>
<tr>
<td>260</td>
<td>Generic Bus Error Data Register 3</td>
<td>378</td>
</tr>
<tr>
<td>261</td>
<td>Generic Bus Error Address Register 0</td>
<td>379</td>
</tr>
<tr>
<td>262</td>
<td>Generic Bus Error Address Register 1</td>
<td>379</td>
</tr>
<tr>
<td>263</td>
<td>Generic Bus Error Parity Register</td>
<td>379</td>
</tr>
<tr>
<td>264</td>
<td>Output Drive Control Register 0</td>
<td>379</td>
</tr>
<tr>
<td>265</td>
<td>Output Drive Control Register 1</td>
<td>380</td>
</tr>
<tr>
<td>266</td>
<td>Output Drive Control Register 2</td>
<td>381</td>
</tr>
<tr>
<td>267</td>
<td>Output Drive Control Register 3</td>
<td>381</td>
</tr>
<tr>
<td>268</td>
<td>Source for PCMCIA Card Enable Signals</td>
<td>386</td>
</tr>
<tr>
<td>269</td>
<td>PCMCIA 3.3V and 5V VCC Power Enable Truth Table</td>
<td>387</td>
</tr>
<tr>
<td>270</td>
<td>PCMCIA VPP Power Enable Truth Table</td>
<td>388</td>
</tr>
<tr>
<td>271</td>
<td>Example Flash Card AC Specs</td>
<td>391</td>
</tr>
<tr>
<td>272</td>
<td>Example Generic Bus Timing parameters</td>
<td>393</td>
</tr>
<tr>
<td>273</td>
<td>PCMCIA Configuration Register</td>
<td>394</td>
</tr>
<tr>
<td>274</td>
<td>PCMCIA Status Register</td>
<td>395</td>
</tr>
<tr>
<td>275</td>
<td>GPIO Pins and Alternate Uses</td>
<td>398</td>
</tr>
<tr>
<td>276</td>
<td>GPIO Edge Clear Register</td>
<td>399</td>
</tr>
<tr>
<td>277</td>
<td>GPIO Interrupt Type Register</td>
<td>399</td>
</tr>
</tbody>
</table>
Table 278: GPIO Read Register.................................................................................................................. 399
Table 279: GPIO Input Invert Control Register.......................................................................................... 399
Table 280: GPIO Glitch Filter Select Register............................................................................................. 400
Table 281: GPIO Direction Register.......................................................................................................... 400
Table 282: GPIO Pin Clear Register ............................................................................................................ 400
Table 283: GPIO Pin Set Register ............................................................................................................. 400
Table 284: Other Pins that can be Used as General Inputs or Outputs ....................................................... 401
Table 285: Supported SMBus Transfer Types............................................................................................... 406
Table 286: Command/Address Options ...................................................................................................... 408
Table 287: Write Data Options ................................................................................................................. 408
Table 288: Read Data Options ................................................................................................................. 409
Table 289: SMBus Clock Frequency Registers............................................................................................ 416
Table 290: SMBus Command Registers .................................................................................................... 416
Table 291: SMBus Control Registers ...................................................................................................... 416
Table 292: SMBus Status Registers......................................................................................................... 417
Table 293: SMBus Data Registers ........................................................................................................... 417
Table 294: SMBus Extra Data Registers.................................................................................................. 417
Table 295: SMBus Packet Error Check Registers....................................................................................... 418
Table 296: SMBus Start and Command Registers SMBus Mode ............................................................. 418
Table 297: SMBus Start and Command Registers Extended Mode .......................................................... 419
Table 298: JTAG Signals............................................................................................................................ 422
Table 299: JTAG Instructions .................................................................................................................. 424
Table 300: JTAG Device ID Register ........................................................................................................ 425
Table 301: JTAG Wafer ID Register .......................................................................................................... 426
Table 302: System Control Scan Chain ................................................................................................... 428
Table 303: Performance Counter Scan Chain .......................................................................................... 430
Table 304: Trace Control Scan Chain ...................................................................................................... 431
Table 305: Trace Current Count Scan Chain ............................................................................................ 431
Table 306: Ring Oscillator Scan Chain ................................................................................................... 432
Table 307: CPU and Probe Accesses .................................................................................................... 436
Table 308: JTAG Address Register Scan Chain ....................................................................................... 439
Table 309: Data Register Scan Chain .................................................................................................... 439
Table 310: EJTAG Control Register ........................................................................................................ 440
Table 311: Internal Register Addresses by Function.................................................................................. 446
Table 312: Internal Registers Ordered by Address .................................................................................. 464
Section 1: Introduction

THE SiBYTE BROADBAND PROCESSOR FAMILY

The SiByte Broadband Processors form a family of high performance system-on-a-chip parts targeted at network centric tasks. Examples include:

- In-line packet processing
- Exception processing
- Switch control and management
- Higher layer switching and filtering
- Protocol conversion (VoIP Gateway)
- Network appliances (file servers, web cache, print servers)
- VPN access, firewalls, gateways

The different members of the family target different performance points and applications, while retaining software compatibility. This User Manual covers the dual-processor BCM1250, and the uni-processor BCM1125 and BCM1125H parts.

All the parts use the SB-1 CPU core. This is a high performance implementation of the standard MIPS64 instruction set architecture. The core supports a 4-issue enhanced skew pipeline and can dispatch up to two memory and two ALU (Integer, Floating point or MIPS-3D) instructions per cycle.
THE BCM1250

The core of the BCM1250 is an on-chip multiprocessor (CMP) system consisting of two Broadcom SB-1 high performance MIPS64 CPUs, a shared 512K L2 cache and a DDR SDRAM memory controller. Three integrated 10/100/1000 Ethernet MACs enable easy interfacing to LANs. The three network interfaces can also be operated together to give two 16 bit wide interfaces that can run full-duplex at OC-48 rates. Two serial ports are provided for use as UARTs or for WAN connections at up to T3/OC-1 rates (55Mbit/s). High speed I/O is provided using the HyperTransport (formerly called “Lightning Data Transport” or “LDT”) I/O fabric and a 66 MHz PCI (rev 2.2) local bus. To enable low chip count systems the BCM1250 includes a configurable generic bus that allows glueless connection of a boot ROM or flash memory and simple I/O peripherals. On-chip debug, trace and performance monitoring functions assist both hardware and software designers in debugging and tuning the system. The system can be run either big endian or little endian.

Figure 1: BCM1250 Block Diagram
THE BCM1125 AND BCM1125H

The core of the BCM1125 and BCM1125H is a uniprocessor system consisting of a Broadcom SB-1 high performance MIPS64 CPU, a 256K L2 cache and a DDR SDRAM memory controller. Two integrated 10/100/1000 Ethernet MACs enable easy interfacing to LANs. The two network interfaces can also be operated together to give a 16 bit wide interface that can run full-duplex at OC-48 rates. Two serial ports are provided for use as UARTs or for WAN connections at up to T3/OC-1 rates (55Mbit/s). High speed I/O is provided on the BCM1125H using the HyperTransport (formerly called “Lightning Data Transport” or “LDT”) I/O fabric. Both the BCM1125 and BCM1125H have a 66 MHz PCI (rev 2.2) local bus. To enable low chip count systems both parts include a configurable generic bus that allows glueless connection of a boot ROM or flash memory and simple I/O peripherals. On-chip debug, trace and performance monitoring functions assist both hardware and software designers in debugging and tuning the system. The system can be run either big endian or little endian.

Figure 2: BCM1125/H Block Diagram

AUDIENCE

This User Manual includes information that is needed when writing software for the BCM1250, BCM1125 or BCM1125H. It provides a system overview for Systems and Hardware Designers. This User Manual is common across the parts, reflecting their software compatibility. Each part has an individual Data Sheet containing detailed hardware information. (Note that some hardware details of the BCM1250 that were in previous versions of the User Manual are now found only in the Data Sheet.)
OTHER DOCUMENTATION

Most of the example designs and Application Notes for the BCM1250 also apply to the BCM1125 and BCM1125H. Documentation useful to users of all the devices includes:

- SB-1 Core Processor User Manual (SB-1-UM00-R)
- BCM1250 Board Routability Considerations Application Note (1250-AN101-R)
- BCM1250 Generic Bus Interface to ATA/ATAPI PIO Mode 3 (IDE) Hard Disk Application Note (1250-AN201-R)
- BCM1250 DRAM Support Application Note (1250-AN302-R)
- BCM1250 Power Supply Application Note (1250-AN4xx-R)
- BCM1250 Design Considerations for Fast Memory Systems Application Note (1250-AN5xx-R)
- BCM1250 Big Memory System Application Note (1250-AN6xx-R)
- BCM1250 Generic Bus Interface to 3-Volt Intel StrataFlash Memory (1250-AN7xx-R)
- HyperTransport I/O Link Specification Revision 1.03 from the HyperTransport Technology Consortium
- PCI Specification - Revision 2.2
- MIPS64 Architecture for Programmers: http://www.mips.com/publications/index.html
  - Volume I: Introduction to the MIPS64 Architecture (MD00083)
  - Volume II: The MIPS64 Instruction Set (MIPS MD00087)
  - Volume III: Privileged Resource Architecture (MIPS MD00091)
  - Volume IV-b: The MDMX ASE to MIPS64 (MIPS MD00095)
  - Volume IV-c: The MIPS-3D ASE to MIPS64 (MIPS MD00099)
- MIPS EJTAG Specification revision 2.5
- Corelis Application Note #00-117 BCM1250 EJTAG Connector

Documentation specific to the BCM1250 device includes:

- BCM1250 Data Sheet (1250-DS02-R)
- BCM91250A Evaluation Card for the BCM1250 Product Brief (91250A-PB03-R)
- BCM91250E PCI Evaluation Card for the BCM1250 Product Brief (91250E-PB01-R)
- BCM1250 Package Thermal Report (available on request)

Documentation specific to the BCM1125 and BCM1125H includes:

- BCM1125/H Data Sheet (1125H-DS01-R)
- BCM91125E PCI Evaluation Card for the BCM1125/H Product Brief (91125E-PB01-R)
- BCM1125/H Package Thermal Report (available on request)
TERMINOLOGY

**Numbers** used in data fields of this document follow the verilog convention of giving the field size in decimal, followed by a quote ('), a character representing the base (b for binary, d for decimal or h for hexadecimal) and the number. Thus the decimal number 250 in a 12 bit field will be written as 12'h0FA or 12'b00001111010.

**Block** or **Cache Block** is used to refer to 32 bytes of data with a base address aligned to a 32 byte boundary (the low 5 address bits are zero).

**Line** or **Cache Line** is used to refer to the 32 bytes of memory in a cache that holds a cache block.

**Physical Addresses** are written as 40 bit hexadecimal numbers spaced with underbars (_) for legibility. For example 01_2345_6789.

**Virtual Addresses** are only meaningful within a CPU. They are written as 64 bit hexadecimal numbers spaced with underbars (_) for legibility. For example 0123_4567_89AB_CDEF.

**UNPREDICTABLE** operations or behaviors can give arbitrary results, that may differ from device to device or as a function of time on the same device. However, the system will continue to operate and any section that has been placed in an UNPREDICTABLE state can be restored to deterministic operation under software control.

**UNDEFINED** operations or behaviors can result in any outcome from no change in the state of the system to creating an environment in which the system no longer continues to operate. Following an UNDEFINED operation assertion of the reset signal may be required to restore deterministic operation.

**Register Definition Tables** include the register name and physical address in their titles. The table shows all bits that are implemented. Registers are all allocated a 64 bit field, any bits that are not implemented will return UNPREDICTABLE data if read. The default values show the register value after a system reset. If the default is given as "ext" the value following reset depends on an external pin setting. If no defaults are given the field is UNPREDICTABLE after reset.

**Reserved** register bits and fields are not used in the current device, but may be used in future versions. They must be written with zeros to get the documented behavior and the behavior is UNDEFINED if they are written with any other value. Future parts may have well defined operation with these bits set.

**Not Implemented** register bits and fields are not used in the current device and do not have any implementation supporting them. Reads will return UNPREDICTABLE values, writes will have no effect. There is no need to write these bits (for example if the top 32 bits of a register are Not Implemented then a store word can be used to set the lower bits).

**Read Only** registers and bit fields provide status information should only be read by the CPUs. Writes to these fields are ignored.

**Write Only** registers and locations should be written with data, but cannot be read back. Unless otherwise specified, reads will return UNPREDICTABLE data.
PERIPH_REV3 refers to the third major revision of the peripherals. These are used in the BCM1125, BCM1125H (All steppings) and later production versions of the BCM1250 (stepping C0 and later, also called “Pass 3”). This manual describes both the PERIPH_REV2 and PERIPH_REV3 revisions and tags information that only applies to PERIPH_REV3. The revision number in the system_revision register can be used by software to determine the available features. Note that code written for the earlier parts will work on PERIPH_REV3 parts (provided that it correctly observes Reserved fields). An earlier version of the manual should be used for the prototype parts marked BCM12500 which lack some of the peripheral functionality.

Broadcom Use Only registers and operations are intended for use by Broadcom in testing the device. In most cases this manual does not describe these options in sufficient detail for them to be safely used. Incorrect settings of these registers or inappropriate use of operations may cause the system to behave in UNDEFINED ways, or require the system to be power cycled to restore operation.
Section 2: Signal Overview

BCM1250 SIGNAL GROUPS

The signal pins of the BCM1250 can be divided into functional groups, primarily related to the peripheral to which they are attached. Hardware designers should refer to the BCM1250 Data Sheet for full pinout and timing details.

<table>
<thead>
<tr>
<th>Group</th>
<th>Pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>216 Pins</td>
</tr>
<tr>
<td>HyperTransport</td>
<td>46 Pins</td>
</tr>
<tr>
<td>PCI</td>
<td>58 Pins</td>
</tr>
<tr>
<td>MAC</td>
<td>86 Pins</td>
</tr>
<tr>
<td>Generic Bus</td>
<td>46 Pins +4 shared</td>
</tr>
<tr>
<td>Serial Ports</td>
<td>16 Pins</td>
</tr>
<tr>
<td>GPIO</td>
<td>15 Pins +4 shared</td>
</tr>
<tr>
<td>SMBus</td>
<td>4 Pins</td>
</tr>
<tr>
<td>JTAG</td>
<td>5 Pins</td>
</tr>
<tr>
<td>Clock / Misc</td>
<td>9 Pins</td>
</tr>
<tr>
<td>Spare/No Connect</td>
<td>4 Pins</td>
</tr>
</tbody>
</table>

Figure 3: BCM1250 Signals
BCM1125/H Signal Groups

The signal pins of the BCM1125 and BCM1125H can be divided into functional groups, primarily related to the peripheral that they are attached to. Hardware designers should refer to the BCM1125/H Data Sheet for full pinout and timing details.

Figure 4: BCM1125/H Signals
Section 3: System Overview

INTRODUCTION

A logical block diagram of the BCM1250 and BCM1125/H family is shown in Figure 5. This figure does not exactly match the implementation details, but it gives a more useful model for programmers and system designers to use. The system is based around the ZBbus, a high speed split-transaction multiprocessor bus. It connects the CPU(s), the level 2 cache (L2), the memory controller, two I/O bridges and the System Control and Debug unit (SCD).

Figure 5: Logical Block Diagram of BCM1250 and BCM1125/H
This section gives an overview of the whole system. Each of the system elements are described in detail in later sections.

The ZBbus is the on-chip multiprocessor system bus:

- Data is 256 bits wide, running at half the CPU frequency. (For an 800 MHz part this gives (800/2)*256 = 102.4 Gbit/s bus bandwidth).
- The address is 40 bits (matching the CPU physical address and the HyperTransport I/O fabric address).
- The separate address and data sections are arbitrated for independently, and operate as a split transaction bus.
- All transactions are tagged, data responses can come back in any order.
- The cache block ownership protocol (MESI) ensures memory coherence is maintained.
- Individual buffer-full indications from each bus target allow selective flow-control by requestors.

The processors are Broadcom SB-1 CPUs implementing the MIPS64 architecture. These are described in detail in the SB-1 User Manual. On the BCM1250 the two CPUs are identical in all respects apart from the processor number that will be read from the Processor Identification register (CP0 register 15). The reset logic in the System Control and Debug unit (SCD) is different for the two processors. Following a system reset only CPU 0 will be brought out of reset, it can then setup the system and release CPU 1 when ready. In the BCM1125/H parts the processor is CPU 0 and will report that it is a uniprocessor in the Processor Identification Register.

The Level 2 cache (L2) is organized slightly differently than L2 caches in other systems. It is shared by the processor(s) and any I/O DMA masters. It is best understood as a cache on the front of the memory (as shown in Figure 5), rather than by using the traditional model where the L2 is associated with a CPU. All memory accesses are checked in the L2. The bus includes a signal to indicate that the data should be allocated in the L2 cache on an L2 miss, this signal may be used by DMA masters to write data to the L2 so it can quickly be accessed by the CPU (this has to be done in a controlled way to avoid disrupting the normal gains of having a L2 cache).

The I/O bridges isolate the peripherals from the bus, and implement the bus and coherence protocols. They include buffering to allow multiple outstanding I/O requests, and support for DMA masters. The PCI and HyperTransport expansion buses share an I/O bridge, so any peer-to-peer traffic (between PCI and HyperTransport devices) is hidden from the ZBbus.

The part can be run as a big endian system or a little endian system. This is set by an input that is read at reset time. Internally the few data paths that need to will swap byte lanes, however the system is designed to have as few changes as possible. The CPU endian mode bit (the BE bit in the CP0 Config register) will always reflect the endian mode of the system. The interface to the PCI and HyperTransport expansion buses includes byte lane swappers that can be used if required to interface to these devices.
**INTERNAL REGISTERS**

There are a large number of internal registers. Their definitions are given in the sections of this manual that describe their use. Section: "Internal Register Addresses by Function" on page 445 has a summary of all the register addresses. Each register is 1 byte, 2 bytes, 4 bytes or 8 bytes. However, even if a smaller size is implemented registers are always allocated in an aligned 8 byte (64 bit) field. Most registers can be read or written using a double-word (8 bytes), word (4 bytes), half-word (2 byte) or single byte access. However, some of the registers can only be written with their full width (in these registers a wider write will work, but a narrower write will leave the register value UNPREDICTABLE). All registers can be read with any size read, if the size requested is larger than the size that is implemented in the register then the extra bits will be filled with UNPREDICTABLE data (unless otherwise documented).

The address given in this manual for all registers is aligned to a double-word (i.e. the final hex digit is a 0 or an 8), reflecting the field size rather than the register size. As is illustrated in Figure 6 the base address may be used for all access widths in a little endian system.

In a big endian system, if the access to the registers is as a double-word the base address should be used, for a word access the address is the base address plus 4, for a half-word access use the base address plus 6 and for a byte access use the base address plus 7. For example, consider a register with 1 valid byte and base address 1234_0000. Its double-word access address is 1234_0000; word access address is 1234_0004; half-word access address is 1234_0006; and single byte access address is 1234_0007. This fits with the big endian model for data position in double-word fields.

![Figure 6: Internal Control and Status Register Alignment](image)

The internal registers should be referenced by the CPUs in uncacheable space, so transactions will never be wider than 8 bytes (and will never span more than a single 8 byte aligned range).

One source for initialization errors happens when a bit in a register is defined to have an UNPREDICTABLE (1’bx) value after reset. It is possible that on all parts used in a small prototype build the bit has the same value, but on a few parts in a larger build it has the other value. Thus if it is not initialized the error may only be discovered late in the testing process.
When a register is marked as cleared by a read it will be cleared by any read to it. Care should therefore be taken to ensure that a compiler or debugger does not split an access to the register into multiple reads. For example if a 64-bit access is split into two 32-bit reads, the first of these reads will get valid data and clear the register and the second will get zeros (compilers will probably do this if they are configured to only use 32-bit registers). Similarly, the value of these registers cannot be read from a byte-by-byte memory dump. Most (but not all) of the registers with read side effects have an alias (with _debug appended to the register name) that does not have the side effect for a debugger to use.

**COHERENCE**

The system maintains coherence for all memory operations including DMA. This coherence means that a read reference to memory space will return the most up to date version of the data. The L2 cache, processors, memory controller and I/O bridges all cooperate to deliver the correct version of each cache block. The coherency is managed by hardware, other than selecting the correct cachability mode no action is needed by software.

Coherence works at cache line granularity, for each cache block (aligned 32 byte block of memory) there is at all times an owner. The default owner of a block is either the L2 cache or main memory, they work together to service bus requests. Blocks that are being shared between bus agents are owned by the L2 or memory. Any agent that wishes to modify a cache line must become exclusive owner. When it makes the request the current owner of the block will give it up and any shared copies will be invalidated. Although the data transfer may occur at any time following the request, the ownership transfer is deterministic. If an agent receives ownership of a block it is possible that it will lose ownership before it has received the data; in this case it can perform one operation on the block before passing the data to the new owner, this is required to avoid live-lock when two agents are trying to write to the same block. Coherent memory references check L1 and L2 tags and the partial-line merge buffer in the I/O bridges at bus speed. If a block is exclusively owned, a request for that line can only be serviced by the owner, if the block is shared it can reside in more than one location, but will be returned from the L2 or memory (as the default owner). When block ownership is transferred (either from one exclusive owner to another, or from an exclusive owner to the default) the data transfer is snooped by the L2/memory so the new owner gets a clean copy of the line.

A line in the level 1 cache (in the processor) will be in one of six states: Invalid, Shared, Exclusive Clean, Exclusive Dirty, Non-coherent Clean or Non-coherent Dirty. Stores are permitted to all valid lines except for Shared ones, which must be upgraded to Exclusive before a store is possible. Dirty lines will be written back if the line is evicted to make way for a new fill, if evicted due to another device requesting the line (an intervention), or if explicitly written back by software (either a CACHE instruction forcing write-back, or a PREFetch instruction with the nudge hint).

Lines will only be put in the L1 cache Non-coherent if their TLB entry has the cacheable non-coherent cache attribute. These lines will not play a part in the ZBus coherence protocol, and will need careful management by software. In general this should only be done for accessing memory that is genuinely non-coherent, which is true for any memory attached on the PCI bus or HyperTransport fabric. Memory attached on any of these interfaces will not be cached in the L2 cache. Memory attached on these is outside the coherency domain: they are I/O connections and do not carry coherency messages, so some other device could update the memory without the CPU being informed that its cached copy is now out of date.
Blocks with main memory addresses should be marked coherent in the CPU. If these blocks are not shared there is no difference in performance between marking them cacheable coherent and cacheable non-coherent, but there is a much higher chance of unexpected behavior if the block is marked non-coherent (for example due to false sharing or missing cache flushes before DMA operations). When these blocks are used for DMA operations the system performance will generally be better if the blocks are marked coherent and the coherence protocol is used to fetch modified data from the CPU cache when it is needed by the DMA engine. Having the blocks marked non-coherent and using CACHE operations to flush the cache prior to initiating DMAs will give a lower system performance as well as being prone to programming errors.

Lines in the L2 cache are in one of three states: Invalid, Clean or Dirty. All cacheable accesses to the address range controlled by the memory controller check the L2 cache.

The coherence mechanism will be circumvented by doing uncached or cacheable non-coherent accesses into memory that has previously been cacheable and coherent. If this is done the results are UNPREDICTABLE. If areas of memory are never cacheable and coherent then it is the responsibility of software to track ownership. The coherence mechanism can also be circumvented if software uses the Invalidate CACHE operation on a block that is held exclusive in the L1 cache. If a snoop request is received on the bus at the same time this instruction executes, the snoop will hit on the line in the L1 but the line is invalidated before it can be evicted. The CPU will respond by returning UNPREDICTABLE data marked with a fatal bus error. There is no problem with using the Writeback&Invalidate CACHE operation.

The generic bus section of memory may be mapped cacheable coherent. The I/O bridge will act as default owner in the MESI protocol (it behaves as the L2/Memory does for memory addresses). The CPUs are able to take exclusive ownership of cache blocks from memory on the generic bus, if ownership is transferred the new owner will acquire the block clean and if needed a writeback will be done to the generic bus. This allows the cache attribute for kseg0 to be set to cacheable coherent to cover both main memory and the Boot ROM and allows for RAMs on the generic bus with no special software management.

The PCI/HyperTransport space should not be mapped cacheable coherent. The I/O bridge will respect an ownership assertion by one of the CPUs (so there will not be two replies to a read request) but it will not act as default owner and will not copy back dirty data on ownership changes. Changes to data may therefore be lost leading to UNPREDICTABLE behavior. In error cases the behavior of cacheable coherent accesses through I/O Bridge 0 is UNDEFINED.

The I/O bridges provide the entry point into the coherent domain for any DMA traffic (from the on-chip network interfaces or PCI/HyperTransport master devices). The bridges will check the address being accessed. For any memory address cacheable coherent accesses will be used, and partial line writes will cause a read (exclusive)-modify-write cycle. For any address that is not to memory uncacheable accesses are done and partial line accesses will result in only some of the byte enables being set.
ORDERING RULES AND DEVICE DRIVERS

The interaction between the ordering rules imposed by the SB-1 CPU, the ZBbus and the peripheral agents simplifies device programming in most situations. In these situations it is the ordering between cacheable coherent memory and uncacheable peripheral registers that is important.

The five important rules for the SB-1 are:

1. Cacheable coherent stores are visible in program order.
2. Uncached loads and stores will issue to the ZBbus in program order, and will only issue when all earlier cacheable coherent stores are visible.
3. The CPU will stall until data returns when it needs the result of an uncacheable load (or a cacheable load that has missed in the cache).
4. Uncached stores will not issue to the ZBbus if there are any uncached loads outstanding.
5. The SYNC instruction will prevent other instructions from issuing until all outstanding cacheable or uncacheable loads are satisfied, all cacheable stores are visible, all uncacheable stores have been sent on the ZBbus, and the write buffer has drained.

The internal I/O bridges and peripheral devices have a simple rule:

6. In all queues transactions will remain in the order of ZBbus Address phase.

The PCI and HyperTransport expansion busses follow the PCI ordering rules (Appendix E of the PCI Specification revision 2.2). Of particular importance is the rule:

7. Posted Writes can pass Delayed (Non-Posted) Reads.

Rule (1) allows the programming style where a processor writes some data and then sets a flag to indicate it is done. Another processor polling the flag will always see the new value of the data if it sees the new value of the flag. Rule (2) allows the “flag” to be a peripheral register: memory data (such as DMA descriptors or data buffers in cacheable coherent space) can be updated and then a control register (in uncached space) written (for example to start the DMA) without the need for any special intermediate operations.

Note that there is no rule in the other direction; a cacheable coherent store may be visible before an uncacheable load or store that precedes it in program order. If this ordering is required it can be enforced by either using a SYNC instruction (rule 5) or performing and using the result of an uncacheable load (by rule 2 and 3). This is important if the processor is using the PCI producer-consumer model and is setting a flag in memory to indicate that it has performed a write to a device. When this model is used the uncacheable write to PCI or HyperTransport space must be flushed from the CPU before the flag is written in cacheable space. The I/O Bridge will prevent the read-data-return from the polling request from passing an uncacheable write, so a SYNC will ensure the uncached write is queued in the bridge before the flag changes.

Rule (6) allows there to be multiple outstanding uncached operations. For internal peripherals and the generic bus all requests to the same peripheral will be seen in program order. Note that there are no such guarantees between different peripherals (which may be on different ZBbus agents, equivalent to being on different PCI busses), but this order only matters if there are back-channels between the devices (which will break any ordering rules).
Rule (7) is a potential problem, since writes could pass earlier reads to the same device (and thus the read could see the state after the write completes). However, rule (4) ensures that this situation never arises for code running on the SB-1. Other than blocking this write following read problem, rule (4) does allow multiple outstanding uncacheable accesses which will reach the peripherals in order. A series of loads performed to a FIFO will give the expected results, and by having multiple loads in flight the FIFO can be driven at full speed.

Cacheable non-coherent requests have the same timing as cacheable coherent requests as far as the CPU L1 data cache is concerned. They will be written to the data cache in program order, and the order with respect to uncached operations will be as described above. However, since these requests are outside the coherence domain it is unknown when writes will be visible to the rest of the system (i.e. flushed to L2 or memory and not hidden behind a different cached non-coherent copy of the block).

Locks can be implemented using the load linked (ll) and store conditional (sc) instructions. These have additional rules:

8. Load linked will not issue speculatively, it is held at the issue point until all preceding instructions have graduated.

9. Store Conditional includes a partial SYNC operation. No instructions will be issued from the time the store conditional is issued until it graduates.

In most cases no extra SYNC instructions are needed when acquiring a lock. The load linked is used to check if the lock is free and the store conditional can be used to claim it. If the load indicates that the lock is in use or the store conditional fails the code should spin waiting for the lock (or the process can be blocked, depending on the situation). If the load indicates the lock is free and the store conditional succeeds then the lock has been acquired.

There is a potential problem when freeing a lock. A normal store is used to release the lock. However, a SYNC may be required before the store to force completion of cacheable loads before the lock is released. This will only be the case if loads are performed while the lock is held, but the use of the data is delayed until the lock has been released. Consider the case where two loads are done and one hits in the L1 cache but the other misses in the L1 cache and is not used until after the lock is released. The CPU will not stall and (if there is no SYNC) can therefore execute the write to release the lock. There is a (very small) chance that the other CPU can claim the lock (by getting the line exclusive from the first CPU and having the load linked/store conditional succeed) and modify the data before the load from the first CPU gets the data. Thus the first CPU will not see the data that was present when it held the lock, which is probably an error. This can be solved using the SYNC, or ensuring there is a use of the load before the lock is released (rule 3).
CPU SPECULATIVE EXECUTION

The SB-1 CPU can speculatively execute cacheable loads (i.e. a read is issued to the ZBbus to get the data before the load is guaranteed to graduate), this only happens if the load is canceled because of an exception on an earlier instruction (branches are always resolved before the load is committed). This is normally not a problem since cacheable reads do not have side effects. However, the MIPS kernel address map includes areas where the cacheability attribute is supplied by the address rather than the TLB (kseg0 and xkphys). If a register contains one of these addresses that forces the cacheable attribute then a speculative load may be made to a peripheral address. To protect against this the internal peripherals check the cacheability of reads, and will prevent cacheable reads from having side effects. The generic bus has a mechanism to block cacheable reads which can be enabled for external peripherals that have read side effects (but should normally be disabled for memories). The only space where speculation can cause a problem is therefore the xkphys aliases for PCI and HyperTransport peripherals.

In the absence of program error (i.e. the register just contains the wrong value) there are two reasonable instruction sequences that could cause this behavior if at the start of the sequence the register happens to hold a kseg0 or xkphys cacheable alias of a peripheral. In both cases an exception is used to fix the address for the load.

```assembly
; at this point r1 contains junk
syscall 73       ; returns address in r1
ld    r2, 0(r1)

; This code intends to check r1 before the load
tle   r1, r2   ; trap if r1 is out of range
```

These two sequences can be fixed by adding two superscalar no-ops (ssnop) instructions before the loads to ensure the load is not speculatively executed before the exception triggers.

A similar situation arises if the `ld` is replaced with an indirect jump (`jr r1`), which can put a junk value into the jump register cache and cause a subsequent speculative instruction fetch to issue a cacheable read from a peripheral address. Again inserting two ssnop instructions will prevent the problem.

The CPU (break) instruction has a special issue rule, so does not suffer from this problem. It is safe for a debugger to use (break) to insert breakpoints in the normal way.
ERROR CONDITIONS

The system makes every effort to ensure that processing is never done based on erroneous data. Error conditions are signalled in several ways, as well as being passed in a flag with the data. All the critical structures where data corruption could occur are protected with ECC (allowing for correction of single bit errors) or parity (in structures that never contain the only copy of the data). As a result of an error condition being detected on data destined for the CPU, it will receive an exception (as required by the MIPS architecture) and will be signalled interrupts from the source that detected the problem or the Bus Watcher (see Section: “Bus Watcher” on page 64) or from both source and Bus Watcher. If the error is detected on data destined for a DMA engine, the DMA channel will stop and raise an interrupt to the CPU and the Bus Watcher will report the error. (Note that reading the Bus Watcher status register will clear it and enable logging of future errors, so it should be the last register read when the Bus Watcher information is dumped.)

CACHE ERROR EXCEPTIONS

The cache error exception is raised on the CPU both by error conditions detected in the internal caches and by error returns that are flagged on the bus as having data errors (but valid addresses). This follows the MIPS Architecture use of the exception. The practical upshot of this is that code diagnosing the exception should examine the state of external error reporting registers in addition to the ones in the CPU. Note that when data is returned to the data cache marked with an error code it will be written to the cache with an uncorrectable ECC error (after ECC is calculated the bottom two bits of each double-word are inverted to force the error) to ensure that it is not used on this processor and that the error is preserved if the line is evicted or snooped out of the cache by another CPU or DMA engine.

The error registers within the CPU are in the CP0 set and are described in the “Error Reporting Registers” part of Section 9 of the SB-1 User Manual. The error control register ErrCtl (register 26, select 0) indicates which cache suffered the error (or was being filled if it is an error signalled over the bus). It also flags recoverable errors from the data cache, these have been corrected and the exception can immediately return unless it needs to gather error statistics. Depending on which cache saw the error either CacheErr-I (register 27, select 0) and EPC or the CacheErr-D (register 27, select 1) and CacheErr-DPA (register 27, select 3) contain additional information. The external registers are in the Bus Watcher, the memory error counter (bus_mem_io_errors) and L2 error counter (bus_l2_errors) indicate how many errors have happened and the bus_err_status register will indicate the type of error and participants involved. Because instructions may be issuing and probing the cache at the same time an external error is found on a fill it is possible that an internal error is raised before the external one can be signalled, therefore it is normally useful for debugging if the error reporting code always dumps the Bus Watcher errors even if it appears to be an internal problem.
BUS ERROR EXCEPTIONS

The bus error exception is raised on the CPU by external errors that are not reporting data corruption. The main cause is when an access is detected to a non-existent address. Cacheable accesses that result in a bus error will be put in the cache with uncorrectable ECC errors (as described for the cache error exception). As with cache errors, both CPU and Bus Watcher registers are useful.

The error registers within the CPU are in the CP0 set and are described in the “Error Reporting Registers” part of Section 9 of the SB-1 User Manual. The BusErr-DPA (register 26, select 1) contains the address of the bus error (or first detected bus error) and the error control register ErrCtl (register 26, select 0) indicates if multiple bus errors have been seen. In the Bus Watcher the bus error counter (bus_mem_io_errors) indicate how many errors happened and the bus_err_status register will indicate the type of error and participants involved. If the bus error is generated from the generic bus (address in the range 00_1009_0000 - 00_3FFF_FFFF responder id=Br1) then the io_interrupt_status register contains more information on the cause of the bus error. Again, error reporting code can usefully dump all these registers.

There are a couple of interesting corner cases for bus errors. The first occurs when code is executed out of some physical address and at a later time that physical address is no longer valid (this is only likely to happen for unmapped addresses such as kseg0 and xkphys, since the software would presumably have removed any TLB entries that point to invalid physical addresses). For example the generic bus chip select or memory controller address parameters may have been changed. If any branches were executed while the address was valid the CPU will store the branch target information in its branch prediction structures. At a later time it may predict that a branch destination is at the now invalid address and issue a speculative instruction fetch. In this case the CPU will not take a bus error exception (because it will discover the speculation was incorrect), but an interrupt may come in from the peripheral indicating a bad access was made and the Bus Watcher will record and report that the error passed over the bus (the transaction ID recorded by the Bus Watcher will help indicate this has happened, see Table 51: “Decode of some TIDs for system revision PERIPH_REV3,” on page 81 and Section: “Overview of the ZBus Protocol” on page 20).

The second case arises because the bus error exception is imprecise. It can be reported on any instruction from the load that causes the bus error until a few instructions after the use of the register that was the target of the load. In most cases the bus error is a programming error and causes the process to be terminated or the system to enter error recovery. However, in systems with devices that can be hot-swapped the first indication of device removal may be a time out or target abort during an access to the device. In this case the imprecise reporting of the bus error may make it hard to determine that the exception handler should signal the device driver to clean up its state and resume operation (eret). One approach to solving this is to have a sync instruction after each load to the hot-swappable device which will force the bus error to be reported on the sync instruction. Alternatively, if two ssnop instructions are put after the instruction that uses the target of the load then the bus error will be signalled on one of the instructions between the load and the second ssnop.
CPU TO CPU COMMUNICATION (BCM1250 ONLY)

The two CPUs in a BCM1250 can communicate and share information in several ways. The MIPS Load Linked (ll and lld) and store conditional (sc and scd) instructions can be used to implement atomic operations. When there is genuine sharing, the L1 cache to L1 cache latency is 28-36 CPU cycles.

Inter-processor interrupts are possible in several ways using a per-CPU mailbox register. This is described in more detail in Section: “Mailbox Registers” on page 46. The generating CPU will do a write across the ZBus to the mailbox, which directly raises the receiving CPUs interrupt line. The total latency for this is 20-30 CPU cycles plus any context switch time needed. 7-12 of these cycles are taken between the receiving CPUs interrupt being raised and its fetching from the exception vector. The interrupt is attached as an exception to a convenient instruction: if the pipeline is running there will immediately be an instruction to use so the shorter time will apply, the longer time is taken when the pipeline is stalled and the instruction that will carry the exception has to be generated.

EXTERNAL INTERRUPTS

External interrupts from the PCI inputs are synchronized into the internal clock and pass through the interrupt mapper (see Section: “Interrupts” on page 47) before raising the CPU interrupt input. Including the 7-12 cycles in the CPU, there is a latency of 17-22 CPU cycles plus about 2ns from the external pin being asserted to the CPU fetching the first instruction from the exception vector. (Note that this is a guideline only, these times are not specified or tested and may vary between parts.)

External interrupts from the GPIO pins are synchronized and filtered to remove glitches before being passed through the interrupt mapper and to the CPU. When configured for the 60ns glitch filter there is a total latency of 20-25 CPU cycles plus about 112ns from the external pin being asserted and the CPU fetching the first instruction of the exception vector. (Note that this is a guideline only, these times are not specified or tested and may vary between parts.)
OVERVIEW OF THE ZBbus PROTOCOL

This section gives an overview of the ZBbus protocol. It contains the details needed to do system and performance debugging using the bus trace unit, but is not a full specification for the protocol. The bus is entirely internal to the part, and it connects the main blocks of the system. Each block that directly connects to the ZBbus is called an agent. Figure 5 shows the agents and Table 1 lists their four bit bus ID number. The bus runs at half the CPU clock speed and can transfer a new request and 256 data bits every cycle.

Table 1: ZBbus Agent IDs

<table>
<thead>
<tr>
<th>Agent</th>
<th>ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU0</td>
<td>0</td>
<td>SB-1 CPU 0.</td>
</tr>
<tr>
<td>CPU1</td>
<td>1</td>
<td>SB-1 CPU 1. (Only in BCM1250)</td>
</tr>
<tr>
<td>IOB0</td>
<td>2</td>
<td>I/O Bridge 0, connects PCI and HyperTransport interfaces to the ZBbus.</td>
</tr>
<tr>
<td>IOB1</td>
<td>3</td>
<td>I/O Bridge 1, connects the MACs and slow speed peripheral interfaces to the ZBbus.</td>
</tr>
<tr>
<td>SCD</td>
<td>4</td>
<td>System Control and Debug Unit.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>L2C</td>
<td>6</td>
<td>L2 Cache.</td>
</tr>
<tr>
<td>MC</td>
<td>7</td>
<td>Memory Controller.</td>
</tr>
</tbody>
</table>

The bus is split into an address section and a data section. These are arbitrated for separately and run independently. For a given transaction, use of the data bus always follows use of the address bus, but there is no other ordering imposed. Many transactions can be in progress at a given time. In theory each agent could have 64 outstanding operations on the bus. In practice, the agents are limited by their internal transaction tracking buffers to fewer than 64. Table 2 lists the signals in each section of the bus.

Table 2: ZBbus Signals

<table>
<thead>
<tr>
<th>Address/Control Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_AD[39:5]</td>
</tr>
<tr>
<td>A_BYEN[31:0]</td>
</tr>
<tr>
<td>A_ID[9:0]</td>
</tr>
<tr>
<td>A_CMD[2:0]</td>
</tr>
<tr>
<td>A_L1CA[1:0]</td>
</tr>
<tr>
<td>A_L2CA</td>
</tr>
<tr>
<td>R_SHD[5:0]</td>
</tr>
<tr>
<td>R_EXC[5:0]</td>
</tr>
<tr>
<td>R_L2HIT</td>
</tr>
</tbody>
</table>
Each transaction is marked with a Transaction ID (TID), a 10 bit number constructed by the requester. The top four bits of the TID are the requester's agent number (given in Table 1), and the lower six bits are chosen by the requester to be unique for all transactions it has in progress (See also Section: “Magic Decoder Ring For Using The Trace Buffer” on page 81).

The bus supports coherent and non-coherent transactions. The bulk of the protocol is used to track ownership of coherent blocks and implement the standard MESI coherent states:

- **M** - modified. This block is dirty (has been updated, and is different from memory). Modified is a subset of exclusive, an agent must be exclusive owner before changing the data.
- **E** - exclusive. The agent that has this data is the only agent that has it. This agent is permitted to modify the block.
- **S** - shared. The line is present in multiple agents, but is clean (memory or the L2 cache has the most current update). No agent may modify the block.
- **I** - invalid. The block is not present in a particular agent.

A transaction on the bus involves three phases and two arbitrations.

1. The requesting agent arbitrates for the address bus. The bus will be granted in a fair manner.
2. The address phase (A-phase) starts the transaction.
3. The response phase (R-phase) gathers the coherency state for the transaction and thus determines which agent is responsible for providing the data. Ownership transfers at the end of the R-phase.
4. The agent providing the data arbitrates for the data bus.
5. The data phase (D-phase) transfers the associated data and ends the transaction.

Normally an agent only arbitrates for the bus when it is prepared to make a transaction. However, occasionally agents (particularly the CPUs) will speculatively request the bus and then discover the transaction has been canceled. In this case the agent will still use the bus cycle, but issues a NOP command. An A-phase NOP will also be caused if the previous cycle on the bus was a transaction for the same address or if the destination agent blocks transactions as the source agent is granted the bus.

## Arbitration

The ZBbus arbitration uses a fair history based system. Requests are granted based on the last time the agent was granted the bus. If all agents request every cycle (this never happens because the bus bandwidth is much higher than the I/O device bandwidths) this will become a simple round robin scheme. In practice most agents request the bus ahead of needing it thus hiding the arbitration cycle.
**ADDRESS PHASE**

In the address phase the requesting agent puts the TID, address, byte-enables, command and cache attributes onto the address bus. All agents will examine the request and respond by setting their R_SHD and R_EXC bits in the R-phase.

The address has cache block granularity, and is therefore bits [39:5] of the physical address being accessed. Thirty-two byte enables indicate which bytes within the block are to be transferred.

The bus commands are in Table 3. Uncached and non-coherent accesses use the same commands, but will be responded to by the agent that owns the address. The default owner of a block is the L2 or memory controller for all addresses in the memory address space, the generic bus is the default owner for addresses in the generic bus range.

**Table 3: ZBbus Commands**

<table>
<thead>
<tr>
<th>A_CMD[2:0]</th>
<th>Command</th>
<th>Action</th>
<th>Data Bus</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_SHD</td>
<td>Read</td>
<td>The block is provided by the current owner and becomes shared. If the current owner is L2/memory and no other agent has a shared copy then the acquiring agent may take exclusive ownership (with no other action).</td>
<td></td>
</tr>
<tr>
<td>000</td>
<td>Shared</td>
<td></td>
<td>Data will be supplied by the exclusive owner if one is identified in the response phase, otherwise the default owner will supply it.</td>
</tr>
<tr>
<td>READ_EXC</td>
<td>Read</td>
<td>The block is provided by the current owner and becomes exclusively owned by the acquiring agent. All other copies of the block are invalidated.</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>Exclusive</td>
<td></td>
<td>Data will be supplied by the exclusive owner if one is identified in the response phase, otherwise the default owner will supply it.</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write</td>
<td>The block is written back to L2/memory.</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
<td>Data will be supplied by the requester.</td>
</tr>
<tr>
<td>WRITE_INV</td>
<td>Write</td>
<td>The requesting agent acquires ownership of the block and invalidates all copies (even if they are dirty), it then releases ownership and writes the block back to L2/memory. This is an optimization that avoids the need for a READ_EXC and transfer of data that will be discarded, it is used when the requesting agent is overwriting a complete block with new data (for example when a packet is DMAed from the network).</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>Invalidate</td>
<td></td>
<td>Data will be supplied by the requester.</td>
</tr>
<tr>
<td>INVALIDATE</td>
<td>Invalidate</td>
<td>The requesting agent acquires ownership of the block and all copies are invalidated (even if they are dirty). It is used when an agent upgrades a shared line to exclusive (in which case none of the copies will be dirty), or if the agent needs to become exclusive owner and guarantees to overwrite the complete line.</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>Block</td>
<td></td>
<td>There is no data phase for this command.</td>
</tr>
<tr>
<td>101</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>This command is used when the agent arbitrated for the bus and was granted, but is unable to complete the transaction. This will happen occasionally if the access is blocked by a buffer becoming full between the time the agent requested the bus and the address phase. It can also happen as a result of an exception in the requesting agent.</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
<td>There is no data phase for this command.</td>
</tr>
</tbody>
</table>
Two fields describe the cache attributes of a transfer. The first, shown in Table 4, describes the base (Level 1 cache) attribute. This must be set consistently by all agents using a block or all accesses to the block become UNPREDICTABLE.

Table 4: ZBus Level 1 Cache Attributes

<table>
<thead>
<tr>
<th>A_L1CA[1:0]</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Cacheable non-coherent. The block will be cached but will not play a part in the coherence protocol. Any agent with a copy of the block is free to write it. Software must manage the coherence.</td>
</tr>
<tr>
<td>01</td>
<td>Cacheable coherent. The block will be cached and uses the full coherence protocol. During usual operation all blocks with main memory addresses should use this attribute.</td>
</tr>
<tr>
<td>10</td>
<td>Uncacheable. The data will not be cached, and will not play a part in the coherence protocol. (Although not required, this attribute is normally used for uncacheable data that consists of 8 bytes or less.)</td>
</tr>
<tr>
<td>11</td>
<td>Uncacheable. The data will not be cached and will not play a part in the coherence protocol. (Although not required, this attribute is normally used for uncacheable data that has been merged in a buffer and may therefore be any number of bytes.)</td>
</tr>
</tbody>
</table>

The second attribute is only used for cacheable transactions that have an address in the memory controller range. The A_L2CA (Level 2 Cache Allocate) signal indicates that the block should be allocated in the L2 cache if it misses. Following the D-phase of a transaction with this attribute the data will always be in the L2 cache. Note that this attribute only controls the behavior on a cache miss: all cacheable accesses to the memory controller range are checked in the L2 cache, reads will always be serviced by the L2 if they hit and writes will always update the L2 if they hit.

**RESPONSE PHASE**

The response phase determines which agent will supply the data for a read command and is the point of ownership transfer for a block. Following the R-phase the new owner is responsible for the block (even if it does not yet have the data).

Each agent has R_EXC and R_SHD status signals. It asserts the R_EXC signal to indicate it has exclusive ownership of the block and that it will provide read data. It asserts the R_SHD signal to indicate it has a shared copy of the block. The memory controller and L2 cache use these to determine if they must supply or accept the data. The requesting agent will examine the status after doing a Read (shared) command, if no other agent has a shared copy of the data then the requesting agent is permitted (but not required) to take exclusive ownership of the block as if it had issued a Read Exclusive command.

An agent can assert both R_EXC and R_SHD to indicate an error. This is done if there is a parity or ECC error in the tags and the agent is unable to determine if it has ownership. If an exclusive owner can be identified then it supplies the block as normal. If there is no other exclusive owner then the memory controller will respond with UNPREDICTABLE data marked with a Fatal Error.

The L2 cache will assert the R_L2HIT to indicate to the memory controller that it will act as the default owner for the block. The default owner supplies data for a read if there is no exclusive owner, and captures the data on a write or ownership transfer.
DATA PHASE

The data transfer ends a transaction, and takes place in the Data phase. Since the latency between the A-phase request and the data becoming available can be very long, the D-phase is completely decoupled from the A-phase. The transaction ID is used to tie the data to the corresponding request. An agent must always be prepared to accept the data; it must not issue a read command unless it has space to accept the read-data-return, and it must block all writes when its receive buffers are full.

The D-phase takes place on the data section of ZBbus. A separate fair arbiter is used to get control of the data lines. The D-phase must always follow the A-phase, so agents are not permitted to arbitrate for the data bus until they have been granted the address bus. The data bus signals, shown in Table 2, include the actual data being sent (data smaller than a block is sent on its natural byte lanes) and status information.

The byte lanes that data uses on the ZBbus depends on the system endian mode. When viewed as 64 bit double-words there is no change (i.e. address bits [4:3] always select which 64 bit section of the ZBbus is used), but the byte address within the 64 bit double-words is swapped when the endian mode changes. Table 5 shows the mapping from byte address to data bits and byte enables.

Table 5. ZBbus Byte Lane Assignments

<table>
<thead>
<tr>
<th>Data</th>
<th>Byte Enable</th>
<th>Big Endian Addr[4:0] (hex)</th>
<th>Little Endian Addr[4:0] (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>58</td>
<td>00 01 02 03 04 05 06 07 08</td>
<td>07 06 05 04 03 02 01 00 0F</td>
</tr>
<tr>
<td>14</td>
<td>70</td>
<td>09 0A 0B 0C 0D 0E 0F</td>
<td>08 07 06 05 04 03 02 01 00</td>
</tr>
<tr>
<td>36</td>
<td>92</td>
<td>10 11 12 13 14 15 16 17 18</td>
<td>17 16 15 14 13 12 11 10 1F</td>
</tr>
<tr>
<td>58</td>
<td>70</td>
<td>19 20 21 22 23 24 25 26 27</td>
<td>22 23 24 25 26 27 28 29 30</td>
</tr>
<tr>
<td>00</td>
<td>31</td>
<td>32 33 34 35 36 37 38 39 40</td>
<td>41 42 43 44 45 46 47 48 49</td>
</tr>
</tbody>
</table>

The D_CODE indicates the status of the data being transferred as shown in Table 6.

Table 6: ZBbus Data Status Codes

<table>
<thead>
<tr>
<th>D_CODE[2:0]</th>
<th>Status of Data on D_DA</th>
<th>Sources of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>NOP, data invalid (gets counted as arbitrated but not used). This is used if an agent was granted the data bus but some exception causes it to be unable to supply the data.</td>
<td>All agents can generate NOPs.</td>
</tr>
<tr>
<td>001</td>
<td>Data valid.</td>
<td>All agents can generate valid data.</td>
</tr>
<tr>
<td>010</td>
<td>Data valid, tag was corrected for ECC error.</td>
<td>L2 Cache: Tag ECC corrected.</td>
</tr>
<tr>
<td>011</td>
<td>Data valid, data was corrected for ECC error.</td>
<td>CPU: Data cache had correctable error. Memory Controller: Data ECC corrected. L2 Cache: Data ECC corrected.</td>
</tr>
</tbody>
</table>
If an error is signalled during a data transfer the system takes care to maintain the error condition. If bad data is sent to the L2 cache or memory controller an uncorrectable ECC pattern is written with the data, so that subsequent reads will also see an error and bad data will not be used in processing. This can lead to a cascade of errors during the period between the initial detection of an error and software recovery code running (the SCD error log will record the first problem).

The D_MOD signal is asserted if the data being transferred is dirty (i.e. differs from L2/memory). In this case the L2 cache or memory controller will take a copy of the data, so that the new owner receives it clean.

The memory controller and L2 cache work together to supply data for any of the memory regions of the address space. Figure 7 shows the decision process that is used on reads and writes to the memory region to determine which agent supplies the data and which accepts write data. It also shows which of the memory or L2 will write back data when dirty data is supplied by an exclusive owner.

### Table 6: ZBbus Data Status Codes (Cont.)

<table>
<thead>
<tr>
<th>D_CODE[2:0]</th>
<th>Status of Data on D_DA</th>
<th>Sources of Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Bus error. A CPU will take a bus error exception if it receives this error.</td>
<td>I/O bridge0: PCI parity error, master or target abort. HyperTransport NxA or error return. I/O bridge1: Generic bus error (no chip select for the address, time-out during an access or I/O bus parity error). Memory Controller: No chip select decoded for the address. SCD: The SCD will return this error with UNPREDICTABLE data if the bus watcher detects an illegal address.</td>
</tr>
<tr>
<td>101</td>
<td>Fatal bus error. The ownership of the block is unclear. This can be caused by a CPU tag parity error, or because software used a CACHEOP to invalidate an exclusive line and the CPU had committed to supply the line in the window while the operation executed. A CPU will take a bus error exception if it receives this error.</td>
<td>Memory Controller: The memory controller returns UNPREDICTABLE data with this error code if an agent asserts both R_SHD and R_EXC. I/O bridge1: This error code is returned with UNPREDICTABLE data and the io_coh_err bit will be set if some agent asserts both R_SHD and R_EXC during a coherent generic bus access.</td>
</tr>
<tr>
<td>110</td>
<td>Uncorrectable ECC error in the tag. A CPU will take a cache error exception if it receives this error.</td>
<td>CPU: Tag parity error. L2 Cache: Tag ECC error.</td>
</tr>
<tr>
<td>111</td>
<td>Uncorrectable ECC error in the data (L2 cache or memory controller). A CPU will take a cache error exception if it receives this error.</td>
<td>CPU: Data cache had uncorrectable data ECC error. Memory Controller: Data ECC error. L2 Cache: Data ECC error I/O bridge: Write of a Read-modify-write operation that received an uncorrectable data error during the read.</td>
</tr>
</tbody>
</table>
Figure 7: Decision Tree for Memory Space Address Accesses

RESET

The COLDRES_L input is used for power-on reset of the device. It must be held low until the power supplies are within the operating range and the reference clock is stable. The cold reset will read the PLL multiplier bits and start the PLL, an internally generated delay allows the PLL to lock before the system is started. At the end of the cold reset delay reset time configuration information is read from the generic bus IO_AD lines (see below). The RESET_L input is used to cause a warm reset of the device, this resets all the internal logic but does not restart the PLL, read the configuration information or wait the cold reset delay (RESET_L need not be asserted at power-on since the cold reset delay has higher precedence). The end of the cold reset sequence or the deassertion of RESET_L signal starts the internal reset sequence to establish the internal state, the SCD will then release the system and CPU0. The device will source the reset signal for the board (RESETOUt_L), the PCI bus (P_RST_L) and the HyperTransport fabric (LDT_RESET_L and LDT_PWROK if the link needs a cold reset). These are all driven during the internal reset period and other than P_RST_L can be asserted separately under software control.

Reset can also be initiated by software or time-out of the watchdog timers. These can be a soft reset which restarts the part but is not signalled externally or a system reset which will assert RESETOUt_L. The software initiated system reset will re-sample the configuration information (except for the PLL ratio), the watchdog system reset will not.
Table 7 summarizes the reset options.

### Table 7: Operation of Different Reset Sources

<table>
<thead>
<tr>
<th></th>
<th>COLDRES_L&lt;br&gt;pin asserted</th>
<th>Software System Reset&lt;br&gt;system_cfg[60]</th>
<th>Watchdog 2nd time-out&lt;br&gt;Type=system</th>
<th>RESET_L&lt;br&gt;pin asserted</th>
<th>Software Soft Reset&lt;br&gt;system_cfg[58]</th>
<th>Watchdog 2nd Time-out&lt;br&gt;Type=sbsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample pll_div bits</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Restart PLL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clear system_cfg[63]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample configuration</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>bits (other than pll_div)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assert RESETOUT_L</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reset ZBbus</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Reset Agents</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

During reset pull-up and pull-down resistors on the generic Address/Data bus are used to set static system options. These pins have weak internal pull-up or pull-down resistors (as described in the Hardware Data Sheet), external resistors are required to set the other state. These options change the behavior of pins that need to be active at startup time to ensure proper operation.

### Table 8: Static Configuration Options

<table>
<thead>
<tr>
<th>IO_AD Bit</th>
<th>Name</th>
<th>Pulled Up to 3.3V</th>
<th>Pulled Down</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reserved</td>
<td>Reserved</td>
<td>Normal Operation.</td>
<td>N/A</td>
</tr>
<tr>
<td>1</td>
<td>clk100_src</td>
<td>The internal 100MHz clock and IO_CLK100 come directly from the CLK100_p reference. The IO_CLK100 will have a duty cycle only a little worse than the reference clock.</td>
<td>The internal 100MHz clock and IO_CLK100 are generated by dividing down the output of the PLL that feeds the CPU clock. The IO_CLK100 will match the internal clocking and will run at a few MHz (typically less than 10MHz) during COLDRES_L.</td>
<td>Section: “Clocks” on page 31</td>
</tr>
<tr>
<td>2</td>
<td>ldt_minrstcnt</td>
<td>Broadcom Use Only. Enable HyperTransport reset test mode.</td>
<td>Normal Operation.</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>ldt_bypass_pll</td>
<td>Broadcom Use Only. Bypass the HyperTransport PLL.</td>
<td>Normal Operation.</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>pci_test_mode</td>
<td>Broadcom Use Only. Enable PCI test mode.</td>
<td>Normal Operation.</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>iob0_div</td>
<td>IOB0 clock is CPU clock/3 Use with slow CPU clocks.</td>
<td>IOB0 clock is CPU clock/4 Use with fast CPU clocks.</td>
<td>Section: “Clocks” on page 31</td>
</tr>
<tr>
<td>6</td>
<td>iob1_div</td>
<td>IOB1 clock is CPU clock/2 Use with slow CPU clocks.</td>
<td>IOB1 clock is CPU clock/3 Use with fast CPU clocks.</td>
<td>Section: “Clocks” on page 31</td>
</tr>
<tr>
<td>11:7</td>
<td>pll_div</td>
<td>These bits are used to set the PLL clock ratios</td>
<td></td>
<td>Section: “Clocks” on page 31</td>
</tr>
</tbody>
</table>
Table 8: Static Configuration Options (Cont.)

<table>
<thead>
<tr>
<th>IO_AD Bit</th>
<th>Name</th>
<th>Pulled Up to 3.3V</th>
<th>Pulled Down</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>ser0_enable</td>
<td>Serial port 0 is synchronous. This sets the reset configuration, software may change it.</td>
<td>Serial port 0 is UART A.</td>
<td>Section: “Synchronous Mode” on page 336</td>
</tr>
<tr>
<td>13</td>
<td>ser0_rstb_en</td>
<td>GPIO[0] is driven as S0_RSTROBE by serial port 0.</td>
<td>GPIO[0] is a GPIO pin.</td>
<td>Section: “Synchronous Mode” on page 336</td>
</tr>
<tr>
<td>14</td>
<td>ser1_enable</td>
<td>Serial port 1 is synchronous. This sets the reset configuration, software may change it.</td>
<td>Serial port 1 is UART B.</td>
<td>Section: “Synchronous Mode” on page 336</td>
</tr>
<tr>
<td>15</td>
<td>ser1_rstb_en</td>
<td>GPIO[1] is driven as S1_RSTROBE by serial port 1.</td>
<td>GPIO[1] is a GPIO pin.</td>
<td>Section: “Synchronous Mode” on page 336</td>
</tr>
<tr>
<td>17</td>
<td>boot_mode[0]</td>
<td>The IO_CS_L[0] region of the generic bus space that is used for the boot ROM is configured for non-multiplexed operation with 8 data bits and 24 address bits. The SMBus EEPROM boot is configured for large (&gt; 16 kbit) EEPROMs using the eeprom read word protocol.</td>
<td>The IO_CS_L[0] region of the generic bus space that is used for the boot ROM is configured for multiplexed operation with 32 data bits and 32 address/enable bits. The SMBus EEPROM boot is configured for small (&lt;= 16 kbit) EEPROMs using the read word protocol.</td>
<td>Section: “Configuring a Chip Select Region” on page 362</td>
</tr>
<tr>
<td>18</td>
<td>boot_mode[1]</td>
<td>The boot address will access an EEPROM on the SMBus.</td>
<td>The boot address will access a ROM on the generic bus.</td>
<td>Section: “Boot ROM Support” on page 372.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Section: “Booting Using an SMBus EEPROM” on page 412</td>
</tr>
<tr>
<td>19</td>
<td>pci_host</td>
<td>The PCI interface is run as the host bridge.</td>
<td>The PCI interface is run as a regular device.</td>
<td>Section: “Introduction” on page 190</td>
</tr>
<tr>
<td>20</td>
<td>pci_arbiter</td>
<td>The PCI interface uses the internal arbiter. This must not be set if bit [19] is pulled down. The PCI interface uses an external arbiter.</td>
<td></td>
<td>Section: “PCI Arbiter” on page 222</td>
</tr>
<tr>
<td>21</td>
<td>southOnLDT</td>
<td>The South Bridge is on the HyperTransport fabric or on a bus bridged from the HyperTransport fabric. If there is no South Bridge this bit may be set either way.</td>
<td>The South Bridge is on the PCI bus.</td>
<td>Section: “The SouthBridge, VGA and Subtractive Decode” on page 195</td>
</tr>
<tr>
<td>22</td>
<td>big_endian</td>
<td>The system is Big Endian.</td>
<td>The system is Little Endian.</td>
<td>Section: “Introduction” on page 9</td>
</tr>
<tr>
<td>23</td>
<td>genclk_en</td>
<td>Enables output of the generic bus clock on the IO_CLK100 pin. Set the IO_CLK100 pin to a high impedance state.</td>
<td></td>
<td>Section: “Generic Bus Timing” on page 364</td>
</tr>
</tbody>
</table>
After a system reset (or power-on reset) CPU 0 and all peripherals are brought out of reset, but on a BCM1250 CPU 1 continues to be held in reset and will be isolated from the system. This allows CPU 0 to perform essential system initialization before releasing CPU 1. Since both CPUs have the same reset vector (virtual FFFF_FFFF_BFC0_0000, physical 00_1FC0_0000) the initial code will probably branch based on the CPU number which can be read from the PrId CP0 register.

The boot address is in the space that the I/O system uses for the generic expansion bus. At reset time this is configured to allow access to a (slow) boot ROM or flash memory, or can be diverted to SMBus interface 0, which will fetch code from a serial EEPROM. (A debugger may use the EJTAGBOOT option to cause the CPU to fetch the boot code from the JTAG port as described in Section: “EJTAGBOOT Instruction” on page 425.)

Following reset, no coherent accesses should be made until the data cache tags in the CPU(s) and the L2 cache tags have been invalidated. The DMA controllers and requests from the expansion buses can use coherent accesses, so they must not be used until the tags have been cleared. On a single CPU, once the level 1 cache tags have been invalidated cacheable non-coherent accesses may be done to the boot memory space. It is recommended that CPU 0 invalidates its cache tags and the L2 cache tags prior to releasing CPU 1 from reset, CPU 1 should clear its tags and signal CPU 0 that coherent accesses may be used.

It is possible to independently reset the CPUs, either from software or the watchdog timers. This is a potentially hazardous operation since while in reset the CPU is removed from participation in the coherence protocol. When the CPU goes in to reset it may be in the process of supplying data and releasing a exclusive lock on a line, since these operations are aborted some other agent may hang waiting for the data or lock. When the CPU is released from reset it will have stale coherency information in its caches and software initialization will clear all state. The independent reset features can be used to implement error recovery when one CPU fails while maintaining availability by allowing the other to continue to operate, but very careful consideration must be given to the possible state of the system if this is done (or it may be optimistically done with the fall-back position of a full reset if the partial restart fails).
A controlled reset of a running CPU can be done in a safe way. First the data cache must be flushed by doing indexed-writeback-invalidate cache ops on all entries in the cache. Secondly, it must be ensured that the data for all evicts has been sent to the bus (rather than just being visible). After these steps, the CPU will not be involved in any coherent operations and can be safely reset. The sequence of operations can be summarised:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index wb invalidate over whole Dcache</td>
<td>Flush cache</td>
</tr>
<tr>
<td>Sync</td>
<td>Ensure completion</td>
</tr>
<tr>
<td>Uncached store to somewhere that does not matter</td>
<td>Send data to bus</td>
</tr>
<tr>
<td>Sync</td>
<td>Ensure completion</td>
</tr>
<tr>
<td>Uncached store to somewhere that does not matter</td>
<td>Send data to bus</td>
</tr>
<tr>
<td>Sync</td>
<td>Ensure completion</td>
</tr>
<tr>
<td>Uncached store to reset bit</td>
<td>Write to reset bit</td>
</tr>
</tbody>
</table>

The first sync will ensure all the index-writeback-invalidate operations have completed and the CPU will not hit for any future snoops, but up to two writebacks may still be in transit in the bus interface buffers. The uncached stores will queue behind the writebacks, and the second sync ensures they have reached the bus buffers and therefore the writebacks have completed. At this point it is safe for the CPU to write to its reset bit. The data cache state is not affected by reset, so when the CPU comes out of reset the cache (and duplicate tags used by snoops) will still have all entries invalid. If the CPU has been shutdown using this sequence it is therefore safe to release it from reset in a system that is already running with coherent accesses.

The different actions of the resets may be used to identify which reset happened. A sequence similar to the following could be used (this assumes that the device uses a UART on port 0 for the console but the IO_AD[12] is set for synchronous port).

```c
// First check the sw_flag. It is only cleared by coldreset
If system_cfg.sw_flag == 0
    resetType = Coldreset
// Next check for the software initiated full reset, this resamples
// the configuration bits so will set the synchronous port
else if system_cfg.ser0_enable == 1
    resetType = SoftwareSystemReset
// Next check for a soft reset, since the bit is not self-clearing
// it will still be set
else if system_cfg.sb_softres == 1
    resetType = SoftwareSbSoftReset
// No internal way to tell between an external RESET_L
// and a Watchdog timeout
else
    resetType = ResetLOrWatchdog
// later BCM1250 have a flag for watchdog reset
if BCM1250 stepping C0 or later
    if watchdog wd_has_reset bit set
        resetType = Watchdog
    else
        resetType = Reset
// Now set conditions for the next time
// And allow use of the UART, set not-coldreset flag, clear softres
system_cfg.ser0_enable = 0
system_cfg.sw_flag = 1
system_cfg.sb_softres = 0
```
The system_scratch register could be used as an alternative to using the configuration bit for the serial port. The software would have to put a unique value into the register (or could set a bit that is reserved for this purpose) before writing the system_reset bit.

CLOCKS

The device is provided with a 100 MHz clock from which it generates the other internal clocks. The clock is multiplied up with a PLL to provide the CPU clock, which is divided down to provide the internal bus clocks and memory clock. A separate PLL is used to generate the HyperTransport clocks. An internal 100MHz clock is generated either directly from the reference clock or by dividing the CPU clock by the PLL ratio to drive the internal timers, the baud rate generator and for the timing parameters on the generic bus. The internal 100 MHz clock may be driven out on the IO_CLK100 pin. Using the reference source ensures that for all PLL ratios the IO_CLK100 will have a duty cycle only a little worse than the reference clock and that the IO_CLK100 runs at the same 100MHz frequency as the reference clock during cold reset and while the PLL locks. Using the internally generated reference ensures that the IO_CLK100 tracks the internal clocks (it will run slowly during cold reset even if the reference clock has not started), and if an integer PLL ratio is used (i.e. IO_AD[7] was sampled low) ensures the duty cycle is no worse than 40:60 regardless of the reference clock. However, if a nonintegral PLL ratio is used only the rising edge of the internally generated IO_CLK100 is valid, the time the output spends high can vary from cycle to cycle and be as short as 10% of the cycle.

The PCI bus and network interfaces have their own bus clocks that must be externally generated to match the attached components. The synchronous serial port clock either comes from the baud rate generator or from an external source.

Table 9 shows the supported clock ratios for the CPU and HyperTransport interface. Note that the speed grade of the part determines the maximum frequency that is permitted. The ratio for the CPU is set statically using the reset value on IO_AD[11:7] as described in the previous section. The ratio for the HyperTransport may be set directly (by enabling an extension register in the HyperTransport configuration space), but is normally set indirectly by encoding the value from the HyperTransport Link Frequency register.

<table>
<thead>
<tr>
<th>Code</th>
<th>Ratio</th>
<th>Main PLL (Code from Reset Time IO_AD[11:7])</th>
<th>HyperTransport PLL (Code from HyperTransport Frequency Register)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>CPU Clock (MHz)</td>
<td>ZBus Clock (MHz)</td>
</tr>
<tr>
<td>00100</td>
<td>2x</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>00101</td>
<td>2.5x</td>
<td>250</td>
<td>125</td>
</tr>
<tr>
<td>00110</td>
<td>3x</td>
<td>300</td>
<td>150</td>
</tr>
<tr>
<td>00111</td>
<td>3.5x</td>
<td>350</td>
<td>175</td>
</tr>
<tr>
<td>01000</td>
<td>4x</td>
<td>400</td>
<td>200</td>
</tr>
<tr>
<td>01001</td>
<td>4.5x</td>
<td>450</td>
<td>225</td>
</tr>
<tr>
<td>01010</td>
<td>5x</td>
<td>500</td>
<td>250</td>
</tr>
<tr>
<td>01011</td>
<td>5.5x</td>
<td>550</td>
<td>275</td>
</tr>
<tr>
<td>01100</td>
<td>6x</td>
<td>600</td>
<td>300</td>
</tr>
<tr>
<td>01101</td>
<td>6.5x</td>
<td>650</td>
<td>325</td>
</tr>
<tr>
<td>01110</td>
<td>7x</td>
<td>700</td>
<td>350</td>
</tr>
<tr>
<td>01111</td>
<td>7.5x</td>
<td>750</td>
<td>375</td>
</tr>
</tbody>
</table>
The clocks for the I/O bridges that connect the peripherals to the ZBbus (see Figure 5 on page 9) are synchronously divided down from the CPU clock. The divide ratio needs to be set as part of the reset time configuration (see Section: "Reset" on page 26) based on the CPU frequency. The I/O Bridge clock does not affect the speed of the peripheral interfaces directly, but it does affect the bandwidth (and latency) between the peripheral and the ZBbus. Increasing the bridge clock speed will increase the bandwidth, but will use more power. Conversely the speed can be decreased to save power if the full bandwidth is not needed. I/O Bridge 0 is designed for operation with the clock about 200 MHz, it should be set in the range 166-266 MHz (check the data sheet for the maximum frequency for a given speed grade). I/O Bridge 1 is designed for operation with the clock about 266 MHz, it should be set in the range 233-333 MHz (check the data sheet for the maximum frequency for a given speed grade). The maximum frequency for the bridge clocks is given in the Data Sheet as Fbr0 and Fbr1 in the Clock, Reset and Test timing parameters.

<table>
<thead>
<tr>
<th>Code</th>
<th>Ratio</th>
<th>CPU Clock (MHz)</th>
<th>ZBbus Clock (MHz)</th>
<th>HyperTransport Clock (MHz)</th>
<th>HyperTransport Data Rate (Mbps/pair)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>8x</td>
<td>800</td>
<td></td>
<td>400</td>
<td></td>
</tr>
<tr>
<td>10001</td>
<td>8.5x</td>
<td>850</td>
<td></td>
<td>425</td>
<td></td>
</tr>
<tr>
<td>10010</td>
<td>9x</td>
<td>900</td>
<td></td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>10011</td>
<td>9.5x</td>
<td>950</td>
<td></td>
<td>475</td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>10x</td>
<td>1000</td>
<td></td>
<td>500</td>
<td></td>
</tr>
<tr>
<td>10101</td>
<td>10.5x</td>
<td>1050</td>
<td></td>
<td>525</td>
<td></td>
</tr>
<tr>
<td>10110</td>
<td>11x</td>
<td>1100</td>
<td></td>
<td>550</td>
<td></td>
</tr>
</tbody>
</table>
The memory clock is synchronously divided down from the CPU clock. The divide ratio is programmed into the memory controller configuration registers. Figure 8 gives an overview of the internal clock generation.

**Figure 8: Clock Distribution Overview**
MEMORY MAP

The memory map is designed to be usable in systems that only have 32 bit addressing and to provide expansion for systems that can support the full 64 bit virtual address and 40 bit physical address of the SB-1 CPU. There are some additional restrictions that the MIPS architecture imposes:

- The reset vector of the CPU is to physical address 00_1FC0_0000.
- The exception vectors of the CPU are at physical address 00_0000_0000. (They may be offset from the base by a multiple of 64K using the SB-1 MultiProcessor Vector extension in the CPU config register).
- The first 512 MB of memory is addressable uncached (and unmapped) in KSEG1 and is therefore a good place for fixed address peripherals (they can be accessed without using any TLB entries).

In addition devices on the PCI need to be able to DMA into all of the physical memory (within the 32 bit PCI address range) and be able to enable or disable endian swapping on each transaction (see Section: “Endian Policies” on page 201 for a full discussion of PCI and HyperTransport endian policies). An overview of the memory map is given in Table 10 and Figure 9 and a more detailed view is in Table 11 on page 36.

Table 10: Overview of BCM1250 Physical Address Map

<table>
<thead>
<tr>
<th>Base</th>
<th>Top</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_0000_0000</td>
<td>00_A000_0000</td>
<td>Memory controller.</td>
</tr>
<tr>
<td>00_C000_0000</td>
<td>00_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>00_F800_0000</td>
<td>00_FC00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>00_FF00_0000</td>
<td>01_0000_0000</td>
<td>HyperTransport expansion</td>
</tr>
<tr>
<td>01_8000_0000</td>
<td>01_F000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>02_0000_0000</td>
<td>02_3000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>02_4000_0000</td>
<td>02_5000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>02_6000_0000</td>
<td>02_7000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>02_8000_0000</td>
<td>02_9000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>03_0000_0000</td>
<td>03_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>03_C000_0000</td>
<td>03_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>03_F000_0000</td>
<td>03_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>04_0000_0000</td>
<td>04_1000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>04_2000_0000</td>
<td>04_3000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>04_4000_0000</td>
<td>04_5000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>04_6000_0000</td>
<td>04_7000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>04_8000_0000</td>
<td>04_9000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>05_0000_0000</td>
<td>05_1000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>05_2000_0000</td>
<td>05_3000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>05_4000_0000</td>
<td>05_5000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>05_6000_0000</td>
<td>05_7000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>05_8000_0000</td>
<td>05_9000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>06_0000_0000</td>
<td>06_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>06_C000_0000</td>
<td>06_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>06_F000_0000</td>
<td>06_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>07_0000_0000</td>
<td>07_1000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>07_2000_0000</td>
<td>07_3000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>07_4000_0000</td>
<td>07_5000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>07_6000_0000</td>
<td>07_7000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>07_8000_0000</td>
<td>07_9000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>08_0000_0000</td>
<td>08_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>08_C000_0000</td>
<td>08_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>08_F000_0000</td>
<td>08_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>09_0000_0000</td>
<td>09_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>09_C000_0000</td>
<td>09_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>09_F000_0000</td>
<td>09_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0A_0000_0000</td>
<td>0A_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0A_C000_0000</td>
<td>0A_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0A_F000_0000</td>
<td>0A_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0B_0000_0000</td>
<td>0B_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0B_C000_0000</td>
<td>0B_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0B_F000_0000</td>
<td>0B_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0C_0000_0000</td>
<td>0C_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0C_C000_0000</td>
<td>0C_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0C_F000_0000</td>
<td>0C_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0D_0000_0000</td>
<td>0D_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0D_C000_0000</td>
<td>0D_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0D_F000_0000</td>
<td>0D_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0E_0000_0000</td>
<td>0E_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0E_C000_0000</td>
<td>0E_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0E_F000_0000</td>
<td>0E_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0F_0000_0000</td>
<td>0F_A000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0F_C000_0000</td>
<td>0F_E000_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
<tr>
<td>0F_F000_0000</td>
<td>0F_FF00_0000</td>
<td>HyperTransport/PCI configuration space</td>
</tr>
</tbody>
</table>
The memory controller supports up to 1 GB of memory in a system that is restricted to 32-bit physical addresses, and up to 4 GB (2 GB on BCM1125/H) using 512 Mb technology DRAMs (up to 8 GB when 1Gb technology SDRAMs are available, and with an option to double the size at the cost of speed with an external decoder) on systems with a full 40-bit address. The address map includes 512 MB for mapping PCI and HyperTransport memory mapped peripherals that use 32-bit addressing, and an alias for this space that will byte swap accesses when the system is in big endian mode.

Accesses to Reserved or Unused regions will result in UNPREDICTABLE behavior. Writes will be discarded.

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_FFFF_FFFF</td>
<td>Reserved</td>
</tr>
<tr>
<td>FA_0000_0000</td>
<td>PCI Full Access</td>
</tr>
<tr>
<td>F8_0000_0000</td>
<td>HT Devices</td>
</tr>
<tr>
<td>80_0000_0000</td>
<td>Not on BCM1125</td>
</tr>
<tr>
<td>01_0000_0000</td>
<td>SDRAM Expansion</td>
</tr>
<tr>
<td>00_F800_0000</td>
<td>Maps to 00_D800_0000</td>
</tr>
<tr>
<td>00_E000_0000</td>
<td>- 00_DFFF_FFFF</td>
</tr>
<tr>
<td>00_DE00_0000</td>
<td>With &quot;Match Bit&quot; Endian Policy</td>
</tr>
<tr>
<td>00_DC00_0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_D800_0000</td>
<td>PCI/HT Config</td>
</tr>
<tr>
<td>00_D000_0000</td>
<td>PC/HT I/O Space</td>
</tr>
<tr>
<td>00_C000_0000</td>
<td>HT/PCI Special</td>
</tr>
<tr>
<td>00_A000_0000</td>
<td>L2 Direct Access</td>
</tr>
<tr>
<td>00_9000_0000</td>
<td>Fourth SDRAM Region</td>
</tr>
<tr>
<td>00_8000_0000</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_6000_0000</td>
<td>Third SDRAM Region</td>
</tr>
<tr>
<td>00_5000_0000</td>
<td>Second SDRAM Region</td>
</tr>
<tr>
<td>00_4000_0000</td>
<td>PCI/LDT Memory Space</td>
</tr>
<tr>
<td>00_3000_0000</td>
<td>Match Bit Lane Endian Policy</td>
</tr>
<tr>
<td>00_2000_0000</td>
<td>Match Byte Lane Endian Policy</td>
</tr>
<tr>
<td>00_1FC0_0000</td>
<td>Boot ROM (Default for IO_CS0)</td>
</tr>
<tr>
<td>00_2000_0000</td>
<td>Generic Bus Devices</td>
</tr>
<tr>
<td>00_1009_0000</td>
<td>Internal Devices</td>
</tr>
<tr>
<td>00_1006_0000</td>
<td>System Control and Debug</td>
</tr>
<tr>
<td>00_1000_0000</td>
<td>First SDRAM Region</td>
</tr>
<tr>
<td>00_0000_0000</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Figure 9: Memory Map
<table>
<thead>
<tr>
<th>Base</th>
<th>Top</th>
<th>Size</th>
<th>Owner</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_0000_0000</td>
<td>00_0FFF_FFFF</td>
<td>256 MB</td>
<td>MC</td>
<td>Base DRAM.</td>
</tr>
<tr>
<td>00_1000_0000</td>
<td>00_1001_FFFF</td>
<td>2*64 KB</td>
<td>SCD</td>
<td>Reserved/Debug JTAG serviced addresses.</td>
</tr>
<tr>
<td>00_1002_0000</td>
<td>00_1002_0FFF</td>
<td>4 KB</td>
<td>SCD</td>
<td>Reset config, CPU 0 interrupt mapper, timers, addr trap, trace, bus log and counters.</td>
</tr>
<tr>
<td>00_1002_1000</td>
<td>00_1002_1FFF</td>
<td>4 KB</td>
<td>SCD</td>
<td>SCD CPU 0 Mailbox alias.</td>
</tr>
<tr>
<td>00_1002_2000</td>
<td>00_1002_2FFF</td>
<td>4 KB</td>
<td>SCD</td>
<td>CPU 1 interrupt mapper. (Reserved on BCM1125/H)</td>
</tr>
<tr>
<td>00_1002_3000</td>
<td>00_1002_3FFF</td>
<td>4 KB</td>
<td>SCD</td>
<td>CPU 1 Mailbox alias. (Reserved on BCM1125/H)</td>
</tr>
<tr>
<td>00_1002_4000</td>
<td>00_1002_FFFF</td>
<td>48 KB</td>
<td>SCD</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_1003_0000</td>
<td>00_1003_FFFF</td>
<td>64 KB</td>
<td>SCD</td>
<td>ZBbus cycle count.</td>
</tr>
<tr>
<td>00_1004_0000</td>
<td>00_1004_FFFF</td>
<td>64 KB</td>
<td>SCD/L2</td>
<td>L2 registers.</td>
</tr>
<tr>
<td>00_1005_0000</td>
<td>00_1005_FFFF</td>
<td>64 KB</td>
<td>SCD/MC</td>
<td>Memory controller registers.</td>
</tr>
<tr>
<td>00_1006_0000</td>
<td>00_1006_00FF</td>
<td>0.25 KB</td>
<td>I/O</td>
<td>SMBus, GPIO.</td>
</tr>
<tr>
<td>00_1006_0100</td>
<td>00_1006_01FF</td>
<td>0.25 KB</td>
<td>I/O</td>
<td>Duart ch A.</td>
</tr>
<tr>
<td>00_1006_0200</td>
<td>00_1006_02FF</td>
<td>0.25 KB</td>
<td>I/O</td>
<td>Duart ch B.</td>
</tr>
<tr>
<td>00_1006_0300</td>
<td>00_1006_03FF</td>
<td>0.25 KB</td>
<td>I/O</td>
<td>Duart status and control.</td>
</tr>
<tr>
<td>00_1006_0400</td>
<td>00_1006_07FF</td>
<td>1 KB</td>
<td>I/O</td>
<td>Sync serial, dma and HDLC ch 0.</td>
</tr>
<tr>
<td>00_1006_0800</td>
<td>00_1006_0BFF</td>
<td>1 KB</td>
<td>I/O</td>
<td>Sync serial, dma and HDLC ch 1.</td>
</tr>
<tr>
<td>00_1006_0C00</td>
<td>00_1006_0FFF</td>
<td>1 KB</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>00_1006_1000</td>
<td>00_1006_17FF</td>
<td>2 KB</td>
<td>I/O</td>
<td>Generic Bus config.</td>
</tr>
<tr>
<td>00_1006_1800</td>
<td>00_1006_1FFF</td>
<td>2 KB</td>
<td>I/O</td>
<td>Generic Bus status and log, PCMCIA config and status.</td>
</tr>
<tr>
<td>00_1006_2000</td>
<td>00_1006_2FFF</td>
<td>4 KB</td>
<td>I/O</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_1006_3000</td>
<td>00_1006_3FFF</td>
<td>4 KB</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>00_1006_4000</td>
<td>00_1006_4FFF</td>
<td>4 KB</td>
<td>I/O</td>
<td>MAC 0</td>
</tr>
<tr>
<td>00_1006_5000</td>
<td>00_1006_5FFF</td>
<td>4 KB</td>
<td>I/O</td>
<td>MAC 1</td>
</tr>
<tr>
<td>00_1006_6000</td>
<td>00_1006_6FFF</td>
<td>4 KB</td>
<td>I/O</td>
<td>MAC 2. (On BCM1125/H: alias of MAC 0; do not use!)</td>
</tr>
<tr>
<td>00_1006_7000</td>
<td>00_1006_FFFF</td>
<td>36 KB</td>
<td>I/O</td>
<td>Unused</td>
</tr>
<tr>
<td>00_1007_0000</td>
<td>00_1008_FFFF</td>
<td>2*64 KB</td>
<td>I/O</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_1009_0000</td>
<td>00_3FFF_FFFF</td>
<td>767 MB</td>
<td>I/O</td>
<td>Generic/boot interface.</td>
</tr>
<tr>
<td>00_4000_0000</td>
<td>00_5FFF_FFFF</td>
<td>512 MB</td>
<td>PCI/HT</td>
<td>Memory mapped I/O space. Match byte lane endian policy.</td>
</tr>
<tr>
<td>00_6000_0000</td>
<td>00_7FFF_FFFF</td>
<td>512 MB</td>
<td>PCI/HT</td>
<td>Memory space mapped I/O space. Match bit lane endian policy.</td>
</tr>
<tr>
<td>00_8000_0000</td>
<td>00_8FFF_FFFF</td>
<td>256 MB</td>
<td>MC</td>
<td>Second DRAM bank.</td>
</tr>
<tr>
<td>00_9000_0000</td>
<td>00_9FFF_FFFF</td>
<td>256 MB</td>
<td>MC</td>
<td>Third DRAM bank.</td>
</tr>
<tr>
<td>00_A000_0000</td>
<td>00_BFFF_FFFF</td>
<td>512 MB</td>
<td>XX</td>
<td>Unused</td>
</tr>
<tr>
<td>00_C000_0000</td>
<td>00_CFFF_FFFF</td>
<td>256 MB</td>
<td>MC</td>
<td>Fourth DRAM bank.</td>
</tr>
<tr>
<td>00_D000_0000</td>
<td>00_D7FF_FFFF</td>
<td>128 MB</td>
<td>L2C</td>
<td>L2 special test address range. Match byte lane endian policy.</td>
</tr>
</tbody>
</table>
### Table 11: Address Map Details (Cont.)

<table>
<thead>
<tr>
<th>Base</th>
<th>Top</th>
<th>Size</th>
<th>Owner</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_D800_0000</td>
<td>00_D8FF_FFFF</td>
<td>16 MB</td>
<td>HT</td>
<td>EOI signaling. Match byte lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_D900_0000</td>
<td>00_D90F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>Iack signaling. Match byte lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_D910_0000</td>
<td>00_D91F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>System management. Match byte lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_D920_0000</td>
<td>00_D92F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>N/A</td>
</tr>
<tr>
<td>00_D930_0000</td>
<td>00_DBFF_FFFF</td>
<td>45 MB</td>
<td>HT</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_DC00_0000</td>
<td>00_DDFF_FFFF</td>
<td>32 MB</td>
<td>PCI/HT</td>
<td>I/O space (only 25 bits). Match byte lane endian policy.</td>
</tr>
<tr>
<td>00_DE00_0000</td>
<td>00_DEFF_FFFF</td>
<td>16 MB</td>
<td>PCI/HT</td>
<td>Configuration Space. Match byte lane endian policy.</td>
</tr>
<tr>
<td>00_DF00_0000</td>
<td>00_DFFF_FFFF</td>
<td>16 MB</td>
<td>PCI/HT</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_E000_0000</td>
<td>00_EFFF_FFFF</td>
<td>256 MB</td>
<td>XX</td>
<td>Unused</td>
</tr>
<tr>
<td>00_F000_0000</td>
<td>00_F7FF_FFFF</td>
<td>128 MB</td>
<td>XX</td>
<td>Unused</td>
</tr>
<tr>
<td>00_F800_0000</td>
<td>00_F8FF_FFFF</td>
<td>16 MB</td>
<td>HT</td>
<td>EOI signaling. Match bit lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_F900_0000</td>
<td>00_F90F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>Iack signaling. Match bit lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_F910_0000</td>
<td>00_F91F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>System Management Match bit lane endian policy. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>00_F920_0000</td>
<td>FD_F92F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>N/A</td>
</tr>
<tr>
<td>00_F930_0000</td>
<td>00_FBFF_FFFF</td>
<td>45 MB</td>
<td>HT</td>
<td>Reserved</td>
</tr>
<tr>
<td>00_FC00_0000</td>
<td>00_FDFF_FFFF</td>
<td>32 MB</td>
<td>PCI/HT</td>
<td>I/O space (only 25 bits). Match bit lane endian policy.</td>
</tr>
<tr>
<td>00_FE00_0000</td>
<td>00_FEFF_FFFF</td>
<td>16 MB</td>
<td>PCI/HT</td>
<td>Configuration Space. Match bit lane endian policy.</td>
</tr>
<tr>
<td>00_FF00_0000</td>
<td>00_FFFF_FFFF</td>
<td>16 MB</td>
<td>PCI/HT</td>
<td>Reserved</td>
</tr>
<tr>
<td>01_0000_0000</td>
<td>7F_FFFF_FFFF</td>
<td>508 GB</td>
<td>MC</td>
<td>DRAM expansion space.</td>
</tr>
<tr>
<td>80_0000_0000</td>
<td>F7_FFFF_FFFF</td>
<td>480 GB</td>
<td>HT</td>
<td>HyperTransport devices. (Reserved on BCM1125)</td>
</tr>
<tr>
<td>F8_0000_0000</td>
<td>F8_FFFF_FFFF</td>
<td>4 GB</td>
<td>PCI</td>
<td>PCI full address range. Match byte lane endian policy.</td>
</tr>
<tr>
<td>F9_0000_0000</td>
<td>F9_FFFF_FFFF</td>
<td>4 GB</td>
<td>PCI</td>
<td>PCI full address range. Match bit lane endian policy.</td>
</tr>
<tr>
<td>FA_0000_0000</td>
<td>FC_FFFF_FFFF</td>
<td>12 GB</td>
<td>PCI</td>
<td>Reserved</td>
</tr>
<tr>
<td>FD_0000_0000</td>
<td>FD_F7FF_FFFF</td>
<td>3968 MB</td>
<td>HT</td>
<td>Reserved (HyperTransport reserved space).</td>
</tr>
<tr>
<td>FD_F800_0000</td>
<td>FD_F8FF_FFFF</td>
<td>16 MB</td>
<td>HT</td>
<td>Reserved (Interrupt signaling).</td>
</tr>
<tr>
<td>FD_F900_0000</td>
<td>FD_F90F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>Reserved (Iack signaling).</td>
</tr>
<tr>
<td>FD_F910_0000</td>
<td>FD_F91F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>Reserved (System Management).</td>
</tr>
<tr>
<td>FD_F920_0000</td>
<td>FD_F92F_FFFF</td>
<td>1 MB</td>
<td>HT</td>
<td>Reserved (PREQ Protocol).</td>
</tr>
<tr>
<td>FD_F930_0000</td>
<td>FD_FBFF_FFFF</td>
<td>45 MB</td>
<td>HT</td>
<td>Reserved</td>
</tr>
<tr>
<td>FD_FC00_0000</td>
<td>FD_FFFF_FFFF</td>
<td>32 MB</td>
<td>HT</td>
<td>Reserved (PCI I/O).</td>
</tr>
<tr>
<td>FD_FE00_0000</td>
<td>FD_FFFF_FFFF</td>
<td>32 MB</td>
<td>HT</td>
<td>Reserved (PCI Configuration).</td>
</tr>
</tbody>
</table>
### Table 11: Address Map Details (Cont.)

<table>
<thead>
<tr>
<th>Base</th>
<th>Top</th>
<th>Size</th>
<th>Owner</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE_0000_0000</td>
<td>FF_FFFF_FFFF</td>
<td>8 GB</td>
<td>HT</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
This Page is left blank for notes
Section 4: System Control and Debug Unit

INTRODUCTION

The System Control and Debug unit (SCD) includes all the system control functions, interrupt mappers, system level performance monitoring, and debugging functions.

SYSTEM CONTROL

The system controller is used to bring the part out of reset. It holds the reset-time configuration options and contains the extended JTAG interface that allows external control and monitoring of the system.

The JTAG interface is described in detail in Section: “TAP Controller” on page 421. The basic scan chain has been extended to allow an external debugger access to a selection of internal state both in the CPUs and in the system. The JTAG interface can access system debug and performance monitoring features such as the performance counters and the ZBus trace logic. No software support on the target system is required to run this debugging interface.

System Reset is controlled by this section of the SCD. The external COLDRES_L pin must be asserted at system power-up and can be asserted later to cause a cold reset of the system. The RESET_L pin can be used to cause a warm reset of the system. The SCD generates all the required internal reset pulses and timing and drives the external reset output RESETOUT_L.

There are two registers associated with the system controller. One is the system_revision register which identifies the part and gives the chip revision number. The second is the system configuration register system_cfg which reports the states of the reset time configuration options and allows resetting of various sections of the system. The system_cfg register is accessible both from the system and from the JTAG port.

A CPU can cause a system reset by setting the system_reset bit of the system_cfg register. This will behave the same as the COLDRES_L pin of the part being asserted (except the PLL is not restarted), and is the standard way for software to cause a full restart. The restart will clear the system_reset bit. The sb_softres bit is identical except the system_cfg register bits are not restored to their defaults (note that this bit does not self clear). Software may use the other bits of the register to reset an individual CPU or signal an external reset. To prevent lockup, the CPU 0 reset bit automatically clears. JTAG access to the register may also manipulate the reset bits, in this case they do not automatically clear so the JTAG probe has full control of the resets. The JTAG probe can also reset individual ZBus agents. While reset is asserted the agent is isolated from the system, in this state any access to the agent will hang.

The lower bits of the system_cfg register reflect the state that was latched from the generic bus IO_AD pins at reset time. These are described in Section: “Reset” on page 26. Some of these are used to configure device pin options, others are free for interpretation by system software (for example the board revision could be read in this way).

Many bits of the system_cfg port are only writable from the JTAG port. They provide access to test features and are reserved for use by Broadcom. They must be left at zero for normal operation of the part.
The uniprocessor bits and soft-reset bit may be used on the BCM1250 to disable one of the processors. The clock to the disabled processor will be stopped following reset and it will enter a low power state (lower power than being left in reset). The processor will be enabled on the next full reset (RESET_L pin asserted, system_reset bit written with a one or a watchdog double timeout). Following a full reset only CPU0 will be running and CPU1 will be held in reset. If a uniprocessor system is desired, the CPU0 reset sequence should check for a multiprocessor environment by reading the multiprocessor bit in the PrId (CPU CP0 Processor Id register) or the uniprocessor bits in the system_cfg register. If the system is running as a multiprocessor the unicpu0 bit should be set in the system_cfg register, and the sb_softres bit written with a one to cause the system to reboot as a uniprocessor.

The system_scratch register is a read/write register that is not used by the hardware. It is free for software use. For example it may be used by firmware to store a pointer to bootstrap information for use by the operating system, or it may be used to pass (small) messages between the processors before memory is configured. The value of the register is UNPREDICTABLE following cold reset, the value is preserved over other resets. The sw_flag bit in the system_cfg register may be used to indicate the validity of the scratch value, since the flag is cleared on a cold reset and preserved over other resets. The sw_flag may also be used by the system to distinguish between cold resets and others. Software can check the bit after reset: if the bit is zero then the reset was a cold reset and the bit should be set, if the bit is already set then the reset was some other type (also see Section: “Reset” on page 26).

### Table 12: System Identification and Revision Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>reserved</td>
<td>8'hff</td>
<td>Reserved, reads as 8'hff.</td>
</tr>
<tr>
<td>15:8</td>
<td>revision</td>
<td>xx</td>
<td>Revision of the part. See Table 13.</td>
</tr>
<tr>
<td>31:16</td>
<td>Part</td>
<td>xx</td>
<td>Part type.</td>
</tr>
<tr>
<td></td>
<td>BCM1250</td>
<td>16'h1250</td>
<td>BCM1250</td>
</tr>
<tr>
<td></td>
<td>16'h1150</td>
<td></td>
<td>(configured as uniprocessor)</td>
</tr>
<tr>
<td></td>
<td>16'h1125</td>
<td></td>
<td>(configured as uniprocessor, half L2 aka 1125Y)</td>
</tr>
<tr>
<td></td>
<td>BCM1125</td>
<td>16'h1123</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BCM1125H</td>
<td>16'h1124</td>
<td></td>
</tr>
</tbody>
</table>

Bits 27:24 Indicate the number of processors.
Bits 23:20 Indicate the L2 cache size (1-128KB, 2-256KB, 5-512KB, 0-1024K).
Bits 19:16 Indicate the peripheral set
(0,2,5 - BCM1250 peripherals, 3 - BCM1125 peripherals, 4 - BCM1125H peripherals).  

<table>
<thead>
<tr>
<th>Bits</th>
<th>wid</th>
<th>xx</th>
<th>Wafer ID.</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:32</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 13: Part Revisions

<table>
<thead>
<tr>
<th>Part</th>
<th>Stepping</th>
<th>Peripherals</th>
<th>Revision</th>
<th>Nickname</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCM12500</td>
<td>An</td>
<td>PERIPH_REV1</td>
<td>8'h01 or 8'h02</td>
<td>“Pass1”</td>
<td>Prototype only parts, refer to earlier manual.</td>
</tr>
<tr>
<td>BCM1250</td>
<td>An</td>
<td>PERIPH_REV2</td>
<td>8'h03 - 8'h0b</td>
<td>“Pass2”</td>
<td>Initial production BCM1250 parts.</td>
</tr>
<tr>
<td>BCM1250</td>
<td>Bn</td>
<td>PERIPH_REV2</td>
<td>8'h10 or 8'h11</td>
<td>“Pass2.2”</td>
<td></td>
</tr>
<tr>
<td>BCM1250</td>
<td>Cn</td>
<td>PERIPH_REV3</td>
<td>8'h20</td>
<td>“Pass3”</td>
<td>BCM1250 with peripherals upgraded</td>
</tr>
<tr>
<td>BCM1125/H</td>
<td>An</td>
<td>PERIPH_REV3</td>
<td>8'h20 or 8'h21</td>
<td>“Pass1”</td>
<td>Initial production BCM1125/H parts</td>
</tr>
<tr>
<td>BCM1125/H</td>
<td>Bn</td>
<td>PERIPH_REV3</td>
<td>8'h30</td>
<td>“Pass2”</td>
<td></td>
</tr>
</tbody>
</table>

---

*Broadcom Corporation*

Page 42  Section 4: System Control and Debug Unit  Document 1250_1125-UM100-R
### Table 14: Manufacturing Information Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>mid</td>
<td>xx</td>
<td>Manufacturing ID. Broadcom Use Only.</td>
</tr>
</tbody>
</table>

### Table 15: System Configuration Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[0].</td>
</tr>
<tr>
<td>1</td>
<td>clk100_src</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[1] that selects the source for the internal 100MHz clock and IO_CLK100 (if enabled).</td>
</tr>
<tr>
<td>2</td>
<td>ldt_minrstcnt</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[2]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>3</td>
<td>ldt_bypass_pll</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[3]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>4</td>
<td>pci_test_mode</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[4]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>5</td>
<td>iob0_div</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[5] that controls the clock divider for I/O Bridge 0. 0: IOB0 runs at CPU clock/4, for use with fast CPU clocks. 1: IOB0 runs at CPU clock/3, for use with slow CPU clocks.</td>
</tr>
<tr>
<td>6</td>
<td>iob1_div</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[6] that controls the clock divider for I/O Bridge 1. 0: IOB1 runs at CPU clock/3, for use with fast CPU clocks. 1: IOB1 runs at CPU clock/2, for use with slow CPU clocks.</td>
</tr>
<tr>
<td>11:7</td>
<td>pll_div</td>
<td>ext</td>
<td>Read Only, reflects the strap resistors on generic IO_AD[11:7] that select the PLL Divide ratio.</td>
</tr>
<tr>
<td>12</td>
<td>ser0_enable</td>
<td>ext</td>
<td>Read/Write. The default reflects the strap resistor on generic IO_AD[12] but this bit can be written by software to change the setting. 0: Serial interface 0 is in asynchronous (uart) mode. 1: Serial interface 0 is in synchronous mode. If this bit is changed by software it must also re-initialize the interface.</td>
</tr>
<tr>
<td>13</td>
<td>ser0_rstb_en</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[13] that allocates GPIO[0] pin to the synchronous serial interface.</td>
</tr>
<tr>
<td>14</td>
<td>ser1_enable</td>
<td>ext</td>
<td>Read/Write. The default reflects the strap resistor on generic IO_AD[14] but this bit can be written by software to change the setting. 0: Serial interface 1 is in asynchronous (uart) mode. 1: Serial interface 1 is in synchronous mode. If this bit is changed by software it must also re-initialize the interface.</td>
</tr>
<tr>
<td>15</td>
<td>ser1_rstb_en</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[15] that allocates GPIO[1] pin to the synchronous serial interface.</td>
</tr>
<tr>
<td>16</td>
<td>pcmcia_enable</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[16] that configured the PCMCIA mode.</td>
</tr>
<tr>
<td>18:17</td>
<td>boot_mode</td>
<td>ext</td>
<td>Reflects the strap resistor on generic IO_AD[18:17] that configured the boot mode. 00: 32 bit generic bus ROM (multiplexed) 01: 8 bit generic bus ROM (non-multiplexed) 10: SMBus EEPROM &lt;= 16 kbit (read word protocol) 11: SMBus EEPROM &gt; 16kbit (eeprom read word protocol). Bit 17 is Read Only. Bit 18 can be changed by software to clear the SMBus boot mode and allow general use of the SMBus 0 interface.</td>
</tr>
</tbody>
</table>
### Table 15: System Configuration Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>pci_host</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[19], that configures the PCI interface to be host or device mode.</td>
</tr>
<tr>
<td>20</td>
<td>pci_arbiter</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[20], that configures the PCI interface to use an internal or external arbiter. (If the PCI is set in device mode the resistor must be set for an external arbiter)</td>
</tr>
<tr>
<td>21</td>
<td>southOnLDT</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[21], that configures the southbridge to be on the HyperTransport fabric or PCI bus.</td>
</tr>
<tr>
<td>22</td>
<td>big_endian</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[22], that configures the system to be big or little endian.</td>
</tr>
<tr>
<td>23</td>
<td>genclk_en</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[23], that enables output of the generic bus clock on IO_CLK100. If this bit is zero then the IO_CLK100 will be held in a high impedance state.</td>
</tr>
<tr>
<td>24</td>
<td>ldt_test_en</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on IO_AD[24]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>25</td>
<td>gen_parity_en</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[25] that configured the generic bus parity.</td>
</tr>
<tr>
<td>31:26</td>
<td>config</td>
<td>ext</td>
<td>Read Only, reflects the strap resistor on generic IO_AD[31:26]. These configuration bits are available for interpretation by software.</td>
</tr>
<tr>
<td>32</td>
<td>clkstop</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>33</td>
<td>clkstep</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>41:34</td>
<td>clkcount</td>
<td>8'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>42</td>
<td>pll_bypass</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>44:43</td>
<td>pll_ref</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>46:45</td>
<td>pll_vco</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>48:47</td>
<td>pll_vreg</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>49</td>
<td>mem_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the memory controller is held in reset.</td>
</tr>
<tr>
<td>50</td>
<td>l2c_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the level 2 cache is held in reset.</td>
</tr>
<tr>
<td>51</td>
<td>io_reset_0</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the I/O bridge to the PCI and HyperTransport fabric is held in reset.</td>
</tr>
<tr>
<td>52</td>
<td>io_reset_1</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the I/O bridge to the slow speed devices and generic bus is held in reset.</td>
</tr>
<tr>
<td>53</td>
<td>scd_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the SCD is held in reset.</td>
</tr>
<tr>
<td>54</td>
<td>cpu_reset_0</td>
<td>1'b0</td>
<td>Writable via JTAG only. When written with a 1 a standard length reset pulse is delivered to CPU 0. (A device reset will also cause a standard length reset pulse to CPU 0).</td>
</tr>
<tr>
<td>55</td>
<td>cpu_reset_1</td>
<td>1'b1</td>
<td>When set CPU 1 will be held in reset. This bit is set on a device reset, causing the processor to remain in reset until released under software control.</td>
</tr>
<tr>
<td>56</td>
<td>unipcpu0</td>
<td>1'b0</td>
<td>Set to indicate uniprocessor using physical processor 0. (This bit will always be set on the BCM1125/H.)</td>
</tr>
<tr>
<td>57</td>
<td>unipcpu1</td>
<td>1'b0</td>
<td>Set to indicate uniprocessor using physical processor 1. (BCM1250 only)</td>
</tr>
<tr>
<td>58</td>
<td>sb_softres</td>
<td>1'b0</td>
<td>When a write changes this bit from a 0 to a 1 a soft reset will be performed. This will reset of whole chip except for this register. Note that once it is set the bit must be cleared before writing a 1 will again cause a soft reset.</td>
</tr>
<tr>
<td>59</td>
<td>ext_reset</td>
<td>1'b0</td>
<td>When set the RESETOUT_L pin will be asserted.</td>
</tr>
</tbody>
</table>
Table 15: System Configuration Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>system_reset</td>
<td>1'b0</td>
<td>When written with a 1 a full system reset will be performed (thus setting this bit back to zero).</td>
</tr>
<tr>
<td>61</td>
<td>misr_mode</td>
<td>1'b0</td>
<td>Broadcom Use Only.</td>
</tr>
<tr>
<td>62</td>
<td>scd_misr_reset</td>
<td>1'b0</td>
<td>Broadcom Use Only.</td>
</tr>
<tr>
<td>63</td>
<td>sw_flag</td>
<td>1'bx</td>
<td>This read/write bit is cleared by a cold reset. Its value is preserved on any other reset. It may be used by software to detect the reset type, or for any other use.</td>
</tr>
</tbody>
</table>

Table 16: Scratch Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>value</td>
<td>64'hx</td>
<td>This register is available for any use by software. When the chip is powered up its value is UNPREDICTABLE. Its value is preserved over reset.</td>
</tr>
</tbody>
</table>
MAILBOX REGISTERS

Each CPU has a mailbox register that can be used to signal events or small messages. The register may be accessed by the other CPU (on a BCM1250) or a bus master peripheral. If the part is a device on the PCI bus (i.e. not the host bridge) then the host on the PCI can use the mailbox to signal the CPU.

A mailbox register is 64 bits wide and has four interrupt lines, for each group of 16 bits the corresponding interrupt is raised whenever any bit is set. To set a bit in the mailbox a 1 must be written to the appropriate bit position in the `mbox_set` location. To clear a bit in the mailbox a 1 must be written to the appropriate bit position in the `mbox_clear` location. Thus there is no need for an atomic read-modify-write. Typically only the CPU which owns the register would write the clear register and the signalling agent would write the set register. However, there is no protection in the hardware to enforce this (any agent that can access the `mbox_set` and `mbox_clear` locations can change the mailbox state). The interrupt outputs from a particular mailbox are only routed to the interrupt mapper of one CPU, thus CPU0 cannot be interrupted by the setting of bits in `mailbox_1` and CPU1 cannot be interrupted by the setting of bits in `mailbox_0`.

Interrupts from the HyperTransport interface can set bits in the mailbox, allowing peripherals to signal events. This is setup by the way the peripheral interrupts are configured by the system software.

The system software determines the use of the mailbox register and the meaning associated with each of the bits. Other than the mapping from bit positions to interrupt lines the hardware imposes no restrictions.

The simplest way to use the mailbox is as sixty-four event channels grouped into four (or fewer) priority levels. The signalling agent writes to set the event bit and the signaled CPU can clear it when it has responded. One or more of the four interrupts will be raised whenever there are outstanding events in the mailbox. This is likely to be the best method to use if high speed I/O interrupts will be feeding the mailbox.

At the other extreme, each of the quarters of the mailbox could be considered a channel for passing 16 bit messages. An example use is in inter-processor procedure calls. The source of the call will marshal arguments into a shared memory buffer, then make the call by writing the buffer index to the mailbox set location. This raises an interrupt on the service processor, which will read the buffer index from the mailbox and write the mailbox clear location. Since the shared memory buffer is coherent the service processor will always see the correct data, and the ordering imposed by the write-interrupt cleanly transfers buffer ownership. When the procedure is completed the results could be passed back using the reverse mechanism.

In addition to the normal address, the `mbox_set` and (read only) `mailbox` locations have aliases that are the only registers in a 4k page. This page is accessible from the PCI interface (through BAR2 for the CPU 0 mailbox and BAR3 for the CPU1 mailbox). Thus PCI devices can only set bits in the mailbox (to signal the CPU) or read the current mailbox status (to see if a signal was cleared), but can never clear the mailbox. The CPU (or any other agent in the part) can access all the mailbox registers and can therefore both set and clear bits.

<table>
<thead>
<tr>
<th>Table 17: Mailbox Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailbox_cpu_0 - 00_1002_00C0, alias_mailbox_cpu_0 - 00_1002_1xx0 (Read Only)</td>
</tr>
<tr>
<td>mailbox_cpu_1 - 00_1002_20C0, alias_mailbox_cpu_1 - 00_1002_3xx0 (Read Only)</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>mbox_int_0 - if any bit set</td>
</tr>
<tr>
<td>mbox_int_1 - if any bit set</td>
</tr>
<tr>
<td>mbox_int_2 - if any bit set</td>
</tr>
<tr>
<td>mbox_int_3 - if any bit set</td>
</tr>
</tbody>
</table>

When this location is written, any bits that are set will cause the corresponding bit to be set in the mailbox register.

<table>
<thead>
<tr>
<th>mbox_set_cpu_0 - 00_1002_00C8, alias_mbox_set_cpu_0 - 00_1002_1xx8 (Write Only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbox_set_cpu_1 - 00_1002_20C8, alias_mbox_set_cpu_1 - 00_1002_3xx8 (Write Only)</td>
</tr>
</tbody>
</table>

When this location is written, any bits that are set will cause the corresponding bit to be cleared in the mailbox register.
INTERRUPTS

The chip has a large number of interrupt sources. These come from both internal peripherals and external signals. Each CPU has its own interrupt mapper which allows the destination of the interrupt from each source to be set to one of the six architected hardware interrupts, the non-maskable interrupt (NMI) or the Debug Interrupt (DINT). The mapper block includes the CPU mailbox registers described in the previous section.

Most interrupt sources are common to each interrupt mapper. The only sources that are specific to a CPU are directed interrupts from the HyperTransport (as described below, the interrupt message includes which CPU it should be delivered to) and the per-CPU mailbox interrupts.

The basic operation of the interrupt mapper is described in this section. Interrupts from the HyperTransport fabric extend the basic model as described in the next section. Section: “The Full Interrupt Mapper” on page 50 has a diagram of the complete mapper, including system, mailbox and HyperTransport sources and describes all the associated registers.

The interrupt mapper receives the level sensitive interrupts from all sections of the part. Each source has an associated mask bit and a 3 bit map register. If the source is interrupting and not masked out it is driven to one of the CPU interrupt lines according to the mapping. Table 18 shows the mapping, and the corresponding IP bit in the Cause register.

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Interrupt</th>
<th>CP0 Cause Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>NMI</td>
<td>Reset/NMI vector, flagged in status register not cause register.</td>
</tr>
<tr>
<td>111</td>
<td>DINT</td>
<td>CPU debug interrupt, flagged in debug register.</td>
</tr>
</tbody>
</table>

Each source maps to one CPU interrupt line. There is no limit on how many sources may map to a single CPU line. The I5 interrupt line is also used by interrupts generated internal to the CPU, these will be merged with any external requests. The MIPS architecture uses software based interrupt dispatch, so the software can assign priorities to the six regular interrupt lines as it sees fit. A typical system would merge most of the system sources on a few of the interrupt lines, and assign one source per line for sources that need rapid dispatch.
To assist software dispatch there are status registers associated with each of the six CPU interrupt lines, DINT and the NMI. These indicate which sources are interrupting, pass the mask and are routed to this interrupt line. In addition there is a global status register that shows which sources are interrupting regardless of their mask setting or routing.

The debug interrupt (DINT) to the CPU may be raised by the interrupt mapper. It will also be raised if a trace trigger is marked to send DINT (see Section: “Trigger Sequences” on page 73) or if the break control bit is set in the EJTAG control register (see Section: “EJTAG Control Register” on page 439).

There is a diagnostic register built into the interrupt mapper that can be used to force interrupts to be asserted into the mask and map logic.

Since there is a full copy of the mapper for each CPU, system interrupts may be masked and mapped differently on the two processors. In many cases a particular source would always be assigned to a particular processor, delivering it to both requires software to coordinate servicing the interrupt.

**HYPERTRANSPORT INTERRUPTS**

The HyperTransport bus protocol includes passing interrupt messages across the HyperTransport fabric. These messages include identification of the interrupt vector (as described below there are some non-vectored interrupts, carrying meanings specific to the x86 architecture, but these are thought of as having an implicit vector number in the interrupt mapper). The MIPS architecture does not directly support hardware interrupt vectored, so the vector number is used by the hardware to decide which input to the interrupt mapper will be triggered and software can do any required vectoring.

The interrupt packet also includes a destination. This is either a physical processor number or “a system specific logical mapping”. The BCM1250 CPUs (and their related interrupt controllers) are physically numbered 0 and 1. The BCM1125H CPU (and interrupt controller) is physical number 0. The logical mapping used is the example one from the HyperTransport specification, a processor is selected as the target by setting the bit position corresponding to its physical processor number. This scheme allows for up to eight processors. Use of the logical mapping allows broadcast and multicast interrupts (having only two CPUs, these amount to the same thing on the BCM1250). The mapper supports the 8 bit destination from the HyperTransport Specification rev 1.0 and earlier, and will ignore the additional 24 destination bits added by the HyperTransport Specification rev 1.01.

The HyperTransport interrupt messages must maintain ordering with transactions that come across the HyperTransport link ahead of them. They therefore follow the same flow as other HyperTransport transactions and are delivered to the SCD across the ZBus as writes to the `interrupt_ldt_set` register. Writes to this register are decoded and used to set bits in either the `mailbox` register or the `interrupt_ldt` register. The CPU can write the `interrupt_ldt_set` register during testing to simulate the arrival of an interrupt message. The data written is a 64-bit double-word containing the data from the HyperTransport Interrupt message as described in Table 19.
Table 19: Interrupt Message Format for Writes to interrupt_ldt_set Register

<table>
<thead>
<tr>
<th>Data Bits</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>HyperTransport Interrupt Message Type:</td>
<td></td>
</tr>
<tr>
<td>000:</td>
<td>Fixed</td>
<td></td>
</tr>
<tr>
<td>001:</td>
<td>Arbitrated (Low Priority)</td>
<td></td>
</tr>
<tr>
<td>010:</td>
<td>Non-vectored: SMI</td>
<td></td>
</tr>
<tr>
<td>011:</td>
<td>Non-vectored: NMI</td>
<td></td>
</tr>
<tr>
<td>100:</td>
<td>Non-vectored: INIT</td>
<td></td>
</tr>
<tr>
<td>101:</td>
<td>Non-vectored: Startup</td>
<td></td>
</tr>
<tr>
<td>110:</td>
<td>Non-vectored: External Interrupt</td>
<td></td>
</tr>
<tr>
<td>111:</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>HyperTransport Interrupt Trigger Mode 0=Edge, 1=Level (non-vectored interrupts must be edge).</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>HyperTransport Interrupt Destination Mode 0=Physical, 1=Logical.</td>
<td></td>
</tr>
<tr>
<td>12:5</td>
<td>HyperTransport Interrupt Destination.</td>
<td></td>
</tr>
<tr>
<td>20:13</td>
<td>HyperTransport Interrupt Vector.</td>
<td></td>
</tr>
</tbody>
</table>

The HyperTransport defines three sorts of interrupts:

**Non-vectored interrupts** carry a type field rather than source vector information. They are used to carry interrupt messages that are traditionally associated with pins on the x86 CPU. The five types are SMI (system management interrupt), NMI (non maskable interrupt), Init (initialize processor, on x86 resets integer registers but does not affect FP or caches), Startup (used to start application processors in x86 multiprocessor systems) and Ext. Int. (external interrupt, used to signal an interrupt from a PIC style PC interrupt controller). These are statically mapped to source vectors h32 - h36 before being reported to the interrupt mapper of the CPU(s) to which they are directed.

**Fixed interrupts** are the standard form. They include the destination information and the source vector and are delivered to all interrupt mappers to which they are directed.

**Arbitrated (Low Priority) interrupts** behave the same as fixed interrupts. However if multiple destinations are indicated they only get delivered to one of them. The HyperTransport specification requires that the destination selected is either the lowest priority or the CPU that is currently servicing an interrupt from the same source. Since MIPS architecture CPUs give no external indication of which interrupt they are currently servicing the mapper implements this by delivering the interrupt to the lowest numbered CPU to which it is directed.

All HyperTransport interrupts are thus presented to the interrupt mapper as an 8 bit vector number. The top two bits of the vector number are used to determine how the low 6 bits are used:

Table 20: Delivery of HyperTransport Interrupts

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The bit corresponding to Vector[5:0] in the interrupt_ldt register is set, raising that interrupt line to the mapper.</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>The bit corresponding to Vector[5:0] in the mailbox register is set, raising one of the mailbox interrupts to the mapper.</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Interrupt will be discarded.</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Interrupt will be discarded.</td>
<td></td>
</tr>
</tbody>
</table>
HyperTransport interrupt messages specify if the source is edge or level sensitive. Edge sensitive interrupts do not need to be acknowledged, and it is source dependent whether the CPU has to take action before further interrupts can be signalled. Level interrupts require the CPU to send an EOI message to acknowledge the servicing of the request. The interrupt source device will wait until an EOI is received before issuing any further interrupts. In the MIPS architecture the CPU does not produce a signal to indicate EOI. Software must do a write to the EOI space at the end of the interrupt service routine (see Section: “HyperTransport End Of Interrupt (EOI) Signaling Space” on page 198). Since the HyperTransport specification requires that edge and level interrupts are not reported with the same source vector ID, software will know from the vector number that an EOI is needed. The access to the EOI space must be a write, so the CPU is not stalled while the request is issued. (Reads to EOI space have UNDEFINED results.)

Interrupts from a PC style PIC interrupt controller are signaled using the External Interrupt message, which has no additional source information associated with it. The PIC will provide an 8-bit source vector in response to an interrupt acknowledge (IACK) cycle. On the BCM1250 and BCM1125H software is responsible for running the IACK cycle by performing a byte read within the reserved IACK range as described in Section: “Legacy Interrupt Acknowledge (IACK) Space” on page 199. A read to this address range gets routed to the southbridge. If the southbridge is a native HyperTransport device, then this command packet will be routed directly to the southbridge. If the southbridge is connected to a PCI device bridged from the HyperTransport then the command packet will be routed to the intervening HyperTransport-PCI bridge. (If the southbridge is on the PCI interface (set by the southOnLDT configuration bit being clear) the IACK access will be run as a PCI IACK cycle, but in that case the interrupt would not have come in as a HyperTransport External Interrupt message.)

**The Full Interrupt Mapper**

The full interrupt controller is illustrated in Figure 10 on page 51. This shows the mailbox and HyperTransport sources and all the mapper registers.

The system sources and the mailbox interrupts come into the `interrupt_source_status` register. They are combined with the HyperTransport interrupts and the data in the `interrupt_diag` register. This is a read/write register that the CPU can use to activate interrupt request lines. It is primarily used to allow testing of the interrupt routing by simulating device interrupts.

The `interrupt_idt` register bits are set by the HyperTransport interrupt controller as described in the previous section. The register is read by the CPU to determine the HyperTransport interrupt source and cleared by a write to the `ldt_interrupt_clear` register.

The `interrupt_mask` register is used to block interrupts. Masking is done after the three sources (system, HyperTransport and diagnostic) have been combined, if interrupt lines are shared either all sources are masked off or none. All unmasked interrupts are passed to the interrupt mapper. This has a 3 bit Map register for each of the sources which maps the source to one of the CPU interrupt lines (regular INT[5:0], NMI or DINT). Each source can be mapped to only one CPU line, but each CPU line can have any number of sources.

The regular interrupt signals to the CPU and DINT are level sensitive and will continue to be asserted as long as there are interrupting sources. The NMI is edge triggered and will be asserted for a single cycle whenever the mapper output asserts (i.e. after an NMI all interrupting sources must be clear for at least a cycle before a subsequent NMI will be generated).
The **interrupt_trace** register is used to select which interrupts to this CPU are sent to the trace unit to be used as part of a trigger condition. The bits in this register provide an accept mask. Any interrupt that has its corresponding bit set in this register will cause the trigger.

![Diagram of Per-CPU Interrupt Mapper](image)

**Figure 10: Per-CPU Interrupt Mapper (replicated for each CPU; x = 0 or 1)**
The main interrupt registers are all 64 bits wide:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_diag</td>
<td>Setting a bit in this register raises the corresponding interrupt. Software must clear the bit to remove the interrupt. This register is intended for diagnostics only.</td>
</tr>
<tr>
<td>interrupt_ldt_set</td>
<td>This write only register is written with a HyperTransport interrupt message and will decode it into setting a single bit in either the interrupt_ldt or mailbox register, and thus an interrupt will be raised.</td>
</tr>
<tr>
<td>interrupt_ldt</td>
<td>The HyperTransport interrupt controller will set the bit corresponding to the interrupt vector number for vectors in the range 0-63. The CPU can read this register to find which HyperTransport interrupts have been posted.</td>
</tr>
<tr>
<td>interrupt_ldt_clr</td>
<td>This is not a register. Writing a one to a bit in this location clears the corresponding bit in the interrupt_ldt register.</td>
</tr>
<tr>
<td>interrupt_mask</td>
<td>Setting a bit in this register masks the corresponding interrupt. All bits in this register are set at reset, masking out all interrupts.</td>
</tr>
<tr>
<td>interrupt_status_n</td>
<td>Read Only. There is a status register for each of the outputs from the mapper. A bit will be set only if the corresponding source is interrupting, is not masked and is mapped to this output.</td>
</tr>
<tr>
<td>interrupt_source_status</td>
<td>Read Only. A bit will be set if the corresponding source is interrupting regardless of the state of the interrupt mask.</td>
</tr>
<tr>
<td>interrupt_trace</td>
<td>Setting a bit in this register allows the interrupt to trigger the trace unit.</td>
</tr>
<tr>
<td>interrupt_map0 ... interrupt_map63</td>
<td>There are 64 interrupt mapper registers. Each is 3 bits wide and encodes the mapping from the interrupt number given in Table 22 onto the CPU interrupt lines as described in Table 18.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
<th>Method to clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>watchdog_timer_int_0</td>
<td>Watchdog timer 0 timeout</td>
<td>Interrupt is raised on first timeout of the timer (the second will reset the machine). It is cleared by writing to the timer configuration register. See Section: “Watchdog Timers” on page 57.</td>
</tr>
<tr>
<td>1</td>
<td>watchdog_timer_int_1</td>
<td>Watchdog timer 1 timeout</td>
<td>Interrupt is raised on first timeout of the timer (the second will reset the machine). It is cleared by writing to the timer configuration register. See Section: “Watchdog Timers” on page 57.</td>
</tr>
<tr>
<td>2</td>
<td>timer_int_0</td>
<td>General timer 0</td>
<td>Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See Section: “General Timers” on page 58.</td>
</tr>
<tr>
<td>3</td>
<td>timer_int_1</td>
<td>General timer 1</td>
<td>Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See Section: “General Timers” on page 58.</td>
</tr>
<tr>
<td>4</td>
<td>timer_int_2</td>
<td>General timer 2</td>
<td>Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See Section: “General Timers” on page 58.</td>
</tr>
<tr>
<td>Number</td>
<td>Name</td>
<td>Description</td>
<td>Method to clear</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>5</td>
<td>timer_int_3</td>
<td>General timer 3</td>
<td>Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See Section: “General Timers” on page 58.</td>
</tr>
<tr>
<td>6</td>
<td>smb_int_0</td>
<td>SMBus 0 interrupt</td>
<td>Interrupts when transfer completes or an error is signalled by SMBus unit 0. Cleared by reading status register or writing the error bit. See Section: “Programming Model” on page 409.</td>
</tr>
<tr>
<td>7</td>
<td>smb_int_1</td>
<td>SMBus 1 interrupt</td>
<td>Interrupts when transfer completes or an error is signalled by SMBus unit 1. Cleared by reading status register or writing the error bit. See Section: “Programming Model” on page 409.</td>
</tr>
<tr>
<td>8</td>
<td>uart_int_0</td>
<td>UART channel A interrupt</td>
<td>Interrupts from UART channel A are merged into this one signal, cleared as defined in Section: “Interrupts” on page 324.</td>
</tr>
<tr>
<td>9</td>
<td>uart_int_1</td>
<td>UART channel B interrupt</td>
<td>Interrupts from UART channel B are merged into this one signal, cleared as defined in Section: “Interrupts” on page 324.</td>
</tr>
<tr>
<td>10</td>
<td>ser_int_0</td>
<td>Synch serial 0 interrupt</td>
<td>DMA status and errors from the synchronous serial interface are combined into this interrupt, cleared as defined in Section: “Synchronous Serial Interrupts” on page 351.</td>
</tr>
<tr>
<td>11</td>
<td>ser_int_1</td>
<td>Synch serial 1 interrupt</td>
<td>DMA status and errors from the synchronous serial interface are combined into this interrupt, cleared as defined in Section: “Synchronous Serial Interrupts” on page 351.</td>
</tr>
<tr>
<td>12</td>
<td>pcmcia_int</td>
<td>PCMCIA controller interrupt</td>
<td>There are three sources for interrupt from the PCMCIA controller, they are cleared by reading the pcmcia_status register as described in Section: “Using The PCMCIA Card” on page 389.</td>
</tr>
<tr>
<td>13</td>
<td>addr_trap_int</td>
<td>Address trap interrupt</td>
<td>Raised by any of the address traps counting from 1 to 0. Cleared by reading the addr_trap_reg. See Section: “Address Trapping” on page 67.</td>
</tr>
<tr>
<td>14</td>
<td>perf_cnt_int</td>
<td>System performance counter interrupt</td>
<td>Raised by any of the system performance counters reaching its maximum count. Cleared by clearing the counters.</td>
</tr>
<tr>
<td>15</td>
<td>trace_freeze_int</td>
<td>Trace buffer frozen interrupt</td>
<td>This interrupt is raised when the trace buffer is frozen, and cleared when the buffer is reset. See Section: “Trigger Sequences” on page 73.</td>
</tr>
<tr>
<td>16</td>
<td>bad_ecc_int</td>
<td>Uncorrectable ECC error or Bus Error</td>
<td>This interrupt is raised by the bus watcher when a ZBus data transfer is marked as having an uncorrectable ECC error or some other form of fatal error. This indicates an uncorrectable ECC from either the L2 cache or memory, or a fatal error from one of the other agents. It is cleared by a read of the bus_err_status register.</td>
</tr>
<tr>
<td>17</td>
<td>cor_ecc_int</td>
<td>Correctable ECC error</td>
<td>This interrupt is raised by the bus watcher when a ZBus data transfer is marked as having a corrected ECC error. This indicates an corrected ECC from either the L2 cache or memory, or a non-fatal error from one of the other agents. It is cleared by a read of the bus_err_status register.</td>
</tr>
<tr>
<td>18</td>
<td>io_bus_int</td>
<td>Generic bus error: illegal address, timeout waiting ACK, parity error</td>
<td>Raised by an error condition in the generic bus logic. The access that caused the error is logged (see the Generic Bus description). The interrupt is cleared by reading the io_int_status register.</td>
</tr>
</tbody>
</table>
### Table 22: Interrupt Sources (Cont.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
<th>Method to clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>mac_0_int</td>
<td>MAC 0 interrupt</td>
<td>Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the mac_status_0 register or servicing the DMA interrupt.</td>
</tr>
<tr>
<td>20</td>
<td>mac_1_int</td>
<td>MAC 1 interrupt</td>
<td>Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the mac_status_1 register or servicing the DMA interrupt.</td>
</tr>
<tr>
<td>21</td>
<td>mac_2_int</td>
<td>MAC 2 interrupt</td>
<td>Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the mac_status_2 register or servicing the DMA interrupt.</td>
</tr>
<tr>
<td>22</td>
<td>dm_ch_0_int</td>
<td>Data Mover channel 0 interrupt</td>
<td>Raised at the end of data transfer in channel 0 if a data mover descriptor has the interrupt bit set. Cleared by reading the dm_dscr_base_0 register.</td>
</tr>
<tr>
<td>23</td>
<td>dm_ch_1_int</td>
<td>Data Mover channel 1 interrupt</td>
<td>Raised at the end of data transfer in channel 1 if a data mover descriptor has the interrupt bit set. Cleared by reading the dm_dscr_base_1 register.</td>
</tr>
<tr>
<td>24</td>
<td>dm_ch_2_int</td>
<td>Data Mover channel 2 interrupt</td>
<td>Raised at the end of data transfer in channel 2 if a data mover descriptor has the interrupt bit set. Cleared by reading the dm_dscr_base_2 register.</td>
</tr>
<tr>
<td>25</td>
<td>dm_ch_3_int</td>
<td>Data Mover channel 3 interrupt</td>
<td>Raised at the end of data transfer in channel 3 if a data mover descriptor has the interrupt bit set. Cleared by reading the dm_dscr_base_3 register.</td>
</tr>
<tr>
<td>26</td>
<td>mbox_int_0</td>
<td>Mailbox bits [63:48] non zero</td>
<td>Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.</td>
</tr>
<tr>
<td>27</td>
<td>mbox_int_1</td>
<td>Mailbox bits [47:32] non zero</td>
<td>Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.</td>
</tr>
<tr>
<td>28</td>
<td>mbox_int_2</td>
<td>Mailbox bits [31:16] non zero</td>
<td>Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.</td>
</tr>
<tr>
<td>29</td>
<td>mbox_int_3</td>
<td>Mailbox bits [15:0] non zero</td>
<td>Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.</td>
</tr>
<tr>
<td>30</td>
<td>cycle_cp0_int</td>
<td>ZBbus Cycle Count Match 0</td>
<td>Interrupts when the ZBbus cycle count matches compare register 0. Cleared by any write to zbbus_cycle_cp0.</td>
</tr>
<tr>
<td>31</td>
<td>cycle_cp1_int</td>
<td>ZBbus Cycle Count Match 1</td>
<td>Interrupts when the ZBbus cycle count matches compare register 1. Cleared by any write to zbbus_cycle_cp1.</td>
</tr>
<tr>
<td>32</td>
<td>gpio_int_0</td>
<td>GPIO pin 0 interrupt</td>
<td>Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
</tr>
<tr>
<td>33</td>
<td>gpio_int_1</td>
<td>GPIO pin 1 interrupt</td>
<td>Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
</tr>
<tr>
<td>34</td>
<td>gpio_int_2</td>
<td>GPIO pin 2 interrupt</td>
<td>Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
</tr>
<tr>
<td>35</td>
<td>gpio_int_3</td>
<td>GPIO pin 3 interrupt</td>
<td>Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
</tr>
</tbody>
</table>
### Table 22: Interrupt Sources (Cont.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
<th>Method to clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>gpio_int_4</td>
<td>GPIO pin 4 interrupt</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>gpio_int_5</td>
<td>GPIO pin 5 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>gpio_int_6</td>
<td>GPIO pin 6 interrupt</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>gpio_int_7</td>
<td>GPIO pin 7 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>gpio_int_8</td>
<td>GPIO pin 8 interrupt</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>gpio_int_9</td>
<td>GPIO pin 9 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>gpio_int_10</td>
<td>GPIO pin 10 interrupt</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>gpio_int_11</td>
<td>GPIO pin 11 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>gpio_int_12</td>
<td>GPIO pin 12 interrupt</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>gpio_int_13</td>
<td>GPIO pin 13 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>gpio_int_14</td>
<td>GPIO pin 14 interrupt</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>gpio_int_15</td>
<td>GPIO pin 15 interrupt</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Intermits when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the gpio_clr_edge register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the gpio_int_type register to free these vectors for use by HyperTransport interrupts.</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>ldtFatal_int</td>
<td>HyperTransport fatal error interrupt</td>
<td>This bit is set when a fatal error is detected by the HyperTransport interface. The HyperTransport Error Control register is used to classify the HyperTransport errors. Software must clear all fatal error status bits to clear the interrupt.</td>
</tr>
<tr>
<td>49</td>
<td>ldtNonFatal_int</td>
<td>HyperTransport nonfatal error interrupt</td>
<td>This bit is set when a non fatal error is detected by the HyperTransport interface. The HyperTransport Error Control register is used to classify the HyperTransport errors. Software must clear all nonfatal error status bits to clear the interrupt.</td>
</tr>
<tr>
<td>50</td>
<td>ldt_smi</td>
<td>HyperTransport signaled SMI</td>
<td>This bit is reserved in the system interrupt register, and comes only from the ldt_interrupt register. This bit is set when an SMI interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.</td>
</tr>
</tbody>
</table>
### Table 22: Interrupt Sources (Cont.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
<th>Method to clear</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>ldt_nmi</td>
<td>HyperTransport signaled</td>
<td>This bit is reserved in the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NMI</td>
<td>interrupt register, and comes only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>from the ldt_interrupt register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This bit is set when an NMI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interrupt packet directed to this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU had been received from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HyperTransport bus. This bit is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cleared using the interrupt clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register. HyperTransport interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>are directed, so the source for this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit differs for each interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>controller.</td>
</tr>
<tr>
<td>52</td>
<td>ldt_init</td>
<td>HyperTransport signaled</td>
<td>This bit is reserved in the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INIT</td>
<td>interrupt register, and comes only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>from the ldt_interrupt register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This bit is set when an INIT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interrupt packet directed to this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU had been received from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HyperTransport bus. This bit is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cleared using the interrupt clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register. HyperTransport interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>are directed, so the source for this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit differs for each interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>controller.</td>
</tr>
<tr>
<td>53</td>
<td>ldt_startup</td>
<td>HyperTransport signaled</td>
<td>This bit is reserved in the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>STARTUP</td>
<td>interrupt register, and comes only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>from the ldt_interrupt register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This bit is set when a STARTUP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interrupt packet directed to this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU had been received from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HyperTransport bus. This bit is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cleared using the interrupt clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register. HyperTransport interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>are directed, so the source for this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit differs for each interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>controller.</td>
</tr>
<tr>
<td>54</td>
<td>ldt_ext_int</td>
<td>HyperTransport signaled</td>
<td>This bit is reserved in the system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ExtInt</td>
<td>interrupt register, and comes only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>from the ldt_interrupt register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This bit is set when an ExtInt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interrupt packet directed to this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CPU had been received from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HyperTransport bus. This bit is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cleared using the interrupt clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register. HyperTransport interrupts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>are directed, so the source for this</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit differs for each interrupt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>controller.</td>
</tr>
<tr>
<td>55</td>
<td>pci_error_int</td>
<td>PCI error interrupt</td>
<td>This bit is set when an error is seen</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>on the PCI bus. It can be caused by</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCI parity errors, target-abort,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>master-abort, timeout transfer abort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>or a serious system error signalled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>on the SERR pin. The cause can be</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>read from the PCI interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>configuration space (bus=0, dev=0,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>function=0) Status register (bits 8,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11, 12, 13 and 14), and the PCI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Additional Status and Control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register. Software must clear all the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>error status bits to clear the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interrupt.</td>
</tr>
<tr>
<td>56</td>
<td>pci_inta_int</td>
<td>PCI interrupt A</td>
<td>This bit is set when an interrupt is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>signalled on the PCI inta line.</td>
</tr>
<tr>
<td>57</td>
<td>pci_intb_int</td>
<td>PCI interrupt B</td>
<td>This bit is set when an interrupt is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>signalled on the PCI intb line.</td>
</tr>
<tr>
<td>58</td>
<td>pci_intc_int</td>
<td>PCI interrupt C</td>
<td>This bit is set when an interrupt is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>signalled on the PCI intc line.</td>
</tr>
<tr>
<td>59</td>
<td>pci_intd_int</td>
<td>PCI interrupt D</td>
<td>This bit is set when an interrupt is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>signalled on the PCI intd line.</td>
</tr>
<tr>
<td>60</td>
<td>spare_int</td>
<td>Spare interrupt</td>
<td>There is no system interrupt matching</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>this bit. It can be used for vectored</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HyperTransport interrupts.</td>
</tr>
<tr>
<td>61</td>
<td>mac_0_ch1_int</td>
<td>MAC 0 Channel 1 interrupt</td>
<td>Raised when channel 1 transmit or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>receive DMA engine needs service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cleared by any read to mac_status1_0.</td>
</tr>
<tr>
<td>62</td>
<td>mac_1_ch1_int</td>
<td>MAC 1 Channel 1 interrupt</td>
<td>Raised when channel 1 transmit or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>receive DMA engine needs service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cleared by any read to mac_status1_1.</td>
</tr>
<tr>
<td>63</td>
<td>mac_2_ch1_int</td>
<td>MAC 2 Channel 1 interrupt</td>
<td>Raised when channel 1 transmit or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>receive DMA engine needs service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cleared by any read to mac_status1_2.</td>
</tr>
</tbody>
</table>

---

**Note:** The descriptions and methods to clear are specific to the Broadcom system control and debug unit, and may vary depending on the specific hardware and software configurations. Always consult the latest documentation for accurate and comprehensive information.
TIMERS

There are two types of timers on the part: watchdog and general. Watchdog timers can be used to detect the liveliness of the system and will cause a full system reset if software does not prove its correct operation by regularly resetting the timer. General timers are available for any use and can be set to generate a regular interrupt or act as one shot timers. Both sorts of timer count in 1 us steps and have a maximum timeout of over 8 s.

The timers all have a similar programming interface. Each timer module consists of an initial count register, a down counter which is loaded with the initial count and then decrements, and a configuration register. The initial count register should only be written while the timer is disabled. Whenever the timer enable bit in the control register is written with a 1 the down counter is loaded with the initial count and begins decrementing. When the counter reaches zero an interrupt is raised.

WATCHDOG TIMERS

The watchdog timers have a 23 bit counter and are clocked by a 1 MHz clock derived from the 100 MHz reference clock. This allows a maximum count of 8388608, giving the maximum timeout of 8.3888608 s.

A watchdog timer is normally not permitted to timeout. System software should regularly re-enable the timer by writing a 1 to the enable bit in the configuration register. This will restore the count to the initial count and force the counter to restart decrementing from there. If the timer is permitted to reach zero then on the next clock tick an interrupt will be raised, the counter will reload from the initial count and decrementing will continue. This interrupt warns the system software that it has not been re-enabling the timer. The interrupt will be cleared by any write to the configuration register. If the timer times out for a second time (i.e. before the interrupt has been cleared) the default behavior of the watchdog is to signal the system controller to perform a full reset of the part (and any external devices that take reset from the RESETOUT_L, LDT_RESET_L/HT_RESET_L or PCI P_RST_L signals). Once the watchdog has signalled a reset it will be disabled.

There are two watchdog timers on the part. They are identical other than using different registers (distinguished by appending register names with _0 and _1) and different interrupt lines. On the BCM1250 this allows the liveliness of each processor to be separately monitored. This will be of benefit when the two CPUs are running loosely coupled (for example when one runs a real-time task and the other does general system control). The interrupt mapper can be used to route a timeout interrupt from either watchdog to either or both processors (this allows each of them to monitor the other).

The watchdog can be configured to do a more selective reset when it times out for the second time. It can send an sb_softres (as described in Section: “System Control” on page 41), reset just one CPU, or (on the BCM1250) reset both CPUs but no peripherals. Care must be taken if this is done, the reset software will need to probe for devices that are hung or in inconsistent states. The behavior should only be changed from the default in high availability systems or in some cases where the two CPUs are running distinct operating systems.

Care must be taken in systems that boot from an SMBus EEPROM, none of the selective resets set the SMBus channel to boot operation if it has been returned to normal operation. An additional problem with resetting only a CPU is that it will immediately be removed from the coherence machinery. It is possible that the CPU had just been detected as exclusive owner of a cache block and was in the process of providing it when reset. Since the data will never be supplied the ZBbus transaction will never be terminated and the requester will hang.
GENERAL TIMERS

The part has four general timers. Each has a 23 bit counter and is clocked by a 1 MHz clock derived from the 100 MHz reference clock. This allows a maximum count of 8388608, giving the maximum one-shot timeout of 8.388608 s, and a maximum periodic interrupt every 8.388608 s. The timers can be either set to count down from the initial count to zero and stop, or repeatedly count down. A timer should always be disabled before its operating mode is changed.

When the timer is enabled in the one-shot mode the counter is loaded from the initial count register and will decrement to zero. When the count reaches 0, the next clock tick causes an interrupt to be generated and the timer is disabled. The count will remain at 0 until the timer is re-enabled by software. The interrupt is cleared by any write to the configuration register, regardless of whether the timer is kept in the disabled state or enabled. If the timer is re-enabled before its counter reaches 0, its counter is reloaded with the initial count and will restart decrementing.

In the second mode, the counter is loaded from the initial count and will decrement to zero. When the counter reaches zero the next clock tick cause an interrupt to be generated and the counter to be reloaded with the initial count and start counting down again. When the counter is in this mode it will not be reloaded if it is re-enabled (but it will if it is disabled and then enabled again). The interrupt is cleared by any write to the configuration register, if regular interrupts are required this write should re-enable the counter. There is no detection of missed interrupts, once the interrupt has asserted it will only be raised again after it has been cleared and then the counter reaches zero.

TIMER SPECIAL CASES

If the initial count of any timer (regardless of mode) is set to zero and the timer is enabled then an interrupt will be signalled every clock tick. It is UNPREDICTABLE if writing the configuration register will cause the interrupt to deassert. It is UNPREDICTABLE how many ticks a watchdog timer will take to assert reset.

If the initial count register is updated or the mode is changed between one-shot and continuous while the counter is enabled the behavior is UNPREDICTABLE.

ZBBus CYCLE COUNT AND COMPARE

There is an additional read only counter that is initialized to zero and increments every ZBbus cycle. Even in the fastest parts it will take more than 1000 years for the counter to wrap. It can therefore be used to provide a sequence number that is unique and monotonically increasing through the runtime of the system. This is useful for software that needs to assign an order to events. The counter may also be used as a high resolution clock for timing. The read only zbus_cycle_count register is placed at the start of a 64KB memory range, if additional registers are added in this range they will be Read Only with no side effects. This range can therefore be safely mapped into user address space. There are two comparison registers which will raise an interrupt when the count matches the comparison value. The interrupt will remain asserted until the compare register is written. Note that the interrupt is only raised on equality, so writing the compare register with a value lower than the current count will ensure the interrupt is never raised. Since cycle 0 is guaranteed to have passed by the time the CPU executes the first instruction, writing the compare register with zero is always a safe way to clear the interrupt and disable it.
### TIMER REGISTERS

**Table 23: Watchdog Timer Initial Count Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>watchdog_timer_init_cnt</td>
<td>23'bx</td>
<td>Watchdog timer initial count register. Sets the watchdog timeout time in microseconds. This register should only be written when the timer is disabled. If written when the timer is enabled the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td></td>
<td>63:23 reserved</td>
<td>41'h0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Table 24: Watchdog Timer Current Count Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>watchdog_timer_cnt</td>
<td>23'bx</td>
<td>Watchdog timer current count register. When the counter is enabled the count decrements every microsecond.</td>
</tr>
<tr>
<td></td>
<td>63:23 reserved</td>
<td>41'h0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Table 25: Watchdog Timer Configuration Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>watchdog_timer_enable</td>
<td>1'b0</td>
<td>When this bit is written with a 0 the timer will be disabled. When this bit is written with a 1 regardless of its previous state the timer will be loaded from the initial count and start decrementing.</td>
</tr>
<tr>
<td>1</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>4:2</td>
<td>reset_type</td>
<td>3'b0</td>
<td>This field sets the type of reset that is generated when the watchdog times out for the second time. In most situations the (default) full system reset is the correct behavior, since this will clear all state. xx0: Full system reset (Default). 001: SB Soft Reset (full reset retaining system_cfg register value). 011: Reset only CPU 0. 101: Reset only CPU 1. 111: Reset only CPU 0 and CPU 1 (no peripherals).</td>
</tr>
<tr>
<td>5</td>
<td>wd_has_reset</td>
<td>1'bx</td>
<td>This bit is reserved prior to PERIPH_REV3. On PERIPH_REV3 and later it indicates if the watchdog timer has caused a reset. Following power up the value of this bit is UNPREDICTABLE. Whenever this register is written this bit becomes zero. If the watchdog causes a reset this bit is set.</td>
</tr>
<tr>
<td>7:6</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
### Table 26: General Timer Initial Count Registers

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_timer_init_cnt_0 - 00_1002_0070</td>
<td>General Timer Initial Count Register 0</td>
</tr>
<tr>
<td>general_timer_init_cnt_1 - 00_1002_0078</td>
<td>General Timer Initial Count Register 1</td>
</tr>
<tr>
<td>general_timer_init_cnt_2 - 00_1002_0170</td>
<td>General Timer Initial Count Register 2</td>
</tr>
<tr>
<td>general_timer_init_cnt_3 - 00_1002_0178</td>
<td>General Timer Initial Count Register 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22:0</td>
<td>timer_init_cnt</td>
<td>23'bx</td>
<td>Timer initial count register. This sets the maximum value of the counter, and will be one less than the timeout in microseconds for one-shot mode, and one less than the period in microseconds in continuous mode. This register should only be written when the timer is disabled. If written when the timer is enabled the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td>63:23</td>
<td>reserved</td>
<td>41'h0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 27: General Timer Current Count Registers

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_timer_cnt_0 - 00_1002_0080</td>
<td>General Timer Current Count Register 0</td>
</tr>
<tr>
<td>general_timer_cnt_1 - 00_1002_0088</td>
<td>General Timer Current Count Register 1</td>
</tr>
<tr>
<td>general_timer_cnt_2 - 00_1002_0180</td>
<td>General Timer Current Count Register 2</td>
</tr>
<tr>
<td>general_timer_cnt_3 - 00_1002_0188</td>
<td>General Timer Current Count Register 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>22:0</td>
<td>timer_cnt</td>
<td>23'bx</td>
<td>Timer current count register. When the timer is running the count decrements every microsecond.</td>
</tr>
<tr>
<td>63:23</td>
<td>reserved</td>
<td>41'h0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 28: General Timer Configuration Registers

<table>
<thead>
<tr>
<th>Address Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_timer_cfg_0 - 00_1002_0090</td>
<td>General Timer Configuration Register 0</td>
</tr>
<tr>
<td>general_timer_cfg_1 - 00_1002_0098</td>
<td>General Timer Configuration Register 1</td>
</tr>
<tr>
<td>general_timer_cfg_2 - 00_1002_0190</td>
<td>General Timer Configuration Register 2</td>
</tr>
<tr>
<td>general_timer_cfg_3 - 00_1002_0198</td>
<td>General Timer Configuration Register 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>general_timer_cfg_enable</td>
<td>1'b0</td>
<td>When high, enable the general purpose timer. If the timer is disabled when this bit is written with a 1 the timer will be loaded from the initial count register and will start decrementing. If the timer is enabled when this bit is written with a 1 the timer continues to decrement, but if the general_timer_cfg_mode bit is 0 the counter will also be reloaded with the initial count. If this bit is written with a 0 the timer will stop decrementing.</td>
</tr>
<tr>
<td>1</td>
<td>general_timer_cfg_mode</td>
<td>1'b0</td>
<td>When low, the general purpose timer stops at 0. When high, the general purpose timer runs continuously.</td>
</tr>
<tr>
<td>7:2</td>
<td>reserved</td>
<td>6'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
SYSTEM PERFORMANCE COUNTERS

There are four system performance counters. They count system events and interrupt when saturated. All four counters are cleared and/or enabled simultaneously. When one counter saturates, all counters are frozen. Thus, the readings across the four counters cover the same interval. The counters are writable; if they are written while active the count will continue from the new value (note that the time from the CPU issuing a store to the counter getting the new value is UNPREDICTABLE).

The counters are all controlled by the `perf_cnt_cfg` register. This is used to configure the count source for each counter, to enable or disable all the counters and to clear all the counters. The counter clear bit is write only, the counters will all be cleared to zero at the end of the cycle that this bit is written with a 1. If both the enable and clear bits are written with 1 in the same write, the counters are cleared and will start counting the cycle after the write completes.

If any counter reaches its maximum count the counters will all be disabled (the enable bit in the `perf_cnt_cfg` register is cleared) and the `perf_cnt_int` interrupt will be raised. The interrupt is removed and all the counters are cleared when the clear bit in the `perf_cnt_cfg` register is written with a 1.

Since it takes a cycle to disable the counters there is one additional cycle sampled after one (or more) of the counts reaches FF_FFFF_FFFF during which the counter may overflow (if its source is still active). There is an additional bit in the performance counters that gets set when they overflow, thus the counter (or counters) that disabled monitoring could have the value FF_FFFF_FFFF or 100_0000_0000 when the interrupt is seen.

### Table 29: ZBbus Count Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>count</td>
<td>64'h0</td>
<td>Count of ZBbus cycles since reset. Writes will be ignored.</td>
</tr>
</tbody>
</table>

### Table 30: ZBbus Count Compare Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>compare</td>
<td>64'hFFFF_FFFF_FFFF</td>
<td>Value is compared to zbbus_cycle_count and the interrupt corresponding to the register is raised when the count is equal to the value. The interrupt is cleared by any write to the compare register.</td>
</tr>
</tbody>
</table>

### Table 31: System Performance Counter Configuration Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>Counter 0 source</td>
<td>8'b0</td>
<td>Sets the source for counter 0.</td>
</tr>
<tr>
<td>15:8</td>
<td>Counter 1 source</td>
<td>8'b0</td>
<td>Sets the source for counter 1.</td>
</tr>
<tr>
<td>23:16</td>
<td>Counter 2 source</td>
<td>8'b0</td>
<td>Sets the source for counter 2.</td>
</tr>
</tbody>
</table>
Table 31: System Performance Counter Configuration Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>Counter 3 source</td>
<td>8'b0</td>
<td>Sets the source for counter 3.</td>
</tr>
<tr>
<td>32</td>
<td>clear</td>
<td>1'b0</td>
<td>If this bit is set in a write to the register the counters will be reset,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the bit self-clears and always reads as zero.</td>
</tr>
<tr>
<td>33</td>
<td>enable</td>
<td>1'b0</td>
<td>Set to enable the counters, clear to stop them. Cleared automatically by</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>any counter reaching its maximum count.</td>
</tr>
<tr>
<td>63:34</td>
<td>reserved</td>
<td>30'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 32: System Performance Counters

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>Count</td>
<td>40'bx</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>40</td>
<td>oflow</td>
<td>1'b0</td>
<td>This bit is set if the counter overflows.</td>
</tr>
<tr>
<td>63:41</td>
<td>reserved</td>
<td>23'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 33: System Performance Counter Sources

<table>
<thead>
<tr>
<th>Value</th>
<th>Condition Counted</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No count.</td>
</tr>
<tr>
<td>01</td>
<td>Bus cycles are counted. The counter is incremented each ZBbus clock cycle (half</td>
</tr>
<tr>
<td></td>
<td>the CPU clock).</td>
</tr>
<tr>
<td>02</td>
<td>Bus cycles used. The counter is incremented every cycle the address portion of</td>
</tr>
<tr>
<td></td>
<td>ZBbus is used.</td>
</tr>
<tr>
<td>03</td>
<td>Address bus cycles waiting grant. The counter is incremented every cycle some</td>
</tr>
<tr>
<td></td>
<td>agent loses arbitration for the address bus (i.e. there are two or more requesters</td>
</tr>
<tr>
<td></td>
<td>in arbitration).</td>
</tr>
<tr>
<td>04</td>
<td>Address bus cycles arbitrated but not used. The counter is incremented every</td>
</tr>
<tr>
<td></td>
<td>time the address bus is granted but the agent drives a NOP command.</td>
</tr>
<tr>
<td>05</td>
<td>Address match 0 count - counts each bus address cycle that matches the address</td>
</tr>
<tr>
<td></td>
<td>trap comparator. The occurrence counter (that is used to determine if the</td>
</tr>
<tr>
<td></td>
<td>address trap should interrupt) is ignored, the performance counter will increase</td>
</tr>
<tr>
<td></td>
<td>every cycle there is a hit in the trap.</td>
</tr>
<tr>
<td>06</td>
<td>Address match 1 count. (See comments for Address match 0.)</td>
</tr>
<tr>
<td>07</td>
<td>Address match 2 count. (See comments for Address match 0.)</td>
</tr>
<tr>
<td>08</td>
<td>Address match 3 count. (See comments for Address match 0.)</td>
</tr>
<tr>
<td>09</td>
<td>Data bus cycles waiting grant. The counter is incremented every cycle some agent</td>
</tr>
<tr>
<td></td>
<td>loses arbitration for the data bus (i.e. there are two or more requesters in</td>
</tr>
<tr>
<td></td>
<td>arbitration).</td>
</tr>
<tr>
<td>0A</td>
<td>Data bus cycles arbitrated but not used. The counter is incremented every time</td>
</tr>
<tr>
<td></td>
<td>the data bus is granted but the agent drives a NOP command.</td>
</tr>
<tr>
<td>0B</td>
<td>CPU0 EDEN. The counter is incremented every time CPU 0 asserts its External Debug</td>
</tr>
<tr>
<td></td>
<td>Event Notification (EDEN) signal. This can be asserted by software, or watchpoint</td>
</tr>
<tr>
<td></td>
<td>register hits.</td>
</tr>
<tr>
<td>0C</td>
<td>CPU1 EDEN. The counter is incremented every time CPU 1 asserts its External Debug</td>
</tr>
<tr>
<td></td>
<td>Event Notification (EDEN) signal. This can be asserted by software, or watchpoint</td>
</tr>
<tr>
<td></td>
<td>register hits.</td>
</tr>
</tbody>
</table>
Interrupt request cycles. The counter is incremented every bus cycle the int_trace_trigger_x output of the interrupt mapper is asserted (see Figure 10 on page 51). This records the time taken from the interrupt assertion to software clearing the interrupt condition. If a single interrupt source is selected and the count is read at the start of the interrupt service routine and cleared when the interrupt is cleared, software will be able to track the interrupt service time. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.

Interrupt request count. The counter is incremented every time the int_trace_trigger_x output of the interrupt mapper changes from not asserted to asserted (see Figure 10 on page 51). This records the number of interrupts, and can be used with the interrupt request cycles count to determine the average number of cycles between an interrupt being raised and cleared. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.

Agent n Request count - counts number of cycles that agent n requested the bus.

Agent n Grant delay count - counts the total sum of cycles between agent n requesting the bus, and agent n receiving grant on the bus.

Memory channel 0 read accesses. Counts the number of reads to memory.

Memory channel 0 write accesses. Counts the number of writes to memory.

Memory channel 0 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.

Memory channel 0 page hits. Counts the number of page hits in the memory controller.

Memory channel 0 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.

Memory channel 1 read accesses. Counts the number of reads to memory.

Memory channel 1 write accesses. Counts the number of writes to memory.

Memory channel 1 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.

Memory channel 1 page hits. Counts the number of page hits in the memory controller.

Memory channel 1 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.

L2 read requests. Counts the number of read requests to the L2 cache.

L2 read misses. Counts the number of read requests that missed the L2 cache.

L2 write requests. Counts the number of write requests to the L2 cache.

L2 write misses. Counts the number of write requests that missed the L2 cache.

L2 evicts. Counts the number of lines evicted from the L2 cache.

Reserved

Table 33: System Performance Counter Sources (Cont.)

<table>
<thead>
<tr>
<th>Value</th>
<th>Condition Counted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0D</td>
<td>Interrupt request cycles. The counter is incremented every bus cycle the int_trace_trigger_x output of the interrupt mapper is asserted (see Figure 10 on page 51). This records the time taken from the interrupt assertion to software clearing the interrupt condition. If a single interrupt source is selected and the count is read at the start of the interrupt service routine and cleared when the interrupt is cleared, software will be able to track the interrupt service time. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.</td>
</tr>
<tr>
<td>0E</td>
<td>Interrupt request count. The counter is incremented every time the int_trace_trigger_x output of the interrupt mapper changes from not asserted to asserted (see Figure 10 on page 51). This records the number of interrupts, and can be used with the interrupt request cycles count to determine the average number of cycles between an interrupt being raised and cleared. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.</td>
</tr>
<tr>
<td>0F</td>
<td>Reserved</td>
</tr>
<tr>
<td>1n</td>
<td>Agent n Request count - counts number of cycles that agent n requested the bus.</td>
</tr>
<tr>
<td>2n</td>
<td>Agent n Grant delay count - counts the total sum of cycles between agent n requesting the bus, and agent n receiving grant on the bus.</td>
</tr>
<tr>
<td>30</td>
<td>Memory channel 0 read accesses. Counts the number of reads to memory.</td>
</tr>
<tr>
<td>31</td>
<td>Memory channel 0 write accesses. Counts the number of writes to memory.</td>
</tr>
<tr>
<td>32</td>
<td>Memory channel 0 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.</td>
</tr>
<tr>
<td>33</td>
<td>Memory channel 0 page hits. Counts the number of page hits in the memory controller.</td>
</tr>
<tr>
<td>34</td>
<td>Memory channel 0 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.</td>
</tr>
<tr>
<td>35</td>
<td>Memory channel 1 read accesses. Counts the number of reads to memory.</td>
</tr>
<tr>
<td>36</td>
<td>Memory channel 1 write accesses. Counts the number of writes to memory.</td>
</tr>
<tr>
<td>37</td>
<td>Memory channel 1 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.</td>
</tr>
<tr>
<td>38</td>
<td>Memory channel 1 page hits. Counts the number of page hits in the memory controller.</td>
</tr>
<tr>
<td>39</td>
<td>Memory channel 1 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.</td>
</tr>
<tr>
<td>3A</td>
<td>L2 read requests. Counts the number of read requests to the L2 cache.</td>
</tr>
<tr>
<td>3B</td>
<td>L2 read misses. Counts the number of read requests that missed the L2 cache.</td>
</tr>
<tr>
<td>3C</td>
<td>L2 write requests. Counts the number of write requests to the L2 cache.</td>
</tr>
<tr>
<td>3D</td>
<td>L2 write misses. Counts the number of write requests that missed the L2 cache.</td>
</tr>
<tr>
<td>3E</td>
<td>L2 evicts. Counts the number of lines evicted from the L2 cache.</td>
</tr>
<tr>
<td>3F-FF</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
BUS WATCHER

The Bus Watcher monitors ZBbus data transfers, detecting error reports. Corrected ECC errors from memory and the L2 cache are counted. Uncorrectable ECC errors are counted and logged. The bus watcher will raise an interrupt when it detects an error.

When a data bus transaction is made with an error code the bus watcher logs the transaction (in the bus_err_data registers and the the bus_err_status register), and increments the appropriate counter. If the error was an unrecoverable one (uncorrectable ECC error or bus error or fatal error) the log is frozen until the bus_err_status register is read. (The unrecoverable errors are Dcode=4-7).

The bus watcher contains the following counters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_l2_cor_d_ecc</td>
<td>Counts Correctable L2 cache data ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_l2_bad_d_ecc</td>
<td>Counts Uncorrectable L2 cache data ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_l2_cor_t_ecc</td>
<td>Counts Correctable L2 cache tag ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_l2_bad_t_ecc</td>
<td>Counts Uncorrectable L2 cache tag ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_mem_cor_d_ecc</td>
<td>Counts Correctable memory data ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_mem_bad_d_ecc</td>
<td>Counts Uncorrectable memory data ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
<tr>
<td>bus_error</td>
<td>Counts fatal errors and bus errors. The bad_ecc_int interrupt is raised every time this counter increments.</td>
</tr>
</tbody>
</table>

Each counter is 8 bits, and saturates at 8'hff. Software may write any value into the counters, a counter is cleared by writing zero to it. The counters are collected into two 32-bit registers, bus_l2_errors for the L2 cache counts and bus_mem_io_errors for the other counts. These registers should only be written with 32-bit or 64-bit writes, using smaller will result in the value of some of the counters becoming UNPREDICTABLE.

In addition to the type of error the bus_err_status register records information about the transaction that encountered the error. The upper four bits of the transaction id can be used to determine which of the ZBbus agents initiated the transaction that resulted in an error, see Table 1 on page 20 for the number of each agent. In some cases the lower six bits of the transaction id can be decoded to get more information about the request, see Table 51 on page 81. The responder id gives the number of the agent that reported the error. The data that was returned with the error code is also captured, but in many cases it will be of UNPREDICTABLE value.

Note that when a bus error or fatal error is returned to a CPU it will take a bus error exception and when an uncorrectable ECC error is returned it will take a cache error exception. These exceptions are taken with a higher priority than the interrupt from the bus watcher, or any interrupt raised by the source of the error. DMA engines that receive any of these errors will stop and report the error in some way. If any of these errors are returned to the PCI or HyperTransport interfaces they will be converted into the appropriate error returned on the interface and flags will be set in the CSRs to indicate this has been done. Thus an error is likely to be reported several times, ensuring that processing is never done based on corrupted data. The correctable
errors do not raise exceptions or cause the DMA engines to stop, so the Bus Watcher interrupt is likely to be the only way they are reported.

### Table 35: Bus Watcher Error Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>17:8</td>
<td>tid</td>
<td>10'b0</td>
<td>Transfer ID that had error code. Top 4 bits indicate the initiator of the transfer, see Table 1 on page 20 for the mapping. Low 6 bits may give more details on the what within the agent caused the transaction, see Table 51 on page 81.</td>
</tr>
<tr>
<td>21:18</td>
<td>rid</td>
<td>4'b0</td>
<td>Responder ID identifies the agent that reported the error. See Table 1 on page 20.</td>
</tr>
<tr>
<td>24:22</td>
<td>dcode</td>
<td>3'b0</td>
<td>Data transfer error code. See Table 6 on page 24.</td>
</tr>
<tr>
<td>29:25</td>
<td>reserved</td>
<td>5'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>mult_errors</td>
<td>1'b0</td>
<td>Set to indicate multiple errors. The first fatal one, or the most recent non-fatal one is logged (all were counted).</td>
</tr>
<tr>
<td>31</td>
<td>reserved</td>
<td>1'b1</td>
<td>Reserved. Reads as 1.</td>
</tr>
<tr>
<td>63:32</td>
<td>notimp</td>
<td>32'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 36: Bus Watcher Error Status Debug Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>value</td>
<td>64'hx</td>
<td>This register contains the same information as the bus_err_status register, but reads do not have any side effects.</td>
</tr>
</tbody>
</table>

### Table 37: Bus Watcher Error Data Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>data</td>
<td>64'hx</td>
<td>Data from error transaction. The register number is address bits [4:3]. Register _0 contains ZBus bits 255:192 Register _1 contains ZBus bits 191:128 Register _2 contains ZBus bits 127:64 Register _3 contains ZBus bits 63:0</td>
</tr>
</tbody>
</table>
### Table 38: Bus Watcher L2 ECC Counter Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td><code>l2_cor_d_ecc</code></td>
<td>8'b0</td>
<td>Count of correctable L2 cache data errors, saturates at 8'hff, write the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register with zero to clear the count.</td>
</tr>
<tr>
<td>15:8</td>
<td><code>l2_bad_d_ecc</code></td>
<td>8'b0</td>
<td>Count of uncorrectable L2 cache data errors, saturates at 8'hff, write the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register with zero to clear the count.</td>
</tr>
<tr>
<td>23:16</td>
<td><code>l2_cor_t_ecc</code></td>
<td>8'b0</td>
<td>Count of correctable L2 tag errors, saturates at 8'hff, write the register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>with zero to clear the count.</td>
</tr>
<tr>
<td>31:24</td>
<td><code>l2_bad_t_ecc</code></td>
<td>8'b0</td>
<td>Count of uncorrectable L2 tag errors, saturates at 8'hff, write the register</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>with zero to clear the count.</td>
</tr>
<tr>
<td>63:32</td>
<td><code>notimp</code></td>
<td>32'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 39: Bus Watcher Memory and I/O Error Counter Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td><code>mem_cor_d_ecc</code></td>
<td>8'b0</td>
<td>Count of correctable memory data errors, saturates at 8'hff, write the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register with zero to clear the count.</td>
</tr>
<tr>
<td>15:8</td>
<td><code>mem_bad_d_ecc</code></td>
<td>8'b0</td>
<td>Count of uncorrectable memory data errors, saturates at 8'hff, write the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register with zero to clear the count.</td>
</tr>
<tr>
<td>23:16</td>
<td><code>bus_error</code></td>
<td>8'b0</td>
<td>Count of bus errors and fatal bus errors, saturates at 8'hff, write the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>register with zero to clear the count.</td>
</tr>
<tr>
<td>31:24</td>
<td><code>reserved</code></td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:32</td>
<td><code>notimp</code></td>
<td>32'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
ADDRESS TRAPPING

There are four address traps in the SCD. An interrupt is raised by any access that falls within the address range specified by a trap. Hits to a trap can also be used to trigger the performance counters and trace unit. The address traps are mainly used as a debugging aid, but they can also be used to detect (but not prevent) illegal accesses in the event of a system failure.

Each address trap has the same interface consisting of a set of registers. The register names described should have _0, _1, _2, or _3 appended to indicate which trap is being used.

A 40 bit upper address register addr_trap_up sets the top physical address in the trapping range. An address must be less than or equal to this value to be part of the range. A 40 bit lower address register addr_trap_down sets the base physical address in the trapping range. An address must be greater than or equal to this value to be part of the range.

The fields in the addr_trap_cfg register are used to make the trap more selective than just based on an address range. A two bit access_type field selects if the trap is active for read, write or all accesses. The source_id field contains a ZBbus source id that is compared at the same time as the address. Flags determine if the trap is hit when the access is inside or outside the address range, and if the source id must match or differ.

Each trap has a 3 bit occurrence counter in the addr_trap_cfg register that counts down every time the trap condition is hit. Once the occurrence counter is zero it will stick at zero and not continue to decrement.

A simple trap ignores the source information. It is setup by programming the upper and lower address registers to set the range, then writing the configuration register with the access type to trap on, and a greater than 0 occurrence count. Any access of the selected type with an address greater than or equal to the lower register and less than or equal to the upper register matches the range. Thus to trap on a specific address the upper and lower addresses are set to be the same. When an access matches and the occurrence counter is greater than zero, the counter will decrement.

 Provided the address trap interrupt is clear, any address trap match will log the address that matched in the addr_trap_reg and the trap number in the addr_trap_index. The first time an occurrence counter from any trap decrements from 1 to 0 the address trap interrupt is raised, both signalling the event and preventing the log registers from being overwritten. Further trap matches will continue to decrement their counters, but will otherwise be ignored until the interrupt is cleared by a read of the addr_trap_reg. The CPU should read the addr_trap_index before the addr_trap_reg to ensure a consistent result.

More complex traps can be made by inverting the output of the range match (the trap is only hit by addresses outside the range) and considering the source of the access in addition to the address range. The source id that is provided on the ZBbus by the initiator of the transaction (as the transaction ID bits A_ID[9:6]) is used in the comparison. There are four forms of complex trap:

1. Address match, source match. This is used to detect a particular agent accessing a particular memory range.
2. Address differs, source match. This will detect an access by a device outside of the range of addresses that it is expected to use.
3. Address match, source differs. If a region of memory is reserved for use by a single agent (for example CPU 1) this will detect an access to the region by any other agent. (Note that this provides detection only, not prevention).
4. Address differs, source differs. If all of the memory can be used by one agent (for example the master CPU), but all other agents are confined to a particular region, this will detect illegal accesses by other agents. It could be used if only the master CPU were given access to I/O addresses and other agents are confined to memory.
The complex trap can also use the cacheability attributes to determine if the request matches the trap. The trap can be set to trigger if the access is Uncached, Cacheable Coherent or Cacheable Non-coherent. It can also only trigger for requests that do not match the expected cacheability. This can aid debugging by detecting accesses to areas of the address space that are made with the wrong attribute. The complex traps use the occurrence counter in the same way as simple traps.

When an access hits the trap (ignoring the value of the occurrence counter) a signal is provided to the trace trigger logic (described in Section: “Trigger Events” on page 70) and to the performance counter selector (Section: “System Performance Counters” on page 61).

| Table 40: Address Trap Trigger Index Register |
| addr_trap_index - 00_1002_00B0 |
| Bits | Name | Default | Description |
| 3:0 | addr_trap_index | 4’b0 | Trap that triggered the interrupt. |
| 7:4 | reserved | 4’h0 | Reserved |

| Table 41: Address Trap Trigger Debug Register |
| addr_trap_reg_debug - 00_1002_0460 |
| Bits | Name | Default | Description |
| 63:0 | value | 64hx | This register contains the same information as the addr_trap_reg register, but reads do not have any side effects. |

| Table 42: Address Trap Trigger Address Register |
| addr_trap_reg - 00_1002_00B8 |
| Bits | Name | Default | Description |
| 4:0 | addr_trap | 5’h0 | Always reads as zero. |
| 39:5 | addr_trap | 35’h0 | Address that triggered the interrupt. |
| 63:40 | reserved | 24’h0 | Reserved |

| Table 43: Address Trap Range Top Address Registers |
| addr_trap_up_0 - 00_1002_0400 |
| addr_trap_up_1 - 00_1002_0408 |
| addr_trap_up_2 - 00_1002_0410 |
| addr_trap_up_3 - 00_1002_0418 |
| Bits | Name | Default | Description |
| 39:0 | addr_trap_up | 40’hx | Top address in range to compare. |
| 63:40 | reserved | 24’h0 | Reserved |
### Table 44: Address Trap Range Base Address Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:39</td>
<td>addr_trap_down</td>
<td>40'hx</td>
<td>Base address in range to compare.</td>
</tr>
<tr>
<td>40:63</td>
<td>reserved</td>
<td>24'h0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 45: Address Trap Configuration Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:2</td>
<td>addr_trap_cnt</td>
<td>3'b0</td>
<td>Address trap count. Decrement on address range matches, sticking at 0. The 1 to 0 transition will trigger an interrupt. Note that this count is not applied when an address trap hit is used to trigger a trace event.</td>
</tr>
<tr>
<td>3</td>
<td>addr_trap_access_type</td>
<td>1'b0</td>
<td>When low, indicates read trap. When high, indicates write trap.</td>
</tr>
<tr>
<td>4</td>
<td>all_accesses</td>
<td>1'b0</td>
<td>Ignore the state of bit 3, and trap on any matching access.</td>
</tr>
<tr>
<td>5</td>
<td>addr_inv</td>
<td>1'b0</td>
<td>These bits enable complex matches by allowing the source agent of the transaction to be compared, and inverting the sense of both address and source comparisons.</td>
</tr>
<tr>
<td>6</td>
<td>use_source</td>
<td>1'b0</td>
<td>Usefull values are:</td>
</tr>
<tr>
<td>7</td>
<td>source_inv</td>
<td>1'b0</td>
<td>x00: Trap for all sources if address is in the range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x01: Trap for all sources if address is not in the range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010: Trap if top 4 bits of tid matches source and address is in the range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011: Trap if top 4 bits of tid matches source and address is outside the range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110: Trap if top 4 bits of tid does not match source and address is in the range.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111: Trap if top 4 bits of tid does not match source and address is outside the range.</td>
</tr>
<tr>
<td>11:8</td>
<td>source_id</td>
<td>4'b0</td>
<td>Agent ID to use in source comparison.</td>
</tr>
<tr>
<td>13:12</td>
<td>cattr</td>
<td>2'b00</td>
<td>These bits enable complex matches by allowing the cacheability attribute of the transaction to be compared, and having the trap match if the access has or does not have the expected attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Usefull values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>x00: Ignore the cacheability attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>001: Trap if Uncachable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010: Trap if Cacheable Non-coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011: Trap if Cacheable coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>101: Trap if cacheable (not Uncacheable).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110: Trap if not Cacheable Non-coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111: Trap if not Cacheable Coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The cacheability is decoded from the A_L1CA signal:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2'b00: Cacheable Non-coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2'b01: Cacheable Coherent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2'b1x: Uncacheable.</td>
</tr>
<tr>
<td>63:15</td>
<td>notimp</td>
<td>49'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
TRACE UNIT

The trace unit allows ZBbus activity to be non-intrusively traced. There is a 12KB buffer into which a trace of the address or data activity on the bus can be written. The trace may be read out either by a CPU or through the JTAG interface. Triggers and filters allow control of the activity that is traced.

The trace buffer contains 256 entries, each of 384 bits. Each entry can hold either three address/control bundles (each 128 bits) or one full bundle containing address/control information (128 bits) and a databus snapshot (256 bits). When a full bundle is captured it will always start a new buffer entry, marking any unused address/control slots in the previous entry empty.

The trace buffer is reset either by software or using the JTAG interface. A trigger is used to start collection of a trace. Once collection is enabled, filters are applied to every bus cycle which select not sampling, taking an address sample or taking a complete sample that cycle (the filter can also select that every cycle is recorded). The buffer pointer will advance modulo 256, thus the trace RAM is used as a circular buffer leaving the latest 256 entries available. A trigger can stop collection, the trace buffer pointer maintains its value and will continue filling the buffer when a start trigger is again encountered. A trigger can also freeze collection. Following this, collection cannot be restarted until the trace buffer is reset. An option can be set to freeze collection the first time the buffer fills, this turns the circular buffer into a linear buffer.

The same mechanism is used to construct both triggers and filters, they only differ in the flags that are set in the control register. There are two components: events and sequences. A trigger event is signalled when the cycle on the ZBbus (or some other system state) matches a condition. Up to four events are concatenated to make a sequence. The sequence also includes an action, which is one of the trace buffer commands (start, stop or freeze) and an indication if the trace buffer should record information from the ZBbus cycle that is in progress as the sequence completes. When all the events in the sequence have been signalled sequentially the sequence is complete and the action associated with the sequence will take place.

TRIGGER EVENTS

A trigger event is specified as a conditional match on the ZBbus address phase signals, or a match on ZBbus data phase signals, or the external debug pin being asserted, or a CPU interrupt being raised. There are eight trigger events (event0 - event7), all identical.

The address phase match consists of three parts, all of which must be true for the trigger to happen:

1. True if
   the req_id_match bit is set and the transaction id source on the bus matches the req_id field.
   Always true if the req_id_match bit is clear.

2. True if
   the addr_match0 bit is set and the address on the bus matches the address trap0 range,
   or the addr_match1 bit is set and the address on the bus matches the address trap1 range,
   or the addr_match2 bit is set and the address on the bus matches the address trap2 range,
   or the addr_match3 bit is set and the address on the bus matches the address trap3 range.
   Always true if none of the addr_match bits are set.

   The complex (source and cache attribute) address trap match is used to determine a hit, but the trigger does not use the count in the address trap.

3. True if
   the read bit is set and the bus command is READ_SHD or READ_EXC,
   or the write bit is set and the command is WRITE, WRITE and INVALIDATE, or INVALIDATE.
   Note that having both read and write bits clear disables the address section of the trigger.
The event is triggered by the data phase if the data_id_match bit is set and the top four bits of the transaction ID on the data bus (which identify the agent that initiated the transfer) match the data_id field, or the resp_id_match bit is set and the responder ID (which provided the data) matches the resp_id field. However, if both data_id_match and resp_id_match bits are set then both IDs have to match for the event to trigger.

A CPU interrupt can trigger the event. Each interrupt mapper contains an interrupt trace mask register. Bits are set in this register to indicate that when the corresponding interrupt is raised the event should be triggered. The trace event will trigger when the int_trace_trigger output (see Figure 10 on page 51) of either interrupt mapper is asserted. This can be used to start tracing when the CPU gets an interrupt from the device being studied. The trigger is edge sensitive and only happens on the change from neither int_trace_trigger outputs being asserted to at least one being asserted. The trigger will not happen again until after the trace outputs from both mappers have cleared.

An external device can trigger an event by pulling the DEBUG_L pin low. DEBUG_L is a bidirectional open collector signal and must have an external pull-up to 3.3V. The SCD can signal the completion of a trigger event sequence by pulling the signal low for 10 ZBus cycles (see Section: “Trigger Sequences” on page 73). The state of the DEBUG_L signal is ignored during the 10 cycles that the SCD is pulling it low and for 5 ZBus cycles after the SCD stops driving, so an event will not be triggered by the SCD signalling a sequence completion. The trigger is edge sensitive, it will happen once when DEBUG_L changes from high to low and will not happen again until DEBUG_L has returned high.

Expressing the event trigger in pseudo-code:

```pseudocode
if ((
    // Address section
    (((trace_event_N[15:12] == zb_req_id) | ~trace_event_N[4]) &
      (trace_event_N[0] & addr_match0) |
      (trace_event_N[1] & addr_match1) |
      (trace_event_N[2] & addr_match2) |
      (trace_event_N[3] & addr_match3) |
      (trace_event_N[3:0] == 4'h0)) &
    (trace_event_N[10] & write))
    ) |

    // Data section
    ((trace_event_N[23:20] == zb_data_id) &
    ) |

    // Interrupt
    (trace_event_N[7] & edge_detect (interrupt)) |
    // External pin
    (trace_event_N[9] & edge_detect (~debug_pin))
) then trigger event N
```
Each trigger event has a counter associated with it. If the counter is greater than zero when the trigger condition is true then the counter decrements. If the counter is zero then the event is signaled to the trigger sequencers, and the counter will be reloaded with its initial value.

The trace event register is shown in Table 46. There are eight such registers `trace_event_0` to `trace_event_7`.

### Table 46: Trace Event Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>Addr_match[3:0]</td>
<td>4'b0</td>
<td>Each of these bits corresponds to an address trap register (see Section: “Address Trapping” on page 67). If none of these bits are set then the address on the bus is not considered as part of the trigger. If any of these bits are set then the address is considered part of the trigger, and the event only triggers if the address falls within the range specified by the corresponding trap. Both simple and complex traps may be used. The event will trigger whenever there is a hit in the trap, the occurrence counter in the trap is ignored.</td>
</tr>
<tr>
<td>4</td>
<td>req_id_match</td>
<td>1'b0</td>
<td>When set the address portion of the event only occurs if the requester ID of the transaction matches the req_id field of this register.</td>
</tr>
<tr>
<td>5</td>
<td>data_id_match</td>
<td>1'b0</td>
<td>When set the data portion of the event only occurs if the data ID of the transaction matches the data_id field of this register.</td>
</tr>
<tr>
<td>6</td>
<td>resp_id_match</td>
<td>1'b0</td>
<td>When set the data portion of the event only occurs if the responder ID of the transaction matches the resp_id of this register.</td>
</tr>
<tr>
<td>7</td>
<td>interrupt</td>
<td>1'b0</td>
<td>When set the event occurs when an interrupt that is enabled for tracing occurs. The int_trace_trigger outputs from the two interrupt mappers (see Figure 10 on page 51) are ORed together to signal the event. (No record is made of which interrupt triggered the event). The event triggers once on the assertion of the OR of the two interrupt mapper outputs. Before the event can trigger again the outputs of both mappers must be deasserted.</td>
</tr>
<tr>
<td>8</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>debug_pin</td>
<td>1'b0</td>
<td>When set the event occurs if the external debug pin is pulled low. The event will trigger once on the falling edge of the DEBUG_L pin. DEBUG_L must return to its deasserted (high) state before the event will trigger again.</td>
</tr>
<tr>
<td>10</td>
<td>write</td>
<td>1'b0</td>
<td>When set the event only occurs if the transaction is a write.</td>
</tr>
<tr>
<td>11</td>
<td>read</td>
<td>1'b0</td>
<td>When set the event only occurs if the transaction is a read.</td>
</tr>
<tr>
<td>15:12</td>
<td>req_id[3:0]</td>
<td>4'b0</td>
<td>Requester ID used for matching.</td>
</tr>
<tr>
<td>19:16</td>
<td>resp_id[3:0]</td>
<td>4'b0</td>
<td>Responder ID used for matching.</td>
</tr>
<tr>
<td>23:20</td>
<td>data_id[3:0]</td>
<td>4'b0</td>
<td>Data ID used for matching.</td>
</tr>
<tr>
<td>31:24</td>
<td>count[7:0]</td>
<td>8'b0</td>
<td>If the count is not 0 then the event does not take place. Instead the counter is decremented. If the counter is 0 then the event occurs and the count is restored to its initial value. Writing this register sets the initial value of the counter, reading will return the current value.</td>
</tr>
</tbody>
</table>
TRIGGER SEQUENCES

A trigger sequence is a set of up to four conditions that must be satisfied in sequence and a trace function that will be executed when the sequence is complete. The conditions are combinations of trigger events. There are eight trigger sequencers. Sequencers 0-3 have events 0-3 as their sources, sequencers 4-7 have events 4-7 as their sources.

The following chart shows the combinations for sequencers 0-3 (for sequencers 4-7 add 4 to the event numbers):

<table>
<thead>
<tr>
<th></th>
<th>event0</th>
<th>event1</th>
<th>event2</th>
<th>event3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>event0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>event1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>event2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>event3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>event0</td>
<td>event1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>event1</td>
<td>event2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>event2</td>
<td>event3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>event0</td>
<td>event1</td>
<td>event2</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>event0</td>
<td>event1</td>
<td>event2</td>
<td>event3</td>
</tr>
<tr>
<td>9</td>
<td>event0 &amp; event1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>event0 &amp; event1 &amp; event2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>event0 &amp; event1 &amp; event2 &amp; event3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>(event0 &amp; event1)</td>
<td>event2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(event0 &amp; event1)</td>
<td>(event2 &amp; event3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>(event0 &amp; event1)</td>
<td>event2</td>
<td>event3</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>ignored</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The codes from these events are combined to make the sequence. For example the sequence 0123 requires that event0 happens, then event1, then event2, following this when event3 happens the function associated with the sequencer will be executed. Sequences of less than four events have the ignored code (F) in their lower bits, hence a sequence that will execute its function every time event 1 or event 2 triggers is expressed as 5FFF.

The functions that can be performed when a sequence is complete are:

1. **Start** collecting samples. This enables collection of samples into the trace buffer, unless collection has been frozen. The sample that triggers the start will be recorded if the sequence is marked with one of the filter bits described below.

2. **Stop** collecting samples. This suspends collection of samples into the trace buffer until a start function is performed. The sample that triggers the stop will be recorded if the sequence is marked with one of the filter bits.

3. **Freeze** the trace buffer. This stops collection of samples into the trace buffer until the buffer is reset by software or via the JTAG port. The sample that triggers the freeze will be recorded if the sequence is marked with one of the filter bits. An interrupt is raised when the buffer is frozen.

In addition there are three filter flags that are only used if collection is in progress:
**A Sample** causes an address/control bundle to be recorded in the trace buffer. These samples are packed three per line of the trace memory, so a maximum of 768 samples can be in the buffer. Note that if the final trigger of the sequence is a databus trigger then the A-phase signals captured may not be valid (the latches within the trace unit will only capture address information when an agent is driving the bus, this data is held in the latches until the next time the bus is driven).

**D Sample** causes an address/control bundle and the 32 bytes of data from the bus to be recorded in the trace buffer. A D sample always starts a new line in the trace buffer (if the previous line was only partially full from A samples the empty spaces are marked unused). So a maximum of 256 D samples can be taken. Since a D sample is a superset of an A sample, if both filter bits are set a D sample is taken. When a D sample is taken the A-phase signals may not be valid.

**Clear Use** causes the counter that determines the buffer usage to be cleared. Thus following the current sample 255 more entries will be filled before the buffer full condition is raised. This flag is normally used with the **Start** function when the start trigger may repeat and the freeze should only come when the buffer fills after the last start.

The final two flags allow the completion of a sequence to be signalled outside the trace logic (these occur regardless of the state of trace collection).

**Debug Pin** causes the SCD to pull the DEBUG_L pin low for 10 ZBbus clock cycles.

**Debug CPU** causes the SCD to assert the debug interrupt to both CPUs. The debug interrupt is cleared by a read of the **trace_cfg** register.

The outputs from all the sequencers are combined. If multiple functions are signalled, **freeze** has the highest priority, then **stop**, then **start**.

Note that when an address based trigger is used, the data bus may not be valid. So, forcing a D sample from such a trigger could waste trace buffer entries.

There are eight sequence control registers **trace_sequence_0** to **trace_sequence_7**, as shown in Table 47.

**Table 47: Trace Sequence Control Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>Event select 4</td>
<td>4'b0</td>
<td>These bits set the event selection that forms the fourth event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than four events.</td>
</tr>
<tr>
<td>7:4</td>
<td>Event select 3</td>
<td>4'b0</td>
<td>These bits set the event selection that forms the third event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than three events.</td>
</tr>
<tr>
<td>11:8</td>
<td>Event select 2</td>
<td>4'b0</td>
<td>These bits set the event selection that forms the second event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than two events.</td>
</tr>
<tr>
<td>15:12</td>
<td>Event select 1</td>
<td>4'b0</td>
<td>These bits set the event selection that forms the first event in the sequence. The bits should all be set (the ignore code) if the sequence should always trigger (used with the filter bits to capture all active bus cycles).</td>
</tr>
</tbody>
</table>
The special event selection sequence 16'hFFFF written in one of the trace sequence registers will cause all bus cycles to be selected. There are several possible actions depending on which of bits 18,19,23 and 24 are set:

- **Asample** (bit 18): An address/control sample is taken every time the address bus is used.
- **Dsample** (bit 19): An address/control and data sample is taken every time the data bus is used.
- **Alld_a** (bit 23): An address/control sample is taken every time the data bus is used.
- **Asample and Alld_a** (bits 18 and 23): An address/control sample is taken every time either address bus or data bus is used.
- **Asample and Dsample** (bits 18 and 19): An address/control sample is taken every time the bus is used, with data also being captured if the data bus is used.
- **All_a** (bit 24): An address/control sample is taken on every bus cycle.

Using these sequences a trace of the entire active bus can be taken. To avoid the count of cycles between samples overflowing, the forceCnt bit can be set in the trace_cfg register to force an address/control sample to be taken just before overflow. The sample collected when a section of the bus is not used is UNPREDICTABLE. The Avalid and Dvalid bits in the address/control bundle are used to indicate which bits in the bundle are valid. The All_a flag can be used in profiling to monitor the blocking and interrupt signals on every cycle.
USING THE TRACE BUFFER

The trace buffer starts off empty and tracing is disabled. The trigger events and sequencers are configured by software or over the JTAG debug interface. Then the system is run and the trace is recorded in the buffer. When the buffer freezes the trace can be read out, again either by software on the BCM1250 or BCM1125/H or via the JTAG link.

Control of the trace features is done through the trace_cfg register. The JTAG interface or debug software can write this register to trigger the start, stop or freeze functions, and also to reset and enable the trace buffer. The trace unit can also be set to execute the Freeze function and/or the Debug Pin function when the buffer fills (these functions are described in Section: “Trigger Sequences” on page 73).

A 64 bit register is provided for reading out the collected trace. The buffer should be frozen before samples are read out, if it is not then the results of the read are UNPREDICTABLE. The startread bit in the trace_cfg register is written with a one to start the read process. Then each read of the trace_read location (00_1002_0A08) will provide the next 64 bits from the trace memory. Software (either on the target, or external debug agent) must post-process the trace to extract the information. (Broadcom provides code to do this.) The buffer can only be read once after samples have been collected.

The trace control register is shown in Table 48.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reset</td>
<td>1'b0</td>
<td>When this bit is written with a 1, the trace buffer pointers will be reset ready for collection to begin. This bit always reads as zero.</td>
</tr>
<tr>
<td>1</td>
<td>startread</td>
<td>1'b0</td>
<td>When this bit is written with a 1, the trace buffer read pointer and latch are prepared for reading. This bit always reads as zero. Note that the buffer can only be read once after a collection, if this bit is written with a 1 for a second time the data read back will be UNPREDICTABLE.</td>
</tr>
<tr>
<td>2</td>
<td>start</td>
<td>1'b0</td>
<td>When this bit is written with a 1, the start function is triggered, and collection will begin if the buffer is not frozen. When read this bit will be set if collection is in progress.</td>
</tr>
<tr>
<td>3</td>
<td>stop</td>
<td>1'b0</td>
<td>When this bit is written with a 1, the stop function is triggered, and collection will be suspended until a start if the buffer is not frozen. When read the bit will be set if collection is stopped and not frozen.</td>
</tr>
<tr>
<td>4</td>
<td>freeze</td>
<td>1'b0</td>
<td>When this bit is written with a 1, the trace buffer is frozen. When read this bit will be set if the buffer is frozen. The buffer must be reset to allow further collection.</td>
</tr>
<tr>
<td>5</td>
<td>freezeFull</td>
<td>1'b0</td>
<td>If this bit is set the buffer will freeze when if becomes full. In order to prevent overflow the buffer will freeze as soon as there is any data in the final entry, if it freezes with space for two more Address/Control bundles (but no room for a Data sample) the trcFull bit will not be set.</td>
</tr>
<tr>
<td>6</td>
<td>debugFull</td>
<td>1'b0</td>
<td>If this bit is set the DEBUG_L pin will be asserted when the buffer becomes full. The DEBUG_L pin is deasserted again when the first read of the trace buffer begins or the trace is reset.</td>
</tr>
<tr>
<td>7</td>
<td>trcFull</td>
<td>1'b0</td>
<td>Read Only. This bit is set when the trace buffer is full (if this is set 256 entries can be read from the buffer)</td>
</tr>
<tr>
<td>8</td>
<td>forceCnt</td>
<td>1'b0</td>
<td>If this bit is set and collection is in progress then an address/control sample is forced to be collected before the COUNT field overflows regardless of the state of the filters. This ensures an exact count between events (at the expense of some additional entries used in the buffer.</td>
</tr>
<tr>
<td>9</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
All entries in the trace buffer contain an address/control bundle. The bundle includes the response phase signals that match with the address phase signals that are captured. The data phase signals are captured at the same time as the A phase signals, and have no relationship with the transaction that the A-phase and R-phase are part of. The address/control bundle is 128 bits organized as shown in Table 49.

### Table 49: Trace Buffer Address/Control Bundle

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>D_MOD</td>
<td>D-phase indication that the data is dirty (this has no relationship to the A and R phase signals).</td>
</tr>
<tr>
<td>7:4</td>
<td>D_RSP[3:0]</td>
<td>Data phase responder ID (this has no relationship to the A and R phase signals).</td>
</tr>
<tr>
<td>18</td>
<td>R_L2HIT</td>
<td>The L2 hit during the response phase of the A-phase transaction.</td>
</tr>
<tr>
<td>31</td>
<td>A_L2CA</td>
<td>L2 cacheability bit. 0: Do not allocate on miss. 1: Allocate on miss.</td>
</tr>
<tr>
<td>36:32</td>
<td>reserved</td>
<td>Always reads zeros.</td>
</tr>
<tr>
<td>37</td>
<td>int_trace_1</td>
<td>Records the int_trace_trigger_1 output from the CPU 1 interrupt mapper (see Figure 8)</td>
</tr>
<tr>
<td>38</td>
<td>int_trace_0</td>
<td>Records the int_trace_trigger_0 output from the CPU 0 interrupt mapper (see Figure 8)</td>
</tr>
<tr>
<td>39</td>
<td>Dvalid</td>
<td>Set if the data bus had a valid cycle when the sample was captured (indicates the D_ signals are valid)</td>
</tr>
<tr>
<td>40</td>
<td>Avalid</td>
<td>Set if the address bus had a valid cycle when the sample was captured (indicates the A_ and R_ signals are valid).</td>
</tr>
<tr>
<td>51:41</td>
<td>BLOCK[10:0]</td>
<td>ZBBus blocker signals.</td>
</tr>
</tbody>
</table>
The data trace consists of two 128 bit entries. The first contains bits [255:128] from the databus, the second has bits [127:0].
READING THE TRACE BUFFER

Each entry in the trace buffer is 384 bits, and falls into one of two possible formats. Format 1 is used when the entry contains one to three address/control samples, the first bundle (t0) will always come from an address trigger (the ATRIG bit will be 1) and will indicate format 1 by not having an associated data sample (the DTRIG bit will be 0). Note that if the sample was forced by an all-bus-cycle trigger, a data trigger or to avoid the intersample interval count overflowing, the Avalid bit may be clear indicating the A-phase information is invalid. The Dvalid bit indicates if there was databus activity when the sample was taken and the D-phase information is valid. Format 2 covers the address and data sample, in this case there may or may not have been an address trigger (ATRIG could be either 0 or 1) but there will always have been a data trigger (DTRIG will be 1). Again the Avalid and Dvalid bits in the control bundle indicate what activity was on the bus and thus which fields of the sample are valid. Software or the JTAG probe must perform six reads to collect the entire sample before decoding it. Table 50 describes the entry formats and the order in which they are read.

Table 50: Trace Entry Format and Read Order

<table>
<thead>
<tr>
<th>Bits</th>
<th>Read Order</th>
<th>Format 1 (Address/Control)</th>
<th>Format 2 (Address/Data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>383:320</td>
<td>1</td>
<td>Address/Control Bundle t2 [127:0]</td>
<td>Data [255:0].</td>
</tr>
<tr>
<td>319:256</td>
<td>2</td>
<td>This is the third sample recorded in this entry. It may not be valid.</td>
<td></td>
</tr>
<tr>
<td>255:192</td>
<td>3</td>
<td>Address/Control Bundle t1 [127:0]</td>
<td></td>
</tr>
<tr>
<td>191:128</td>
<td>4</td>
<td>This is the second sample recorded in this entry. It may not be valid.</td>
<td></td>
</tr>
<tr>
<td>127:64</td>
<td>5</td>
<td>Address/Control Bundle t0 [127:0]</td>
<td>Address/Control Bundle t0 [127:0]</td>
</tr>
<tr>
<td>63:0</td>
<td>6</td>
<td>This is the first sample recorded in this entry. It will always be valid for a format 1 sample. The DTRIG bit will be 0.</td>
<td>This is the address sample captured with the data. It may not be valid. The DTRIG bit will be 1.</td>
</tr>
</tbody>
</table>

When all the entries have been read from the buffer the ATRIG and DTRIG bits are forced to zeros in any subsequent reads. This condition can be used to terminate reading the buffer, or the trcFull and trcAddr fields of the trace_cfg register can be used to compute the number of entries that should be read. The startread bit in the trace_cfg register must be written with a 1 before the trace can be read out. The buffer can only be read out once, any additional attempts will return UNPREDICTABLE data.
An algorithm for decoding the bundle is:

```c
if (b[127]==1)
{
    if (b[126]==1)
    {
        /* Have valid address and valid data sample */
        Format = 2
    }
    else
    {
        Format = 1
        if (b[255] == 0)
        {
            /* Have 1 valid address sample */
        }
        else if (b[383] == 0)
        {
            /* Have 2 valid address samples */
        }
        else
        {
            /* Have valid address and no data sample */
        }
    }
    else if (b[383] == 0)
    {
        /* Have 2 valid address samples */
    }
    else
    {
        /* Have valid address and no data sample */
    }
}
```

The entries are returned with the newest one first and the earliest available read out last. The postprocessing software must take care to reverse the order to show causality.
MAGIC DECODER RING FOR USING THE TRACE BUFFER

This section includes some notes for using the Trace Buffer when debugging. Nothing in this section is part of the official specification of the device and things (in particular the TID information) could change from revision to revision.

The trace control entries include the transaction ID generated by the source of the transaction. In many cases additional information about the transaction can be deduced from the TID. For chips with system revision indicating PERIPH_REV3 Table 51 gives some details.

Table 51: Decode of some TIDs for system revision PERIPH_REV3

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent ID</td>
<td>See Table 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>000x</td>
<td>11nnnn</td>
<td>Instruction Fetch.</td>
<td></td>
</tr>
<tr>
<td>000x</td>
<td>000nnn</td>
<td>Read for read queue entry nnn.</td>
<td></td>
</tr>
<tr>
<td>000x</td>
<td>10nnnn</td>
<td>Write or Evict.</td>
<td></td>
</tr>
<tr>
<td>Bridge 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>00pnnn</td>
<td>Read from port p (1=PCI, 0=HyperTransport).</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>01pnnn</td>
<td>Write from port p (1=PCI, 0=HyperTransport).</td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>10pnnn</td>
<td>Read-modify-write (i.e. partial line write) from port p (1=PCI, 0=HyperTransport).</td>
<td></td>
</tr>
<tr>
<td>Bridge 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>00pnnn</td>
<td>Read from DMA engine p (1=Serial, 0=MAC).</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>01pnnn</td>
<td>Write from DMA engine p (1=Serial, 0=MAC).</td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>10pnnn</td>
<td>Read-modify-write (i.e. partial line write) from DMA engine p (1=Serial, 0=MAC).</td>
<td></td>
</tr>
<tr>
<td>SCD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>cc00aa</td>
<td>Read for buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>cc01aa</td>
<td>Write of buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>cc11aa</td>
<td>Read for read-modify-write with buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>cc1000</td>
<td>Descriptor fetch for channel cc. (cc bits always 00 prior to PERIPH_REV3)</td>
<td></td>
</tr>
<tr>
<td>0100</td>
<td>001001</td>
<td>JTAG Probe transaction. (010000 prior to PERIPH_REV3)</td>
<td></td>
</tr>
</tbody>
</table>
### Table 52: Encoded Byte Enables for CPU Transactions

<table>
<thead>
<tr>
<th>CPU Operation</th>
<th>DoubleWord A_DW</th>
<th>Byte A_BYT</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cacheable fill or evict or</td>
<td>1111</td>
<td>11111111</td>
<td>Cacheable operations always move a full block, so the byte enables can be ignored.</td>
</tr>
<tr>
<td>Instruction Fetch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncacheable Instruction Fetch</td>
<td>1100 0011</td>
<td>11111111</td>
<td>Uncacheable instruction fetches always fetch four instructions (either the first or second half of a cache block).</td>
</tr>
<tr>
<td>Uncacheable lb/sb</td>
<td>One bit set</td>
<td>One bit set</td>
<td>The byte can be identified from the encoded enables.</td>
</tr>
<tr>
<td>Uncacheable lh/sh</td>
<td>One bit set</td>
<td></td>
<td>The half word can be identified from the encoded enables.</td>
</tr>
<tr>
<td>Uncacheable lw/sw</td>
<td>One bit set</td>
<td>11110000 00001111</td>
<td>The word can be identified from the encoded enables.</td>
</tr>
<tr>
<td>Uncacheable ld/sd</td>
<td>One bit set</td>
<td>11111111</td>
<td>The double word can be identified from the A_DW code.</td>
</tr>
<tr>
<td>Uncacheable lwl/lwr/swl/swr</td>
<td>One bit set</td>
<td>1-4 bits set</td>
<td>The bytes can be identified from the encoded enables (but some combinations will match word/halfword/byte accesses).</td>
</tr>
<tr>
<td>Uncacheable ldl/ldr/sdl/sdr</td>
<td>One bit set</td>
<td>1-8 bits set</td>
<td>The bytes can be identified from the encoded enables (but some combinations will match double/word/halfword/byte accesses).</td>
</tr>
<tr>
<td>Uncached Accelerated Writes</td>
<td>1-4 bits set</td>
<td>1-8 bits set</td>
<td>If only doubleword stores are involved they can be identified. If word stores are used then information may be lost (reconstruction may be possible from the source code).</td>
</tr>
</tbody>
</table>

```
CONNECTIONS TO THE TRACE LOGIC

The trace logic connects to many other parts of the SCD. Figure 11 shows these connections.

Figure 11: Connections to Trace Logic
TRACE EXAMPLE 1: ALL CPU0 ACTIVITY

The code running on CPU0 is being monitored, for a particular section of code all address bus activity will be recorded to check the L2 cache behavior.

The code is patched to do a dummy access to an address at the start and end points of the code to trace, and the trace buffer is set up:

```
Addrmatch0 = start trigger
Addrmatch1 = stop trigger

trace0 = reqid=cpu0 & (read|write)
trace1 = reqid=cpu0 & Addrmatch0 & (read|write)
trace2 = reqid=cpu0 & Addrmatch1 & (read|write)
trace_cfg = freezeFull

seq0 = trace1, ignore, ignore, ignore -> Start
seq1 = trace0, ignore, ignore, ignore -> A-sample
seq2 = trace2, ignore, ignore, ignore -> Stop
```

When CPU0 accesses the dummy start address trace1 and seq0 will cause trace collection to be started. Then trace0 and seq1 will cause every address requested by CPU0 to be written to the buffer, until the dummy access that causes trace2 and seq2 to stop collection (the A-sample bit could be set on this sequence to record the dummy access if a separator was required).

When the trace buffer is full, the `trace_cfg` setting causes it to freeze, and raise an interrupt. CPU0 would service this interrupt and (with as little cache disturbance as possible) read out the trace into a bigger in-memory buffer. At the end of the ISR the trace buffer is reset and started (it must have been collecting for it to fill) and collection continues.

At the end of the test the buffer would be frozen by software and the final samples copied to the memory buffer. The memory buffer would then be post-processed for analysis.
TRACE EXAMPLE 2: NETWORK PACKET HEADERS

This example uses the trace buffer as a non-obtrusive record of the most recent network packet headers that were received from one of the MACs. It could be used, for example, when there is a bug that is thought to be caused by the processing of some rare sequence of network packets.

The trick is to notice that the DMA controller will always write the receive DMA status back into the DMA descriptor at the end of packet reception. This is used to prime collection of the first two transfers of the next packet.

```
Addr0 = range of receive DMA descriptors for the MAC
Addr1 = range of receive buffers

trace0 = reqid=iobr1 & Addrmatch0 & write (A phase of descr write)
trace1 = reqid=iobr1 & Addrmatch1 & write (A phase of write)
trace2 = rspid=iobr1 & dataid=iobr1 (write data from iobridge)

seq0 = trace0, ignore, ignore, ignore -> Start
seq1 = trace1, ignore, ignore, ignore -> A-sample
seq2 = trace1, trace2, ignore, ignore -> D-sample
seq3 = trace1, trace2, trace1, trace2 -> Stop
```

The first address trap is set up to detect the range of addresses that cover the receive DMA descriptor ring. trace0 detects a write to this space from the I/O bridge that connects the MACs. This write is done at the end of a packet reception to put the status flags into the descriptor (the requester ID is needed here because the CPU will also be writing the ring to update the descriptors). seq0 uses this to start the trace collection. When a packet arrives the DMA will write it to memory in 32 byte cache line chunks. The address associated with the write request triggers trace1, and seq1 filters it to be recorded (this may not be needed in this example, and will be fairly costly because it will unalign the trace buffer pointer from the entry boundary).

The trace2 is used to detect a write data transfer from the MAC to the buffer memory. It does this by detecting a transfer on the data bus where the data source (in the resp_id field) and the initiator of the transaction (in the data_id field) are the same. This is enough to detect that some write data is in flight from the I/O bridge. The seq2 sequence will first need trace1 to detect the write request to the buffer addresses before looking at trace2 to detect the data transfer part of the transaction. When the sequence is triggered the data sample is taken and written to the trace buffer.

Since the network packet header is in the first 64 bytes of the packet only the first two DMA transfers should be recorded. seq2 will trigger on both, causing data to be written to the trace buffer. seq3 has to see both transfers before it triggers and stops collection.

This process repeats, using the trace memory as a circular buffer, until the Freeze command is given (by software writing the trace_cfg register). At this point the trace buffer can be read out, showing the last 64 packet headers. (Two lines in the buffer are used up for each DMA transfer recorded -- the first has the seq1 address sample and two empty slots, and the second the seq2 address/data sample. Since the first two transfers of each packet are recorded there are 4 trace buffer lines per packet).
TRACE EXAMPLE 3: PCI DRIVER DEBUGGING

Device drivers for PCI devices quite often rely heavily on macros and library input/output routines. Particularly when porting code from a little endian (e.g. x86) driver to the BCM1250 or BCM1125/H running big endian, it can be hard for the programmer to track down exactly what is being transferred back and forth to the device on the PCI bus. This example uses the trace buffer to record every transfer between the CPU and the PCI bus, and any DMA activity initiated by the PCI device.

```
Addr0 = PCI range
trace0 = reqid=cpu0 & Addrmatch0 & (read|write)
trace1 = reqid=iobr0 & (read|write)
trace2 = rspid=cpu0 & dataid=cpu0
trace3 = rspid=iobr0 & dataid=cpu0
trace4 = dataid=iobr0
seq0 = trace0, ignore, ignore, ignore -> A-sample
seq1 = trace0, trace2 | trace3, ignore, ignore -> D-sample
seq2 = trace1, ignore, ignore, ignore -> A-sample
seq4 = trace4, ignore, ignore, ignore -> D-sample
```

The driver code has writes to the trace_cfg register added to start and stop samples, and to freeze and read out the results.

Any access from the CPU to the PCI range is detected by trace0. If the transaction is a write then trace2 will detect the data transfer; the resp_id indicates that the data source is the CPU and the data_id indicates that the transaction was initiated by the CPU. (This detects any write data from the CPU, it is used in sequence with trace0 to detect this particular write.) If the transaction is a read then trace3 detects the data returning from the I/O bridge in response to the CPU request (the I/O space read is uncacheable and the CPU only has one outstanding, so in sequence with trace0 this always catches the correct data).

Master mode accesses from the PCI are detected by trace1 for addresses and trace4 for the data. These just use the transaction id to identify the I/O bridge as the initiator of the operation. Note that since trace4 is used the detection sequence had to move to seq4 on the second sequencer block. This will detect DMAs from devices on the HyperTransport fabric too, so this is not a perfect trace. However, since it is being used to debug during the device driver bring-up, it should be relatively easy to prevent other DMA devices from being active.
This Page is left blank for notes
Section 5: L2 Cache

INTRODUCTION

The on-chip second level cache is shared by the processors and any I/O DMA masters. On the BCM1250 the L2 cache is 512 KBytes, on the BCM1125/H it is 256 KBytes. This section describes the normal operation of the L2 cache, a mode where banks of it can be used as an on-chip SRAM and the management interface to it. After reset the management interface must be used to initialize the L2 cache before it can be used, if a cacheable reference is made before the L2 is initialized multiple cache lines may be selected causing an internal bus clash.

NORMAL OPERATION

The L2 cache is 256 or 512 KBytes, organized into 32 byte cache lines four way set associative. All accesses to the L2 are in full cache blocks. It is a non-inclusive/non-exclusive cache, thus there are no restrictions on which cache blocks can be in the L2. A random replacement policy is used when a victim line must be found.

The L2 runs internally at the CPU core speed and is fully pipelined. This allows one access per ZB bus cycle, including all the required housekeeping needed to evict dirty lines.

The L2 cache tags and the data blocks are ECC protected. Single bit errors in either are corrected while an access is in progress (the L2 will internally take time to perform the full recovery, but the system continues running). The ECC cleanup is started by a cache hit on a line with a single bit error. Cleanup will be pre-empted by a write to another way at the same index, so occasionally a single bit error will be flagged twice. Tags with uncorrectable errors result in UNPREDICTABLE data being returned to the requestor with an uncorrectable tag error signalled. Data with uncorrectable errors will be returned to the requestor with an uncorrectable data error signalled. In either error case software recovery is required. The bus watcher in the SCD will log the data associated with the error and raise an interrupt. The L2 will record the tag associated with the error in the l2_ecc_tag register (this can be read to give the tag from the most recent ECC error). The management interface to the L2 can be used to invalidate the line and clear the ECC bits. Any data that is received by the L2 cache marked with an error is written into the cache with an uncorrectable ECC error, so subsequent reads will also get an error.

The L2 cache is physically one of the ZB bus agents, but architecturally it sits between the system bus and the main memory, and there are dedicated signals between the L2 and memory controller that coordinate them. Every bus access is accompanied with the L1 cache and coherence attributes for the access. In the CPU these are set in the TLB and indicate cacheable-coherent, cacheable-non-coherent, uncacheable, and uncacheable-accelerated (which indicates writes may have been merged). For DMA accesses the L1 attributes are provided from the DMA control registers and should match the attributes the CPUs use for that area of memory. The L2 is coherent with memory for all L1 cacheable accesses (non-cacheable accesses to main memory will bypass the L2 cache, they should never be done to areas of memory that have previously been cacheable). The L2 cache is checked by all L1 cacheable memory accesses put on ZB bus. As described in Section: “Coherence” on page 12, it acts in conjunction with the memory controller to provide any blocks that are not owned exclusively by another bus agent.
Every bus request is accompanied by an L2 cacheability flag, if this is set and the block is not in the L2 cache then it will be allocated. On a write miss the L2 will allocate space for the block and accept the data when it is transmitted. On a read miss the memory controller will supply the data to the requesting agent, the L2 will allocate space for the block at the time the request is made and will take a copy of the data as it passes on the bus. The L2 cacheability flag only affects the behavior on an L2 miss. Regardless of the L2 cacheability flag, if a request that is L1 cacheable hits in the L2 then it will supply the data for a read and receive the data on a write.

The L2 cache behaves the same whatever the source of the request, so DMA masters will read data from the L2 if it is there. More interestingly, DMA writes can be done with the L2 cacheability bit (see A_L2CA in Section: “Address Phase” on page 22) set to cause the data to be written to the L2 cache (and on to memory only if it gets evicted). This can be used to reduce the access latency for any data the CPU will need to manipulate. However, this operation can potentially pollute the L2 cache so it must be used with care. The internal DMA controllers have a control bit to enable this behavior, its use by external DMA masters is described in the host bridge section for the buses.

To allow programs further control of the L2 cache, the CPU provides a mechanism to prevent the L2 cacheability flag from being set on accesses that are cacheable in the L1 cache. This is of particular use for reducing L2 pollution with data that is highly likely to be referenced only by one of the CPUs and used for a short time. There are two ways this can be done. Pages can be marked in the TLB with one of the cacheable coherent no L2 attributes (codes 0 and 1), which will cause all blocks in the page to be accessed with the L2-allocate flag clear. Alternatively, the pages can be marked with the usual cacheable coherent attributes (codes 4 and 5) and the PREFetch instruction with a streaming hint can be used to access individual lines around the L2 cache. In the second case since the block is marked for streaming when it is put in to the L1 it is flagged as the first of the 4 lines at that index to be evicted (it will be evicted first regardless of the LRU information).

When the L2 cache is accessed address bits [16:5] ([15:5] on a part with or configured for a 256KB cache and [14:5] for a part with or configured for a 128KB cache) are used to form an index into the cache array, and the four ways are examined at that index. When designing for best performance the potential for bad aliasing must be taken into consideration. For example in an embedded system using 64K pages and a simple memory allocator, it is possible that memory is always allocated in 64K chunks starting at a page boundary. Thus (for a 512KB cache) data in the first cache block of allocated objects will be found at one of 8 locations in the L2 cache (the index will either be zero or have just bit [16] set, and there are 4 ways at each possible index). A programmer unaware of the allocation policy could see a large amount of unexpected cache thrashing, and thus get significantly lower performance than expected. A similar (though less dramatic) effect can be seen if network packet buffers are allocated as 1KB long aligned at a 1KB boundary and the DMA engine is set to cache 64 byte headers in the L2. The packet headers will all have address bits [9:6] as zero, so will be limited to use 1/16th of the cache. If 64 bytes of padding is put between the buffers (or the buffer size is increased to 1088) then all of the cache will be used.

Statistics for the L2 cache are gathered by the debug/performance monitor unit. These are discussed in the Section: “System Performance Counters” on page 61.
USING THE L2 CACHE AS MEMORY

Some applications require more control over the on-chip memory than is provided by a cache, and would work best with a local RAM. A fast RAM can be provided by removing ways from the L2 cache. Each way that is removed from the cache provides a 128 KByte memory that can be accessed at the same speed as the L2 (or 64KB memory on a 256KB cache).

Ways are removed from cache use (they are removed from the random replacement algorithm) by clearing the bit corresponding to the way number in the l2_wayan[3:0] register. This is a control register hidden in the L2 cache that can be updated by doing a write to the l2_way_enable space as shown in Figure 12. Whenever any write is made to this address range the data is ignored, and bits [11:8] of the address get written to bits [3:0] of the register that enables the way. This register resets to have all bits set, so all ways of the cache are enabled. To remove way 2 from the cache and enable all the other ways the access address would be to the l2_way_enable space with address bit 10 clear and the others set, giving an address of 00_1004_1B00.

Software should not clear all four bits in the l2_wayan register, the resulting behavior of the cache is UNPREDICTABLE if there is any L2 cacheable activity in the system (the system will not hang, but data corruption will occur in one way of the cache). On parts with system revision indicating PERIPH_REV3 and later the register can be read back from the l2_misc_value register.

![Figure 12: Level 2 Cache Way Disable Access Address](image)

The memory removed from the cache must always be accessed as cacheable space. Cacheable transfers are always done as full blocks and the L2 cache always operates on full cache lines. Writes smaller than a cache line (which will be the case for most uncacheable stores) will result in the whole line being written with UNPREDICTABLE data. The ECC logic remains active for the memory. On writes the correct ECC is generated and written. On reads the ECC is checked, correctable errors are fixed and uncorrectable errors are flagged as data errors.

Removing ways from the L2 cache reduces both its size and associativity. This is likely to impact the performance of the processors. A system design trade-off must be made between the control given by having the local fast memory, and the degradation of cache performance. In general purpose systems using all the memory as cache is most likely to be the best solution, in well-characterized embedded applications removal of one way of the L2 cache can improve the predictability of critical loops.

There are two main methods for using ways removed from the L2 cache. The simplest is just to use the memory as a block of on-chip RAM with a fast access time (that initially contains random data). The second method is to load data from main memory into the L2 cache and then lock it in place by preventing it being replaced.
STANDARD RAM

Ways removed from the L2 cache can be used as a standard RAM block by accessing the memory using the management access address range with the way bits set appropriately and the special control bits set to zero (the management mode is described in Section: “Cache Management Access” on page 94). This provides a bank of memory that the system will need to initialize and that has no corresponding main memory locations. If consecutive ways are removed from the L2 cache the banks provided will be contiguous, forming a larger memory. Table 53 shows the address range that should be used for each RAM bank. Note that special management accesses can be invoked by using addresses other than those listed, potentially corrupting the data.

Table 53: Addresses for Memory Banks

<table>
<thead>
<tr>
<th>Way</th>
<th>Base</th>
<th>End (for 512KB cache)</th>
<th>End (for 256KB cache)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00_D018_0000</td>
<td>00_D019_FFFF</td>
<td>00_D018_FFFF</td>
</tr>
<tr>
<td>1</td>
<td>00_D01A_0000</td>
<td>00_D01B_FFFF</td>
<td>00_D01A_FFFF</td>
</tr>
<tr>
<td>2</td>
<td>00_D01C_0000</td>
<td>00_D01D_FFFF</td>
<td>00_D01C_FFFF</td>
</tr>
<tr>
<td>3</td>
<td>00_D01E_0000</td>
<td>00_D01F_FFFF</td>
<td>00_D01E_FFFF</td>
</tr>
</tbody>
</table>

Figure 13 on page 95 gives details on how these addresses were derived. The special ECC diagnostic mode bits are clear, the valid and dirty bits are set and the way bits match the desired way. Bits [26:23] are ignored by the access (although they get written to the cache tag), so there are 15 aliases to the addresses in Table 53 that will work equally well.

MEMORY LOCKED IN THE L2 CACHE

The second method for using the L2 cache as a controlled memory is to lock data into it. The L2 is initialized with the data from main memory before the way is removed from the replacement algorithm, then any accesses to those main memory addresses will always access the L2 cache. This scheme is more complicated to set up and care must be taken that the addresses to be locked do not collide in the cache (since only one way is being locked only one address can be used per cache index), however it may be useful for code or data that cannot be easily relocated.

One method to initialize the cache is to ensure there are no copies of the code in L1 or L2 caches (since the initialization would most likely run as the system boots, it may be possible to arrange this by design). Three ways of the L2 cache are then disabled and cacheable reads are used to fetch the code or data into the fourth way. Once the data is in place the three ways are enabled and the fourth way disabled, locking the data in the cache. As with any L2 cache manipulation, care must be taken to ensure that there is no other activity in the system during this process.
COMMENTS ON USING THE L2 AS MEMORY

This section discusses some of the system design trade-offs associated with using the L2 cache as memory, and some of the associated questions.

In normal operation the L2 cache is 512KB or 256KB and 4 way associative. There is no on chip memory. In general this is the best combination, since the goal of a cache is to automatically keep recently used information in fast on-chip memory and spill older data out to the main memory. However this is statistical, there are no guarantees of behavior because sometimes a full memory cycle will be needed. Turning a way of the cache into a memory bank will give some memory with guaranteed (fast) access time (subject only to bus contention), but the performance of the cache is degraded both because it is smaller and because the associativity has reduced. There is no easy way to tell what the performance impact will be for an embedded system without modeling the details (there are many studies of general computing systems but they are less useful because most uses of the part will have a much higher amount of I/O buffer and descriptor manipulation than the studies typically include). The L2 cache is non-inclusive and non-exclusive (i.e. what is in the L2 and what is in the L1 need not be related) which mitigates the effects some amount. But with a smaller L2 there is very likely to be reduced performance. On the other hand on a BCM1250 a 256KB 2 way associative cache (i.e. with half of the cache set aside as on chip memory) may be sufficient for the control processor if the task is split such that the other CPU is running a real-time-loop. The real time processor can reduce its impact on the L2 cache by keeping its main code loop in its L1 Instruction cache and most of its data either in its L1 Data cache or fetched around the L2 from a large table in memory (using PREFetch for streaming or by mapping the page with the no-L2 allocate cache attribute).

When the L2 cache is used as memory there is a restriction that writes must always be of full 32 byte cache lines. In practice this is not a problem. The space must be mapped cacheable in the CPUs (which is reasonable for data that needs a fast access time), this is not a burden for I/O devices doing DMA since the coherence protocol will ensure the most up to date copy of the data is returned for a DMA read and the latency is about the same regardless of whether the data comes from the L1 or L2 cache. In cacheable space the CPU will only do full 32 byte block reads (to fill the L1 cache line) and full 32 byte block writes (when a line is evicted from the L1 cache), so the full line write requirement is met. The I/O bridges (the point at which any DMA traffic hits the coherent domain) recognize the L2-as-memory addresses as coherent memory space so on a read will use the coherence protocol to get the latest data, and if a partial block write (i.e. <32 bytes or not cache aligned) is made the I/O bridge will do a read (exclusive)-modify-write of the full line to coherently merge in the new data. Therefore the full line write requirement is met in this case too. The Data Mover has the same restriction as the CPU: the space must be marked cacheable to ensure full line writes.

Another option that should be considered rather than using the L2 cache as memory is to make use of the L2 as a cache in which I/O devices can cache data. The ZBbus L2CA flag causes an L2 cache allocate on a miss. This flag accompanies every bus command (see Section: “Address Phase” on page 22). The on-chip DMA engines can be configured to assert the flag to cache descriptors and/or packet headers. HyperTransport devices can assert the flag by setting the isochronous bit in a HyperTransport command or (with revision 3 of the interface system revision PERIPH_REV3) hitting in an address range defined by the IsocBAR and IsocMask. PCI devices can assert it through the mapping in the BAR0 translation registers. Using this rather than partitioning the cache will allow the data to behave as a cache and store recently accessed data. This will work well in situations where the CPUs are expected to be manipulating packet headers and descriptors shortly before (on transmit) or after (on receive) the I/O device. The drawback of this method is that occasionally the data cached by the I/O device will cause some CPU cached data to be evicted and force the CPU to re-fetch it from memory (this is always true of the other CPU as well), and in overload, newly arrived packets will displace older ones in the cache, so as the processors try to catchup, they will have to re-fetch from memory (this can be easily prevented by detecting overload and flipping the bit in the DMA controllers to disable the L2CA flag).
REDUCED CACHE SIZE

The BCM1250 can be used to develop code for lower end members of the SiByte processor family. To allow performance tuning, the L2 cache size can be reduced to match the other parts, otherwise any performance numbers may be skewed by the large L2.

A 4 bit register `l2_cache_disable` is used for this. It is similar to the `l2_way_enable` register in that it is hidden and written from address bits. The register is accessed by writing dummy data to address 00_10042r00, bits [11:8] (denoted `r`) are the control. On parts with system revision indicating PERIPH_REV3 and later the register can be read back from the `l2_misc_value` register.

- To restore the full 512K L2 cache `r` should be 4'h0.
- For a 256K 4 way associative L2 cache `r` should be 4'h1 or 4'h2.
- For a 128K 4 way associative L2 cache `r` should be 4'h5 or 4'h6 or 4'h9 or 4'hA.
- Other values of `r` will give UNDEFINED results.

CACHE MANAGEMENT ACCESS

In addition to regular accesses to the L2 cache there is a management mode. This is used to invalidate the cache when the system is reset, and is used during recovery from uncorrectable ECC errors. It can also be used to force dirty lines to be flushed from the L2 cache, however this should never be necessary in normal operation. There are two memory mapped registers associated with the management mode.

Performing cache operations using the management mode requires software to maintain ordering and control of traffic to the L2 cache. Accesses to the L2 cache by the other CPU or DMA engines could cause the cache operation to fail. In normal systems management accesses are only required as part of system startup or to recover from uncorrectable ECC errors, so it should be possible to have this exclusive access.

A portion of the address space (00_D000_0000 - 00_D7FF_FFFF) is allocated to L2 cache management operations. Accesses in this range (called “management reads” or “management writes” in this section) will be ignored by the memory controller and will always hit in the cache even when the access made is uncacheable. The usual lower address bits select the cache index, two address bits select the way of the cache. Figure 13 on page 95 and Table 54 on page 95 show how a cache management address is created.
Figure 13: Cache Management Address

Table 54: Management Address

<table>
<thead>
<tr>
<th>Bits</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>Offset (always 0, or treated as such by L2).</td>
</tr>
<tr>
<td>16:5</td>
<td>Index. The cache is divided into sets, each containing four 32 byte lines. These bits select which set is indexed. The BCM1250 has all 4096 sets, the BCM1125/H has only 2048 so bit 16 is not part of the index. Bits 16:15 are also written to the tag (they are used as tag bits in 256 KB and 128 KB cache configurations).</td>
</tr>
<tr>
<td>18:17</td>
<td>Way. These bits select which of the four lines in a set are accessed. During management writes these address bits are written to the tag.</td>
</tr>
<tr>
<td>19</td>
<td>Dirty. The dirty bit indicates that the line held in the L2 cache contains more recent data than the block in main memory. If this bit is set the block must be written back to memory before the line can be used for another block. During management writes this address bit is written to the dirty bit and the tag.</td>
</tr>
<tr>
<td>20</td>
<td>Valid. The valid bit indicates that the line contains valid data. During management writes this address bit is written to the valid bit and tag.</td>
</tr>
<tr>
<td>22:21</td>
<td>ECC_diag[1:0]. If this field is nonzero during a management mode access an ECC diagnostic access is performed. During management writes these address bits are written to the tag.</td>
</tr>
<tr>
<td>26:23</td>
<td>During management writes these address bits are written to the tag.</td>
</tr>
<tr>
<td>39:27</td>
<td>$b0000000011010$ ($00_D000_0000$ - $00_D7FF_FFFF$) this address range selects management access. During management writes these address bits are written to the tag.</td>
</tr>
</tbody>
</table>
Management accesses may be done with cacheable or non-cacheable operations. However, the L2 cache will always operate on a full 32 byte cache line. On reads there is not a problem since the destination will filter the data (data is always carried on its natural byte lane on the ZBbus), but writes that are smaller than a cache block will cause random values to be written to the other bytes in the line. Uncacheable writes will normally be smaller than a full cache block (the exception being uncached-accelerated writes that have merged to a full block in the CPU write buffer), so they cannot be used to set particular values in the cache. However, uncached writes may be used for initializing the cache (where the data is unimportant, since the goal is to clear the valid bit for all the lines in the cache) or clearing data ECC errors by overwriting the bad data. This works because the other bytes on the bus during an uncached write will always be driven with valid (although UNPREDICTABLE) data.

The management accesses will still signal hit or miss to the performance counters based on the comparison between the management access address and the cache tags.

Following any management access to the cache that is to an enabled way, the replacement algorithm state is set such that the way of the cache that was accessed is used for the victim (if one is needed) on the next cache access (note this is the next access not the next miss). Therefore a line can be flushed from the cache by doing a management read of the line to be flushed followed by a regular cacheable read of a block that uses the same index and is not currently in the L2. The management access selects the way as the victim for the subsequent fill. To avoid the line being flushed ending up in the L1 cache, the management access is likely to be uncacheable, however the regular access must be cacheable. The ordering must be maintained by software (an ALU operation can be performed on the data returned by the uncached management access to ensure it has completed before the cacheable load is issued). If multiple lines are to be flushed care should be taken that the dummy cacheable fetch has completed before the next uncacheable management access (i.e. an ALU operation should be performed on the dummy data or a SYNC used).

Management mode accesses flow through the same paths as normal accesses. If a write is done to a line followed by a read of the same address, the data used to satisfy the read may be bypassed from the cache input queue. This increases the performance in normal operation, but is a problem when the cache RAM is being tested. There must be more than six ZBbus cycles between the write data (i.e. the ZBbus D-phase for the write transaction) and the read request (i.e. the ZBbus A-phase for the read transaction) arriving at the cache to ensure the read is from the RAM array. This can be arranged by doing two reads from the same location. Alternatively, a sequence of writes to other addresses can be done before the read.
STANDARD MANAGEMENT MODE ACCESSES (BOTH ECC_DIAG ADDRESS BITS ZERO)

Standard management mode accesses are any made to the management address range with the ECC_diag address bits clear.

Management reads will return the line from the addressed index and way. The normal ECC checking and correction is performed. Data with correctable ECC errors will be returned with the errors corrected, flagged as ECC corrected. Data with uncorrectable errors will be flagged with the uncorrectable data ECC error code. Similarly, if the tag has a correctable ECC error it will be corrected and the data returned with the tag ECC corrected flag, and if the tag has an uncorrectable ECC error data is returned with the uncorrectable tag error code.

As the management read is done the tag associated with the entry is transferred to the l2_read_tag register. This register will contain the tag after ECC correction and the raw tag ECC bits. The register can be read by the CPU with an uncacheable read. If cacheable reads are used for the management read that gets the data care is required since the CPU implements run-under-miss and there is no ordering guaranteed between cacheable and uncacheable reads. The ordering can be forced by the CPU performing a SYNC or an ALU operation on the data read from the cache management access (e.g. add zero) before reading the tag register. There is no ordering problem if the management access is done using an uncacheable read because the CPU will maintain the order between the management access and the l2_read_tag register, but uncached access is less efficient if all the data in the line needs to be examined.

Management writes will write the data to the addressed index and way. The data is written with correct ECC bits. The previous data in the addressed cache line will be lost, evicts are never done. If the previous data had an ECC error this will not be reported. A new cache tag (with correct ECC) is written, tagging the line with the address used in the write (the bits put in the tag are shown in Figure 13 on page 95). The valid and dirty bits of the tag are copied from bits 20 and 19 of the address.

The L2 cache can be invalidated by doing standard management writes to each index and way of the cache using an address that causes the tag to be written with the valid bit clear. In addition to invalidating the cache this will cause correct ECC to be written to all the cache blocks and tags, so there will not be spurious ECC errors on the first use of a cache line. This must be done during system startup before the cache can be used.

ECC error recovery code should not directly read the erroneous data from the cache. If the data contains an uncorrectable error it will still be signalled as such, so the CPU would see it as a cache error. However the bus watcher in the SCD logs the data whenever an uncorrectable error is reported, and the L2 controller logs the address and tag (with the error) in the l2_ecc_tag register.
ECC Diagnostic Management Accesses (ECC_diag bits nonzero)

Management accesses with the ECC_diag field nonzero modify the behavior of the cache to allow access to the raw memory bits for testing.

Management reads with the ECC_diag [0] bit set will return the raw data from the addressed index and way. The data ECC checking and correction is disabled, the raw data from the cache memory array will be returned and will always be marked valid.

As the management read is done the tag associated with the entry is transferred to the l2_read_tag register. Again the ECC correction is modified. The register will contain the raw tag data and the raw tag ECC bits. The register can be read by the CPU as in the standard management access, with the same care required about read ordering.

The ECC_diag address field modifies the behavior of management writes to allow data and tag ECC errors (of any number of bits) to be written into the cache, to test both the hardware and software ECC detection and recovery mechanisms.

Management writes with the ECC_diag address field nonzero are used to check the ECC bits. The combinations allow direct writes from the data into the cache tag, and can write using ECC information from a previous write. Table 55 summarizes the operations.

<table>
<thead>
<tr>
<th>ecc_diag[1:0]</th>
<th>Read</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>2'b00</td>
<td>Normal Normal</td>
<td>Write data to the cache, write the tag from the data bits. Generate tag ECC bits. The tag address bits are written from ZBbus bits [23:0]. The tag valid bit is written from ZBbus bit [24]. The tag dirty bit is written from ZBbus bit [25]. Note that these are from the double-word at offset h18.</td>
</tr>
<tr>
<td>2'b01</td>
<td>Raw Access ECC disabled</td>
<td>Write current data with the ECC bits from the previous cache write.</td>
</tr>
<tr>
<td>2'b10</td>
<td>Reserved</td>
<td>Write data to the cache, write the tag from the data bits (as in code 2'b01). Use tag ECC bits from the previous write.</td>
</tr>
<tr>
<td>2'b11</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

Table 55: ECC Diagnostic Operations

Data ECC errors can be stimulated with two writes. The first write (which is a normal write or standard management write) sets the index and way for the test along with the data ECC bits to be used for the test. The second write (an ECC diagnostic write of type 2'b01) replaces only the actual data. If the data of the second write differs by a single bit from the data in the first write a subsequent standard read will get the data from the first write flagged with a corrected ECC error. If the second write had two bits difference the subsequent read will get an uncorrectable ECC error.

Tag ECC errors can also be stimulated with two writes. The first write writes the tag bits with correct ECC to be used for the test (using ECC diagnostic write of type 2'b10). The second write changes the tag bits but uses the ECC bits from the first write (using an ECC diagnostic write of type 2'b11). If the tag generated by the first write differs by a single bit from the tag in the second, a subsequent standard management read will be flagged with a corrected tag ECC error and the l2_read_tag register will hold the reconstructed tag for the second write. If the tag generated by the first write had two bits difference the subsequent read will get an uncorrectable tag ECC error.
CACHE CONFIGURATION REGISTER

There is a 4 bit register l2_misc_config that can be used to modify the L2 behavior. It is similar to the l2_way_enable register in that it is hidden and written from address bits. In most systems this register should not be modified. The register is accessed by writing dummy data to address 00_10043r00. Bits 8 and 9 are reserved and should be written as zero. If bit 10 is set the L2 cache will use the low priority memory controller blocker rather than the high priority one. If bit 11 is set the automatic ECC cleanup is disabled. On parts with system revision indicating PERIPH_REV3 and later the register can be read back from the l2_misc_value register.

EXAMPLE STARTUP CODE TO CLEAR THE L2 CACHE

The following code will work on the BCM1250 and BCM1125/H using 4096 sets. It could be optimized for the BCM1125/H to only clear 2048 sets.

```assembly
# Save the old status register, and set the KX bit.
# This allows 64 bit addressing through xkphys.
   mfc0  t2,C0_SR
   or   t1,t2,M_SR_KX
   mtc0  t1,C0_SR

# Start the index at the base of the cache management
# area, but leave the address bit for "Valid" zero.
# Note that the management tags are at 00_D000_0000,
# which cannot be accessed through kseg1,
# so generate a 64-bit xkphys uncacheable address to get to it.
# The PHYS_TO_XKPHYS mapping adds 9000_0000_0000_0000 to the address
   dli   t0,PHYS_TO_XKPHYS(A_L2C_MGMT_TAG_BASE)

# Loop through each entry (4096) and each way (4)
   li    t1,L2C_ENTRIES_PER_WAY*L2C_NUM_WAYS

# Write a zero to the cache management register at each
# address. Align and do four per loop to run faster since the code
# is running in uncached space.
   .align 4
   1:
      sd   zero,0(t0)
      sd   zero,CACHE_LINE_SIZE(t0)
      sd   zero,2*CACHES_LINE_SIZE(t0)
      sd   zero,3*CACHES_LINE_SIZE(t0)
      daddu t0,(4*CACHES_LINE_SIZE) # size of a cache line
      subu t1,4
      bne   t1,0,1b

# # Restore old KX bit setting
   mtc0  t2,C0_SR

   j     ra  # return to caller
```
REGISTERS

Table 56: Level 2 Cache Tag Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>0</td>
</tr>
<tr>
<td>14:5</td>
<td>Index</td>
</tr>
<tr>
<td>15</td>
<td>Tag bit 39 (holds index bit 15 in a 256KB or 512KB cache and is a tag bit in a 128KB cache)</td>
</tr>
<tr>
<td>16</td>
<td>Tag bit 40 (holds index bit 16 in a 512KB cache and is a tag bit in a 256KB or 128KB cache)</td>
</tr>
<tr>
<td>38:17</td>
<td>Tag bits</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>45:40</td>
<td>Tag ECC (raw).</td>
</tr>
<tr>
<td>47:46</td>
<td>Way</td>
</tr>
<tr>
<td>48</td>
<td>Dirty</td>
</tr>
<tr>
<td>49</td>
<td>Valid</td>
</tr>
<tr>
<td>59:50</td>
<td>Data ECC (raw)</td>
</tr>
<tr>
<td>63:60</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

The L2 tag registers are UNPREDICTABLE until the first event that causes them to be written. The performance of reads to the registers will be low, while a read is in progress hardware will delay accesses to the L2 Cache registers and any registers in the SCD.

Table 57: Level 2 Cache Settings Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>Cache quadrant information [t,b,r,l] (Broadcom Use Only)</td>
</tr>
<tr>
<td>7:4</td>
<td>I2_cache_disable register value</td>
</tr>
<tr>
<td>8</td>
<td>Reads back enable for use of low priority memory blocker from value written as bit [10] in the I2_misc_config register address</td>
</tr>
<tr>
<td>9</td>
<td>Reads back disable of ECC cleanup from value written as bit [11] in the I2_misc_config register address</td>
</tr>
<tr>
<td>13:12</td>
<td>Reads back I2_way_enable register</td>
</tr>
<tr>
<td>63:14</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Section 6: DRAM

INTRODUCTION

The part incorporates a DDR SDRAM controller that works closely with the level 2 cache to provide a high performance memory system. The BCM1250 memory controller includes two channels each providing a 64 bit data path with 8 bit ECC. The BCM1125/H parts have a memory controller with a single channel providing a 64 bit data path with 8 bit ECC. Each channel can directly support up to two standard two physical bank JEDEC 184 pin DDR DIMMs (for a total of four DIMMs on a BCM1250) running at 133MHz clock (266MHz data rate, sometimes called PC2100 DIMMs based on PC266A parts) and allows for performance to increase as the DIMMs support higher data rates. In a more controlled electrical environment, with the DRAMs mounted on the main board and traces tightly controlled the memory controller can run up to 200MHz clock giving a 400MHz data rate. The peak memory bandwidth for a single channel using standard (133 MHz clock) DIMMs is 16 Gbit/s and increases up to 50 Gbit/s for a high speed (200 MHz clock) design using both channels.

The memory controller supports three types of memory all of which use SSTL_2 signalling levels and the conventional multiplexed address bus.

- **Standard DDR SDRAM** which may be mounted on board or on DIMMs.
- **Standard DDR SGRAM** which can be on DIMMs but are designed to be mounted on the main board and run with faster cycle times than standard DDR. The memory controller does not make use of any of the graphics features.
- **Fujitsu/Toshiba/Samsung DDR FCRAM** (Fast Cycle RAM) which is designed to be mounted on the main board. These parts have smaller rows and a much simplified command set allowing them to have both faster cycle times and lower access latency than standard DDR.

Each channel can support up to 1 GByte of memory using 256 Mbit technology parts. As larger DRAMs become available this will increase to 2 GByte with 512 Mbit parts and 4 GByte with 1 Gbit parts (note that gigabit parts that use 14 row address bits are only supported on the BCM1125/H and versions of the BCM1250 supporting PERIPH_REV3 or later). A special large memory mode allows these sizes to be doubled, but requires the use of an external decoder and may be lower performance.

A COMMENT ON THE TERM BANK

The term "Bank" is unfortunately used in a couple of different ways when discussing DDR SDRAM memory systems.

The first use of bank is to describe the organization within a memory device, most devices have four internal banks and thus need two bank address bits (BA[1:0]). The number of banks on a device sets the number of pages that it can have open at a given time.

The second use of bank is to describe the physical groups of devices that are enabled by a single chip select. Standard DDR 184 pin DIMMs can have two physical banks and thus need two chip selects. Each physical bank corresponds to one load on the data and DQS signal lines. The number of loads on the address lines for a physical bank depends on the organization of the memory devices. For the 64 bit wide data path used by the memory channel using x8 devices gives 8 address loads per physical bank (9 if ECC is used). Using x16 devices gives 4 (or 5) loads and using x32 devices gives 2 (or 3) loads.

In this chapter the unqualified term bank will only be used to refer to internal banks. Physical banks will always be described as such.
MEMORY CONTROLLER ARCHITECTURE

The memory controller consists of a ZBbus interface and one or two memory channel interfaces. The channels on the BCM1250 are numbered 0 and 1, the BCM1125/H only has channel 1. The block diagram is shown in Figure 14 on page 105.

The ZBbus interface consists of:

- The request queue (RQQ).
- A data buffer (DBF) for read and write.
- The scheduler for issuing requests to memory (out-of-order).

The Request Queue (RQQ) is a 16 entry shift-register queue. Entries are added in-order as requests come from ZBbus and are issued out-of-order to the memory channels. Any entry in the RQQ can be issued to memory. Entries in the queue can be reserved for use by network DMA, this reduces the latency of reads and improves performance when using slower or highly loaded memory systems.

The Data Buffer (DBF) is also 16 entry and is shared between reads and writes. Each entry in the buffer is a full 256 bit block wide. The buffer has multiple pairs of read and write ports; one pair for ZBbus which can transfer a full block each access, and a pair for each of the memory ports that transfer 64 bit double words.

When a request is received from ZBbus, entries are allocated in both the RQQ and the DBF. Entries are normally added to the RQQ in sequential order. However, checks are first done to detect any write to read or write to write address dependencies.

The RQQ holds the memory access information and initially the DBF is empty. If the request is a write then the request is not ready for submission to the memory until the data has been received from ZBbus and written to the DBF, at which time the RQQ entry becomes valid for the memory scheduler. Both entries are freed when the write request is issued to the SDRAMs. If the request is a read the RQQ entry becomes valid as soon as the L2 cache reports a miss, and the DBF entry is used to stage the data when it returns from memory. In this case the entries are freed when the data is sent on ZBbus.

The memory controller calculates an ECC during writes, and will check it during reads. This provides correction of all single bit errors and detection of double bit errors. An extra 8 bit ECC bus accompanies the 64 bit data bus, matching the standard 72 bit DDR DIMMs. ECC checking can be disabled.

The memory controller will service uncacheable requests from ZBbus. A read-modify-write cycle for the full line is needed for uncacheable writes, merging the new data into the existing block and recalculating the ECC. The additional memory accesses and the poor utilization of the DBF will cause the memory system performance to degrade if many uncacheable accesses are done.
Entries are selected for issue from the RQQ based on the state of the other entries and the status returned by the memory pipeline controller for each of the channels. The issues can be out-of-order, and use the following rules:

- Any access to configuration registers causes serialization of the queue. These accesses can be in progress for some time (40-50 ZBbus clocks), so should not be done during normal operation.
- Any access with address conflict with an entry already in the queue will be only issued when it reaches the head of the queue. (It can be bypassed by other requests).
- If priority is enabled in the `mc_config_1` register, any read requests from I/O bridge 1 are given priority and are issued in-order. If a read request conflicts with an existing write, the write is also given priority.
- If a channel is currently doing a read, all reads to that channel will be issued ahead of any writes.
- If a read has been bypassed enough times to reach its age_limit (see below) it will go ahead of other reads.
- If a channel is currently doing a write, all writes to that channel will be issued ahead of any reads up to a maximum number of consecutive writes that are set in the channel configuration register.
- If the open page policy is being used an entry that hits on an open page goes first.
- If the open page policy is disabled an entry that is to a precharged bank (closed bank) goes first.
- If a channel is currently idle, reads will be issued before writes.
- Otherwise entries are issued in the order they were allocated.
When a channel switches from writes to reads (and back) there is a delay while the data bus lines and timing strobes (DQSs) turnaround from the controller driving to the memory driving (or the reverse). To get the best bandwidth from the channel the controller batches writes and reads to minimize the number of these turnaround delays. If the controller is doing reads then any writes get buffered in the RQQ. Eventually there will either be no new read requests or the RQQ will become full of writes, the controller will then only have writes to service and can drain them. Since all writes are posted (once they have been sent the sender receives no reply) there is no problem in delaying them in this way.

When the controller is performing a write burst any reads will suffer additional latency as they wait in the RQQ for the writes to drain. The simple case is that the read is trapped behind all the writes that have been buffered up during the previous read burst. However, since the writes drain the RQQ additional writes could be inserted into the queue further delaying the read. To minimize the extra latency a read will suffer during a write burst, the controller counts the number of writes done in the burst and will switch to reads if the burst is longer than the wr_limit (and there are reads waiting). The write limit is set per-channel in the mc_config register. The memory bus utilization can be increased by increasing the limit, the average read latency will be decreased by decreasing the limit.

A similar situation arises for a channel that is just doing reads. Reads to an open page will be issued ahead of reads to a closed page. Once a read is completed the data can be returned and the RQQ entry freed. It is therefore possible for more requests to the open page to be added to the queue, these will also bypass the read to the closed page. To limit the length of time the read to the closed page is blocked, the controller limits the number of reads that may pass it. Every time a read is bypassed its age is incremented, when it reaches the age_limit no new entries will be permitted to pass it and it (and any entries ahead of it) will drain from the queue. The age_limit is set per-channel in the mc_config register. The trade-off is the same as for the wr_limit: the memory bus utilization will be increased by increasing the limit, the average read latency will be decreased by decreasing the limit.

Reads from I/O bridge 1 can be given priority to ensure timely servicing. In most systems this should be enabled (which is the default) to avoid transmit buffer underruns at high data rates. The low relative frequency of I/O bridge 1 DMA requests minimizes the impact of this prioritization on other requests, but ensures that the latency sensitive requests are not delayed behind a high frequency request stream (for example from the CPU or Data Mover). Along with this RQQ entries may be reserved for I/O bridge 1 requests. If the number of entries in the queue reaches the limit set in the iob1_qsize field in the mc_config_1 register then all agents apart from I/O bridge 1 will back off from accessing the memory until the number of entries in the queue falls below the iob1_qsize set in mc_config_0. Some hysteresis is provided by having the two limits. The number of buffers that should be reserved depends on the number of active DMA channels, their bandwidth, and the amount of other memory activity in the system. For a relatively high load on the system a reasonable staring point is to reserve 5 buffers. Note that if zeros are used for the iob1_qsize fields the memory controller performance will be extremely poor.

Each memory channel interface consists of:

- The memory pipeline control queue and scheduler (MCQ).
- Memory data FIFO registers (MFIFO).
- Configuration registers (CRREG).

The MCQ keeps track of the activities of the memory cycle, open rows and banks. It supplies this information to the issue logic for the request queue and the memory scheduler.
MEMORY ACCESS SEQUENCING

Certain applications need more control over the memory access pattern than is available using the normal operations in the memory controller, for example when the memory is being used for transaction logging. The controller was designed to optimize the normal case, but there are several methods that software can use to force the sequence.

The ultimate control is to set the force_seq bit in the mc_config register. This forces the controller to send requests to the SDRAMs in the same order their address phase happens on the ZBbus. This prevents most of the memory optimizations and is likely to be low performance.

In many cases a memory barrier operation is sufficient. This can be provided by an access to the memory controller configuration registers. An access to one of the channel registers will force all reads/writes ahead of it to complete before any behind it. In the normal case a CPU doing a write of A to address 1 followed by a write of B to address 2 provides no guarantee that A gets written into the SDRAM before B. However, if the CPU performs a write of A to address 1 followed by a write of 0 to the mc_test_ecc register and finally the write of B to address 2 then A is always written to memory before B. All these operations are posted, so the CPU has no way of knowing when they complete.

If the CPU needs to both place a memory barrier and be informed of the completion then a read may be done to one of the controller configuration registers. If the CPU performs a write of A to address 1 followed by a read of the mc_test_ecc register and finally a write of B to address 2 then A is always written to memory before B and when the CPU sees the result of the read it knows that A will have been written to the SDRAM (but B may still be in the queue). A sync instruction or use of the read result can be used to stall the CPU until the read completes.

CLOCK RATIOS AND CLOCKING SCHEME

The memory clock range is from 133 MHz - 200 MHz produced by dividing down the ZBbus clock, which is half the CPU core frequency. The ratio of the clocks can be set from 1:2 to 1:4.5 in steps of 0.5, by programming the channel configuration register. The channels may run at different speeds.

<table>
<thead>
<tr>
<th>CPU Clock</th>
<th>ZBbus Clock</th>
<th>Memory Clock 4.5:1</th>
<th>Memory Clock 4:1</th>
<th>Memory Clock 3.5:1</th>
<th>Memory Clock 3:1</th>
<th>Memory Clock 2.5:1</th>
<th>Memory Clock 2:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>600</td>
<td>133.3</td>
<td>150.0</td>
<td>171.4</td>
<td>200.0</td>
<td>240.0</td>
<td>300.0</td>
</tr>
<tr>
<td>1150</td>
<td>575</td>
<td>127.8</td>
<td>143.8</td>
<td>164.3</td>
<td>191.7</td>
<td>230.0</td>
<td>287.5</td>
</tr>
<tr>
<td>1100</td>
<td>550</td>
<td>122.2</td>
<td>137.5</td>
<td>157.1</td>
<td>183.3</td>
<td>220.0</td>
<td>275.0</td>
</tr>
<tr>
<td>1050</td>
<td>525</td>
<td>116.7</td>
<td>131.3</td>
<td>150.0</td>
<td>175.0</td>
<td>210.0</td>
<td>262.5</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>111.1</td>
<td>125.0</td>
<td>142.9</td>
<td>166.7</td>
<td>200.0</td>
<td>250.0</td>
</tr>
<tr>
<td>950</td>
<td>475</td>
<td>105.6</td>
<td>118.8</td>
<td>135.7</td>
<td>158.3</td>
<td>190.0</td>
<td>237.5</td>
</tr>
<tr>
<td>900</td>
<td>450</td>
<td>100.0</td>
<td>112.5</td>
<td>128.6</td>
<td>150.0</td>
<td>180.0</td>
<td>225.0</td>
</tr>
<tr>
<td>850</td>
<td>425</td>
<td>94.4</td>
<td>106.3</td>
<td>121.4</td>
<td>141.7</td>
<td>170.0</td>
<td>212.5</td>
</tr>
<tr>
<td>800</td>
<td>400</td>
<td>88.9</td>
<td>100.0</td>
<td>114.3</td>
<td>133.3</td>
<td>160.0</td>
<td>200.0</td>
</tr>
<tr>
<td>750</td>
<td>375</td>
<td>83.3</td>
<td>95.8</td>
<td>107.1</td>
<td>125.0</td>
<td>150.0</td>
<td>187.5</td>
</tr>
<tr>
<td>700</td>
<td>350</td>
<td>77.8</td>
<td>87.5</td>
<td>100.0</td>
<td>116.7</td>
<td>140.0</td>
<td>175.0</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>72.2</td>
<td>81.3</td>
<td>92.9</td>
<td>108.3</td>
<td>130.0</td>
<td>162.5</td>
</tr>
</tbody>
</table>
Table 58: Clock Speed (Cont.)

<table>
<thead>
<tr>
<th>CPU Clock</th>
<th>ZBus Clock</th>
<th>Memory Clock 4.5:1</th>
<th>Memory Clock 4:1</th>
<th>Memory Clock 3.5:1</th>
<th>Memory Clock 3:1</th>
<th>Memory Clock 2.5:1</th>
<th>Memory Clock 2:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>300</td>
<td>66.7</td>
<td>75.0</td>
<td>85.7</td>
<td>100.0</td>
<td>120.0</td>
<td>150.0</td>
</tr>
<tr>
<td>550</td>
<td>275</td>
<td>61.1</td>
<td>68.8</td>
<td>78.6</td>
<td>91.7</td>
<td>110.0</td>
<td>137.5</td>
</tr>
<tr>
<td>500</td>
<td>250</td>
<td>55.6</td>
<td>62.5</td>
<td>71.4</td>
<td>83.3</td>
<td>100.0</td>
<td>125.0</td>
</tr>
<tr>
<td>450</td>
<td>225</td>
<td>50.0</td>
<td>56.3</td>
<td>64.3</td>
<td>75.0</td>
<td>90.0</td>
<td>112.5</td>
</tr>
<tr>
<td>400</td>
<td>200</td>
<td>44.4</td>
<td>50.0</td>
<td>57.1</td>
<td>66.7</td>
<td>80.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Table 58 shows possible memory clock frequencies for different divide ratios and CPU clock speed. Table 59 shows the percentage difference for the closest frequency below some common DIMM frequencies for each of the CPU clock speeds (entries are left blank if the closest frequency is more than 10% under the required one).

Table 59: Percent Deltas from Popular DIMM Frequencies

<table>
<thead>
<tr>
<th>CPU Clock</th>
<th>ZBus Clock</th>
<th>133 MHz DIMM Clock</th>
<th>143 MHz DIMM Clock</th>
<th>166 MHz DIMM Clock</th>
<th>183 MHz DIMM Clock</th>
<th>200 MHz DIMM Clock</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>600</td>
<td>0.0</td>
<td>-6.8</td>
<td>-10.0</td>
<td>-6.5</td>
<td>0.0</td>
</tr>
<tr>
<td>1150</td>
<td>575</td>
<td>-4.1</td>
<td>-</td>
<td>-1.4</td>
<td>-</td>
<td>-4.2</td>
</tr>
<tr>
<td>1100</td>
<td>550</td>
<td>-8.3</td>
<td>-3.8</td>
<td>-5.7</td>
<td>0.0</td>
<td>-8.3</td>
</tr>
<tr>
<td>1050</td>
<td>525</td>
<td>-1.5</td>
<td>-8.2</td>
<td>-10.0</td>
<td>-4.5</td>
<td>-</td>
</tr>
<tr>
<td>1000</td>
<td>500</td>
<td>-6.2</td>
<td>-0.1</td>
<td>0.0</td>
<td>-9.1</td>
<td>0.0</td>
</tr>
<tr>
<td>950</td>
<td>475</td>
<td>-</td>
<td>-5.1</td>
<td>-5.0</td>
<td>-</td>
<td>-5.0</td>
</tr>
<tr>
<td>900</td>
<td>450</td>
<td>-3.5</td>
<td>-10.1</td>
<td>-10.0</td>
<td>-1.8</td>
<td>-10.0</td>
</tr>
<tr>
<td>850</td>
<td>425</td>
<td>-8.9</td>
<td>-0.9</td>
<td>-</td>
<td>-7.3</td>
<td>-</td>
</tr>
<tr>
<td>800</td>
<td>400</td>
<td>0.0</td>
<td>-6.8</td>
<td>-4.0</td>
<td>-</td>
<td>0.0</td>
</tr>
<tr>
<td>750</td>
<td>375</td>
<td>-6.2</td>
<td>-</td>
<td>-10.0</td>
<td>-</td>
<td>-6.3</td>
</tr>
<tr>
<td>700</td>
<td>350</td>
<td>-</td>
<td>-2.1</td>
<td>-</td>
<td>-4.5</td>
<td>-</td>
</tr>
<tr>
<td>650</td>
<td>325</td>
<td>-2.5</td>
<td>-9.1</td>
<td>-2.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>600</td>
<td>300</td>
<td>-10.0</td>
<td>-</td>
<td>-10.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>550</td>
<td>275</td>
<td>-</td>
<td>-3.8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>500</td>
<td>250</td>
<td>-6.2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>450</td>
<td>225</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>400</td>
<td>200</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
MEMORY CONFIGURATIONS

Before the first SDRAM access starts, the memory configuration registers must be initialized, and the correct startup sequence issued to the memory chips. This is normally done by the BootROM code. The Broadcom CFE firmware includes memory controller initialization which uses the Serial Presence Detect ( SPD) EEPROM information to support a wide range of memory types. This code provides the best starting point for memory initialization.

MAPPING

The physical address space is mapped to memory address space. The physical address map has four 256 Mbyte regions and one expanded 508 Gbyte region of the address space allocated to the memory controller. The memory controller will map the four 256 Mbyte regions into a single contiguous 1 Gbyte region of memory address space for use by the memory channels. Memory address space is only used in the configuration of the memory controller.

The constraints on the physical memory map of the part (see Section: “Memory Map” on page 34) caused the main memory address ranges in the low 4GB of the address space to be allocated in non-contiguous blocks. However, the memory controller configurations are more flexible if it sees them as a single contiguous block. For example, consider a system where the actual memory consists of one single physical bank (i.e. only one chip select) 512 MB DIMM. Since there is only one chip select this must be configured to have a single address range. But the physical address map has one 256 MB memory block starting at zero (which contains the CPU exception vectors and must be present) and the next 256 MB block starts at 00_8000_0000. If the memory controller used physical addresses there would not be a way to configure the address range to use the whole DIMM. However, in memory address space the two memory blocks can be made contiguous and the DIMM can be configured.

<table>
<thead>
<tr>
<th>Physical Address Space (used to access the memory)</th>
<th>Memory Address Space (used to configure the controller)</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>00_0000_0000 - 00_0FFF_FFFF</td>
<td>00_0000_0000 - 00_0FFF_FFFF</td>
<td>256 MB</td>
</tr>
<tr>
<td>00_8000_0000 - 00_8FFF_FFFF</td>
<td>00_1000_0000 - 00_1FFF_FFFF</td>
<td>256 MB</td>
</tr>
<tr>
<td>00_9000_0000 - 00_9FFF_FFFF</td>
<td>00_2000_0000 - 00_2FFF_FFFF</td>
<td>256 MB</td>
</tr>
<tr>
<td>00_C000_0000 - 00_CFFF_FFFF</td>
<td>00_3000_0000 - 00_3FFF_FFFF</td>
<td>256 MB</td>
</tr>
<tr>
<td>01_0000_0000 - 7F_FFFF_FFFF</td>
<td>01_0000_0000 - 7F_FFFF_FFFF</td>
<td>508 GB</td>
</tr>
</tbody>
</table>

The mapping for the four physical address ranges that have zero for the upper eight bits is configured in the channel configuration register. They can be mapped into the first four 256 MB blocks of the memory address space. The default mapping is shown in Table 60 on page 109. There is normally no need to change from the defaults.

CHANNEL SELECT

On the BCM1250 the two channels may be interleaved or they may be independent. If they are interleaved then they must match in all parameters. When they are interleaved a single address bit is used to select between them. When the channels are independent the start and end address set for the chip select generation will also select between the two channels.
CHIP SELECT

A memory controller channel has four active low Chip Selects: CS[3:0]. The address range that they become active for is configurable. On the BCM1250 if the two channels are interleaved then the chip selects for each channel are programmed to an identical address range, which is twice the size of the memory attached to one of the channels. Within a channel there are three methods for generating the chip selects: interleaved-CS, msb-CS and mixed-CS.

- **msb-CS** divides the memory space into contiguous or non-contiguous sub-spaces based on address ranges. The start and end address for each select line is individually programmable. For example CS[3] covers the top portion of memory space and CS[0] covers the bottom portion. The memory size covered by each CS can be different.

- **interleaved-CS** decodes the chip select from two non-MSB memory address bits, for example the lower column bits or the lower row bits. This increases the possibilities for having active banks. In this configuration the channel must be assigned a contiguous memory address range and each chip select covers the same memory size.

- **mixed_CS** allows interleaving between a pair of chip selects (either CS[0] and CS[1], or CS[1] and CS[2], or CS[2] and CS[3]) and allows the others to be set using their start and end registers. This mode allows devices to be interleaved when only two chip selects are in use.

![Figure 15: Chip Select Options](image)

**Figure 15: Chip Select Options**

Figure 15 on page 110 shows some chip select options. On the left msb_cs is used to separate the four chip selects into different regions. The diagram in the center shows a full interleaved_cs, which requires all the chip select regions are the same size. On the right mixed_cs has the SDRAMs on chip select 0 and 1 interleaved and keeps the SDRAMs on chip select 2 and 3 separate. For the memory covered by chip selects 0 and 1 there can be twice as many active banks (although there maybe a chip-to-chip bubble on some accesses).
The address range covered by each of the chip selects is set in the mc_cs_start and mc_cs_end registers. The start address register holds address bits [39:24] that when extended with zeros in bits [23:0] give the lowest address in the range. The end address register holds address bits [39:24] that when extended with zeros in bits [23:0] give the first address above the range. The comparison is done after addresses have been translated from physical addresses to memory addresses (see Section: “Mapping” on page 109).

If two chip select regions are interleaved (mixed_cs mode) their start address registers must be set to the start of the common range, and their end address registers are set to the end address of the common range plus 1 (i.e. they are programmed identically, to a range twice the size of one of them). The address bit that selects between the two chip selects is configured by writing a single one in that bit position of the mc_cs_interleave register.

If all four chip select regions are interleaved (interleaved_cs mode) their start address registers must be set to the start of the common range, and their end address registers are set to the end address of the common range plus 1 (i.e. they are programmed identically, to a range four times the size of one of them). The address bits that selects between the two chip selects is configured by writing two ones in adjacent bit positions of the mc_cs_interleave register.

If a chip select is not used its start and end address should both be set to zero.

If the two channels are interleaved then the chip select region sizes will be doubled and must be identical for both channels.

If a request is received by the memory controller that does not match in any of the chip select regions no SDRAMs will be selected. On writes the data will be discarded. On reads the memory controller will terminate the cycle by reading UNPREDICTABLE data. The controller can be configured to return either a bus error or a valid data flag with the data. In normal operation the bus error would be used to signal that a bad memory address was used. During system testing and sizing it may be useful to have the controller always return valid data. The controller will not hang during an access with no chip select asserted.
EXAMPLE CHANNEL AND CHIP SELECT CONFIGURATIONS

Configuring the memory map for channels and chip selects is usually straightforward, but there are a few special cases. In this section some example configurations are presented. The values that are set for chip select start and end addresses are shown in the figures, these represent bits [39:24] of the address.

The simplest memory system uses a single chip select on a single channel. Using 256Mb memory technology a memory system could be built using 4 16Mx16 memory chips (5 if ECC is required) to give a total of 128MB of memory.

Figure 16 shows how this could be configured. The standard mapping from physical to memory addressing is used, although only the First SDRAM region is needed for 128MB from physical address 00_0000_0000 to 00_07FF_FFFF. This maps to the same address in memory space. The controller chip selects are all disabled apart from channel 0 chip select 0 which is set with start and end to cover this range. Any reads to the space that is not configured will result in a bus error or UNPREDICTABLE data being returned depending on the setting of the berr_disable bit in the mc_config register.

Using the 512Mb memory technology and using 8 (or 9 with ECC) 64Mx8 allows each chip select to have 512MB of memory. Two physical banks will therefore allow a 1GB memory system to be built. There are three interesting configurations (two on the BCM1125/H). In these advantage is taken of the memory address space collapsing the configuration range into a contiguous space. In the physical address space the CPU and DMA engines use the 1GB is split up into four 256MB chunks, but in the actual memory there are just two 512MB physical banks. The translation into memory address space pulls the four physical address regions into a single 1GB chunk. In the simplest configuration this can be split in two and allocated to chip selects.
Figure 17 shows the first configuration. This configuration has chip select 0 for the low half and chip select 1 for the upper half and the start and end addresses of their range are set to reflect this.

![Diagram of DRAM configuration]

**Figure 17:** Example 1GB with two chip selects on one channel
Figure 18 is an alternative that uses chip select interleaving to (on average) increase the number of active banks. Notice the chip select ranges are now the same for both selects.

On the BCM1250 there is a higher performance possibility. Rather than putting both physical banks on the same channel one can be put on each channel. This doubles the number of data bits in use (since the controller can run both channels in parallel), allowing twice the peak bandwidth.
Figure 19 shows this, with the interleave on a relatively low address bit. Notice that again the chip select ranges are the same for both selects, but in this case they are for different channels.

Figure 19: Example 1GB with two chip selects interleaved across both channels

These examples have benefited from the memory address space map having a contiguous 1GB range. Additional memory could be added in the expansion space. However, there are some configurations that require a 2GB chip select region. For example if the configurations used in Figure 17 - Figure 19 were upgraded to use 1Gb memory technology (rather than 512 Mb) the size will double. The same effect will be seen if two additional physical banks are added and 4-way chip select interleave used in the single channel case or channel and chip select interleaving used in the dual channel case. Systems using the Big Memory mode (see Section: “Larger Memory Systems” on page 124) will also face this issue even when using lower density memory parts.

The 2GB problem can be solved with a little help from software. The hardware is configured in a way that creates an alias of the low 1GB of memory which software should ensure is never used (for example by the virtual-physical address translation in the TLB). Rather than creating a 2GB chip select range, the chip selects are programmed for a 6GB range and address bits [32:31] are not selected for use as a row, column or bank address. Thus the real 2GB range appears 3 times in the range given to the chip selects.
In Figure 20 it can be seen that 3GB of the aliases are hidden in the memory address space that cannot be accessed from the physical space. The only alias that needs to be considered is the 01_0000_0000 - 01_3FFF_FFFF alias of 00_0000_0000 - 00_3FFF_FFFF. A particular memory location must be accessed through only one of these ranges or the coherency algorithm will break down. The CPU exception vectors are all at the bottom of memory so in most systems the 00_0000_0000 - 00_3FFF_FFFF space will be used. Some systems may choose to use 00_0000_0000 - 00_1FFF_FFFF (accessible through kseg0) and 01_2000_0000 - 01_3FFF_FFFF to keep the upper range contiguous with the second 1GB -- however this only allows DMA access to the low 256MB from the PCI (or from 32-bit addressing peripherals bridged from the HT interface) since the DMA and CPU accesses must use the same physical range. (PCI accesses translated by the BAR0 map could be used.)

![Figure 20: Example 2GB with two chip selects interleaved on one channel](image-url)
ROW, COLUMN AND BANK CONFIGURATION

The memory channel can be structured in many different ways with many different types of SDRAMs on different types of DIMMs and different DIMM counts per memory channel. In order to support such different configurations, the assignment of the address bits for row, column, and internal bank selection are fully software configurable along with the SDRAM timing parameters and operating modes.

Each memory channel is 8 bytes (64 bits) wide with ECC (8 bits), so the three lowest physical address bits play no part in the memory addressing. The memory controller always accesses memory with a burst of four 64-bit double-words, to access a full cache line. Therefore bits [4:3] of the physical address are always used as the lowest two column address bits. (The SDRAMs count the burst internally, so the memory controller only ever drives 00 on these two bits.)

<table>
<thead>
<tr>
<th>Table 61: Address Bits Used by a Memory Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM Address</td>
</tr>
<tr>
<td>Row Address[13:0]</td>
</tr>
<tr>
<td>Row Address[14:0] (FCRAM)</td>
</tr>
<tr>
<td>Column Address[1:0]</td>
</tr>
<tr>
<td>Column Address[12:11, 9:0]</td>
</tr>
<tr>
<td>Bank Address[2:0]</td>
</tr>
<tr>
<td>Chip Selects (4)</td>
</tr>
</tbody>
</table>

Table 61 shows the maximum number of address bits that can be used for the row, column and bank parts of the SDRAM address. Each of the memory channels can theoretically address 4GB of memory (for a total of 8GB attached on a BCM1250). This comes from the number of RAS (13 or 14), CAS (12 or 11), bank (2), chip select (2) and bytes in dword(3) = 32 address bits per channel. (The gigabit SDRAMs are mostly configured with 14 row and 11 column address bits, to support these the extra A13 pin was added in PERIPH_REV3 of the BCM1250 and on the BCM1125/H.) However, there are only four chip selects (hence the 2 bit equivalent shown above) per channel, and four bit wide devices are not supported. The maximum size is therefore achieved using four physical banks of 8 (or 9 with ECC) eight bit wide devices. Using 256Mb technology parts this limits the total size to 1GB per channel or 2GB total. Using 512Mb parts this doubles and will double again when the 1Gb DDR parts come out (reaching the theoretical maximum of 8GB).

The memory controller can use fast DDR parts (200 MHz clock). To work at this speed the signal trace lengths between the controller and the memory devices must be carefully controlled, maintaining tight tolerances and avoiding long stubs. Using a carefully chosen termination scheme DIMMs may be run at this speed.
Section: “Larger Memory Systems” on page 124 describes a mode of the memory controller that allows external generation of eight chip selects rather than four. This mode does not increase the theoretical maximum size of the memory, but it does double the maximum achievable size for the 256 Mb and 512 Mb parts. However, the address loading may require the use of registered DIMMs, the data loading will limit the maximum clock speed and there are limits on the supported memory configurations because the controller acts as if only two chip selects are in use.

The controller is programmed per chip select to determine the address bits used to generate the bank, row and column address bits sent to the SDRAM. The row and column address size depends on the devices used. The controller is programmed with bit masks that are applied to the memory address to extract the bank, row and column address. Bits are set in the mask to indicate the corresponding address bit should be used. The bank address is normally two bits corresponding to the BA[1:0] pins on the controller. If three bits are set in the bank address mask the controller will use the topmost address bit (A[12] for regular SDRAM, A[14]/CS[3] for FCRAM) as an extra bank bit and will work with 8 bank parts. With two exceptions described below, the set bits in the masks must be contiguous, so the right-most bit sets the lowest bit used and the number of set bits should match the number of address bits needed by the device.

Example masks are shown in Table 62. The bank is selected from the lowest bits, the column next and the row from the upper bits. Note that the column masks need not have bits [4:3] set, these address bits are always used as the low column bits.

<table>
<thead>
<tr>
<th>Table 62: Example for 128 MByte CS Region with 4K Rows, 1K Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Row Address Bits [26:15]</strong></td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Column</td>
</tr>
</tbody>
</table>

One problem with this example is that the internal bank is switched every cache line, so streaming data that transfers in 64 byte blocks (for example HyperTransport reads or DMAs to the MACs) will not make good use of open pages in the SDRAM. The interleaving for banks (or chip selects or channels) is likely to work better with a larger block. The two exceptions to the rule that the masks must be contiguous allow for this.

The column mask may be split so that bit 5 or bits [6:5] may be set in addition to the contiguous bits. This allows for the interleave to be on 64 or 128 byte chunks. Changing the example to use 64 byte blocks gives the masks shown in Table 63, and using 128 byte blocks gives the masks in Table 64. This example only shows interleaving the internal banks of a single device, if there were chip select or channel interleaving there would be more than two zeros separating the set bits in the column mask.

<table>
<thead>
<tr>
<th>Table 63: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 64 Byte Interleave</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Row Address Bits [26:15]</strong></td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>Row</td>
</tr>
</tbody>
</table>
The previous examples show interleaving between the banks within a device. It is also possible to use interleave between chip selects on a channel. This is done by setting an appropriate mode in the `mc_config` register. If there are two chip selects available on a channel (i.e. there are two physical banks of memory of the same size) then the mixed_cs mode can be used to allow them to be interleaved. When this mode is set the `mc_cs_interleave` register can have a single bit set to program which bit of the address is used to switch between chip selects. Table 65 gives an example of this, the same 128 byte interleave is done between banks on the same device, but the upper row and column bits have been moved to allow for switching between the devices on the channel using bit [9].

### Table 64: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 128 Byte Interleave

<table>
<thead>
<tr>
<th>Row Address Bits [26:15], Column Address Bits [14:9,6:5,4:3], Bank Address Bits [8:7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Bank</td>
</tr>
</tbody>
</table>

The previous examples show interleaving between the banks within a device. It is also possible to use interleave between chip selects on a channel. This is done by setting an appropriate mode in the `mc_config` register. If there are two chip selects available on a channel (i.e. there are two physical banks of memory of the same size) then the mixed_cs mode can be used to allow them to be interleaved. When this mode is set the `mc_cs_interleave` register can have a single bit set to program which bit of the address is used to switch between chip selects. Table 65 gives an example of this, the same 128 byte interleave is done between banks on the same device, but the upper row and column bits have been moved to allow for switching between the devices on the channel using bit [9].

### Table 65: Example for 256 MByte Region with 4K Rows, 1K Columns, two CS, and 128 Byte Interleave

<table>
<thead>
<tr>
<th>Row Address Bits [27:16], Column Address Bits [15:10,6:5,4:3], Bank Address Bits [8:7], mixed_cs Selected by Bit [9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Bank</td>
</tr>
<tr>
<td>CS Interleave</td>
</tr>
</tbody>
</table>

If there are four physical banks of memory then the interleave_cs mode can be used to allow a full interleave between the physical banks. This is shown in Table 66.

### Table 66: Example for 512 MByte Region with 4K Rows, 1K Columns, four CS, and 128 Byte Interleave

<table>
<thead>
<tr>
<th>Row Address Bits [27:17], Column Address Bits [16:11,6:5,4:3], Bank Address Bits [8:7], Interleaved_cs Selected by Bit [10:9]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Bank</td>
</tr>
<tr>
<td>CS Interleave</td>
</tr>
</tbody>
</table>
Combining the bank, row and column masks with the chip select selection allows for more complicated mixing on a single channel. Table 67 shows a configuration using 128 MByte single chip select DIMM on chip select 0, and a 128 MByte dual chip select DIMM (also called a two bank DIMM using the term bank to refer to the physical banks) on chip select 1 and 2. Mixed-cs mode is used to allow the first DIMM to be standalone and the other to be interleaved. In this case the switch between the two physical banks is made based on address bit 15, so the full column addressed page is contiguous in the address space.

**Table 67: Example for 128 MByte + 64 MB + 64 MB Mixed_CS Mode**

<table>
<thead>
<tr>
<th>CS0</th>
<th>Row Bits [26:15], Column Bits [14:9,6:5,4:3], Bank Bits [8:7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row</td>
<td>00000000_00000111_11111111_10000000_00000000</td>
</tr>
<tr>
<td>Column</td>
<td>00000000_00000000_00000000_01111110_01100000</td>
</tr>
<tr>
<td>Bank</td>
<td>00000000_00000000_00000000_00000000_00000000_10000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CS1,2</th>
<th>Row Bits [26:16], CS Interleave Bit [15], Column Bits [14:9,6:5,4:3], Bank Bits [8:7]</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS Interleave</td>
<td>00000000_00000000_00000000_10000000_00000000</td>
</tr>
<tr>
<td>Row</td>
<td>00000000_00000000_00000000_01111111_01111111_01111111_01100000_00000000_10000000</td>
</tr>
<tr>
<td>Column</td>
<td>00000000_00000000_00000000_00000000_00000000_00000000_10000000_00000000_00000000</td>
</tr>
<tr>
<td>Bank</td>
<td>00000000_00000000_00000000_00000000_00000000_00000000_00000000_00000000_00000000</td>
</tr>
</tbody>
</table>

**CHOOSING INTERLEAVE PARAMETERS**

The best method of interleaving the memory is highly system dependant, so there is no single rule about how the memory controller should be configured. However, there are some general guidelines that give a starting point. The general goal is to maximize the number of banks (internal and physical) of the memory that are in use, which in turn leads to more overlap of transactions and less unused time on the channels. There are different costs involved in switching between banks: if the driver of the databus switches from one device to another (a change in physical banks) there must be an additional turnaround cycle which is not needed for a bank to bank switch in the same device (a change in internal banks). There are several measurable parameters: memory bandwidth and average access latency are normally the most important, but maximum access latency may also be important. Some things (e.g. a faster memory clock) will improve all of them, but many of the options improve one at the expense of others. In particular reductions in average memory latency are often at the expense of increasing the maximum latency.

In most cases the system will use more than one physical bank of memory. These can be put on different chip selects on the same channel or on the different channels. From a performance standpoint using both memory channels is almost always the correct choice. The peak memory bandwidth increases because the two channels have physically separate data buses and operate independently of one another. In some cases using both channels may not be possible on the BCM1250 (for example if the system has a single DIMM slot then a two physical-bank DIMM is necessarily on a single channel), but both channels should be used if possible (to continue the example: a second DIMM slot should go on the other channel in preference to using the remaining two chip selects of the first channel). On the BCM1125/H, two DIMM slots with two physical-bank DIMMs should be used if possible and full chip select interleaving should be enabled.
The simple configuration with two channels each with one physical bank of memory provides a good illustration of the trade-offs involved. The most straightforward way to assign the channels is to have one cover address 0 to N-1 and the other address N to 2N-1. However, most of the time there is a reasonable degree of locality (for example, it is likely that the program and its data will reside entirely in the first N locations of memory) so all activity is going to a single channel -- the effective memory bandwidth has been halved and the access latency will increase because of contention for the channel. In some systems this may be desirable, the second channel will have a lower and more deterministic access latency since it is so lightly loaded. But this gain has come at a very large cost to the system performance as a whole. At the other extreme the channels could be interleaved every cache line. This seems a good choice since the distribution of accesses is likely to be equal across even and odd cache lines, so both channels will be equally used. But since a contiguous access (e.g. a packet streaming in or out of the system) will flip back and forth between the two channels there is less likelihood that good use can be made of open pages in the memory. A good compromise is to interleave every four cache lines.

Note that the argument of the previous paragraph will also apply to packet buffers. Dedicating one channel to packet buffers and one to the program and associated data will quite often result in the packet buffer channel (which has CPU accesses as well as inbound and outbound DMA) being very heavily used and the other channel (which will only be used on accesses that miss in both the L1 and L2 caches) being under-used. In this case the network traffic is limited to half the memory bandwidth and will incur latency associated with using a busy channel. The system performance will be improved by interleaving the channels and thus removing the hot spot and allowing the bandwidth to be shared more evenly.

A good general starting point applies the principle from the previous examples: keep a few cache lines contiguous to allow for page mode accesses, then use low bits to interleave across the two channels, the internal banks within a device, and the physical banks (chip selects) on a channel. Using the low bits make it likely that even over short periods of time there is a reasonably even distribution of accesses across the regions. The address is therefore broken up as:

RRRR...R CCCC...C NN BB P CC cc000

This format can be used to set the mask bits.

1. The bottom 3 bits are ignored and should be set to zero. The next two bits (cc) are also ignored, but are always used as column bits, so they must be considered when the total number of column bits in the device is checked.
2. The next two bits are used for column interleave. For 32-byte blocks (and no column interleave), do not use any column bits here. For 64-byte blocks, use one column bit here, and for 128 byte blocks use two column bits here. These bits will be set in the mc_csN_col registers.
3. As discussed above, the next bit is a good one to use for interleaving between the two channels. This bit number is assigned in the mc_config register.
4. The next bits are used to select the bank within a memory device. Most devices have four banks, so two bits will be set in the mc_csN_ba registers.
5. When interleaving across physical banks on a channel via chip-selects one or two bits must be set in the mc_cs_interleave register. If there are only two physical banks then mixed_cs mode will be used and one bit set, if there are four physical banks then interleaved_cs mode is used and both interleave bits are needed.
6. The remaining column bits are set in the mc_csN_col registers. These must be contiguous and the number of bits set will be the number of column address bits needed minus the number set in steps (1) and (2).
7. The remaining address bits form the row address in the mc_csN_row registers. The number of bits should match the number of row address bits needed by the memory device.
This will give a good starting point and should give acceptable results in most systems. Modifications can then be made as a result of performance monitoring.

**PAGE POLICY**

The memory controller supports page mode access to the DDR SDRAMs, and provides several different policies for controlling page accesses. The FCRAMs do not support page mode (they always auto-precharge following CAS).

The Closed Page policy is the simplest. Pages are always closed by using CAS cycles with AutoPrecharge. If the memory uses FCRAMs the controller must be configured to use this policy.

The CAS time check policy will check the request queue just before the CAS cycle is issued to a channel. If there are requests in the queue that hit on the page then it is left open (and the same test will be performed when the new request is serviced) otherwise it is closed by issuing the AutoPrecharge command along with the CAS. If a sequence of accesses are done to the same page there is a good chance the page will be open when required.

The Hint Based policy is the same as the CAS time check, except that it accepts a hint with each memory request. The hint is part of the command on the ZBus, and will be asserted by the DMA engines, PCI and HyperTransport interfaces when doing block moves. If the hint is set on a request then even if there are no more requests the page is left open following its CAS time check. If the hint is clear for the request then the behavior matches the CAS time check policy. Pages that are left open by hints will be closed by subsequent accesses that have the hint clear and fail the CAS time check, or will be closed using an explicit Precharge instruction if a page conflict occurs to the same bank.

The Open Page policy leaves the page open until a page conflict occurs to the same bank or an AutoRefresh is done. In the case of a page conflict an explicit Precharge command is performed to close the open page, delaying the RAS cycle to open the new page.

The page policy is set per chip select, allowing different areas of memory to have different policies. The optimal policy can therefore be selected based on the data access patterns. A good starting point is to use the CAS time check policy with hints enabled.
SUPPORTED DRAMs AND DIMMs

DDR SDRAMs

The memory controller supports standard JEDEC DDR SDRAMs in x8 and x16 configurations, and also the x32 DDR SGRAMs. In addition the FCRAMs are supported in both x8 and x16 configurations. The x32 SGRAM parts only use one DQS signal to cover all 32 data bits, the DQS from the DRAM should be connected to the corresponding four DQS pins on the controller (M_DQS[7:4] for the upper word and M_DQS[3:0] for the lower word); the controller will only drive on one of the lines but needs to receive on all of them. X4 configuration parts are not supported since the interface has insufficient DQS signals.

If the memory channel is configured for SGRAMs the A10 and A8 signals may need to be swapped. The memory controller follows the DDR standard and puts the AutoPrecharge flag on the A10 address bit, but many SGRAMs need it on A8. (In parts with system revision indicating PERIPH_REV3 or greater the pre_on_A8 bit can be set in the mc_drammode register to cause the controller to put the AutoPrecharge flag on A8 instead of A10.)

Non-standard 8 bank DDR SDRAMs use A[12] as the third bank address bit, 8 bank FCRAMS use A[14] as the third bank address bit. This is enabled if three bits are set in the bank address selection register. For regular SDRAMs if the ram_with_A13 bit is set in the mc_drammode register the BA[2] is brought out on the A13 pin and A12 reverts to being a row address.

DDR FCRAMs

Memory channels can be configured to use FCRAMs. These require two extra row address bits and use a function (FN) flag to distinguish between a read and write access. The JEDEC defined footprint for standard DDR and FCRAM parts are identical with the exception of pins 21, 22, and 23. On a DDR SDRAM these pins are WE#, CAS#, and RAS#, respectively. On the FCRAM, they become A14, A13, and FN (effectively WE#). The BCM1250 or BCM1125/H uses the same mapping when a channel is configured for FCRAM use, so:

- Mn_WE_L becomes Mn_A[14] for FCRAM.
- Mn_RAS_L becomes Mn_FN for FCRAM.
- Mn_CKE is called Mn_PD_L but is the same.

This allows the standard footprint to be used and either type of memory populated. The FCRAM parts generally have a CAS to write data latency that is one less than the CAS latency, this is supported by setting the tCwD parameter in the SDRAM timing register.

Table 68: Supported SDRAMs

<table>
<thead>
<tr>
<th>DDR SDRAMs</th>
<th>Technology</th>
<th>Size (row x col)</th>
<th>Size (row x col)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JEDEC std (4 bank)</td>
<td>64 Mb</td>
<td>8Mx8 (4k x 512)</td>
<td>4Mx16 (4k x 256)</td>
</tr>
<tr>
<td></td>
<td>128 Mb</td>
<td>16Mx8 (4k x 1k)</td>
<td>8Mx16 (4k x 512)</td>
</tr>
<tr>
<td></td>
<td>256 Mb</td>
<td>32Mx8 (8k x 1k)</td>
<td>16Mx16 (8k x 512)</td>
</tr>
<tr>
<td>Non-std (8 bank)</td>
<td>64 Mb</td>
<td>2Mx32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>128 Mb</td>
<td>4Mx32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>256 Mb</td>
<td>8Mx32</td>
<td></td>
</tr>
<tr>
<td>FCRAM (4 bank)</td>
<td>256 Mb</td>
<td>32Mx8 (32k x 256)</td>
<td>16Mx16 (32k x 128)</td>
</tr>
</tbody>
</table>
The FCRAM includes a write mask or variable write (VW) function. The memory controller always does a full burst write and will set the bits accordingly (A[14:11] = 4'b1010 during LAL command).

**DIMMs**

All standard DDR SDRAM DIMMs of 64+8bit data path are supported, except those that use x4 parts (which replace DM pins with extra DQS pins). Both registered and unbuffered DIMMs can be used (for registered DIMMs the timing parameters must be adjusted to take account of the additional cycle for the latch on the address and command lines). The controller provides four chip selects per channel, allowing two DIMM slots that use dual physical bank modules (i.e. need two chip selects) or four DIMMs with single physical bank modules (each using a single chip select). Software should read the DIMM configuration information and program the memory controller accordingly.

On a single channel it is not possible to mix registered and unregistered DIMMs.

**LARGER MEMORY SYSTEMS**

The memory controller can support larger memory systems by externally decoding 8 chip selects per channel. This may limit the maximum speed of the memory and reduces the number of available configurations. The increased loading requires use of registered DIMMS in the large memory system. This mode is enabled by setting the large_memory bit in the DRAM type field of the mc_drammode register. Application Note 1250-AN600-R “BCM1250 Big Memory System” describes the design and implementation of a board using the large memory extension.

The extension works by configuring the memory controller to only use two chip selects, and driving two additional address bits on the other two chip select outputs. Externally each of the chip selects is used to enable a 2-to-4 decoder to generate the actual chip selects for the physical memory banks (the delay through the decoder must be taken in to account when timing analysis is done, but since the chip select signals will be registered on the DIMM this should not be a critical timing parameter). Internally the memory controller accounts for the switch of chips when the decoded chip select changes, but it only keeps track of the standard four open pages for the two internal chip selects (potentially degrading performance by not making use of open pages).

The external connections are:

- M_CS_l[0] and M_CS_l[1] are the two chip select lines. They should be connected to the active low enable input of the two halves of a dual 2-to-4 decoder.
- M_CS_[3:2] provide two additional address bits to qualify the chip selects. They should be connected to the address inputs of the 2-to-4 decoder
- The (active low) outputs of the 2-to-4 decoder provide 8 chip selects for DRAMs.

The internal configuration should be:

- The full address space should be covered using the mc_cs0 and mc_cs1 address selection registers. The mc_cs2_start, mc_cs2_end, mc_cs3_start and mc_cs3_end registers must all be set to zero.
- Two additional row address bits (corresponding to row_addr[14:13]) should be set in the mc_cs0_row and mc_cs1_row registers. These address bits will be output on the M_CS_[3:2] pins to qualify the chip select. Note that these bits must come from the memory address bits [34:28].
- The large_memory bit in the mc_drammode register should be set.
Initialization of the DRAMs must be done before the large_memory bit is set. It must be done for each of the real chip selects by stepping through the eight patterns of the chip select field in the mc_drammode register. (Note that setting a bit in the chip select field will assert the active low external signal, so to cover all eight external chip selects \{cs3,cs2\} should count through the four values with \{cs1,cs0\} set to both \texttt{2'b01} and \texttt{2'b10}.)

If the large memory mode is used with parts that use fourteen row address bits (i.e. that need A13) then the ram_with_A13 bit must be set in the mc_drammode register, this causes bits \[15:14\] from the row address mask to be passed to CS\[3:2\] rather than bits \[14:13\].

The page policy should be set to always close page or cas-time-check. Memory accesses will have UNPREDICTABLE results if other page policies are used.

The refresh rate should be set in the usual way based on the requirement of the SDRAM parts used. The controller will issue refresh to two (of the eight) chip selects at a time (rather than one of four).

### ECC

The memory controller supports an 8 bit ECC calculated over the 64 bit data. Single bit errors will be detected and automatically corrected as the data passes through the controller. Double bit errors are detected. In either case the data is flagged appropriately when it is sent on ZBus, correctable errors are counted by the SCD; uncorrectable errors will be counted and report a data error at their destination.

Single bit ECC errors will be corrected during the read and the corrected data will be written back into the memory. They will continue to occupy the RQQ entry until the data has been written. Thus a memory location should not return multiple single bit ECC errors. Double bit errors are left as such, so repeated reads of the data will continue to give the error.

ECC checking can be disabled either to support non-ECC DIMMs or during memory testing.

The memory controller can be set to write bad ECC to the memory in order to check the error handling logic and code. There are two registers mc_test_data which is 64 bits and mc_test_ecc which is 8 bits. If either of these are non-zero then during memory writes any bits that are set in these two registers will cause the corresponding data or ECC bit to be inverted during a memory write. This will force an error into the memory. If the location is subsequently read an ECC error should be reported.

### SDRAM TIMING

The timing for DDR SDRAM accesses is set per channel, so all parts on a single channel have common timing. The memory pipeline control queue and scheduler is driven from the timing information programmed for the channel and will compute its schedule to match the rules. The base timing is set by the memory clock programmed in the mc_clock_cfg register, this is divided down from the ZBus clock as outlined in Section: “Clock Ratios and Clocking Scheme” on page 107.

The mc_timing1 and mc_timing2 registers are used to configure the number of memory clock cycles required between critical events. These can be obtained from the data sheet for the SDRAM that is being used (or the common JEDEC timing specification for standard parts). Most of the fields in these registers are four bits so can be set to a value between 0 and 15, however the acceptable range for the parameters is smaller than this in many cases. Programming the fields outside their acceptable ranges has UNDEFINED results.
SDRAM REFRESH

The memory controller uses the SDRAM auto-refresh command to refresh the memory during normal operation. The refresh interval for a channel is set (in memory clocks) in the mc_clock_cfg register. In parts where the system_revision indicates PERIPH_REV3 or later this register also has fields to completely disable refresh (useful only during debugging, to avoid the scope being triggered by refresh cycles) or disable refresh for particular (i.e. unused) chip selects.

Software can cause Auto Refresh commands to be sent to the SDRAM as described in Section: “SDRAM Initialization and Commands” on page 126. This has no effect on the refresh interval counter.

SDRAM INITIALIZATION AND COMMANDS

The DDR SDRAMs accept a number of commands that are not used as part of regular operation. These can be issued under software control by writing the mc_dramcmd register with the command to be performed and the chip selects that should be asserted when the command is executed. The memory controller will insert a synchronization point before the command is executed so that all buffers are empty. In some cases the commands require additional data encoded in the address lines. The mc_drammode register holds the data to be encoded (on address lines A[11:0] for standard DDR parts and A[14:0] for FCRAMs).

Table 69: Commands that can be Issued Through mc_dramcmd Register

<table>
<thead>
<tr>
<th>Name</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMRS</td>
<td>Write Extended Mode Register</td>
<td>Write the value in the mc_drammode register to the Extended Mode Register in the selected parts.</td>
</tr>
<tr>
<td>MRS</td>
<td>Write Mode Register</td>
<td>Write the value in the mc_drammode register to the Mode Register in the selected parts.</td>
</tr>
<tr>
<td>PRE</td>
<td>Precharge all banks</td>
<td>Issue a precharge command to the SDRAMs with address bit 10 high to cause all banks to be precharged. If the pre_on_8 bit is set then this command will run the precharge command with address bit 8 high.</td>
</tr>
<tr>
<td>AR</td>
<td>Auto Refresh</td>
<td>Issue an Auto Refresh command to the SDRAMs.</td>
</tr>
<tr>
<td>SREF_set</td>
<td>Enter Self Refresh mode</td>
<td>Set the SDRAM into Self Refresh mode by issuing a Self Refresh command and deasserting CKE then stopping the clock.</td>
</tr>
<tr>
<td>SREF_clr</td>
<td>Exit Self Refresh mode</td>
<td>Remove the SDRAM from Self Refresh mode by starting the clock, asserting CKE and issuing NOP commands to the part. Software must ensure that the memory is not accessed for the required time after the part is removed from Self Refresh mode (normally 200 memory clocks).</td>
</tr>
<tr>
<td>PDN_set</td>
<td>Enter Power Down mode</td>
<td>Set the SDRAM into Power Down mode by issuing a NOP command and deasserting CKE.</td>
</tr>
<tr>
<td>PDN_clr</td>
<td>Exit Power Down mode</td>
<td>Remove the SDRAM from Power Down mode by asserting CKE and issuing NOP commands to the part. Software must ensure that the memory is not accessed for the required time after the part is removed from Power Down.</td>
</tr>
</tbody>
</table>
When the part is reset the SDRAM channels are put in power-down mode with the clock stopped and CKE deasserted. Because CKE is deasserted all devices on the memory channel will place their outputs in a high impedance state, this will prevent multiple bus drivers as the channel starts up. To initialize a channel software must first start the clock by programming the **mc_clock_cfg**, then wait for 200us for the clock to stabilize before giving the PDN_clr command to assert CKE. Next the DDR SDRAMs must be initialized (this also needs to be done when exiting power down mode if the DLL has been disabled). The initialization sequence is specified in the SDRAM data sheet, however the JEDEC recommended sequence should be a superset of all manufacturer requirements. It is:

1. Precharge all banks (PRE).
2. Enable the DLL and set the appropriate output drive strength in the extended mode register (EMRS).
3. Reset the DLL in the mode register (MRS). From this point no access other than the subsequent initialization steps (4 through 7) may be done for 200 memory cycles.
4. Wait for 2 cycles (tMRD).
5. Precharge all banks (PRE).
6. Issue two auto refresh cycles (AR, AR). (The JEDEC spec allows these to be done before the Precharge in step 5)
7. Configure the mode register for Normal Operation, appropriate CAS latency, Sequential burst and burst length of 4. (MRS)
8. Once 200 memory cycles have passed since the DLL was reset, the SDRAM is ready for normal operation.

The reset initialization sequence is different for FCRAM from regular DDR parts. The memory controller will not release the enable (CKE) when the PDN_clr command is used to start the clock, but waits for a PRE command to be issued before it asserts CKE and automatically runs the address sequencing initialization sequence for all chip selects. The sequence of commands needed is therefore:

1. Set DRAM type to FCRAM. This MUST be done first. This will start the memory clock on Mn_CLK.
2. Wait for minimum of 200us. This is required by the FCRAM which must have a stable clock for a minimum time before being enabled.
5. Set the Mode Register and issue EMRS command.
6. Set the Mode Register and issue MRS command.
8. Wait for 200 or 300 memory cycles (see FCRAM data sheet).
9. Do a dummy write to each bank of each chip select.
10. The device is now ready for normal operation.
I/O CONTROL

The memory controller supports a wide range of SDRAM speeds and configurations. In particular the signal loading and propagation delays will be very different in a system that is fully populated with DIMMs than it is in a system with point-to-point connections between the part and SDRAMs soldered to the main board. The termination schemes may differ too. The timing budget and electrical characteristics must be calculated for each board the BCM1250 or BCM1125/H is used on. In most cases the correct values for the I/O control registers will be calculated during board design and verified during bring up.

The electrical characteristics of the memory controller outputs can be adjusted to match the environment it is in. The drive strength (and thus slew rate for a particular loading) of the clock, address/command and data output drivers can be set in the channel configuration register (mc_clock_cfg). There is one bit of coarse control which changes the driver configuration between SSTL_2 Class 1 (7.6mA min. drive) and SSTL_2 Class 2 (15.2mA min. drive). In addition there are three slew bits that control the slew rate of the driver from fast (3'b111) to slow (3'b000). In most cases the correct setting will be the fast slew rate, but the slower rates may work better for systems with lower loading and when non-standard termination is used.

IMPORTANT! The rest of this description explains the DLLs for the BCM1250 and BCM1125/H where the system_revision indicates PERIPH_REV2 or later. The address DLL was organized differently on the early-access prototype pass-1 parts (marked BCM12500), please see an earlier revision of this User Manual for description of those parts.

An internal master DLL is used to control slave DLLs to allow adjustment of input and output timing. Each DLL has a fixed Offset, a linear delay adjustment (6 bit unsigned number N) and a proportional delay adjustment (4 bit unsigned number M). The total DLL delay is given by:

\[ Delay = Offset + (N \times dll\_step) \times (1 + ((M-8) \times 5\%)) \]

This is the ideal case, in practice there is a part dependent error to the 5%, and it rolls off to more like 4% at the low end. Table 70 gives the typical multiplier mult that results from the \((1 + ((M-8) \times 5\%))\) term. The master DLL is fixed to have \(M=8\) and derives \(N_{\text{master}}\) from the memory clock period \(T_{\text{mclk}}\) such that:

\[ Delay = Offset + (N_{\text{master}} \times dll\_step) = T_{\text{mclk}}/4 \]

The Offset is typically 500ps, and the resolution of dll_step is sufficient to cover memory clock frequencies from 95-200MHz. At lower frequencies the delay may saturate, however at these speeds the cycle time is so long that the proportional adjustment provided by the DLL is not required.

<table>
<thead>
<tr>
<th>M</th>
<th>M-8</th>
<th>Adjustment%</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simulation Result for (M-8)*5%</td>
<td>Simulation Result for 1+((M-8)*5%)</td>
</tr>
<tr>
<td>4'b0000</td>
<td>-8</td>
<td>-35%</td>
<td>0.65</td>
</tr>
<tr>
<td>4'b0001</td>
<td>-7</td>
<td>-31%</td>
<td>0.69</td>
</tr>
<tr>
<td>4'b0010</td>
<td>-6</td>
<td>-27%</td>
<td>0.73</td>
</tr>
<tr>
<td>4'b0011</td>
<td>-5</td>
<td>-23%</td>
<td>0.77</td>
</tr>
<tr>
<td>4'b0100</td>
<td>-4</td>
<td>-19%</td>
<td>0.81</td>
</tr>
<tr>
<td>4'b0101</td>
<td>-3</td>
<td>-15%</td>
<td>0.85</td>
</tr>
<tr>
<td>4'b0110</td>
<td>-2</td>
<td>-10%</td>
<td>0.90</td>
</tr>
<tr>
<td>4'b0111</td>
<td>-1</td>
<td>-5%</td>
<td>0.95</td>
</tr>
</tbody>
</table>
The slave DLLs are adjusted to have their \( N \) set to \( \textit{Nmaster} \) and \( M \) set to the four bit \( \text{dq}_i\_\text{skew} \), \( \text{dq}_o\_\text{skew} \) or \( \text{addr}\_\text{skew} \) field from the \( \text{mc}\_\text{clock}\_\text{cfg} \) register. This value \( M \) can be converted to a multiplier \( \text{mult} \) according to Table 70. The slave DLL delay is given by:

\[
\text{SlaveDelay} = \text{Offset} + (\text{Nmaster} \times \text{dll}\_\text{step}) \times \text{mult}
\]

Substituting for \( \text{Nmaster} \):

\[
\text{SlaveDelay} = \text{Offset} + \left(\frac{T_{mclk}}{4} - \text{Offset}\right) / \text{dll}\_\text{step} \times \text{mult}
\]

\[
\text{SlaveDelay} = \text{Offset} \times (1 - \text{mult}) + \left(\frac{T_{mclk}}{4}\right) \times \text{mult}
\]

The DLLs are used in three places: to shift the received DQS into the center of the data valid window, to shift the relationship between the output clock and signals, and to shift the DQS driven into the center of the data window.

The first is the most straightforward. During a Read the DRAM will drive the data at the same time as it changes DQS. Since the board traces between the DRAM and the controller are length matched (across a group of 8 data bits and one DQS for regular DDR parts, and a group of 32 data bits and DQS for SGRAM) the data and DQS transitions are still aligned when they are received at the memory controller. The controller delays the DQS using a DLL controlled by \( \text{dq}_i\_\text{skew} \). The data is valid for half of a memory cycle, so the quarter cycle base delay (with \( M = 4'b1000 \) thus \( \text{mult} = 1 \)) will position the delayed DQS at the center of theoretical data window. The DLL can be adjusted about 8.75% of the memory cycle (or 35% of the quarter cycle) either side of this point.

The second use of the DLL is to position the address and data signals relative to the memory clock. This is done by driving the data/address/control from a fixed internal clock and using the DLL to move the external \( Mn\_\text{CLK} \) with respect to the internal one. Since it is the clock moving rather than the data the direction of a change can seem backwards! Figure 21 shows the external view with timings relative to the clock, this is the view needed when designing the memory system.

The position of the clock relative to the point at which the address and control signals change is set by the \( \text{addr}\_\text{skew} \) parameter. Figure 21 shows the range as t0 to t1. To get the address change earliest in the external clock cycle (shown by t0) the clock needs to be delayed by its maximum, so the \( \text{addr}\_\text{skew} \) (which sets the DLL \( M \) parameter from Table 70) needs to be 4'b1111. The other extreme is shown by t1, which has the \( \text{addr}\_\text{skew} \) set to 4'b0000. The range gives the delay from the rising \( Mn\_\text{CLK} \) edge to the address transition point of approximately 16% to 34% of the memory clock cycle time.

---

**Table 70: Adjustment Percentages and Multiplier for Values of DLL M (Cont.)**

<table>
<thead>
<tr>
<th>M</th>
<th>M-8</th>
<th>Adjustment%</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>4'b1000</td>
<td>0</td>
<td>center</td>
<td>1.0</td>
</tr>
<tr>
<td>4'b1001</td>
<td>1</td>
<td>5%</td>
<td>1.05</td>
</tr>
<tr>
<td>4'b1010</td>
<td>2</td>
<td>10%</td>
<td>1.10</td>
</tr>
<tr>
<td>4'b1011</td>
<td>3</td>
<td>15%</td>
<td>1.15</td>
</tr>
<tr>
<td>4'b1100</td>
<td>4</td>
<td>20%</td>
<td>1.20</td>
</tr>
<tr>
<td>4'b1101</td>
<td>5</td>
<td>25%</td>
<td>1.25</td>
</tr>
<tr>
<td>4'b1110</td>
<td>6</td>
<td>30%</td>
<td>1.30</td>
</tr>
<tr>
<td>4'b1111</td>
<td>7</td>
<td>35%</td>
<td>1.35</td>
</tr>
</tbody>
</table>

4'b1000 0 center 1.0
4'b1001 1 5% 1.05
4'b1010 2 10% 1.10
4'b1011 3 15% 1.15
4'b1100 4 20% 1.20
4'b1101 5 25% 1.25
4'b1110 6 30% 1.30
4'b1111 7 35% 1.35
The drive position of the data relative to clock (t4 and t5) is controlled by the addr_skew parameter in the same way as the address, except the data changes on both edges of the clock. The data transition point is thus offset from both edges of the Mn_CLK by about 16% to 34% of the memory clock cycle time.

The third use of a DLL is to position the DQS strobe relative to the data during a write to the memory. The DQS transition needs to be shifted to the center of the data valid window. This is done by shifting the DQS transition from the data transition using a DLL controlled by the dqo_skew. The minimum delay between the data and DQS (t6 in Figure 21) is about 16% of the cycle time when dqo_skew is 4'b0000, the maximum (shown as t7) is about 34% of the cycle time with dqo_skew set to 4'b1111.

The second and third uses of the DLL interact to set the position of the DQS output relative to the clock. The difference between the addr_skew and dqo_skew controls this delay (if they are set to the same value the strobe will line up with the clock). The early DQS shown by t2 results from the addr_skew being set to M=4'b1111 and dqo_skew to M=4'b0000, the difference in the delays will make the DQS precede the clock by (35 - (-35)) = 70% of a quarter cycle or 17.5% of the cycle time. The late DQS shown by t3 is the reverse situation and will have the DQS 17.5% of the cycle time after the clock.

![Figure 21: Timing Relationships Set by DLLs](image-url)
User Manual

10/21/02

TIMING PARAMETER GUIDELINES

The memory controller timing parameters must be configured to match the devices and board delays in the system. This section gives some guidelines for the parameters.

During a read from memory the controller will use the DQS strobe supplied with each byte of the data to latch the four data-beats. The data is subsequently read out and passed to the DBF data buffer. Since there will be skew across the nine DQS lines, the different bytes are captured from the memory at different times, but they are eventually read out of the latch as a single value. There are three factors that influence the timing. (1) The controller must not start looking for DQS until the SDRAM drives it to a valid low as part of the preamble. (2) The controller must start looking for the DQS before the earliest DQS returns. (3) The data must not be read from the latch before the byte with the latest DQS has been written and allowed to settle. Taken together these constraints fix the window in which the first DQS of each byte lane must be received at the controller. The tCrD, tCrDh and tFIFO parameters in the mc_timing1 register describe this window.

As a starting point the tCrD is the integer part of the CAS latency, the tCrDh is set for half-cycle CAS latency and the tFIFO is set to 1 for normal delay in reading out from the data latch. The full story is shown in Figure 22 which shows the window for the first DQS relative to clock for different settings of the three parameters. The count n represents the number of cycles since CAS, so for a CAS latency n device the ideal DQS would be at the rising edge that starts cycle n.

<table>
<thead>
<tr>
<th>ticks after CAS</th>
<th>n-1</th>
<th>n</th>
<th>n+1</th>
<th>n+2</th>
</tr>
</thead>
<tbody>
<tr>
<td>mclk</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n,0,1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n,1,1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n,0,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n,1,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n+1,0,1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n+1,1,1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n+1,0,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n+1,1,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n-1,0,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[n-1,1,2]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 22: Nominal Windows at 133MHz for First Edge of DQS for Various Settings of [tCrD, tCrDh, tFIFO]
This figure shows where in time the first DQS edges must occur for various settings of the parameters \( tCrD, tCrDh, tFIFO \). The following table describes the opening and closing of the windows for different parameter settings:

**Table 71: First DQS Window Opening and Closing (Typical)**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Window opening</th>
<th>Window closing</th>
</tr>
</thead>
<tbody>
<tr>
<td>([n, 0, 1])</td>
<td>1.5ns before rising edge</td>
<td>3.5ns before rising edge of cycle ( n+1 )</td>
</tr>
<tr>
<td>([n, 1, 1])</td>
<td>1.5ns before falling edge</td>
<td>3.5ns before rising edge of cycle ( n+1 )</td>
</tr>
<tr>
<td>([n, 0, 2])</td>
<td>1.5ns before rising edge</td>
<td>2.0ns before falling edge of cycle ( n+1 )</td>
</tr>
<tr>
<td>([n, 1, 2])</td>
<td>1.5ns before falling edge</td>
<td>2.5ns before falling edge of cycle ( n+1 ) plus ( \frac{1}{4} ) cycle</td>
</tr>
</tbody>
</table>

Note that both the MCLK and the DQS are observed at the pins of the controller. When a memory system is analyzed the loop delay must be considered: the clock is driven from the controller and takes time to propagate to the SDRAM, the SDRAM will drive DQS in some window around when it receives the clock edge and DQS will take time to propagate back. In a heavily loaded channel (or with lower drive strength) there may also be a contribution to the delay from the rise/fall time of the clock and DQS. The DQS signals are all being driven by different SDRAM chips and may have different propagation delays (since the board trace length difference between the 8 data bits and their DQS is more tightly controlled than the difference across the DQS bundles). In systems with DIMMs there may be a non-negligible difference in response between a device on the first DIMM and a device on the last DIMM of the channel, increasing the uncertainty in DQS arrival time.

The positions of the windows are affected by the settings of the DLLs described in the previous section. They are shown for the center DLL position \( M=4'b1000 \). Increasing the value of \( addr\_skew \) will move the windows to the left (this is not true on the initial pass1 prototypes with the earlier address DLL arrangement, in those parts increasing \( dqo\_skew \) will move the window to the left). Increasing the value of \( dqi\_skew \) will move the windows to the left (this is true on all parts).

The positions of the windows are also affected by the settings of the clock_class and clock_drive parameters in the memory clock configuration register. The windows in the figure are shown for clock_class=1 and clock_drive=7 (i.e. maximum drive strength SSTL_2 Class II). Values other than these will move the windows to the left.

If the memory system is running slower than 95MHz, it is possible that the DLLs in the memory controller will saturate (i.e., reach their maximum possible delay). When this happens, the windows will move to the right.

A few of the settings are not useful in practice. There is no reason to use the setting \( [n,1,1] \) since it is a subset of the \( [n,0,1], [n,0,2] \) and \( [n,1,2] \) settings and has a narrower window that all nine DQS signals must hit. The setting for \( [n-1,0,2] \) is also not particularly useful. It is only shown for completeness.

The setting for \( [n-1,1,2] \) is probably only useful to systems running with such a slow memory clock that the DLL's have saturated. This setting is misleading the controller about the actual CAS latency of the SDRAM. The controller will therefore expect the SDRAM to release the bus earlier, which must be compensated for by always setting r2wIdle to 1 to restore the cycle in a read to write turnaround.
As an example, consider a memory system with parts soldered to the mainboard using CAS latency 2.5 parts. The simple view would suggest [2, 1, 1] as the starting point. However referring to the figure, we note that with this setting at 133MHz, the window has closed before the earliest point where we might expect DQS to return. As a general rule, systems using CAS latency 2.5 should use either [2,0,2] or [2,1,2]. Which of these settings is better depends on the memory system round trip delay, the DLL settings, and the setting for memory clock drive strength.

Continuing the example, suppose we run the memory at CAS latency 2. In this case the expected setting, [2,0,1], is generally the correct setting. But now suppose that for some reason the memory clock needs to be run slow (below 95MHz). All of the windows in the figure will move right. And in the lightly loaded system the DQS arrival will be relatively early. The width of the [2,0,1] window will continue to work for some time because it can move for a reasonable distance until the opening point reaches the early DQS (which the DDR spec would allow to be before the clock edge). At some frequency the first DQS will not be correctly used to strobe the data and the data received later in the burst will overwrite the earlier (the memory controller reads into the low numbered bits of the ZBus first, so when this happens the data in the low memory address end of a cache line will be UNPREDICTABLE and the data at the high address will have the value expected from the low address). This is where the memory controller needs to be programmed with a smaller CAS latency (and r2wIdle set to 1) to use the right shifted [1,1,2] window.

Most systems will likely want to use one of the following settings:

- [n,0,1] for whole cycle CAS latencies and relatively short board delays.
- [n,0,2] for whole cycle CAS latencies with moderately long board delays.
- [n,1,2] for half-cycle CAS latencies. (Due to a performance bug systems with half-cycle CAS latencies using BCM1250 pass 1 prototype parts should use [n,0,2] if possible).

There are three additional parameters r2wIdle, w2rIdle, and r2rIdle that address system timing concerns, specifically bus conflicts caused by the time delay between signals driven by the controller and signals driven by the SDRAMs.

The r2wIdle parameter sets the turnaround time from the memory driving the databus (on a read) to the controller driving the databus (for a subsequent write). It is independent of whether the accesses are to the same physical bank or different physical banks. Assuming a whole cycle CAS latency for the SDRAM, a setting of 0 for the r2wIdle parameter nominally provides 0.75 cycles of non-overlap time for the DQ lines. But at the controller, this non-overlap time is reduced by the clock delay from the controller to the SDRAM and the DQ delay from the SDRAM to the controller. Further, if the SDRAM CAS latency is set to some half-cycle value, the non-overlap time is reduced by another 0.5 cycle. So systems using SDRAMs with a half-cycle CAS latency value almost certainly need to set the r2wIdle parameter to a 1, increasing the non-overlap time by one cycle. Systems using whole-cycle CAS latency values may need to set this parameter to a 1 if the round-trip delays between the controller and the SDRAMs are significant compared to the clock period.

The w2rIdle parameter sets the turnaround time for the controller driving the databus to the memory driving the databus when physical banks are changed. It should always be set to 1.

The r2rIdle parameter sets the turnaround time between a read from one physical bank to a read from a different physical bank.
A value of 0 for the r2rIdle parameter nominally provides 1 cycle of non-overlap for the DQ lines at the physical bank of SDRAMs being read first. At the physical bank that is the target of the second read, this nominal value is reduced by the DQ delay between the two physical banks. The DQS non-overlap time is nominally 0, but since both physical banks are driving the DQS lines low during the turnaround, there is nominally 0.5 cycle of margin before a conflict occurs on the DQS lines at the first physical bank. Once again, at the second physical bank of SDRAMs, this 0.5 cycle of nominal no conflict time is reduced by the DQS delay between the two physical banks. Therefore it is unlikely that a system will require a non-zero value of the r2rIdle parameter.

To summarize, most systems using half-cycle CAS latency will set r2wIdle to a 1, w2rIdle to a 1, and r2rIdle to a 0. Systems using SDRAMs with a whole-cycle CAS latency will set r2rIdle to 1 and the other parameters to 0. Systems configuring an n-1 value for tCrD will need to set r2wIdle to 1.

**PERFORMANCE MONITORING FEATURES**

The memory controller provides signals to the performance counters in the SCD (see Section: “System Performance Counters” on page 61).

Add discussion.

**ZBBUS MONITORING**

The functionality described in this section in previous releases of the BCM1250 manual was for the prototype chips only and is no longer available. The trace buffer in the SCD (See “Trace Unit” on page 70) should be used to monitor the ZBBus.
**CONFIGURATION REGISTERS**

All memory controller configuration registers are 64 bit wide and they must be written using double-word accesses. If writes smaller than 64 bits are done to a control register its contents become UNPREDICTABLE. 

*On the BCM1125/H, all registers of memory channel 0, except mc_config_0 are Not Implemented.*

Table 72: Memory Channel Configuration Register on BCM1250

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>15:8</td>
<td>channel_sel</td>
<td>8'h0</td>
<td>If this field is zero the two channels are not interleaved. If this field is non zero it specifies the address bit that selects between the two channels, and must have a value in the range 5-35. If the channels are interleaved the other configuration parameters must match.</td>
</tr>
<tr>
<td>19:16</td>
<td>bank0_map</td>
<td>4'h0</td>
<td>Value of physical address bits [31:28] that should map to 0 (1st 256MB block)</td>
</tr>
<tr>
<td>23:20</td>
<td>bank1_map</td>
<td>4'h8</td>
<td>Value of physical address bits [31:28] that should map to 1 (2nd 256MB block)</td>
</tr>
<tr>
<td>27:24</td>
<td>bank2_map</td>
<td>4'h9</td>
<td>Value of physical address bits [31:28] that should map to 2 (3rd 256MB block)</td>
</tr>
<tr>
<td>31:28</td>
<td>bank3_map</td>
<td>4'hC</td>
<td>Value of physical address bits [31:28] that should map to 3 (4th 256MB block)</td>
</tr>
<tr>
<td>39:32</td>
<td>probe_mode</td>
<td>8'b0</td>
<td>Reserved, Broadcom Use Only. Setting these bits to any value other than zero will result in UNDEFINED behavior and can cause the ECC lines to be continually driven regardless of the direction of the data transfer.</td>
</tr>
<tr>
<td>43:40</td>
<td>iob1_qsize</td>
<td>4'ha</td>
<td>These fields are used to set the range of queue entries reserved for use by I/O Bridge 1. The two channel configuration registers set a range to give hysteresis to the blocking. mc_config_0: Queue size to start blocking agents other than IOB1. mc_config_1: Queue size to stop blocking agents other than IOB1.</td>
</tr>
<tr>
<td>47:44</td>
<td>age_limit</td>
<td>4'h8</td>
<td>Maximum number of younger reads that can pass a read in the request queue, before the read is serviced. See Section: “Memory Controller Architecture” on page 104.</td>
</tr>
<tr>
<td>51:48</td>
<td>wr_limit</td>
<td>4'h5</td>
<td>Maximum number of writes in a burst to memory before reads will be serviced. See Section: “Memory Controller Architecture” on page 104.</td>
</tr>
<tr>
<td>52</td>
<td>iob1_priority</td>
<td>1'b1</td>
<td>mc_config_1: If this bit is set reads from I/O bridge 1 will be given high priority (to reduce their latency). If clear all requests have the same priority. mc_config_0: Reserved.</td>
</tr>
<tr>
<td>55:53</td>
<td>reserved</td>
<td>3'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>59:56</td>
<td>cs_mode</td>
<td>4'h0</td>
<td>Chip Selection mode 0000: msb-CS mode: CS[3:0]are determined by the corresponding CS start/end+1 address registers. 1111: Interleaved-CS mode: the values of start/end+1 addresses of CS[3:0] are all same in this mode, CS[3:0] are decoded by the two bits in the interleaved CS position register. 1100: Mixed-CS mode: CS[1] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS3 and CS2, determined by one bit of the interleaved CS position register. The values of the CS[3:2] start/end+1 addresses are the same. 0110: Mixed-CS mode: CS[3] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS2 and CS1, determined by one bit of the interleaved CS position register. The values of the CS[2:1] start/end+1 addresses are the same. 0011: Mixed-CS mode: CS[3] and CS[2] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS1 and CS0, determined by one bit of the interleaved CS position register. The values of the CS[1:0] start/end+1 addresses are the same.</td>
</tr>
</tbody>
</table>
### Table 72: Memory Channel Configuration Register on BCM1250 (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>ecc_disable</td>
<td>1'b0</td>
<td>ECC disable. If this bit is set ECC checking will not be performed and data will never be reported to have an error.</td>
</tr>
<tr>
<td>61</td>
<td>berr_disable</td>
<td>1'b0</td>
<td>Bus error disable. This bit sets the behavior when a read is done to an address that does not match any chip select range. 0: Generate a bus error data return 1: Generate a valid data return containing UNPREDICTABLE data. On the BCM1250 if this bit is set in either channel, bus errors will be disabled for all reads.</td>
</tr>
<tr>
<td>62</td>
<td>force_seq</td>
<td>1'b0</td>
<td>Force Sequential. If this bit is set requests will be issued to memory in the same order their A-phase happens on the ZBbus.</td>
</tr>
<tr>
<td>63</td>
<td>debug</td>
<td>1'b0</td>
<td>This bit enables debug functions in the controller. It should be clear during normal operation. The action if the bit is set is different for the two channels: Channel 0: Reserved Channel 1: Broadcom Use Only (was Enable ZBbus monitoring mode).</td>
</tr>
</tbody>
</table>

### Table 73: Memory Channel Configuration Register on BCM1250/H

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>reserved</td>
<td>16'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>19:16</td>
<td>bank0_map</td>
<td>4'h0</td>
<td>mc_config_1: Value of physical address bits [31:28] that should map to 0 (1st 256MB block) mc_config_0: Reserved.</td>
</tr>
<tr>
<td>23:20</td>
<td>bank1_map</td>
<td>4'h8</td>
<td>mc_config_1: Value of physical address bits [31:28] that should map to 1 (2nd 256MB block) mc_config_0: Reserved.</td>
</tr>
<tr>
<td>27:24</td>
<td>bank2_map</td>
<td>4'h9</td>
<td>mc_config_1: Value of physical address bits [31:28] that should map to 2 (3rd 256MB block) mc_config_0: Reserved.</td>
</tr>
<tr>
<td>31:28</td>
<td>bank3_map</td>
<td>4'hC</td>
<td>mc_config_1: Value of physical address bits [31:28] that should map to 3 (4th 256MB block) mc_config_0: Reserved.</td>
</tr>
<tr>
<td>39:32</td>
<td>probe_mode</td>
<td>8'b0</td>
<td>mc_config_1: Reserved, Broadcom Use Only. Setting these bits to any value other than zero will result in UNDEFINED behavior and can cause the ECC lines to be continually driven regardless of the direction of the data transfer. mc_config_0: Reserved.</td>
</tr>
<tr>
<td>43:40</td>
<td>iob1_qsize</td>
<td>4'ha</td>
<td>These fields are used to set the range of queue entries reserved for use by I/O Bridge 1. The two channel configuration registers set a range to give hysteresis to the blocking. mc_config_0: Queue size to start blocking agents other than IOB1. mc_config_1: Queue size to stop blocking agents other than IOB1.</td>
</tr>
<tr>
<td>47:44</td>
<td>age_limit</td>
<td>4'h8</td>
<td>mc_config_1: Maximum number of younger reads that can pass a read in the request queue, before the read is serviced. See Section: “Memory Controller Architecture” on page 104. mc_config_0: Reserved.</td>
</tr>
<tr>
<td>51:48</td>
<td>wr_limit</td>
<td>4'h5</td>
<td>mc_config_1: Maximum number of writes in a burst to memory before reads will be serviced. See Section: “Memory Controller Architecture” on page 104. mc_config_0: Reserved.</td>
</tr>
</tbody>
</table>

---

Broadcom Corporation
### Table 73: Memory Channel Configuration Register on BCM1125/H (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>iob1_priority</td>
<td>1'b1</td>
<td><strong>mc_config_1</strong>: If this bit is set reads from I/O bridge 1 will be given high priority (to reduce their latency). If clear all requests have the same priority. <strong>mc_config_0</strong>: Reserved.</td>
</tr>
<tr>
<td>55:53</td>
<td>reserved</td>
<td>3'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
| 59:56 | cs_mode          | 4'h0    | **mc_config_1**: Chip Selection mode  
0000: msb-CS mode: CS[3:0] are determined by the corresponding CS start/end+1 address registers.  
1111: Interleaved-CS mode: the values of start/end+1 addresses of CS[3:0] are all same in this mode, CS[3:0] are decoded by the two bits in the interleaved CS position register.  
1100: Mixed-CS mode: CS[1] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS3 and CS2, determined by one bit of the interleaved CS position register. The values of the CS[3:2] start/end+1 addresses are the same.  
0110: Mixed-CS mode: CS[3] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS2 and CS1, determined by one bit of the interleaved CS position register. The values of the CS[2:1] start/end+1 addresses are the same.  
0011: Mixed-CS mode: CS[3] and CS[2] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS1 and CS0, determined by one bit of the interleaved CS position register. The values of the CS[1:0] start/end+1 addresses are the same.  
**mc_config_0**: Reserved. |
| 60    | ecc_disable      | 1'b0    | **mc_config_1**: ECC disable. If this bit is set ECC checking will not be performed and data will never be reported to have an error. **mc_config_0**: Reserved. |
| 61    | berr_disable     | 1'b0    | **mc_config_1**: Bus error disable. This bit sets the behavior when a read is done to an address that does not match any chip select range.  
0: Generate a bus error data return  
1: Generate a valid data return containing UNPREDICTABLE data.  
**mc_config_0**: Reserved. |
| 62    | force_seq        | 1'b0    | **mc_config_1**: Force Sequential. If this bit is set requests will be issued to memory in the same order their A-phase happens on the ZBbus.  
**mc_config_0**: Reserved. |
| 63    | reserved         | 1'b0    | Reserved                                                                                                                                                                                                   |
### Table 74: Memory Clock Configuration Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>clk_ratio</td>
<td>4'h4</td>
<td>ZBclk/mclk ratio, Mclk when ZBclk=400 MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00_1005</td>
<td>2x 200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100_2x</td>
<td>1.5x 160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0101_2.5x</td>
<td>1.33 144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110_3x</td>
<td>1.33 114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0111_3.5x</td>
<td>1.14 100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000_4x</td>
<td>1.14 88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001_4.5x</td>
<td>0.88 80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Others</td>
<td>Reserved</td>
</tr>
<tr>
<td>7:4</td>
<td>cs_absence</td>
<td>4'h0</td>
<td>On parts with system_revision ≥ PERIPH_REV3 setting these bits will disable the refresh cycle for the corresponding chip select. On earlier revisions all chip selects will always be refreshed.</td>
</tr>
<tr>
<td>15:8</td>
<td>ref_rate</td>
<td>8'hC4</td>
<td>Refresh Rate: (value+1) x 16 MCLK examples are: 8'h62 99(x16) MCLK cycles at 100MHz (15.84us). 8'h81 130(x16) MCLK cycles at 133MHz (15.64us). 8'hc4 197(x16) MCLK cycles at 200MHz (15.76us). Note that the controller will refresh one chip select at a time (to save the power spike of all the SDRAMs refreshing simultaneously) so refresh cycles will be seen on the channel four times as often as this field implies (in large memory mode two chip selects are refreshed at a time).</td>
</tr>
<tr>
<td>18:16</td>
<td>clock_drive</td>
<td>3'b111</td>
<td>This sets the drive strength (and therefore slew rate) of the output drivers for the clocks. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>19</td>
<td>clock_class</td>
<td>1'b1</td>
<td>This bit should be clear to have the clock drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.</td>
</tr>
<tr>
<td>22:20</td>
<td>data_drive</td>
<td>3'b0</td>
<td>This sets the drive strength (and therefore slew rate) of the output drivers for the data lines. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>23</td>
<td>data_class</td>
<td>1'b0</td>
<td>This bit should be clear to have the data drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.</td>
</tr>
<tr>
<td>26:24</td>
<td>addr_drive</td>
<td>3'b0</td>
<td>This sets the drive strength (and therefore slew rate) of the output drivers for the address and control lines. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>27</td>
<td>addr_class</td>
<td>1'b0</td>
<td>This bit should be clear to have the address and control drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.</td>
</tr>
<tr>
<td>29:28</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>ref_disable</td>
<td>1'b0</td>
<td>On BCM1250 prior to PERIPH_REV3: Reserved On other parts: If this bit is set refresh cycles are disabled. This can be useful during debugging, but must not be set in normal operation.</td>
</tr>
<tr>
<td>31</td>
<td>dll_bypass</td>
<td>1'b0</td>
<td>Set this bit to bypass the DLLs. This should not be set in normal operation.</td>
</tr>
<tr>
<td>35:32</td>
<td>dqi_skew</td>
<td>4'b1000</td>
<td>DQS to data input skew control. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>39:36</td>
<td>reserved</td>
<td>4'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>43:40</td>
<td>dqs_skew</td>
<td>4'b1000</td>
<td>DQS to data output skew control. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>47:44</td>
<td>reserved</td>
<td>4'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>51:48</td>
<td>addr_skew</td>
<td>4'b1000</td>
<td>Address/control output delay from rising memory clock. See Section: &quot;I/O Control&quot; on page 128.</td>
</tr>
<tr>
<td>55:52</td>
<td>reserved</td>
<td>4'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 74: Memory Clock Configuration Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>61:56</td>
<td>dll_default</td>
<td>6'b010000</td>
<td>This is the value to use in place of the DLL output when DLL bypass is enabled.</td>
</tr>
<tr>
<td>63:62</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 75: DRAM Command Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>command</td>
<td>Command to issue to the SDRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000: EMRS Write the extended mode register (EMR) (BA[1:0] = 01)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001: MRS Write the mode register (MR) (BA[1:0] = 00)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0010: PREcharge all banks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0011: Auto Refresh (AR)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100: Set self-refresh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0101: Clear self-refresh</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110: Set power-down</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0111: Clear power-down</td>
</tr>
<tr>
<td>4</td>
<td>cs0</td>
<td>If this bit is set, cs0 is asserted when the command is issued.</td>
</tr>
<tr>
<td>5</td>
<td>cs1</td>
<td>If this bit is set, cs1 is asserted when the command is issued.</td>
</tr>
<tr>
<td>6</td>
<td>cs2</td>
<td>If this bit is set, cs2 is asserted when the command is issued.</td>
</tr>
<tr>
<td>7</td>
<td>cs3</td>
<td>If this bit is set, cs3 is asserted when the command is issued.</td>
</tr>
<tr>
<td>63:8</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 76: DRAM Mode Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:0</td>
<td>emode</td>
<td>15'b0</td>
<td>This value is written to the Extended MODE register via the address lines when the EMRS command is issued to the mc_dramcmd register.</td>
</tr>
<tr>
<td>14:13</td>
<td></td>
<td></td>
<td>These bits provide the additional two bits to send on address bits 14:13 during an EMRS command to FCRAMs. They are ignored for standard DDR SDRAMs.</td>
</tr>
<tr>
<td>15</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>28:16</td>
<td>mode</td>
<td>15'h22</td>
<td>This value is written to the MODE register via the address lines when the MRS command is issued to the mc_dramcmd register. The mode register must always be programmed for sequential burst order and a burst length of 4. If not memory accesses will be UNPREDICTABLE.</td>
</tr>
<tr>
<td>30:29</td>
<td></td>
<td></td>
<td>These bits provide the additional two bits to send on address bits 14:13 during an MRS command to FCRAMs. They are ignored for standard DDR SDRAMs.</td>
</tr>
<tr>
<td>31</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 76: DRAM Mode Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34:32</td>
<td>dram_type</td>
<td>3'b0</td>
<td>DRAM type: 000: JEDEC DDR (one DQS for every 8 DQ bits) 001: FCRAM 010: DDR SGRAM (one DQS for every 32 DQ bits) Others Reserved</td>
</tr>
<tr>
<td>35</td>
<td>large_mem</td>
<td>1'b0</td>
<td>If this bit is set large memory support (using an external chip select decoder) is enabled. See Section: “Larger Memory Systems” on page 124.</td>
</tr>
<tr>
<td>36</td>
<td>pre_on_A8</td>
<td>1'b0</td>
<td>If this bit is set the AutoPrecharge flag will be output on the A8 address bit rather than A10. (Only supported on parts with system revision PERIPH_REV3 or greater).</td>
</tr>
<tr>
<td>37</td>
<td>ram_with_A13</td>
<td>1'b0</td>
<td>This bit should be set when the large_mem bit is set if the row address uses A13. It causes the CS[3:2] to be driven from bits [15:14] of the row address mask. It also needs to be set to cause BA[2] to be output on A13 rather than A12 for non-standard 8 bank DRAMs. (Only supported on parts with system revision PERIPH_REV3 or greater).</td>
</tr>
<tr>
<td>63:38</td>
<td>reserved</td>
<td>26'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 77: SDRAM Timing Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Range</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>tRCD</td>
<td>1-6</td>
<td>4'h3</td>
<td>RAS to CAS delay. This should be set to 1 for FCRAMs.</td>
</tr>
<tr>
<td>6:4</td>
<td>tCrD</td>
<td>1-6</td>
<td>3'h2</td>
<td>Cycles from CAS(read) to data (the CAS latency).</td>
</tr>
<tr>
<td>7</td>
<td>tCrDh</td>
<td>0-1</td>
<td>1'b0</td>
<td>If this bit is set the data capture is delayed by a half cycle from the integer part of the CAS latency set in tCrD. See the discussion in “Timing Parameter Guidelines” on page 131.</td>
</tr>
<tr>
<td>11:8</td>
<td>tCwD</td>
<td>1-2 or 1-4</td>
<td>4'h1</td>
<td>Delay from CAS to Write data. On parts with system revision PERIPH_REV3 or greater the range is extended to 1-4.</td>
</tr>
<tr>
<td>15:12</td>
<td>reserved</td>
<td>0</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>19:16</td>
<td>tRP</td>
<td>1-4</td>
<td>4'h4</td>
<td>Cycles required from PRECHARGE to RAS.</td>
</tr>
<tr>
<td>23:20</td>
<td>tRRD</td>
<td>1-4</td>
<td>4'h2</td>
<td>Cycles required from RAS to RAS for a different bank.</td>
</tr>
<tr>
<td>27:24</td>
<td>tRCw-1</td>
<td>1-15</td>
<td>4'ha</td>
<td>One less than cycles required from RAS(write) to next RAS of the same bank.</td>
</tr>
<tr>
<td>31:28</td>
<td>tRCr-1</td>
<td>1-15</td>
<td>4'h9</td>
<td>One less than cycles required from RAS(read) to next RAS of the same bank.</td>
</tr>
<tr>
<td>35:32</td>
<td>reserved</td>
<td>0</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>39:36</td>
<td>reserved</td>
<td>0</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>43:40</td>
<td>tCwCr</td>
<td>0-15</td>
<td>4'h4</td>
<td>Cycles of separation from write to read. This field is calculated as: tCwCr = tRCw - (tRCD + tCwD + 2 + tWTR) Note that tRCw is one greater than the value set in tRCw-1 field and tWTR is the internal write to read delay for the SDRAM (normally specified as 1 cycle).</td>
</tr>
<tr>
<td>47:44</td>
<td>reserved</td>
<td>0</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>51:48</td>
<td>reserved</td>
<td>0</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>55:52</td>
<td>tRFC</td>
<td>0-15</td>
<td>4'hc</td>
<td>Cycles required from Auto-refresh to Active or Auto-refresh.</td>
</tr>
<tr>
<td>59:56</td>
<td>tFIFO</td>
<td>0-1</td>
<td>4'h1</td>
<td>Additional cycles required for data capture FIFO.</td>
</tr>
</tbody>
</table>
Table 77: SDRAM Timing Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Range</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>w2rIdle</td>
<td>0-1</td>
<td>1'b1</td>
<td>Number of idle cycles required for write to read turnaround. This bit must always be set.</td>
</tr>
<tr>
<td>61</td>
<td>r2wIdle</td>
<td>0-1</td>
<td>1'b1</td>
<td>Number of idle cycles required for read to write turnaround. If this bit is 0 one idle cycle is used, if this bit is 1 two idle cycles are used. Usually set for half cycle CAS latencies and cleared otherwise.</td>
</tr>
<tr>
<td>62</td>
<td>r2rIdle</td>
<td>0-1</td>
<td>1'b1</td>
<td>Number of idle cycles required for read to read of a different chip turnaround. If this bit is 0 one idle cycle is used, if this bit is 1 two idle cycles are used. This bit is usually 0.</td>
</tr>
<tr>
<td>63</td>
<td>reserved</td>
<td>0</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 78: SDRAM Timing Register 2

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default Ch0</th>
<th>Default Ch1</th>
<th>Default Ch1 (BCM1125/H)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>reserved</td>
<td></td>
<td></td>
<td></td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 79: Chip Select Start Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default Ch0</th>
<th>Default Ch1</th>
<th>Default Ch1 (BCM1125/H)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>cs0_start</td>
<td>16'h0000</td>
<td>16'h0100</td>
<td>16'h0000</td>
<td>Chip select 0,1,2,3 region start address bits [39:24]. These addresses are in the memory address space (after the translation described in Section: &quot;Mapping&quot; on page 109).</td>
</tr>
<tr>
<td>31:16</td>
<td>cs1_start</td>
<td>16'h0010</td>
<td>16'h0000</td>
<td>16'h0010</td>
<td></td>
</tr>
<tr>
<td>47:32</td>
<td>cs2_start</td>
<td>16'h0020</td>
<td>16'h0000</td>
<td>16'h0020</td>
<td></td>
</tr>
<tr>
<td>63:48</td>
<td>cs3_start</td>
<td>16'h0030</td>
<td>16'h0000</td>
<td>16'h0030</td>
<td></td>
</tr>
</tbody>
</table>

Table 80: Chip Select End Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default Ch0</th>
<th>Default Ch1</th>
<th>Default Ch1 (BCM1125/H)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>cs0_end</td>
<td>16'h0010</td>
<td>16'h0200</td>
<td>16'h0010</td>
<td>Chip Select 0,1,2,3 region end address + 1 bits [39:24]. These addresses are in the memory address space (after the translation described in Section: &quot;Mapping&quot; on page 109).</td>
</tr>
<tr>
<td>31:16</td>
<td>cs1_end</td>
<td>16'h0020</td>
<td>16'h0000</td>
<td>16'h0020</td>
<td></td>
</tr>
<tr>
<td>47:32</td>
<td>cs2_end</td>
<td>16'h0030</td>
<td>16'h0000</td>
<td>16'h0030</td>
<td></td>
</tr>
<tr>
<td>63:48</td>
<td>cs3_end</td>
<td>16'h0040</td>
<td>16'h0000</td>
<td>16'h0040</td>
<td></td>
</tr>
</tbody>
</table>
### Table 81: Chip Select Interleave Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>reserved</td>
<td>5'b0</td>
<td></td>
</tr>
<tr>
<td>24:5</td>
<td>interleave</td>
<td>20'b0</td>
<td>For mixed_cs mode single set bit should be written to this register in the bit position that should be used to select between the two chip selects in the interleaved portion. For interleaved_cs mode two adjacent bits should be set in this register in the bit positions that should be used to select between the four chip selects.</td>
</tr>
<tr>
<td>63:25</td>
<td>reserved</td>
<td>39'b0</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

### Table 82: Row Address Bits Select Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:0</td>
<td>reserved</td>
<td>Address bits are selected for use as the row address by setting the corresponding bits in this register. The number of bits set should match the number of rows of the SDRAM. The set bits must be contiguous.</td>
</tr>
<tr>
<td>34:10</td>
<td>select</td>
<td></td>
</tr>
<tr>
<td>63:35</td>
<td>reserved</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

### Table 83: Column Address Bits Select Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>reserved</td>
<td>Reserved (The controller always acts as if bits 4:3 are set.)</td>
</tr>
</tbody>
</table>
| 20:5 | select            | Address bits are selected for use as the column address by setting the corresponding bits in this register. The number of bits set should match the number of columns of the SDRAM. The set bits must be contiguous, with two exceptions:  
  - Bit 5 set and the other bits contiguous (this allows for bank, CS or channel interleave on 64 byte blocks)  
  - Bit 5 and 6 set and the other bits contiguous (this allows for bank, CS or channel interleave on 128 byte blocks) |
### Table 83: Column Address Bits Select Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_cs0_col_0</td>
<td>00_1005_1220</td>
<td></td>
</tr>
<tr>
<td>mc_cs1_col_0</td>
<td>00_1005_1280</td>
<td></td>
</tr>
<tr>
<td>mc_cs2_col_0</td>
<td>00_1005_12E0</td>
<td></td>
</tr>
<tr>
<td>mc_cs3_col_0</td>
<td>00_1005_1340</td>
<td></td>
</tr>
<tr>
<td>mc_cs0_col_1</td>
<td>00_1005_2220</td>
<td></td>
</tr>
<tr>
<td>mc_cs1_col_1</td>
<td>00_1005_2280</td>
<td></td>
</tr>
<tr>
<td>mc_cs2_col_1</td>
<td>00_1005_2340</td>
<td></td>
</tr>
<tr>
<td>mc_cs3_col_1</td>
<td>00_1005_23E0</td>
<td></td>
</tr>
</tbody>
</table>

#### Default:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:21</td>
<td>reserved</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

### Table 84: Bank Address Bits Select Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_cs0_ba_0</td>
<td>00_1005_1240</td>
<td></td>
</tr>
<tr>
<td>mc_cs1_ba_0</td>
<td>00_1005_12A0</td>
<td></td>
</tr>
<tr>
<td>mc_cs2_ba_0</td>
<td>00_1005_1300</td>
<td></td>
</tr>
<tr>
<td>mc_cs3_ba_0</td>
<td>00_1005_1360</td>
<td></td>
</tr>
<tr>
<td>mc_cs0_ba_1</td>
<td>00_1005_2240</td>
<td></td>
</tr>
<tr>
<td>mc_cs1_ba_1</td>
<td>00_1005_22A0</td>
<td></td>
</tr>
<tr>
<td>mc_cs2_ba_1</td>
<td>00_1005_2300</td>
<td></td>
</tr>
<tr>
<td>mc_cs3_ba_1</td>
<td>00_1005_2360</td>
<td></td>
</tr>
</tbody>
</table>

#### Default:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>36:5</td>
<td>select</td>
<td>Address bits are selected for use as the bank address by setting the corresponding bits in this register. The number of bits set should match the number of banks of the SDRAM. The set bits must be contiguous.</td>
</tr>
<tr>
<td>63:37</td>
<td>reserved</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

### Table 85: Chip Select Attribute Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_cs_attr_0</td>
<td>00_1005_1380</td>
<td>2'b01</td>
<td>Page policy for Chip Select 0. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
<tr>
<td>mc_cs_attr_1</td>
<td>00_1005_2380</td>
<td>2'b01</td>
<td>Page policy for Chip Select 1. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
</tbody>
</table>

#### Default:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>cs0_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 0. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
<tr>
<td>15:2</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>17:16</td>
<td>cs1_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 1. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
<tr>
<td>31:18</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>33:32</td>
<td>cs2_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 2. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
</tbody>
</table>

#### Default:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>cs0_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 0. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
<tr>
<td>15:2</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>17:16</td>
<td>cs1_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 1. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
<tr>
<td>31:18</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>33:32</td>
<td>cs2_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 2. (See Section: &quot;Page Policy&quot; on page 122.)</td>
</tr>
</tbody>
</table>
Table 85: Chip Select Attribute Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47:34</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
<tr>
<td>49:48</td>
<td>cs3_page</td>
<td>2'b01</td>
<td>Page policy for Chip Select 3. (See Section: “Page Policy” on page 122.) 00: Closed Page. AutoPrecharge every CAS. 01: CAS time check, AutoPrecharge unless there is another request in the queue. 10: Hint Based check, as CAS time check but including hint signal. 11: Open Page. The page is always left open.</td>
</tr>
<tr>
<td>63:50</td>
<td>reserved</td>
<td>14'h0</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>

Table 86: ECC Test Data Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>invert</td>
<td>64'h0</td>
<td>This value is XORed with the data written to memory. Any bits set in this register will cause the corresponding data bit to be inverted compared to the data the ECC bits were calculated for. If only one bit is set a correctable ECC error should result from a read. If two bits are set an uncorrectable ECC error should result from a read.</td>
</tr>
</tbody>
</table>

Table 87: ECC Test ECC Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>invert</td>
<td>8'h0</td>
<td>This value is XORed with the ecc code written to memory. Any bits set in this register will cause the corresponding ecc bit to be inverted compared to value that was calculated from the data. If only one bit is set a correctable ECC error should result from a read. If two bits are set an uncorrectable ECC error should result from a read.</td>
</tr>
<tr>
<td>63:8</td>
<td>reserved</td>
<td>56'h0</td>
<td>Must be zero.</td>
</tr>
</tbody>
</table>
This Page is left blank for notes
Section 7: DMA

DMA CONTROLLERS

There are several DMA controllers. The programming interface is similar for all of them and is described in this section.

Each synchronous serial port has a transmit and a receive DMA channel and each ethernet interface has two transmit and two receive DMA channels. A DMA channel has a group of descriptors which may either be organized as a ring (in which case each descriptor can point to two memory buffers) or a chain (in this case there is only one memory buffer associated with a descriptor and the other pointer is the link pointer to the next descriptor in the chain). Each descriptor contains some control bits that are used to select interface options, and following packet transfer the DMA controller will update some flag bits.

The configuration registers for the ethernet and serial interfaces are the same. The descriptions in this section use generic names. The actual register names for the ethernet interfaces are generated by appending _mac_N_rx_ch_M for receive channel M (0,1) of ethernet interface N (0,1 for the BCM1125/H and 0,1,2 for the BCM1250), and _mac_N_tx_ch_M for the transmit channels. The register names for the serial interfaces are generated by appending _ser_N_rx and _ser_N_tx for serial interface N (0,1).

There is an additional generic Data Mover for doing transfers between arbitrary addresses (both memory and I/O can be addressed). This is similar in style to the other DMA controllers, but has a slightly different programming interface since it needs to support both source and destination addresses. The general discussion in the following sections applies to the Data Mover, specific details are given in Section: “Data Mover” on page 176.

DATA BUFFERS AND DESCRIPTORS

There are two fundamental elements in the DMA system: data buffers and descriptors. Data buffers are sized areas of the physical address space that hold data. Descriptors point to data buffers and hold their parameters. Receive DMA controllers move bytes between an interface and a data buffer, transmit DMA controllers move bytes between a data buffer and an interface, and Data Mover DMA controllers move bytes between two data buffers.

In many systems a data buffer resides in the main memory. However, there is no need for this to be the case, the address generated by the DMA controller is interpreted in the same way that a CPU physical address would be, thus the controller in the ethernet interface could transfer packets directly to a device on the HyperTransport fabric without the data going through memory (for example if incoming packets are encrypted and must be passed through a decryption engine before the CPU can use them).
Figure 23 shows a data buffer as used by the ethernet and serial interfaces (Data Mover buffers are the same but the alignment restrictions are removed). Three parameters describe a data buffer:

**Table 88: Data Buffer Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start address</td>
<td>The start address of a data buffer is the 40 bit physical address at which the buffer starts. In the ethernet and serial port DMA engines it must be cache block aligned, so the lowest five bits of the address must be zero. The Data Mover buffers can have arbitrary alignment. The Ethernet MAC DMA engines in parts where system_revision indicates PERIPH_REV3, or later, have an additional mode that allows arbitrary alignment.</td>
</tr>
<tr>
<td>Size</td>
<td>The size of the data buffer is given in cache blocks, i.e. the buffer must be a multiple of 32 bytes in size. Prior to PERIPH_REV3, if a transmit channel is configured in two block fetch mode (by setting the tbx_en bit in the dma_config0 register) the buffer size must be a multiple of 64 bytes The maximum size buffer is 512 blocks (16 KBytes). The Data Mover does not use the size field, the buffer length matches the length parameter described below.</td>
</tr>
<tr>
<td>Owner</td>
<td>The data buffer is either owned by one of the DMA controllers or by the system software. Ownership is not explicit, each DMA controller has a count of the number of descriptors it owns and will use all buffers pointed to from those descriptors. Software needs to keep track of the buffers and must update the count when passing ownership. (When software writes to the counter the value written will be added to the existing count, the software does not need to do a read-modify-write operation).</td>
</tr>
</tbody>
</table>

There are an additional two parameters that describe the data in buffers:

**Table 89: Data Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>This is the length in bytes of the valid packet data in the buffer. The length may be greater than the current buffer size, in which case the packet spans multiple buffers. If the packet spans multiple descriptors then the length field in the first descriptor is used, and the length given in the subsequent descriptors is ignored. The maximum packet size for interface DMA engines is 16 KB-1, the maximum size for the Data Mover is 1 MB.</td>
</tr>
<tr>
<td>Offset</td>
<td>This is the offset in bytes of the start of the data. The offset can be between 0 and 31 bytes. If a packet spans multiple buffers, the offset is only applied to the first buffer. The offset can be used to align the header of the packet more conveniently. For example a packet with a 14 byte ethernet header could be started with an offset of 2 bytes to align the IP header to a 64 bit doubleword boundary. Alternatively the offset may be used to easily add or strip an encapsulation header.</td>
</tr>
</tbody>
</table>
The DMA descriptor contains these parameters for each data buffer. A descriptor consists of two 64 bit double-words, dscr_a and dscr_b, that are stored in memory at consecutive locations aligned to a 16 byte boundary. Figure 24 shows a descriptor. It can point to up to two buffers. The offset is only used at the start of a packet and normally applies to buffer A (since a new packet will always start a new descriptor). However, if the offset_b bit is set the offset is applied to the first B buffer in a packet (no offset is applied on buffer A so it will completely fill before the offset is applied at the start of buffer B). In addition to the fields describing the data the descriptor contains per-packet options that are passed to the interface, and status flags that can be written back into the descriptor by the interface. The options and status for the ethernet descriptors are summarized in Section: “Option and Flag Bits for Ethernet MACs” on page 171, those for the serial interface are summarized in Section: “Control and Flag Bits for Synchronous Serial Interface” on page 174. The DMA engines access the descriptors in cacheable-coherent memory, they must also be mapped this way in the CPU.

The descriptor also has three control bits. If the INT bit is set then the DMA channel will raise an interrupt when it has completed processing of the descriptor. The second control bit indicates if the buffer B details are valid, if this bit is clear the descriptor only has one associated buffer and the buffer B details are ignored (they are preserved when the status information is written back in to the descriptor, so could be used by software to store state information). The third control bit allows the offset field to be applied to the B buffer rather than the A buffer.

Figure 25 on page 150 shows a packet spanning three buffers. The start of the packet is offset from the start of the first buffer. Following the last byte in the first buffer the packet continues to fill the entire second buffer. Note that the offset is not applied in the continuation buffer. The packet ends in the third buffer, again the offset is not used. The packet length (which is only put in the DMA descriptor associated with the first buffer) is the sum of the length used in the first buffer (i.e. the number of bytes from the offset to the end of the buffer), the full length of the second buffer, and the length used in the third buffer (i.e. the number of bytes from the buffer start to the end of the valid data).
Figure 25: Packet Spanning Three Buffers

Length = L1 + L2 + L3 in bytes
RINGS AND CHAINS

Each DMA engine works from a set of descriptors. These may be organized either as a ring or a chain.

A descriptor ring is shown in Figure 26. Each descriptor can point to two buffers and the next descriptor to use is in memory just after the current descriptor. The number of descriptors in the ring is configured when the ring is established. The DMA engine will detect when it has reached the last descriptor (at the highest memory location) by checking the ring size, its successor is the descriptor at the lowest memory location, which is pointed to by the descriptor base register in the configuration block. The ring size can be from 1 to 65536 descriptors. The DMA controller keeps track of its current position in the ring (initially the base). Software passes ownership of descriptors (and the associated pair of buffers) to the controller by incrementing the count of the number of descriptors owned by the controller. Writing to the count register adds the value written to the current count. The software may write the register at any time, the hardware prevents the potential conflict of the software incrementing as the controller decrements. There is no overflow protection on the count, so software should take care not to let the count pass 65535 descriptors owned by the DMA controller. When the controller has finished with a descriptor it will return ownership by decrementing the count. If the descriptor interrupt bit is set an interrupt will be raised when ownership of the descriptor is returned.
In a receive channel the interface flags the start of a packet when it sends data to the DMA engine. In a transmit channel the length field for one packet allows the DMA engine to directly determine when to expect the start of the next packet. In either case the DMA engine will always advance to a new descriptor when a new packet is started. Thus the first buffer in the pair can be used to point to small packet header buffers and the second buffer can be large and used for packet bodies. (This does not apply for the Data Mover since each transaction has its own descriptor.)

The descriptor associated with the first buffer in a packet is treated specially. It contains per-packet options that are passed to the interface describing any special treatment that this packet should receive. Once a packet has been completely transferred status information is written back into the first descriptor. Received packets accumulate many status bits (described in Section: “Option and Flag Bits for Ethernet MACs” on page 171 and Section: “Control and Flag Bits for Synchronous Serial Interface” on page 174), and additionally have the packet length written in to the descriptor. Only the first descriptor of the packet is written and the SOP flag bit will always be set. Software can clear the status word in all descriptors used by the receive engine and then use the SOP bit to identify descriptors that start packets. The SOP bit will only be set when the full packet has been received. If the system revision indicates PERIPH_REV3 or greater then the Ethernet DMA provides the option of overwriting the a_size field in the SOP descriptor with the number of descriptors that were used by the received packet. If this is enabled (by setting the xtra_status bit in the dma_config1 register) and the standard descriptor format is used then the software will need to keep track of the A buffer size by some method other than reading the descriptor and will need to restore the a_size field before the descriptor can be used for a subsequent DMA.

For transmitted packets the only change in the status flags is that the SOP bit is cleared, this can be used by software to detect the packet transmission has completed. If software does not make use of this feature the bandwidth used for descriptor management through the I/O bridge (and thus the latency for other accesses) can be reduced by disabling writing back the transmit status.

In the Ethernet DMA engine the SOP bit must always be set for the descriptor associated with the start of a packet, if the engine sees this bit clear in a descriptor that it believes should be the start of a packet then it will report a descriptor error and stop until software fixes the problem. If the SOP bit is set then the options field of the transmit descriptor must be nonzero and is used to enable operations on the data as it is transmitted, if the SOP bit is clear then the options field of the descriptor must be zero.

If a channel is disabled, when it is enabled it will start from the descriptor pointed to by the base address. The count will not be changed. The CPU must either zero the count (by adding the two's-complement of its current value) or reorder the descriptors to put valid ones at the start of the ring or chain. If the system revision indicates PERIPH_REV3 or greater then an Ethernet transmit DMA channel may be paused by setting the pause_channel_en bit in the dma_config1 register. This will pause the channel (but not the other transmit channel) at the next packet boundary. Transmission will be resumed when the bit is cleared. This can be used for example by software to stop transmission of a best-effort queue when the link partner indicates it is only able to buffer priority traffic.

The Ethernet interface will normally respond in hardware at the MAC layer to pause frame flow control requests. However, if the system_revision indicates PERIPH_REV3 or greater it can also be set to implement the pause at the DMA layer (when this is done the interface may stop more slowly because of any packets buffered in the fifo between the DMA engine and MAC). If this option is enabled (by setting the channel_base_fc_en bit in the mac_vlantag register) then the fc_pause_en bit in the dma_config1 register determines if the channel is affected by pause frames or not. Thus, provided the link partner is appropriately configured, the interface can be set to pause only one channel and allow the other (high priority) to continue even when the flow control is requested.
Figure 27 shows a descriptor chain. Each descriptor points to a single buffer, and contains a pointer to the next descriptor. Ownership of descriptors is passed in the same way as ring mode, the count refers to the number of valid descriptors. The last valid descriptor must contain a link pointer to the location where software will place the next descriptor, when the count reaches zero the controller will remember the link value and fetch the new descriptor from that address when the count becomes non-zero.

In both ring and chain modes reading the `dma_cur_dscr_addr` register gives address of the descriptor currently being processed and the current count of descriptors owned by the controller. If the count field in this register is non-zero then the address is of the descriptor that the controller is currently using. If the count is zero then the address is where the controller will fetch the next descriptor from when the count becomes non-zero.
UNALIGNED BUFFER DESCRIPTOR FORMAT FOR ETHERNET DMA

The alignment restrictions in the standard DMA descriptors may be hard to work with in some existing systems and when a lot of scatter/gather operations are done to form packets for transmission. The DMA engine in the Ethernet/Packet Fifo Interface on parts with system revision indicating PERIPH_REV3 or greater supports a third descriptor format that allows arbitrary alignment of the start and end of buffers. The descriptors are slightly modified to support this and there are a number of restrictions. Figure 28 on page 154 shows the descriptor format for the standard and unaligned buffer formats (a full definition is in Table 102 and Table 103). The changes and restrictions are:

- A buffer pointers always use the offset field (i.e. are a full 40 bit physical address).
- The size of the A buffer is specified in bytes in bits 21:8 of the dscr_b (where the buffer B address is in the standard format). Note that the descriptor field must be set to the size of the data plus the low 5 bits of the start address, so the maximum buffer size for arbitrary alignment is 16K-32 bytes.
- The a_size field is not used and can therefore always be used for the optional extra status information.
- No B Buffers are available.
- Only Ring mode can be used.
- The Packet Length, Status, Interrupt and Options fields work as in the standard mode.

The descriptor format is selected separately for each DMA channel. The unaligned mode is particularly useful for the transmit descriptors when packet manipulations are performed, but the standard mode may be more useful for the receive side where there is more control of the buffer locations and advantage can be taken of A and B buffers.

![Figure 28: Standard and Unaligned Buffer DMA Descriptors](image-url)
This format is useful for scatter/gather operations, in particular when transmit packets are being composed and headers inserted or removed. Care must be taken since the format allows for very low performance settings: for example if all buffers are only 3 bytes then the overhead of fetching descriptors will completely dominate the data transfers.

When the unaligned buffer format is used for packet reception and the start and end of a buffer are not aligned to a cache block boundary the DMA engine provides two options. The high performance option is for the interface to always write full cache lines (using the ZBbus WriteInvalidate command) and therefore overwrite the bytes before and after the buffer with UNPREDICTABLE data. The lower performance option is to use a read-modify-write to ensure only the valid bytes are modified, this will have a tendency to back up the read-modify-write queue (since the write must be queued for the read latency) and should therefore be avoided if possible.

**DMA COHERENCE AND CACHE OPTIONS**

All DMA transfers are coherent with the CPU L1 caches, the L2 cache and memory. On reads the latest copy of the data will be retrieved from the current owner and on writes any existing copies of the data will be invalidated or updated.

The DMA controllers can drive a L2-cache request attribute with transfers. This will cause the data to be allocated in the L2 cache if the cache misses (if the L2 cache hits it will always be updated regardless of the attribute). The channel configuration sets how many blocks from the start of a packet should be marked for allocation in the L2 cache (this can be set larger than the maximum packet size to have complete packets sent to the cache). The L2 is still being used as a cache, the data still has a memory address and can be evicted from the cache to memory if the L2 replacement algorithm requires it.

The network and serial port interfaces have data buffers that are aligned to a cache block boundary and are an integral number of cache blocks in size, so on incoming transfers they never need to perform read-modify-write operations (the actual packet data may be offset from the start of the buffer and may finish short of the end of the buffer, but the DMA will always be padded out to a full cache line at each end). The writes to memory are always made with the write-invalidate command, which invalidates any copies of the block in L1 caches, and returns ownership of the block to the L2 cache and memory system (the default owner).

In the standard aligned buffer mode, or the unaligned buffer mode with WriteInvalidate, device driver code has to take care not to store any additional data in the cache block that a buffer starts or ends in. Particular care must be taken with buffer link pointers and buffer header information at the start of the buffer. Even if an offset is applied in the descriptor the buffer is still aligned and any bytes before the offset will be overwritten with UNPREDICTABLE values when a packet is received. In the standard mode the end of the buffer is always aligned so there is not a problem, but in the WriteInvalidate unaligned mode if the buffer has an unaligned end the bytes for the rest of the cache block that contain the end of the buffer will be overwritten with UNPREDICTABLE data.
DMA CONFIGURATIONS

There are many ways that the DMA structure can be configured allowing use with many existing systems. The CPU should always use the cacheable coherent mode to access descriptors and buffers to allow the system to take care of the coherence and avoid the need for software to manage the L1 caches. The DMA engines access descriptors and buffers through the I/O bridge which will always use cacheable coherent accesses to memory addresses and uncacheable (accelerated) accesses for other addresses.

Consideration should be given to the descriptors and their access pattern. In memory space all reads from the controller will be of a full cache block, so if the system were configured to use chain mode descriptors and 32 byte buffers the number of memory accesses fetching descriptors will equal the number of memory accesses for data. The system performance will be lower than expected because half of the bandwidth is consumed by descriptor management. In a new system it is recommended that the descriptors be organized in ring mode and the controller be permitted to prefetch descriptors (tdx_en set in the dma_config0 register). This reduces the descriptor fetch overhead compared to the number of buffer transfers. In most cases receive descriptors should also be marked for allocation in the L2 cache (dscr_l2ca set in the dma_config1 register) since they are a shared resource and they are expected to be accessed over a short time period by the DMA controller (twice for the header descriptor which will be updated with the length and status information) and the CPU. When the DMA controller updates a descriptor it must use a read-modify-write operation. If software does not need the SOP bit to be cleared in transmit descriptors, the read-modify-write can be avoided by setting the no_dscr_updt bit in the dma_config1 register. If this is done, transmit descriptors do not need to be marked for L2 cache allocation by the DMA engine, the CPU will have them cacheable in both L1 and L2 caches so the DMA engine will read them directly from the cache and it never writes them back.

The buffer size can also be increased to reduce the descriptor overhead. While the hardware does support buffers as small as 32 bytes (and these may be ideal as header buffers) the system becomes more efficient as the buffer size is increased. In the transmit channel the tbx_en bit should be set in the dma_config0 register to allow the controller to prefetch buffer data whenever possible. If this bit is set the controller will mostly fetch pairs of cache blocks back to back and is more likely to make use of an open page in the SDRAM. On the receive side cache block writes are posted as soon as the data is available and this bit has no effect.

The most sensitive interface is the transmit side of the Ethernet (or Packet FIFO in GMII mode). Once transmission of a packet has started there must be no interruptions. This is achieved by using a small data FIFO in the interface and priority in the memory controller. The data FIFO is intended just for speed match buffering. There is a threshold which sets how much of a packet must be in the FIFO for transmission to begin, since the FIFO drains at a constant rate this directly translates into the memory latency that will be covered by the buffer. In general the FIFO will fill quickly (particularly when tbx_en allows fetching of pairs of cache blocks), and drain during any high latency memory reads. The worst case is when a new descriptor must be fetched during a packet transmission, since the next block of data cannot be fetched until the descriptor read completes. To allow this to work with a relatively small FIFO the memory controller implements a priority scheme. This is needed because the CPUs and data mover can easily swamp the memory controller (they can access data at much higher bandwidths and frequencies than the I/O DMA engines, so this is an ideal place for priority to protect the low request rate interface from being dominated by a high request rate one). Any reads from the I/O bridge 1 are prioritized over other memory accesses, if they conflict with a write to memory the priority of the write is also raised. In addition some of the memory controller buffers can be reserved for use by the I/O bridge 1 DMA engines. The priority scheme is described in Section: “Memory Controller Architecture” on page 104. If memory is accessed across the HyperTransport there is no priority scheme and the transmit FIFO is likely to underflow during packet transmission (however, because the writes are posted it should be possible to receive into buffers in a memory across the HyperTransport fabric).
ETHERNET AND SERIAL DMA ENGINES

The Ethernet and serial DMA engines are identical except for using different per-packet options and status flags. Each channel works as described above using either a ring or a chain. The DMA engines have the restriction that neither the descriptors nor data buffers may be at an address serviced by I/O bridge 1, the system behavior is UNDEFINED if this is violated. In practice this restriction just rules out DMA access from the Ethernet or Serial DMA to the generic bus, since the other devices connected through I/O Bridge 1 are all based on control registers and are unreasonable DMA targets. Figure 5 on page 9 shows the devices that are behind I/O Bridge 1.

Both Ethernet and serial DMA controllers are connected to the ZBbus through I/O Bridge 1. In the direction from the ZBbus to the devices there is a separate command queue for read and writes to peripheral registers and response queue for data returning to descriptor or buffer DMA read requests. This ensures that the DMA traffic is not blocked by requests to slow peripherals (such as a slow device on the generic bus). In the direction towards the ZBbus it is important that if a status read reports completion of a command the response does not pass that command, so read and write requests from the DMA engine and responses to register reads are held in a single queue. Generic bus responses do not need this ordering and are given their own return queue.

Each channel has an interrupt associated with it. In the ethernet interface the status for the two receive and two transmit DMA channels are combined into the mac_status register for the interface (see Table 182 on page 309). The serial interface transmit and receive channel status are combined into the ser_status register for each channel that can be read to determine the interrupt cause (see Table 240 on page 356). Reading the status will clear all bits in the status register and clear the interrupt.

DESCRIPTOR COUNT WATERMARKS

The count of descriptors owned by the DMA controller is compared to two watermark registers. The high watermark is set in the high_watermark field of the dma_config0 register, and the low watermark is set in the low_watermark field. It is expected that the high watermark is programmed to a larger value than the low watermark. The CPU can opt to be interrupted when the descriptor count falls below either (or both) watermarks.

In a transmit engine setting the low watermark register to 1 and disabling the high watermark interrupt will give an interrupt when all queued transmissions are complete.

In a receive engine the interface can be configured to use automatic flow-control. If this is enabled then when the number of descriptors falls below the low watermark the interface will be signalled to use flow control to apply back pressure, which will only be removed when sufficient descriptors have been supplied to push the count above the high watermark. If the engine is configured to interrupt the CPU when the descriptor count falls below the high watermark, software has a chance to allocate more receive descriptors before flow control is asserted. If automatic flow control is enabled and the high watermark is smaller than the low watermark the behavior of the engine is UNPREDICTABLE. The watermark based flow control will not activate until after buffers have been added to the descriptor queue for the first time, so the link will not be jammed as the interface comes out of reset.
The watermark status, which is used to raise an interrupt, is continually updated. If the number of descriptors falls below a watermark just as the CPU is doing a write to the count register to increase the number of descriptors owned by the DMA engine, the interrupt line may be briefly asserted but all status bits will be clear when the CPU has entered the interrupt service routine and reads the status register. Software should be written to work correctly if these spurious interrupts occur (or only add buffers in the ISR or with the interrupts disabled).

If the receive DMA engine runs out of descriptors during a packet the tail of the packet is lost and the dscr_err bit is set in the receive status. In this case the status includes enough information for the software to compute the size of data actually transferred, but in most cases the packet will just be dropped. If the receive engine is out of descriptors when the start of a packet is received the whole packet is dropped and the interface will signal a packet dropped interrupt. In either case packet reception will resume with the start of the first packet received after the count has been written to make more descriptors available. The Ethernet receive DMA channels count the packets that are dropped because the channel is out of descriptors when the start of packet is received, on parts with system revision indicating PERIPH_REV3 or greater this count can be read from the dma_oodpktlost register. Any write to this register (regardless of data) will zero the count.

**COMPLETION INTERRUPTS**

The controller has a packet counter and will generate an interrupt after transferring a configurable number of packets. This is of most use in the receive channel. If the interrupt count is set to one then an interrupt will be raised after every packet. Since this can swamp the system with interrupts, the count would typically be set higher and the receive interrupt service routine will be written to accept a batch of packets. In order to avoid imposing a high delay before packets are serviced when they are arriving at a low rate, the interrupt can also be raised by a timer. The timer starts counting when the first packet reception is complete and will increment every 768 CPU clocks (i.e. the counter is clocked at CPUclock/(3*256)). If the interrupt has not been raised because the packet count threshold has been reached it will be forced when the timer has counted to a programmed limit. Reading the interface interrupt status register will clear both the packet counter and resets the timer, both will remain at zero until the next packet transfer is complete. The device driver must be written based on this behavior and should handle (or queue) any interrupts that are marked when the status is read.

The completion interrupts are also available for transmit interfaces. In this case the counter will increment when a packet transmission has completed, and the timer will start running when the first transmission has completed. This could be used to detect the transmitter being unable to send for an unacceptably long period.

The interrupt is cleared by reading the channel's interrupt status register, this disables the timer and zeros the received packet count preparing the system for the next batch of packets.

On parts with system revision indicating PERIPH_REV3 or greater the current value of the received packet count can be read from the dma_oodpktlost register.

**EXPLICIT DESCRIPTOR INTERRUPTS**

The descriptors include a flag to cause the controller to raise an interrupt when it has finished using the marked descriptor. The flag can be set on the last descriptor in a batch of packets that are queued for transmission, software will then receive notification when they have all been sent even if more packets have been queued.
ASIC MODE TRANSFERS

The ethernet and serial interface receive DMA engines have a special "ASIC mode" that enables received packets to be passed through an ASIC on the HyperTransport fabric or PCI bus. This is in addition to just sending the packet by address to an I/O destination. It allows connection of a simple ASIC that is logically inserted into the path from the interface to memory.

The ASIC mode is intended for use in configurations where there is an assist ASIC on the HyperTransport or PCI through which some or all of a packet must be passed. Examples include header classification engines and encryption/decryption devices. Figure 29 shows an example packet flow.

ASIC mode is enabled by setting the asic_xfr_en bit in the dma_config1 register. The normal DMA descriptors are fetched for each packet. Rather than sending the packet header to the first buffer in the descriptor, the packet is directed to the address range set in the dma_asic_addr register. The asicxfr_size is set to the number of cache lines of the packet that should be sent to the ASIC; if there is an offset specified this is one less than the maximum number of cache lines that should be sent to the ASIC address range (i.e. if the asicxfr_size is 0 and asci_xfr_en is set then one cache line will be sent to the ASIC). The size sent to the ASIC must match the size set for the first descriptor or behavior of the engine is UNDEFINED. If the packet length is shorter than the size specified by the asicxft_size field then the entire packet is sent to the ASIC, otherwise after the configured size has been sent to the ASIC the DMA controller advances to the second buffer in the descriptor and will transfer to memory in the normal way. If the packet spans to subsequent descriptors they will be used to send the data to memory; data is only directed to the ASIC following a start of packet.
The pre_addr_en bit enables prepending of the dscr_a doubleword of the DMA descriptor to the data sent to the ASIC. If this bit is set then the offset in the DMA descriptor must be set to 8 (or greater than 8) and the asicxfr_size must be adjusted as outlined above. The descriptor information can be used by the ASIC to write the processed data back into memory at the address specified in the original DMA descriptor. Since the entire descriptor doubleword is sent the ASIC has access to the size of the buffer, and flags can be passed from the software to the ASIC by setting them up in the status field. (This field is not used by the DMA controller, so can contain any value when it reads the descriptor. However, the descriptor in memory will be updated by the DMA engine on completion of the packet reception, so any value passed from software to the ASIC in these bits will be overwritten). Note that the offset is not sent to the ASIC, these 5 bits are UNPREDICTABLE. The ASIC does not receive the packet status information, except for an error indication described below.

Cache blocks of the packet will be sent to the ASIC tagged by address according to Figure 30 and Table 90.

```
<table>
<thead>
<tr>
<th>39</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>5</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC Base Addr</td>
<td>Rx DMA</td>
<td>Channel</td>
<td>Number</td>
<td>Block</td>
<td>Type</td>
<td>Byte</td>
<td>Count</td>
<td>Ignored</td>
<td>Unused</td>
<td>in block</td>
<td>address</td>
</tr>
</tbody>
</table>
```

**Figure 30: ASIC Mode Address Generation**

The dmaasic_addr specifies the base address for a 1 MB region of memory. The first cache block in a packet (with the descriptor prepended if enabled) will be sent to the Start Of Packet (SOP) address, subsequent blocks are sent to the Middle Of Packet (MOP) address range and the last cache block in the packet will be set to either the End Of Packet (EOP) address or the EOP/Error address. The block sent to the EOP address may contain less than a full cache block of valid data, the byte_count indicates the number of valid bytes which will be packed towards the low memory address end of the block sent.

**Table 90: Address Used for ASIC Mode Transfers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>These bits are not present, since a full cache line is always transferred.</td>
<td></td>
</tr>
<tr>
<td>10:5</td>
<td>ignored</td>
<td>Ignore</td>
</tr>
<tr>
<td>15:11</td>
<td>byte_count</td>
<td>In EOP packets this is set to the number of valid bytes in the block. This field is ignored in SOP or MOP blocks. There can be from 1-32 valid bytes, the field is 0 to indicate 32 bytes valid.</td>
</tr>
<tr>
<td>17:16</td>
<td>type</td>
<td>Set to the position of the block in the packet.</td>
</tr>
<tr>
<td></td>
<td>00: Middle of packet (MOP).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>01: Start of packet (SOP), the block includes the prepended descriptor if enabled.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>10: End of packet with no errors (EOP), the byte_count field indicates the number of valid bytes in the cache block.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>11: End of packet with errors (EOP/Error), the byte_count field indicates the number of valid bytes in the cache block.</td>
<td></td>
</tr>
<tr>
<td>19:18</td>
<td>channel</td>
<td>Set to the dma channel number.</td>
</tr>
</tbody>
</table>
Figure 31 shows a full packet being sent using ASIC mode. In this case the asicxfr_size is set to its maximum value (which is greater than the maximum packet size) ensuring the complete packet is sent to the ASIC.

The first cache block is sent to the SOP address, formed by adding the channel number bits and the 01 SOP code bits to the base address. This line includes the prepended first doubleword of the DMA descriptor (dscr_a) in the lowest 8 bytes. Subsequent blocks are sent to the MOP address until the last block. The final block is sent to the EOP address formed by adding the channel number bits, the 10 EOP code and the number of valid bytes to the base address. The valid byte count is needed because a full 32 byte block is always sent. Once the complete packet has been sent the DMA controller will write the length and status flags back into the descriptor in the normal way. The next descriptor will be fetched for the next packet (so the B buffer is never used). The ASIC can process the packet, and write data back to memory at the address given in the prepended dscr_a.

The ASIC needs a way to inform the CPU that its processing is complete. The DMA controller will generate the end of packet (or completion or watermark) interrupt when it has sent the full packet to the ASIC and has written the status back into the descriptor. There is a latency in the buffering between the interface and the ASIC, the ASIC will take time to process the packet and there will be latency writing the result back to memory, so any interrupts from the DMA controller are likely to be seen by the CPU before the ASIC has completed processing the packet. A simple way for the ASIC to do this is to send a HyperTransport interrupt message after writing the last data. The HyperTransport protocol, and host bridge will ensure that the ordering is maintained and the CPU does not receive the interrupt until the data of the write is visible in the coherency protocol.
Figure 32 illustrates the ASIC mode being used to pass only the header of a packet (for example for classification). The asicxfr_size is set to the number of header cache blocks (typically 2, causing 64 bytes of header to be sent to the ASIC with no prepended descriptor or 1 causing 56 bytes of header to be sent with the prepended descriptor). The dscr_a size must be set to the same value.

The SOP address is still used for the first block in the packet, and MOP for subsequent blocks (if the packet is only two blocks the second will be sent to the EOP address). Once the header size has been sent to the ASIC address the DMA controller advances to buffer B and sends the rest of the packet into memory in the normal way, when it is done it will write the length and flags back into the descriptor. The ASIC can perform its header processing and write results and the header back using the buffer A address. If CPU intervention is required an ASIC on the HyperTransport fabric could opt to write the results and header back with HyperTransport Isochronous writes, which causes the data to be allocated in the L2 cache. If the results are greater than 8 bytes (the offset that was specified to allow the dscr_a to be prepended) the actual buffer size may need to be greater than indicated in the descriptor, since the size given in the descriptor must match the size sent to the ASIC. The system software must manage this difference by allocating the larger buffer and adjusting the size in the descriptors.
ETHERNET AND SERIAL DMA CONTROL REGISTERS

The BCM1250 has MACs 0, 1 and 2.

The BCM1125/H has MACs 0 and 1. Accesses made to the address range allocated to a nonexistent MAC may cause all MACs to exhibit UNPREDICTABLE behavior.

Table 91: Ethernet and Serial DMA Configuration Register 0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>drop</td>
<td>1'b0</td>
<td>Set to cause all packets to the channel to be dropped. MAC receive channels only, setting this bit in other channels causes UNDEFINED behavior.</td>
</tr>
<tr>
<td>2:1</td>
<td>dscr_type</td>
<td>2'b0</td>
<td>This field sets the descriptor format:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 - Descriptor Ring, Standard Aligned buffer format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 - Descriptor Chain, Standard Aligned buffer format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 - Descriptor Ring, Unaligned buffer format, WritelInvalidate overwrites a full cache block at start/end of a buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 - Descriptor Ring, Unaligned buffer format, read-modify-write at start/end of unaligned buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Formats 10 and 11 are only supported by the Ethernet DMA engine and only if the system revision indicates PERIPH_REV3 or greater.</td>
</tr>
<tr>
<td>3</td>
<td>eop_int_en</td>
<td>1'b0</td>
<td>Set to enable interrupt at end of packet. The interrupt will be generated when the number of packets specified in int_pktcnt have been received, or if the receive interrupt timer has timed out. See Section: “Completion Interrupts” on page 158.</td>
</tr>
<tr>
<td>4</td>
<td>hwm_int_en</td>
<td>1'b0</td>
<td>Set to enable an interrupt when the number of descriptors owned by the DMA controller falls below the high watermark.</td>
</tr>
<tr>
<td>5</td>
<td>lwm_int_en</td>
<td>1'b0</td>
<td>Set to enable an interrupt when the number of descriptors owned by the DMA controller falls below the low watermark. The receiver will assert flow control when the number of descriptors falls below this watermark.</td>
</tr>
<tr>
<td>6</td>
<td>tbx_en</td>
<td>1'b0</td>
<td>Set to cause the transmit DMA engine to fetch two 32 byte blocks at a time (increasing the open page hit rate in the SDRAM). Clear to only fetch one block at a time. The tx_wr_thrsh threshold must be set to match the fetch size. MAC transmit channels only, setting this bit in other channels causes UNDEFINED behavior. If system revision indicates a peripheral revision earlier than PERIPH_REV3 then when this bit is set the buffer size should be a multiple of 64 bytes, if the system revision indicates PERIPH_REV3 or greater there is no restriction.</td>
</tr>
<tr>
<td>7</td>
<td>tdx_en</td>
<td>1'b0</td>
<td>Set in ring mode to allow the DMA engine to fetch two descriptors at a time. Clear if the engine should use 16 byte fetches to get a single descriptor. If this bit is set and there is only a single descriptor available then the invalid prefetched data is discarded. It is recommended that this bit is set in any high bandwidth DMA channels.</td>
</tr>
<tr>
<td>15:8</td>
<td>int_pktcnt</td>
<td>8'b0</td>
<td>This sets the number of packets that must be received before an end of packet interrupt is raised. See Section: “Completion Interrupts” on page 158.</td>
</tr>
<tr>
<td>31:16</td>
<td>ringsz</td>
<td>16'b0</td>
<td>When the channel is operating in ring mode this sets the number of descriptors in the ring. It should only be changed when the channel is disabled. If this field is set to 0 the ring will contain 65536 descriptors.</td>
</tr>
</tbody>
</table>
47:32 high_watermark 16'b0 This specifies the high watermark for generating an interrupt to the CPU based on the number of descriptors. If the MAC receiver is asserting flow control and the number of descriptors rises above this watermark it will remove the flow control.

63:48 low_watermark 16'b0 This specifies the low watermark for generating an interrupt to the CPU based on the number of descriptors. The MAC receiver will assert flow control when the number of descriptors falls below this watermark.

Table 92: Ethernet and Serial DMA Configuration Register 1

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hdr_cf_en</td>
<td>1'b0</td>
<td>If this bit is set then the L2_cacheable bit will be asserted during header transfers to cause the L2 cache to be allocated.</td>
</tr>
<tr>
<td>1</td>
<td>asic_xfr_en</td>
<td>1'b0</td>
<td>Set to enable special support for transferring packets to an ASIC on the HyperTransport fabric (or PCI bus).</td>
</tr>
<tr>
<td>2</td>
<td>pre_addr_en</td>
<td>1'b0</td>
<td>Set when asic_xfr_en is set to enable prepending of the DMA descriptor to the packet sent to the ASIC.</td>
</tr>
<tr>
<td>3</td>
<td>flow_ctl_en</td>
<td>1'b0</td>
<td>Set to cause the controller to send a flow control request to the interface when the descriptor count falls below the low watermark, and only remove the request when the count goes above the high watermark. MAC receive channels only, setting this bit in other channels causes UNDEFINED behavior.</td>
</tr>
<tr>
<td>4</td>
<td>no_dscr_updt</td>
<td>1'b0</td>
<td>Set to prevent the descriptor being written with the status at the end of a packet transfer, saving bandwidth through the bridge. (Transmit channels only.)</td>
</tr>
<tr>
<td>5</td>
<td>dscr_l2ca</td>
<td>1'b0</td>
<td>This bit sets the L2 cacheability for descriptors. If it is set the L2CA bit will be set in descriptor requests, causing them to be cached in the L2 cache. If clear the descriptors will not be allocated in the L2 on a miss.</td>
</tr>
<tr>
<td>6</td>
<td>(rx) xtra_status</td>
<td>1'b0</td>
<td>Ethernet Receive channels only: This bit should only be set for the Ethernet receive engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set the a_size field will be overwritten with additional status information in the start of packet descriptor.</td>
</tr>
<tr>
<td></td>
<td>(tx) cpu_pause_en</td>
<td></td>
<td>Ethernet Transmit channels only: This bit should only be set for the Ethernet transmit engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set the channel will pause at the end of the current packet and will resume only when the bit is cleared.</td>
</tr>
<tr>
<td>7</td>
<td>fc_pause_en</td>
<td>1'b0</td>
<td>Ethernet Transmit channels only: This bit should only be set for the Ethernet transmit engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set and the interface is set for DMA channel based flow control (ch_base_fc_en bit set in the mac_vlantag register) then this channel will be paused by a flow control frame, if this bit is clear and the interface is set for DMA channel based flow control the channel will not be paused by a flow control frame. If the interface is not configured for DMA channel based flow control this bit is ignored.</td>
</tr>
<tr>
<td>15:8</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>Bits</td>
<td>Name</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>20:16</td>
<td>zero</td>
<td>5'b0</td>
<td>These bits must be zero.</td>
</tr>
<tr>
<td>29:21</td>
<td>hdr_size</td>
<td>9'b0</td>
<td>This sets the header length in cache blocks. It sets the number of cache blocks for which the L2 cacheable attribute will be attached if hdr Cf_en is asserted. This count is taken from the start of the buffer (the first cache line contains only 32 - offset bytes from the packet).</td>
</tr>
<tr>
<td>31:30</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>36:32</td>
<td>zero</td>
<td>5'b0</td>
<td>These bits must be zero.</td>
</tr>
<tr>
<td>45:37</td>
<td>asicxfr_size</td>
<td>9'b0</td>
<td>This field sets the size (in cache lines) of the portion of the packet that will be sent to the ASIC ifasic_xfr_en is set. Packets shorter than this size will be sent in their entirety using the EOP protocol described in Section: &quot;ASIC Mode Transfers&quot; on page 159 to signal the packet end. Care must be taken to adjust the number set here when an offset is used (which must be the case if the pre_addr_en bit is set). The number of cache lines sent to the ASIC will be: Offset zero: The number in this field. (If this field is zero then one cache line is transferred.) Offset nonzero: One greater than the number in this field.</td>
</tr>
<tr>
<td>47:46</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:48</td>
<td>int_timeout</td>
<td>16'b0</td>
<td>This field sets the timeout for interrupt generation. See Section: &quot;Completion Interrupts&quot; on page 158. If this field is zero then timing of the assertion of the eop_timer interrupt flag is UNPREDICTABLE.</td>
</tr>
</tbody>
</table>
### Table 93: Ethernet and Serial DMA Descriptor Base Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>zero</td>
<td>4'b0</td>
<td>These bits must be zero.</td>
</tr>
<tr>
<td>39:4</td>
<td>base</td>
<td>36'b0</td>
<td>This is the base address of the descriptor ring, or the pointer to the first descriptor in a chain. This register must only be changed when the channel is disabled. When a channel is disabled and enabled again it will start fetching descriptors from this address.</td>
</tr>
<tr>
<td>64:40</td>
<td>reserved</td>
<td>24'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 94: ASIC Mode Base Address

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>19:0</td>
<td>zero</td>
<td>20'b0</td>
<td>These bits must be zero.</td>
</tr>
<tr>
<td>39:20</td>
<td>base</td>
<td>36'b0</td>
<td>This is the base of the ASIC address space.</td>
</tr>
<tr>
<td>64:40</td>
<td>reserved</td>
<td>24'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 95: Descriptor Count Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>count</td>
<td>16'b0</td>
<td>This is the number of descriptors owned by the DMA engine. Reads will return the number of descriptors owned (Count in Figure 26 on page 151 and Figure 27 on page 153). Data that is written to this register will be added to the count (assigning that number of additional descriptors to the controller). The count is a 16 bit unsigned number, there is no overflow protection. Software should ensure it does not unintentionally cause the count to wrap.</td>
</tr>
<tr>
<td>64:16</td>
<td>reserved</td>
<td>48'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 96: Current Descriptor A Debug Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>cur_a</td>
<td>64'b0</td>
<td>The current descriptor first double word can be read from this register (intended for debugging only).</td>
</tr>
</tbody>
</table>

### Table 97: Current Descriptor B Debug Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>cur_b</td>
<td>64'b0</td>
<td>The current descriptor second double word can be read from this register (intended for debugging only).</td>
</tr>
</tbody>
</table>

### Table 98: Current Descriptor Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>dscr_addr</td>
<td>40'b0</td>
<td>The current descriptor address can be read from this field.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If count is nonzero the address is the descriptor being used.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If count is zero the address is where the next descriptor will be fetched from.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>55:40</td>
<td>count</td>
<td>16'b0</td>
<td>The current count of descriptors owned by the DMA engine can be read from this field. (It provides the same information as reading the <code>dma_dscr_count</code> register).</td>
</tr>
</tbody>
</table>
56: ch_pause_on 1'b0 Only valid if System revision indicates PERIPH_REV3 or greater and only valid for Ethernet transmit channels. This bit is set if the channel has paused because the cpu_pause_en bit is set in the dma_config1 register or a flow control frame has been received the fc_pause_en bit is set in the dma_config1 register and the ch_base_fc_en bit is set in the mac_vlan_tag register. Note that this bit is set when the channel is actually pausing, so when the cpu_pause_en bit is set the ch_pause_en will remain clear until any packet in progress has been sent and the channel is stopped between packets.

63:57 reserved 7'b0 Reserved

Table 99: Ethernet Receive Packet Drop Registers (Only if System Revision >= PERIPH_REV3)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>oodlost</td>
<td>16'b0</td>
<td>This is the count of packets that were dropped by the channel because it was out of descriptors when the start of the packet was received. (The count will not be incremented if the channel has descriptors available but is disabled, or if the channel has descriptors but the drop bit is set.) The counter will stick at 16'hFFFF to indicate 65535 or more packets dropped.</td>
</tr>
<tr>
<td>23:16</td>
<td>eop_count</td>
<td>8'b0</td>
<td>This field contains the current value of the counter that controls the completion interrupt.</td>
</tr>
<tr>
<td>63:24</td>
<td>reserved</td>
<td>40'bx</td>
<td>This field is reserved. Reads will return UNPREDICTABLE data.</td>
</tr>
</tbody>
</table>
## ETHERNET AND SERIAL DMA DESCRIPTORS

### Table 100: DMA Descriptor First Doubleword

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>offset</td>
<td>The offset in the buffer that the header should start at.</td>
</tr>
<tr>
<td>39:5</td>
<td>a_addr</td>
<td>The base address of the data buffer (cache block aligned). In ring mode this is the base address of the first data buffer.</td>
</tr>
<tr>
<td>48:40</td>
<td>a_size</td>
<td>The size of the data buffer (in cache blocks). If this field is zero the buffer is 512 cache blocks. In a transmit channel which has the tbx_en configuration bit set bit [40] must be zero.</td>
</tr>
<tr>
<td>49</td>
<td>interrupt</td>
<td>If this bit is set an interrupt will be raised when the DMA engine has finished using this descriptor.</td>
</tr>
<tr>
<td>50</td>
<td>offset_b</td>
<td>If this bit is set then the offset field will be applied to the first buffer pointed to by descriptor B in a packet rather than the first buffer pointed to by descriptor A. The DMA controller behavior is UNPREDICTABLE if this bit is set in chain mode. If this bit is set, the b_valid bit must be set in the decr_b or the behavior od the DMA engine will be UNPREDICTABLE.</td>
</tr>
<tr>
<td>63:51</td>
<td>status</td>
<td>In the first descriptor of a received packet this field will be updated with the packet status at the end of reception. The flags differ between the interfaces and are described in Section: “Option and Flag Bits for Ethernet MACs” on page 171 and Section: “Control and Flag Bits for Synchronous Serial Interface” on page 174.</td>
</tr>
</tbody>
</table>

### Table 101: DMA Descriptor Second Doubleword

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>options</td>
<td>This field sets the per packet options that are sent to the interface. The options depend on the channel, and are outlined below.</td>
</tr>
<tr>
<td>4</td>
<td>addr</td>
<td>In chain mode this bit provides the additional address bit that is used with bits 39:5 to form the pointer to the next descriptor. In ring mode the bit is reserved and should be zero.</td>
</tr>
<tr>
<td>39:5</td>
<td>b_addr</td>
<td>In ring mode this gives the base address of the second data buffer (cache block aligned). In chain mode it and bit 4 contain a pointer to the next descriptor.</td>
</tr>
<tr>
<td>48:40</td>
<td>b_size</td>
<td>The size of the second data buffer (in cache blocks). If this field is zero the buffer is 512 cache blocks. In a transmit channel which has the tbx_en configuration bit set bit [40] must be zero.</td>
</tr>
<tr>
<td>49</td>
<td>b_valid</td>
<td>If this bit is set the buffer is valid and will be used. If this bit is clear the B buffer is not used and the b_addr and b_size fields are ignored. (Ring mode only.)</td>
</tr>
<tr>
<td>63:50</td>
<td>pkt_size</td>
<td>In the first descriptor of a packet this field contains the packet length. In a transmit channel the length is written by the CPU as part of setting up the descriptor and is passed to the interface. In a receive channel the length is written back into the descriptor at the end of reception.</td>
</tr>
</tbody>
</table>
### Table 102: Unaligned Buffer Format DMA Descriptor First Doubleword

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>addr</td>
<td>The base address of the data buffer.</td>
</tr>
<tr>
<td>47:40</td>
<td>unused</td>
<td>Unused. If extra_status is enabled the receive DMA engine will overwrite this field.</td>
</tr>
<tr>
<td>48</td>
<td>reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>49</td>
<td>interrupt</td>
<td>If this bit is set an interrupt will be raised when the DMA engine has finished using this descriptor.</td>
</tr>
<tr>
<td>50</td>
<td>reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>63:51</td>
<td>status</td>
<td>In the first descriptor of a received packet this field will be updated with the packet status at the end of reception. The flags are described in Section: “Option and Flag Bits for Ethernet MACs” on page 171.</td>
</tr>
</tbody>
</table>

### Table 103: Unaligned Buffer Format DMA Descriptor Second Doubleword

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>options</td>
<td>This field sets the per packet options that are sent to the interface. The options depend on the channel, and are outlined below.</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>21:8</td>
<td>size</td>
<td>This field specifies the buffer size in bytes. This field must be set to the number of bytes of data in the buffer plus bits [4:0] of the start address in the dscr_a addr field. The DMA engine behavior is UNPREDICTABLE if this size is set to be less than the value of bits [4:0] of the address in dscr_a.</td>
</tr>
<tr>
<td>47:22</td>
<td>reserved</td>
<td>Reserved.</td>
</tr>
<tr>
<td>49:48</td>
<td>pkt_size_msb</td>
<td>This field provides bits [15:14] of the packet size. (Note that if packets greater than 16K-1 are used only the low 14 bits of the length is added to the RMON counters.)</td>
</tr>
<tr>
<td>63:50</td>
<td>pkt_size</td>
<td>In the first descriptor of a packet this field contains the packet length. In a transmit channel the length is written by the CPU as part of setting up the descriptor and is passed to the interface. In a receive channel the length is written back into the descriptor at the end of reception. In unaligned buffer mode this field is extended to 16 bits using the pkt_size_msb field as the upper two bits.</td>
</tr>
</tbody>
</table>
### Option and Flag Bits for Ethernet MACs

**Table 104: Status Flags for Ethernet Receive Channel**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Flags Written to dscr_a size_a field if extra_status is enabled</td>
</tr>
<tr>
<td>47:40</td>
<td>descr_cnt</td>
<td>Number of descriptors used by this packet. If more than 255 descriptors were used to receive the packet then this field will be 0 and software must compute the actual number. This field is valid for Ethernet and Packet Fifo modes.</td>
</tr>
<tr>
<td>48</td>
<td>reserved</td>
<td>This bit may be used for status information in the future.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flags Written to dscr_a options field</td>
</tr>
<tr>
<td>51</td>
<td>bad_ip4cs</td>
<td>This bit is clear if at the offset specified in the iphdr_offset field of the mac_adfilter_cfg register the receive packet has a standard length (no options) IPv4 header with a valid header checksum. Otherwise it is set. This flag is only valid in Ethernet mode.</td>
</tr>
<tr>
<td>52</td>
<td>descr_err</td>
<td>Descriptor error. During packet reception the channel ran out of descriptors, the tail of the packet is lost. When this bit is set, the length field can be used to determine the number of bytes transferred before the receiver ran out of descriptors. If the system_revision indicates PERIPH_REV3 or later, the length field indicates the number of bytes transferred. In older revisions the field must be corrected to account for the offset at the start of the packet. Software should compensate by subtracting the low three bits of the offset field: ActualBytesTransferred = dscr_b.pkt_size - 32 - (dscr_a.offset &amp; 7) This flag is valid for Ethernet and Packet Fifo modes.</td>
</tr>
<tr>
<td>54:53</td>
<td>rx_ch</td>
<td>Receive channel number (always 2’b00 for DMA channel zero)</td>
</tr>
<tr>
<td>57:55</td>
<td>pkt_type</td>
<td>Packet type. Contains an encoded version of the Ethernet Type field. See Section: &quot;Packet Type Identification&quot; on page 281. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)</td>
</tr>
<tr>
<td>58</td>
<td>match_hash</td>
<td>This bit is set if the packet passed the hash match filter. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)</td>
</tr>
<tr>
<td>59</td>
<td>match_exact</td>
<td>This bit is set if the received packet passed the exact addresses filter. (The packet will pass if it matched an address and the filter was true, or the packet did not match and the filter was inverted) This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)</td>
</tr>
<tr>
<td>60</td>
<td>bcast</td>
<td>This bit is set if the received packet has a broadcast address. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)</td>
</tr>
<tr>
<td>61</td>
<td>mcast</td>
<td>This bit is set if the received packet has a multicast address. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)</td>
</tr>
<tr>
<td>62</td>
<td>bad</td>
<td>This bit is set if the received packet has errors (length error or dribble error or code error or CRC error or is a runt packet or is an oversize packet). See Table 162 on page 275 for a description of the errors. In Packet Fifo mode, the error bit will be set by a CRC error or the packet is terminated with a link Error (only possible in GMII style and Encoded modes).</td>
</tr>
</tbody>
</table>
This bit is set to indicate the start of the packet. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.

### Flags Written to descr_b options field

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bad_tcpcs</td>
<td>The value of this bit is only valid if the interface is in Ethernet mode and the bad_ip4cs bit is clear. When valid, this bit is clear if the packet received has a correct TCP checksum, and set if the packet has a bad TCP checksum. Since UDP uses the same checksum algorithm this flag also applies to UDP packets.</td>
</tr>
<tr>
<td>1</td>
<td>vlan_flag</td>
<td>This bit is only valid if the system revision indicates PERIPH_REV3 or greater and for Ethernet operation and the vlan_det_en bit is set in the mac_adfilter_cfg register. This bit will be set when a packet with a VLAN tag is received and clear if there is no VLAN tag.</td>
</tr>
<tr>
<td>2</td>
<td>crc_flag</td>
<td>This bit is only valid if the system revision indicates PERIPH_REV3 or greater. Ethernet mode only: this flag is set (in addition to the bad bit) when a CRC error is detected.</td>
</tr>
<tr>
<td>3</td>
<td>reserved</td>
<td>This bit may be used for status information in the future.</td>
</tr>
</tbody>
</table>

### Table 105: Option Flags for Ethernet Receive Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

### Table 106: Status Flags for Ethernet Transmit Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>62:51</td>
<td>ignored</td>
<td>Preserved when the descriptor is written.</td>
</tr>
<tr>
<td>63</td>
<td>SOP</td>
<td>This bit must be set for the first descriptor in a packet. The DMA engine will report a descriptor error if it is starting a new packet and this bit is clear in the corresponding descriptor. This bit is cleared when the descriptor is written.</td>
</tr>
</tbody>
</table>
Table 107: Option Flags for Ethernet Transmit Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>pkt_mod</td>
<td>This field indicates how the transmitter should modify the packet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0000: Descriptor is not start of packet.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0001: Append CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0010: Replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0011: Append CRC and pad.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0100: Append VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0101: Remove VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0110: Replace VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0111: No Modifications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1000: Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001: Replace source address and append CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1010: Replace source address and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1011: Replace source address, append CRC and pad.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1100: Replace source address, append VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1101: Replace source address, remove VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1110: Replace source address, replace VLAN tag and replace CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1111: Reserved</td>
</tr>
</tbody>
</table>

Descriptors with the SOP bit set in their transmit status flags must have a nonzero value in this field. If the SOP bit is clear this field must be zero.

In packet FIFO modes all codes are reserved except for:

- 0000: Descriptor is not start of packet.
- 0001: Append CRC.
- 0111: No Modifications.
### CONTROL AND FLAG BITS FOR SYNCHRONOUS SERIAL INTERFACE

#### Table 108: Status Flags for Synchronous Serial Receive Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>55:51</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>crc_error</td>
<td>This bit is set if the received packet has a bad CRC.</td>
</tr>
<tr>
<td>57</td>
<td>abort</td>
<td>This bit is set if the received packet ended with the abort flag.</td>
</tr>
<tr>
<td>58</td>
<td>octet_error</td>
<td>This bit is set if the size of the received packet was not a multiple of 8 bits.</td>
</tr>
<tr>
<td>59</td>
<td>longframe_error</td>
<td>This bit is set if the size of the received packet is bigger than the maximum frame size.</td>
</tr>
<tr>
<td>60</td>
<td>shortframe_error</td>
<td>This bit is set if the size of the received packet is shorter than the minimum frame size.</td>
</tr>
<tr>
<td>61</td>
<td>overrun_error</td>
<td>This bit is set if the received packet over ran the FIFO and is therefore invalid.</td>
</tr>
<tr>
<td>62</td>
<td>good</td>
<td>This bit is set if the received packet has no errors.</td>
</tr>
<tr>
<td>63</td>
<td>SOP</td>
<td>This bit is set to indicate the start of the packet. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.</td>
</tr>
</tbody>
</table>

#### Table 109: Option Flags for Synchronous Serial Receive Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 110: Status Flags for Synchronous Serial Transmit Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>62:51</td>
<td>ignored</td>
<td>Preserved when the descriptor is written.</td>
</tr>
<tr>
<td>63</td>
<td>Flag</td>
<td>This bit is cleared when the descriptor is written. Software can set this bit in descriptors associated with the start of a packet and the bit will be cleared when the packet has been transmitted. (Unlike the Ethernet there is no requirement that this bit be set for a SOP).</td>
</tr>
</tbody>
</table>

#### Table 111: Option Flags for Synchronous Serial Transmit Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>appendCRC</td>
<td>If this bit is set the computed CRC will be appended to the packet.</td>
</tr>
<tr>
<td>2</td>
<td>appendPAD</td>
<td>If this bit is set the packet will be padded to the minimum packet size.</td>
</tr>
<tr>
<td>3</td>
<td>abort</td>
<td>If this bit is set the packet will be ended with an abort instead of a standard flag.</td>
</tr>
</tbody>
</table>
This Page is left blank for notes
DATA MOVER

The Data Mover can perform transfers between arbitrary addresses. The source for a transfer can be either memory or an I/O device, and the destination of the transfer can be either memory or an I/O device. The alignment of the source and destination can differ, in which case the engine will participate in the MESI coherence protocol to ensure the correct merge is done at the start and end of the transfer. The Data Mover has four channels (in many cases this is sufficient for them to be allocated to processors/processes to avoid the need for locking accesses to the control registers). When multiple channels have requests a modified round robin service schedule is applied, this provides fair access to all the channels while optimizing requests to memory to allow open-bank page mode access. The buffering in the Data Mover is sufficient that it can run at the full memory bandwidth.

DATA MOVER OPERATION

The Data Mover channels can only be configured for ring mode. The descriptor A pointer points to the destination of the transfer and the descriptor B pointer points to the source of the transfer. These addresses can have any alignment. There is no buffer size, just the length parameter which may specify up to 1 MByte of data. The Data Mover reads and writes in 32 byte blocks as much as possible, either using the full coherence protocol or as uncached but merged data (similar to the CPU uncached accelerated). If the full coherence protocol is used and a smaller block needs to be written the Data Mover will read the block exclusive, merge in the new data and write the block back. If uncached accelerated mode is used then the byte enable signals will indicate the valid data and the Data Mover does not do a merge.

The source and destination addresses can be incremented, decremented or held constant for each 32 byte block. This allows movement of overlapping regions; if the source address is less than the destination the data should be moved from the top end of the buffer with both addresses decrementing, if the source address is greater then the data should be moved starting at the bottom of the region and incrementing the addresses. Holding the address constant is useful for transfers to or from I/O devices.

If the source and destination move in different directions the Data Mover will not reorder the bytes within the 32 byte line, and the addresses must be cache aligned. In this case if A[4:0] of either address is nonzero the behavior of the move is UNPREDICTABLE.

There is a level 2 cache flag associated with both the source and destination addresses. This is used to request the data be allocated in the L2 cache on a miss. In most cases it would only be used on the destination side (if at all) since there is little point in caching data that is being moved.

One case where the L2 cacheability attribute would be set on the source is when the prefetch bit is set. In this case the destination is ignored and only the reads are done. If the reads are done with the L2 allocation bit set this will prefetch the data into the L2 cache. In uncacheable I/O space this mode could be used to flush an external FIFO.

There is also a zero_mem bit. If this is set the source address is ignored and destination is written with zeros.

If the data mover is used to perform accesses to the HyperTransport space it can only use a subset of reads. For any access the HyperTransport interface will only accept reads that are less than 8 bytes or are aligned to a 4 byte boundary and a multiple of 4 bytes (up to the full 32 bytes in the block), any other sizes will result in UNDEFINED behavior. Thus a move may need to be broken up into the odd bytes needed to align to a 4 byte boundary, then the main block (a multiple of 4 bytes), then the tail bytes. Any size of write will work correctly.
Channels of the Data Mover can be individually enabled and reset. The reset bit should only be written when the channel is currently disabled or being disabled (i.e. it can be written with either the enable or disable command). The act of writing the reset bit causes the current descriptor pointer to be reset to the ring base address. Note that the reset bit for a channel should always be set to initialize the channel when it is enabled for the first time. There are two ways to disable a channel. The first is to clear the enable bit, which causes the engine to complete the transfer that is in progress and stop until re-enabled when it will continue with the next descriptor (unless reset). The second method is to set the abort bit in the same write that clears the enable bit, this causes the current transfer to be abandoned and the engine stops immediately. When re-enabled after being aborted the Data Mover will start with the next descriptor, so the end of the transfer that was aborted will be lost. The active status bit indicates that the channel is currently in use. If the enable bit is cleared the active bit will remain set until the channel is has finished any transfer and is disabled.

The round robin weights are only used when more than one channel is busy. The data mover will service requests from a channel until it has completed the number of transfers in the weight parameter for that channel or there are no descriptors queued. The next channel is then serviced. Note that the weights are by transfer, not by number of bytes transferred. To achieve true bandwidth sharing the transfer sizes need to be similar (as do the latencies for the reads and writes). If it is important for system behavior to have a better sharing of the data mover, software should break down large transfers into a series of smaller transfers to enable finer granularity on the round robin.

The data mover descriptors include an interrupt bit. If this is set then when the transfer specified in the descriptor is complete the interrupt bit will be set in the dm_dscr_base register and the channel interrupt will be raised. If an error is flagged on any data read (either for descriptors or for data blocks) the error bit will be set, the channel interrupt raised and channel will abort and disable. Both the error and interrupt bits are cleared by reads to the dm_dscr_base register, clearing the interrupt.

A high load can be put on the system by the data mover. As soon as the data returns for one of its four outstanding reads it will write the data and issue another read. To prevent this, data transfers can be throttled on either the read or the write (or both), this is a parameter of the transfer set in the descriptor. When enabled the data mover will check the blocking signal from the agent and will only request if the blocker has been clear for the previous four ZBus cycles. To avoid starvation there is a timeout, if the data mover is forced to backoff 16 times (e.g. the blocker deasserts but some other agent inserts a request causing the block to assert in less than four cycles) it will request the next time the blocker is clear.

In parts with system revision indicating PERIPH_REV3 or greater the data mover will prefetch descriptors by overlapping the descriptor fetch with the data transfer of the previous transaction. This improves performance if small transactions are used. A side effect of this is that the current descriptor and count may be one ahead of the transfer in progress (although the count will not go to zero until the channel is idle). If the channel is disabled a prefetched transaction may be completed before the channel goes inactive. An abort will stop the channel immediately and it is UNPREDICTABLE if it stops during the original or prefetched transaction.
CRC AND CHECKSUM GENERATORS

In parts with system revision indicating PERIPH_REV3 or greater the Data Mover includes a programmable CRC generator and a checksum engine that computes the ones-complement checksum used by the TCP and UDP protocols. These can be used as part of a data copy or with the prefetch (null destination) parameter to read the data and generate the result without a copy. Each of the four channels has its own register to store the current state information for the CRC and checksum in progress in that channel, allowing the channels to be operated independently and with arbitrary interleave.

The CRC and Checksum operations are only supported when the addresses increment, the results of enabling CRC or checksum with a decrementing source or destination address are UNDEFINED.

The basic operation is the same for both CRC and checksum. The generator is controlled by the data mover descriptors and a single CRC or checksum computation may be split over multiple descriptors (there can even be unrelated moves that do not use the generators mixed in). For both generators each descriptor includes an enable flag, a reset flag and an append flag. If the enable flag is set then the function is enabled for this move. When the engine is enabled if the reset flag is set then the partial result is set to the initial value before the move is started and if the append flag is set the final result is written to the destination address after the move has completed. In all cases the software readable partial result register is updated at the end of the move. The partial result register may be saved by software if a computation is interrupted and can be written to restore the result before the computation is resumed (this requires control of the DMA channel and in many cases can be avoided since a different computation can be performed in other channels). The data in the partial result register is only valid when the channel is not currently computing a checksum or CRC, and the register should only be written when the channel is not computing.

If the data mover channel is disabled then the CRC and checksum partial result registers will contain valid data when the active bit becomes clear, and the computation may be resumed when the channel is enabled again. If the channel is aborted then the partial result registers and the results of any in-progress generation become UNPREDICTABLE and it is not possible to resume the computation.

Checksum Generation

If the checksum is enabled then for every 16 bits in the move the data is added to the partial sum using the TCP/UDP ones-complement maths. The value that the sum should be initialized to at the start of a computation must be set in the tcpcs_init field, in most cases this will be zero. At the start of moves that have the reset flag set the current sum is set from this initial value. At the start of any move with the checksum enabled that does not have the reset flag set the current sum is updated from the partial sum register for that channel.

At the end of any move that has the checksum enabled the partial sum register associated with a channel is updated. If there is a byte left over and the append bit is set then the final byte and a byte of zeros is added to the partial sum before the result is appended. If there is a byte left over and the append bit is not set then the byte is included in the sum and a flag set to ensure the first byte of the next description is used to complete the 16-bit sum.

When the sum is appended it behaves just as if the move were two bytes longer, and any partial cache line is merged with a read-modify-write cycle if cacheable. If the prefetch bit is set then no regular data is moved and just the checksum is written to the destination (again with a read-modify-write merge if required).
Note that if the prefetch flag is set no data will be moved and the destination address is used just to write the TCP sum if the append flag is set. If the Prefetch flag is clear then data will be moved as the checksum calculation is done and the TCP sum is appended if the append flag is set. If the zero_mem flag is set then zeros are added to the sum, this has no effect unless there is a retained byte from a previous merge to combine in to the sum. If the prefetch and zero_mem flags are both set no memory is zeroed but zeros will be added to the sum, again this will have no effect unless there is a retained byte or a retained byte is created.

CRC Generation

The programmable CRC generator can be used to generate any CRC up to 32 bits (output fieldwidths of 8, 16 and 32 bits are supported). There are two sets of CRC definition registers and each datamove specifies which should be used. The CRC is programmed using a set of parameters:

- **crc_init**: 32 bit initial value
- **crc_poly**: 32 bit polynomial
- **crc_width**: Sets width of output CRC field (32, 16 or 8 bit)
- **crc_txor**: 32 bit value to xor the final CRC
- **crc_bit_order**: Sets the order of bits within a byte for the computation (0 for bit 0-7, 1 for 7-0)

When the CRC width is less than 32 bits the definition should be put in the high end of the registers. Thus for a 16 bit CRC the width is set to 16, the polynomial must be placed in the upper half of the crc_poly field (i.e. bits 63:48 of the `crc_def` register), the initial value will come from the upper half of the crc_init field (i.e. bits 31:16 of the `crc_def` register) and the partial result will be in the upper bits (31:16) of the `crc_partial` register. For an 8 bit CRC the width is set to 8 the polynomial must be placed in the upper byte (63:48) of the crc_poly field, the initial value will come from the upper byte (31:24) of the crc_init field and the partial result will be in the upper byte (31:24) of the `crc_partial` register. For other CRC widths the next larger output field width should be used and the CRC result will be in the upper bits of the field and the upper bits of the registers are used for the definition. In all cases unused bits must be set to zero or the result will be UNPREDICTABLE.

Before the result is written to memory it is XORed with the crc_txor value. Many CRCs require this to prevent all zero data having a zero CRC.

The polynomial arithmetic behind the CRC algorithms splits a message (and the attached CRC) into the coefficients of a polynomial. This is normally done in the order the bits flow serially over the wire. While all interfaces send bytes in the order of ascending memory address, some choose to send the least significant bit of the byte first and some the most significant bit first. This changes the order of the bits needed for the CRC computation. The generator can compute with either order as set by the crc_bit_order flag. There is a similar issue when the CRC result is written to memory. The data is always written in big endian manner (i.e. the high bits of the result get written to the lower memory addresses) to match the final packet being sent in the order of ascending memory address. But again a swap may be needed within the bits. This final swap is specified in the DMA descriptor by setting the crc_xbit flag. This should only be done in the last descriptor of the computation.

| Table 112: Result in memory of appending the result CRC[31:0] |
|---------------|--------|--------|--------|
|               | Addr+0 | Addr+1 | Addr+2 | Addr+3 |
Table 113 gives some example values for some common CRCs (in a particular use care will need to be taken with the final bit order).

**Table 113: Example CRC configurations**

<table>
<thead>
<tr>
<th></th>
<th>crc_width</th>
<th>crc_poly</th>
<th>crc_init</th>
<th>crc_txor</th>
<th>crc_bit_order</th>
<th>crc_xbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC32 (Ethernet)</td>
<td>32</td>
<td>04C11DB7</td>
<td>FFFFFFFF</td>
<td>FFFFFFFF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CRC32C (iSCSI)</td>
<td>32</td>
<td>1EDC6F41</td>
<td>FFFFFFFF</td>
<td>FFFFFFFF</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CRC-CCITT (HDLC)</td>
<td>16</td>
<td>80050000</td>
<td>00000000</td>
<td>00000000</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If the CRC is enabled then for every byte in the move the data is merged in to the partial CRC. At the end of any move that has CRC enabled the partial sum register associated with a channel is updated. At the start of any move with CRC enabled that do not have the reset flag set the current sum is updated from the partial sum register for that channel. At the start of any move that has the reset flag set the current sum is set to the crc_init value for the selected CRC definition. At the end of a move that has the append bit or the crc_xbit flag set the partial sum register associated with the channel is written with the final CRC value after the XOR and bit-flip have been done.

If the append bit is set then the CRC is appended BEFORE the checksum and is included in the checksum. This matches protocols such as iSCSI where the CRC is internal to the TCP payload. The number of bytes appended will be 1, 2 or 4 to match the field width of the CRC. If required a read-modify-write is done. Note that when a checksum is appended it is not included in the CRC.

When the prefetch flag is set no data will be moved and the destination address is used just to write the CRC if the append flag is set. If the Prefetch flag is clear then data will be moved as the CRC calculation is done and the CRC is appended if the append flag is set. If the zero_mem flag is set then zeros are added to the CRC, this will have an effect on the result. If both prefetch and zero_mem flags are set then the zeros will be added to the CRC but no memory will be zeroed.

**Computation Sizes and Bandwidth**

The TCP checksum is done in 16-bit operations, and CRC calculation is done in 12-bit operations so for a full 32 byte cache line 16 or 24 ZBbus cycles are needed. There will be a few clocks overhead, but this gives a theoretical max sum/crc bandwidth of about 6 or 4 Gbps.
Examples

The following examples illustrate uses of the checksum and CRC engine. These are intended to illustrate use of the data mover generators, in some cases it will be more efficient to do more processing of small buffers on the CPU rather than expending more effort on managing descriptors.

![Diagram of checksum computation](image)

**Figure 33: Example 1 - TCP checksum a packet**

Example 1 illustrated in Figure 33 shows preparing a packet for transmission by computing its TCP checksum. The example has a packet made up of a header buffer (with Ethernet, IP and TCP headers) and two data buffers. Prior to passing the descriptors to the data mover software should compute the checksum of the pseudo-header (the source and destination IP addresses, the protocol type and the TCP Length) and write this (rather than zero) to the checksum field in the TCP header. The descriptors are set up so the first resets the sum and adds the bytes in the TCP header. The pseudo-header sum is included (this takes advantage of the fact that the order of the adds does not matter). The second descriptor adds the first data block, and the third descriptor the second data block. Because the Prefetch flag is set the buffers are only read and no copy is made. The third descriptor has the Append CS flag set so the destination address is used to write out the final checksum. The destination points to the checksum field in the TCP header, so the final checksum is automatically written into the packet. If the packet is going to be sent over the Ethernet interface after only a short delay then consideration should be given to setting the l2ca_src flag in all the descriptors so the packets are only read from memory once and staged in the L2 cache for transmission. In this example l2ca_src should certainly be set in the first descriptor since the buffer will be at most two cache blocks and the data mover will soon be doing a read-modify-write to update the two checksum bytes.
Example 2 illustrated in Figure 34 shows preparing an iSCSI packet for transmission. The example has a packet made up of a single buffer, although split buffers could also be used. Again, software must compute the pseudo-header sum and put it in the TCP header checksum field. The first descriptor initializes the TCP checksum and computes the CRC of the iSCSI header information, when this is inserted into the packet it will be included in the TCP checksum. The second descriptor causes the CRC of the data to be computed and inserted. In this case the TCP checksum is just continued. The third move adds the TCP header (and pseudo-header sum) to the checksum and stores the result in the TCP checksum field. (In practice it may be more effective to just use descriptors 1 and 2 and have the checksum appended after the packet by descriptor 2. The software can then collect the checksum and do the pseudo-header and TCP header computation.)

<table>
<thead>
<tr>
<th>Enable CS</th>
<th>Src</th>
<th>Reset CS</th>
<th>Len=iSCSIHdr</th>
<th>Dst</th>
<th>Src</th>
<th>Reset CRC</th>
<th>Src</th>
<th>Len=</th>
<th>Len(data)</th>
<th>Dst</th>
<th>Src</th>
<th>Append CRC</th>
<th>Dst</th>
<th>Src</th>
<th>Append CS</th>
<th>Len=TCPHdr</th>
<th>Dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>EthHdr</td>
<td>IP</td>
<td>TCPHdr</td>
<td>iSCSIHdr</td>
<td>iHdr</td>
<td>CRC</td>
<td>iSCSIData</td>
<td>iData</td>
<td>CRC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 34: Example 2 - Preparing an iSCSI packet

Example 2 can be extended into a more extreme form to also have the Data Mover schedule the packet for transmission. Assume all iSCSI packets get processed through one channel of the Data Mover and they are the only packets added to one transmit channel of a MAC. If the CPU prepares a MAC DMA descriptor at the same time as the data mover descriptors and places the MAC descriptor in the MAC DMA ring (without incrementing the count) then an additional data move could do a single doubleword copy of the value 1 to the MAC DMA descriptor count register (using an uncached destination to ensure only one doubleword is written). This is a somewhat inefficient use of the data mover but has offloaded the CPU from fielding the datamover interrupt and writing the MAC register.

Example 3 illustrated in Figure 35 on page 183 shows preparing a large iSCSI block for fragmentation before transmission. The example has a packet made up of a single buffer, although split buffers could also be used. The data mover is used to compute the iSCSI CRCs and the partial TCP checksums that cover the data in each fragment. The CRCs are inserted into the data block, the checksums are written to a separate array. The first descriptor resets the checksum for the first fragment and computes and inserts the iSCSI header CRC, the appended CRC will be included in the TCP sum. The second descriptor resets the CRC for the data computation, but keeps the checksum running. At the end of the first fragment the TCP checksum is written into the array (the appended checksum is not included in the CRC so this does not disturb the running CRC). Descriptor 3 covers the second fragment which needs its own checksum, but continues the CRC (if there were more fragments this style of descriptor could be repeated). The fourth descriptor covers the final fragment and will append the data block CRC and include this CRC in the final checksum. There is then a problem because
if the checksum were appended in this descriptor it would be put on the end of the buffer after the CRC (software could do this and pick the final checksum from there). In the example this is solved using the observation that adding zeros does not change the checksum. In the example a final descriptor is used to add 2 zeros to the checksum (i.e. a 16-bit halfword of zero) and place the final checksum into the array. The software can combine the partial checksums from the array with the header checksums when it computes the headers prior to transmitting the packets. This example could be modified to a more extreme form in a similar manner to example 2: if the header buffers were prepared by software before the data mover were started then rather than having the TCP checksums written to an array the Append CS would be removed from the second and third descriptors, and additional descriptors inserted after the second and third and to replace the final one. These descriptors would have the CRC disabled and would add and append the TCP sum in the same way as the third descriptor in example 2. With appropriate setup of the MAC DMA ring moving a single doubleword to the MAC DMA count register would allow the packets to be scheduled for transmission.

**Figure 35: Example 3 - Fragmenting an iSCSI packet**
DATA MOVER CONTROL REGISTERS

Table 114: Data Mover Descriptor Base Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>zero</td>
<td>4'b0</td>
<td>These bits must be zero.</td>
</tr>
<tr>
<td>39:4</td>
<td>base</td>
<td>36'bx</td>
<td>This is the base address of the descriptor ring.</td>
</tr>
<tr>
<td>55:40</td>
<td>ring_size</td>
<td>16'bx</td>
<td>This field sets the total number of descriptors in the ring. If this field is set to 0 the ring will contain 65536 descriptors.</td>
</tr>
<tr>
<td>58:56</td>
<td>priority</td>
<td>3'bx</td>
<td>This field gives the weight for this channel. It sets how many descriptors are processed from this channel before advancing round robin to the next channel. 000: 1 descriptor. 001: 2 descriptors. 010: 4 descriptors. 011: 8 descriptors. 100: 16 descriptors. 101-111: Reserved</td>
</tr>
<tr>
<td>59</td>
<td>active</td>
<td>1'b0</td>
<td>Read Only. This bit is set when a descriptor from the current channel is actively being used. When the enbl bit is cleared the channel will remain enabled until this bit is clear.</td>
</tr>
<tr>
<td>60</td>
<td>interrupt</td>
<td>1'b0</td>
<td>Read Only. This bit is set when the channel is interrupting because of the end of a transfer with the descriptor interrupt bit set. This bit is cleared by a read from the dm_dscr_base register.</td>
</tr>
<tr>
<td>61</td>
<td>error (R/O)</td>
<td>1'b0</td>
<td>On a read this bit is set when the channel is interrupting because of a data transfer error (uncorrectable ECC, Bus Error or Fatal bus error signalled on the D.CODE). If such an error occurs the channel will abort. This bit is cleared by a read from the dm_dscr_base register.</td>
</tr>
<tr>
<td></td>
<td>reset (W/O)</td>
<td></td>
<td>If this bit is written with a 1 the current descriptor pointer is reset to the base address of the descriptor ring (bits 39:0). This should always be done the first time a channel is enabled.</td>
</tr>
<tr>
<td>62</td>
<td>abort(W/O)</td>
<td>1'b0</td>
<td>If this bit is written with a 1 the DMA engine will abort the current transfer and disable the channel. If both this bit and the enbl bit (bit 63) are set in the same write, this bit will override and the channel will be aborted.</td>
</tr>
<tr>
<td>63</td>
<td>enbl</td>
<td>1'b0</td>
<td>This bit must be set to enable the DMA channel. If this bit is cleared while the channel is running the current transfer is completed before the engine stops. (Use the abort bit to cause the engine to stop immediately).</td>
</tr>
</tbody>
</table>

Table 115: Debug Data Mover Descriptor Base Address Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>dscr_base</td>
<td>Reading from this register gives the same data as reading from the dm_dscr_base register (Table 114), but without the side effect of clearing the interrupt and error status bits. It is intended for debugger access to the status.</td>
</tr>
</tbody>
</table>
### Table 116: Data Mover Descriptor Count Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>count</td>
<td>16'b0</td>
<td>This is the number of descriptors owned by the DMA engine. Reads will return the number of unused descriptors. Data that is written to this register will be added to the count (assigning that number of additional descriptors to the controller).</td>
</tr>
<tr>
<td>63:16</td>
<td>reserved</td>
<td>48'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 117: Data Mover Current Descriptor Address

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>dscr_addr</td>
<td>40'bx</td>
<td>The current descriptor address can be read from this field.</td>
</tr>
<tr>
<td>47:40</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:48</td>
<td>cur_count</td>
<td>16'b0</td>
<td>The current count of descriptors owned by the DMA engine can be read from this field.</td>
</tr>
</tbody>
</table>

### Table 118: Data Mover CRC Definition Registers (Only if System Revision >= PERIPH_REV3)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>crc_init</td>
<td>32'bx</td>
<td>This is the initial value for the partial CRC and is loaded into the partial result register before a move that has the crc_reset flag set. For CRCs smaller than 32 bits the initial value should be put into the high bits of this field and the low bits should be zeros.</td>
</tr>
<tr>
<td>63:32</td>
<td>crc_poly</td>
<td>32'bx</td>
<td>This is the polynomial used to define the CRC. For CRCs smaller than 32 bits the polynomial value should be put into the high bits of this field and the low bits should be zeros.</td>
</tr>
</tbody>
</table>

---

**Table 116: Data Mover Descriptor Count Register**

- `dm_dscr_count_0` - 00_1002_0B08
- `dm_dscr_count_1` - 00_1002_0B28
- `dm_dscr_count_2` - 00_1002_0B48
- `dm_dscr_count_3` - 00_1002_0B68

**Table 117: Data Mover Current Descriptor Address**

- `dm_cur_dscr_addr_0` - 00_1002_0B10
- `dm_cur_dscr_addr_1` - 00_1002_0B30
- `dm_cur_dscr_addr_2` - 00_1002_0B50
- `dm_cur_dscr_addr_3` - 00_1002_0B70

**Table 118: Data Mover CRC Definition Registers (Only if System Revision >= PERIPH_REV3)**

- `crc_def_0` - 00_1002_0B80
- `crc_def_1` - 00_1002_0B90
### Table 119: Data Mover CRC/Checksum Definition Registers (Only if System Revision >= PERIPH_REV3)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>crc_txor</td>
<td>32’bx</td>
<td>This value is XORed with the partial result of the CRC computation before the final result is written to memory. For CRCs smaller than 32 bits the value should be put into the high bits of this field and the low bits should be zeros.</td>
</tr>
<tr>
<td>47:32</td>
<td>tcpcs_init</td>
<td>16’bx</td>
<td>This is the initial value for the checksum and is loaded into the partial result register before a move that has the tcpcs_reset flag set.</td>
</tr>
<tr>
<td>49:48</td>
<td>crc_width</td>
<td>2’bx</td>
<td>This sets the field width used when the CRC is written to memory:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>00 - 4 bytes (32 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 - 2 bytes (16 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 - 1 byte (8 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>11 - Reserved</td>
</tr>
<tr>
<td>50</td>
<td>crc_bit_order</td>
<td>1’bx</td>
<td>This selects the bit order used when the CRC engine is reading bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 - Bit 0 is used first and bit 7 last</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 - Bit 7 is used first and bit 0 last</td>
</tr>
<tr>
<td>63:51</td>
<td>reserved</td>
<td>13’bx</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

### Table 120: Data Mover Channel Partial Result Registers (Only if System Revision >= PERIPH_REV3)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>crc_partial</td>
<td>32’bx</td>
<td>Current partial CRC result. After a transfer with the crc_ap append bit set this register will contain the final CRC result (after XOR and bit-flip). For CRCs smaller than 32 bits the value is in the high bits of this field. If the value is saved it may be written back later to allow the CRC computation to resume.</td>
</tr>
<tr>
<td>47:32</td>
<td>tcpcs_partial</td>
<td>16’bx</td>
<td>Current checksum partial result. If the value of this field and the odd_byte bit are saved they may be written back later to allow the checksum computation to resume.</td>
</tr>
<tr>
<td>48</td>
<td>odd_byte</td>
<td>1’bx</td>
<td>This bit is set when the current checksum finished on an odd byte boundary. Since the checksum is calculated in 16 bit halfwords this bit being set indicates that the first byte of the next buffer must be added to the low half of the partial result before 16 bit additions are used. If the value of this bit and the tcpcs_partial field are saved they may be written back later to allow the checksum computation to resume.</td>
</tr>
<tr>
<td>63:49</td>
<td>reserved</td>
<td>15’bx</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>
## DATA MOVER DESCRIPTORS

**Table 121: Data Mover Descriptor First Doubleword**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>dst_addr</td>
<td>The destination address of the transfer (may be any alignment).</td>
</tr>
<tr>
<td>40</td>
<td>un_dest</td>
<td>This bit should be set for an uncached destination, and clear if the destination is cacheable coherent. Uncached accelerated operations will be used.</td>
</tr>
<tr>
<td>41</td>
<td>un_src</td>
<td>This bit should be set for an uncached source, and clear if the source is cacheable coherent. Uncached accelerated operations will be used.</td>
</tr>
<tr>
<td>42</td>
<td>interrupt</td>
<td>If this bit is set an interrupt will be generated at the end of the transfer.</td>
</tr>
<tr>
<td>43</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>45:44</td>
<td>dir_dest</td>
<td>This indicates the direction the destination address should be moved:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00: The address is incremented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01: The address is decremented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10: The address is held constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11: Reserved</td>
</tr>
<tr>
<td>47:46</td>
<td>dir_src</td>
<td>This indicates the direction the source address should be moved:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00: The address is incremented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01: The address is decremented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10: The address is held constant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11: Reserved</td>
</tr>
<tr>
<td>48</td>
<td>zero_mem</td>
<td>If this bit is set the source parameters will be ignored, and zeros will be transferred to the destination address.</td>
</tr>
<tr>
<td>49</td>
<td>prefetch</td>
<td>If this bit is set the destination parameters will be ignored, the data will just be read from the source. This can be used with the L2C_SRC bit to prefetch lines from memory into the L2 cache. Note: when used in this manner, it is most efficient if the src address is 32 byte aligned.</td>
</tr>
<tr>
<td>50</td>
<td>l2c_dest</td>
<td>If this bit is set writes to the destination buffer are marked L2_cacheable and will therefore be allocated in the L2 on an L2 miss. (For cacheable transfers the L2 is always checked, and will always be updated if hit).</td>
</tr>
<tr>
<td>51</td>
<td>l2c_src</td>
<td>If this bit is set reads from the source buffer are marked L2_cacheable and will therefore be allocated in the L2 on an L2 miss. (For cacheable transfers the L2 is always checked, and will supply the data if hit).</td>
</tr>
<tr>
<td>52</td>
<td>rd_bkoff</td>
<td>If this bit is set the data mover will backoff from a read transfer until the destination blocker has been clear for four ZBbus cycles.</td>
</tr>
<tr>
<td>53</td>
<td>wr_bkoff</td>
<td>If this bit is set the data mover will backoff from a write transfer until the destination blocker has been clear for four ZBbus cycles.</td>
</tr>
<tr>
<td>54</td>
<td>tcpcs_en</td>
<td>Set to enable TCP checksum (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>55</td>
<td>tcpcs_res</td>
<td>Set to reset TCP partial sum to init value at start of move (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>56</td>
<td>tcpcs_ap</td>
<td>Set to append TCP checksum at end of move (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>57</td>
<td>crc_en</td>
<td>Set to enable CRC engine (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>58</td>
<td>crc_res</td>
<td>Set to reset partial CRC to init value at start of move (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>59</td>
<td>crc_ap</td>
<td>Set to append CRC at end of move, but before checksum (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>60</td>
<td>crc_dfn</td>
<td>Selects which TCP/CRC definition to use (System revision PERIPH_REV3 or greater)</td>
</tr>
<tr>
<td>61</td>
<td>crc_xbit</td>
<td>Set to swap the bits within the CRC bytes prior to checksumming and appending to the packet and/or updating the partial register. Typically only set for last block of CRC (if the CRC needs the bitflip). (System revision PERIPH_REV3 or greater)</td>
</tr>
</tbody>
</table>
### Table 121: Data Mover Descriptor First Doubleword (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:62</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 122: Data Mover Descriptor Second Doubleword

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>39:0</td>
<td>src_addr</td>
<td>The source address of the transfer (may be any alignment).</td>
</tr>
<tr>
<td>59:40</td>
<td>length</td>
<td>The length of the transfer in bytes. If this field is zero then $2^{20}$ bytes will be transferred.</td>
</tr>
<tr>
<td>63:60</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Section 8: PCI Bus and HyperTransport Fabric

INTRODUCTION

The PCI bus and HyperTransport (formerly called “Lightning Data Transport” or “LDT”) fabric provide the main general expansion buses for the BCM1250 and BCM1125/H parts. PCI is the standard bus used for peripheral connection in many systems and there are a wide variety of peripheral devices that connect to it. HyperTransport (HT) is a high performance replacement for PCI on system boards, it uses a chain of unidirectional point to point links to form a fabric of devices running at much higher data rate. The HyperTransport devices use the standard PCI configuration process, so devices can be migrated from the PCI to HyperTransport with few software changes. Initial devices available for the HyperTransport fabric include bridges to PCI, PCI-X and AGP. The BCM1250 and BCM1125H have both HT and PCI interfaces, the BCM1125 has only PCI.

The PCI interface conforms to revision 2.2 of the PCI standard. It is a 32 bit wide interface, can run up to 66MHz and uses 3.3V signal levels (the interface is not 5V tolerant). PCI special cycles, dual address cycles and the LOCK protocol are not supported. The interface can act as either the host bridge or as a device configured and controlled by another host. These are referred to as Host Mode and Device Mode. When operating in Host Mode an internal arbiter can be used, providing support for four external PCI devices; alternatively an external arbiter can be used to allow more external devices.

The HyperTransport interface conforms to revision 1.03 of the HyperTransport specification although to ensure software backwards compatibility with the earlier implementation (based on 0.17 of the HT specification) there are a few differences, mainly in the mechanisms for error reporting (this was not specified in the earlier version of the standard). The HyperTransport transmit and receive clocks are usually the same frequency (the receive link frequency may be lower than the transmit frequency), generated from the 100 MHz reference clock. The HyperTransport transmit clock can be the reference clock x2, x3, x4, x5 or x6. The interface runs 8 bits wide in each direction, with data sent on both edges of the clock. The HyperTransport interface runs as a host bridge, and therefore is always on one end of the HyperTransport fabric. Support is provided for fabrics with host bridges at each end.
Figure 36 shows the physical and logical organization of the PCI and HyperTransport interfaces.

Physically the PCI and HyperTransport interfaces are both connected to ZBbus through I/O bridge 0 (as shown on the left in Figure 36). The I/O bridge implements the ZBbus protocol, and includes the address and data buffers for transactions between the ZBbus and each interface, the buffers for peer-to-peer transfers between the PCI and HyperTransport, and the buffer for merging partial cache line writes. The PCI and HyperTransport interface units implement their respective bus protocols, and include additional data and command buffering. The bandwidths of the connections between the I/O bridge and interface units are sized appropriately for the interfaces, so the HyperTransport is not constrained by the PCI bandwidths.

However, the interfaces are presented logically to the system as if the HyperTransport was bridged from the PCI bus (shown on the right of Figure 36). This makes the PCI always bus zero to the device enumeration code (this is required by some software) and allows configuration to be based on existing code.
PCI AND HYPERTRANSPORT ADDRESS RANGE

This section describes how devices on the PCI bus and HyperTransport fabric (and any buses bridged from them) are mapped into the memory space. There are two complexities in the mapping. The first is supporting the different types of access that need to be done, and the second is supporting endian swapping when the part is running as a big endian system (both PCI and HyperTransport are little endian). A full discussion of the endian issues is in Section: “Endian Policies” on page 201, the two endian options match bit lanes and match byte lanes are defined there.

All the areas for mapping PCI and HyperTransport devices, except a HyperTransport only expansion area, are put in the low 4 GBytes of the address space so they can be addressed using 32 bits of address in systems that only use 32 bit addressing.

Accesses to the PCI and HyperTransport ranges of the address map should be done as uncacheable or cacheable non-coherent. These addresses pass outside the coherence domain of the system. If cacheable coherent accesses are made to the PCI or HyperTransport space the behavior of the system will become UNDEFINED.
The PCI and HyperTransport portions of the memory map are shown in Figure 37.

Figure 37: Address Ranges for CPU Access to PCI and HyperTransport
MEMORY MAPPED DEVICES

There is a 512MB region for mapping memory mapped PCI and HyperTransport devices. Logically this all maps to the PCI and a segment of it is bridged to the HyperTransport. (On the BCM1125 there is no HyperTransport interface and all accesses to this space are PCI accesses.) Thus the area is divided up into the three regions A, B and C in the address map. Region A contains all the devices on the PCI bus (including bridges) that the configuration code allocated before the HyperTransport bridge. Region B contains the addresses bridged to the HyperTransport fabric, accesses to this address range are detected by the I/O Bridge 0 and sent to the HyperTransport controller directly. Region C consists of the address range allocated to devices on the PCI bus that were configured after the HyperTransport bridge and empty space that was not allocated. When the part is running in big endian mode this area of memory accesses the PCI and HyperTransport using the match byte lane policy.

Region D (offset from the main area by setting address bit 29) is directly mapped onto the A, B and C regions, but uses the match bit lane policy.

HYPERTRANSPORT EXPANSION SPACE

There is a large region of the address space allocated to HyperTransport expansion. This is marked as region N in the address map above (Figure 37 on page 193). This space is Reserved on the BCM1125. Use of this area requires the use of full 40 bit physical addresses. Configuration software will also need modification to use this area (by default it will attempt to put all HyperTransport peripherals in region B since they will seem to be bridged from the PCI bus). The endian policy used in the expansion space is determined by the exp_endian bit in the SRI Command Register. If this bit is clear then all of the expansion space will use the match byte lane policy. If the bit is set then address bit [38] is used to select the endian policy and the address bit is cleared as the request passes through the bridge. Thus:

- exp_endian bit clear:
  - 80_0000_0000 - F7_FFFF_FFFF use the match bytes policy.

- exp_endian bit set:
  - 80_0000_0000 - BF_FFFF_FFFF and E0_0000_0000 - F7_FFFF_FFFF use the match bytes policy.
  - C0_0000_0000 - DF_FFFF_FFFF map to 80_0000_0000 - 9F_FFFF_FFFF and use the match bits policy.

CONFIGURATION SPACE

Configuration cycles can be performed on the PCI bus and HyperTransport fabric by accessing the configuration space. Again, with address bit [29] clear the match byte lane policy is used and with a[29] set the match bit lane policy is selected. See Section: “Configuration of PCI and HyperTransport” on page 234 for a full discussion of configuration.
PCI I/O SPACE

PCI, and for compatibility HyperTransport, has both memory space and I/O space corresponding to the standard and I/O instructions in the x86 architecture. Most PCI devices are mapped into memory space, as memory mapped I/O devices (indeed the PCI standard encourages this) but some use I/O space addresses. The MIPS architecture has no distinction between memory and I/O address ranges, so a range of memory addresses has been defined to map to PCI I/O space.

There is a 32 MB region allocated for I/O space transactions, covering regions H, I, J and K in the address map (Figure 37). When the part is running in big endian mode these regions use the match byte lane endian policy. The I/O address consists of only the low 25 bits of the address, for a 32 bit PCI I/O address bits 31:25 are set to zero. Region H is special and is discussed in the next section. Region I is the space allocated to I/O devices on the PCI that are configured before devices on the HyperTransport, these are accessed using PCI I/O reads and writes. Region J is allocated to devices that are behind the HyperTransport bridge and use I/O addresses, these are accessed by adding the 25 bit I/O address to the base address (FD_FC00_0000) of the special area on the HyperTransport defined for signalling I/O transactions. Region K covers any PCI devices that were configured after the HyperTransport ones.

Region M (offset from the main I/O space area by setting address bit 29) is directly mapped onto the H-K regions, but uses the match bit lane policy.

THE SOUTHBRIDGE, VGA AND SUBTRACTION DECODE

In most x86 systems there is a device called the Southbridge that connects to the PCI bus and provides the interface to a number of slow speed I/O devices (serial ports, parallel ports, keyboard, mouse, more recently USB) and legacy buses (normally the ISA bus), it also contains the legacy interrupt controller (often known as the PIC, after the original Peripheral Interrupt Controller chip).

The interface supports the use of a single southbridge device, which can either be on the PCI bus, bridged from the PCI bus, on the HyperTransport fabric or bridged off the HyperTransport fabric.

There are two problems with addressing I/O devices in a legacy southbridge:

- The I/O addresses of peripherals in the southbridge are normally at the location they have historically been. These addresses are not contiguous, so do not fit well into the PCI address allocation model.
- If the southbridge connects to an ISA bus, there is no easy way to know the memory or I/O addresses of devices on the ISA bus.

On PCI buses subtractive decode is used to access these addresses. Any address that is not claimed by a device on the bus is claimed by the southbridge and used by its internal devices or passed to the ISA bus. As of PCI 2.2 the southbridge may be behind a (or a series of) PCI-PCI bridge(s), the configuration software notices this and sets up all the bridges on the path to act as subtractive decode bridges.

On HyperTransport there is no way to do subtractive decode. Normally commands whose address is not actively decoded and claimed will reach the end of the fabric and an NXA error will be raised (revision 1.0 and later of the HyperTransport standard allows the device at the end of the chain to perform a subtractive decode). Instead, the standard allows packets to be marked with the COMPAT bit, this overrides the active address decoding and will always pass the access to the southbridge (for subtractive decode).
There is a reset time configuration bit southOnLDT (on generic bus IO_AD[21]) that sets whether the route to the southbridge is via the PCI bus or HyperTransport fabric. This is done as a hardware configuration option to allow use of the southbridge before the PCI bus and HyperTransport fabric have been configured (this is in line with section C.3 of the HyperTransport specification rev 0.17, and section E.3.1 of the HyperTransport specification rev 1.0). The PCI and HyperTransport device enumeration will discover the southbridge as it configures the system and can check that the southOnLDT bit is set correctly (if it is not correct there is a serious hardware error, or an option card has been added with a new southbridge -- both these events should be reported). If the southbridge is reached through the HyperTransport bridge then the configuration software should check (and possibly configure, if they are not also setup in hardware) any HyperTransport-HyperTransport and HyperTransport-PCI bridges on the route to accept and forward packets with the COMPAT bit set (and from there on set any PCI-PCI bridges to do subtractive decode). If the southbridge is on the direct PCI bus or is bridged from there by anything other than the internal HyperTransport bridge (which logically appears as if on the PCI), then the southOnLDT bit must be clear (and any bridges off the PCI must be set to do subtractive decode).

Since subtractive decode is only used for legacy devices it will only be used for I/O addresses in the bottom of the range. In this interface, locations in the first 32 KB of the I/O address space can only be used for the legacy subtractive decode devices. This area is shown as region H in Figure 37 on page 193. Any access to this range will be directed according to the southOnLDT bit, if it is set they will be sent out as HyperTransport requests with the COMPAT bit set, if it is clear they will be sent to the PCI for subtractive decode. (On the BCM1125 the access is always made to PCI.)

Similarly, the bottom 16 MB of the memory space region is used for subtractive decode only. If the southOnLDT bits is set any accesses in this region will be sent on the HyperTransport bus as a 24 bit address with the COMPAT bit set, otherwise they are sent on the PCI for subtractive decode.

The other legacy device that the PCI and HyperTransport specifications have as special is the VGA display controller. This has historically used memory addresses 0A_0000-0B_FFFF in the address map and registers X3B0-X3BB and X3C0-X3DF in I/O space (per PCI Spec AD [15:10] are ignored for this decode). These addresses are in the compatibility regions. Rather than being routed based on the SouthOnLDT configuration bit, the VGA range is routed using the VgaEn bit in the HyperTransport Bridge Control Register. If the VgaEn bit is clear (indicating the bridge should not forward VGA accesses) VGA accesses are done to the PCI bus, if the VgaEn bit is set the access is sent to the HyperTransport Fabric but the COMPAT bit is not set allowing the request to be routed by address to the VGA controller. As with all the compatibility space accesses the address is masked to a 24 bit address with the upper bits zero.
Compatibility space routing is summarized in the pseudo-code below:

```c
if (00_4000_0000 <= zbaddr <= 00_7FFF_FFFF)
{
    pcildtaddr = zbaddr & DFFF_FFFF
    endian = zbaddr[29] ? match bit lanes : match byte lanes
    if (pcildtaddr < 00_4100_0000) /* Note after a[29] removal */
    {
        if (pcildtaddr[23:17] == 7'b0000101) /* 0A_xxxx/0B_xxxx */
        {
            if (Bridge Control VgaEn)
                send (pcildtaddr & 00FF_FFFF) to HT with COMPAT bit clear
            else
                send (pciaddr & 00FF_FFFF) to PCI
        }
        else
        {
            if (Southbridge is on LDT)
                send (pcildtaddr & 00FF_FFFF) to HT with COMPAT bit set
            else
                send (pciaddr & 00FF_FFFF) to PCI
        }
    }
    else if (HT mem base <= pcildtaddr <= HT mem limit)
        send to HT
    else
        send to PCI
}
```
For I/O space:

```c
if (0x00_4000_0000 <= zbaddr <= 0x7FFF_FFFF)
    ioaddr = (zbaddr & 0xFF_FFFF)
    endian = zbaddr[29] ? match bit lanes : match byte lanes
if (ioaddr < 0x000_8000)
    {  
        if ((ioaddr[24:16] == 0) &&
            ((ioaddr[9:4]==3B) && (ioaddr[3:2]!=2'b11)) ||
            (ioaddr[9:5]==5'b11110)
        )
        { /* 3b0-3bb or 3c0-3df */
            if (Bridge Control VgaEn)
                send to HT (FD_FC00_0000 + ioaddr) with COMPAT bit clear
            else
                send to PCI ioaddr as I/O access
        }
        else /* ioaddr >= 000_8000 */
        {  
            if (HT bridge header io base <= ioaddr <= HT header io limit)
                send to HT (FD_FC00_0000 + ioaddr)
            else
                send to PCI ioaddr as I/O access
        }
    }
else /* ioaddr >= 000_8000 */
    {
        if (Southbridge is on LDT)
            send to HT (FD_FC00_0000 + ioaddr) with COMPAT bit set
        else
            send to PCI ioaddr as I/O access
    }
```

**HYPERTRANSPORT END OF INTERRUPT (EOI) SIGNALING SPACE**

The HyperTransport interrupt scheme is discussed in detail in Section: “HyperTransport Interrupts” on page 48, which describes all the interrupts on the part. The HyperTransport level sensitive interrupt messages require an End Of Interrupt (EOI) acknowledgement to clear them when the interrupt processing is complete. This is sent as a broadcast message in the HyperTransport address range reserved for interrupt messages.

The interface allows an EOI to be generated by doing a write to the EOI section of the address space, which is marked as region E on the address map (Figure 37 on page 193). Bits 23:16 of the address accessed should be the vector number being acknowledged, bits 15:5 should be zero, bits 4:2 are the message type (MT) field and must be set to 3'b111 to indicate an EOI. The data written is discarded. (Any access to this region will be converted into the broadcast, but it is inadvisable to do a read to generate the EOI since the data returned will be UNPREDICTABLE and since the processor would be doing an uncached access it is likely to stall the CPU for no good reason).
In addition to generating EOI this space can be used to generate HyperTransport interrupt messages. If bits [4:2] of the address are not equal to 3'b111 (i.e. the message is encoding something other than an EOI) rather than sending a Posted Broadcast message the bridge will send a Posted Sized Byte Write with a count of zero, the COMPAT bit clear and a doubleword of zeros as byte masks (the extended destination bits from rev 1.01 and later of the HyperTransport standard are not supported). This is the format of an interrupt message. The address bits 23:16 indicate the vector, bits 15:8 the destination, bit 7 is Reserved and should be zero, bit 6 indicates the destination mode, bit 5 should be zero to indicate an edge interrupt (level cannot be used because the interface will not accept the inbound EOI), and bits 4:2 are the interrupt type.

**LEGACY INTERRUPT ACKNOWLEDGE (IACK) SPACE**

Legacy interrupts from a PIC style interrupt controller in the southbridge require an interrupt acknowledgement (IACK) cycle, this is a byte read that will return the interrupt source vector number. As with the subtractive decode I/O space, the part supports the southbridge being on either the HyperTransport bus or the PCI as set by the southOnLDT configuration bit.

If the southbridge is on the PCI then it will signal the interrupt using its INTR pin, which should be routed to one of the general interrupt inputs. If the southbridge is on the HyperTransport then it signals the interrupt using an Ext. Int. message, which is translated into interrupt 54 being raised in the CPU interrupt mappers (see Section: “HyperTransport Interrupts” on page 48). In either case the interrupt must be acknowledged with an IACK cycle. This is a byte read that returns the interrupt vector information. The address map defines the region F (in Figure 37 on page 193) for the CPU to read from to perform an IACK access; the address is ignored and any access in this range has the same effect. If the southOnLDT bit is set this gets run as a cycle on the HyperTransport to the special address range (FD_F900_0000 - FD_F90F_FFFF) with the COMPAT bit set. If the southOnLDT bit is clear, the read gets run as an IACK cycle on the PCI bus.

**PCI FULL ACCESS SPACE**

Regions O and P in Figure 37 on page 193 are provided to allow the CPU to generate any memory space address on the PCI bus. When the interface is configured in Device Mode the address map used on the PCI bus matches the host system and not the internal address map. The full access space allows the device to access any host system address, as would be expected for a PCI peripheral. The bottom 32 bits of the address are used directly to form the PCI bus address. Address bit [32] is used to select between the two regions and therefore the endian policy used for accesses. The full access space is available when the interface is in Host Mode, but it is less useful. The PCI interface will not respond to requests that it generates, any access through Full Access space to an address with a destination in the part will result in a master abort.

There is no equivalent I/O full access space, since the normal I/O space can be used in Device Mode to generate 25 bit I/O addresses on the PCI bus.

**SPECIAL HYPERTRANSPORT SPACE**

The HyperTransport defines a range of address space for other special cycles. None are directly supported by the interface, however access is provided to the special range using region G in Figure 37 on page 193. Use of addresses in this region requires understanding of the HyperTransport specification. Since no assistance is provided for the special use, accesses in this region may not be useful, and can result in UNDEFINED behavior. Address bit [29] is used to select between the endian policies.
HYPERTRANSPORT READ RESTRICTIONS

The HyperTransport interface maps read requests from the ZBbus into read commands on the HyperTransport fabric. The interface supports all requests that can be generated by CPU instructions (reads of 1-8 bytes and 32 bytes) and any request that includes an aligned number of 32-bit words. Care must be taken when using uncached accelerated accesses from the CPU or the Data Mover, since these can generate accesses that are not supported and will have UNDEFINED results. For best performance the ZBbus read should translate into a single HyperTransport request, accesses that are 1-4 bytes translate directly to a Read (sized) Byte request, and accesses that are an aligned number of 32-bit words translate to the Read (sized) Doubleword request. The interface is able to split ZBbus reads into two HyperTransport requests allowing the 5-7 byte reads that the load doubleword left (LDL) and load doubleword right (LDR) instructions can generate, but the interface is not optimized to give these unusual accesses high performance.

There is no constraint on writes, the interface supports arbitrary writes of 1-32 bytes. Any ZBbus write can be converted into a Write (sized) HyperTransport request.

There are no constraints on inbound accesses. All legal HT read and write requests are supported. (Requests larger than 32 bytes will be converted into two requests to the ZBbus but this is transparent to the HT fabric.)
ENDIAN POLICIES

The system can be run either as a big endian system or as a little endian system. Both the PCI bus and HyperTransport fabric are little endian. When the part is running big endian there are two policies that are used for connecting to the interfaces.

Selection of endian policy is made based on address bits for requests going both from the system to the PCI bus or HyperTransport fabric and from the PCI bus or HyperTransport fabric to the system. Peer-to-peer accesses between the PCI bus and HyperTransport fabric are always passed directly. In most cases address bit [29] being clear indicates that the match byte lane policy will be used, and address bit [29] being set indicates the match bit lane policy will be used. The address bit used to select the endian mode is zeroed as the request passes through the interfaces so that the target always see the match bytes’ address.

LITTLE ENDIAN SYSTEM: NO SWAPS

When the part is run as a little endian system there is no need to do any swapping between the system and PCI or HyperTransport. If the system configuration register is set for little endian then no swapping is done, and the two access addresses become aliases.

This is illustrated in Figure 38. At the top this shows a double-word with the bit numbers and byte addresses used by the CPU and ZBbus. Below it shows how the bytes of the double-word will appear on the PCI byte lanes. The PCI uses byte enables to indicate both the transfer size and the low two address bits, these are shown as the BE#[3:0] signals.
**BIG ENDIAN SYSTEM: MATCH BYTE LANES**

The match byte lanes endian policy will match the byte lanes of 32 bit values on either side of the interface. This preserves the memory address ordering between the ZBbus and PCI or HyperTransport. However, it will scramble the bit ordering. Consequently a 32 bit value that is written into a PCI register from the CPU will have a different interpretation.

The policy is illustrated in the Figure 39. This is similar to Figure 38 on page 201, except that from the CPU the mapping from bytes of the double-word onto addresses is reversed. When the data is passed to the PCI/HyperTransport the address order is maintained and the low two address bits directly set the PCI byte enables. The little endian nature of the PCI is exposed to the processor. If a value is stored from a CPU register into a peripheral device register its bytes will be reversed.

![Figure 39: Match Byte Lane Endian Policy](image)

If a sequence of bytes is moved through the interface the address of each byte is maintained, so the order of the sequence is maintained between the PCI or HyperTransport device and memory. Therefore this is the correct endian policy to use for most DMA transfers.

This policy is also the correct one to use for configuration and control register accesses if the software is written to explicitly endian swap the values in big-endian systems. However, since most drivers will be ported from little endian systems they are unlikely to have the endian swap code. It will normally be better to avoid the software overhead and use the match bits policy for control functions.
**BIG ENDIAN SYSTEM: MATCH BIT Lanes**

The match bit lanes endian policy will match the bit numbers of 32 bit values on either side of the interface. In byte lane terms this is an endian swap of the data. Consequently a 32 bit value that is written into a PCI register from the CPU will have the same interpretation. This policy is therefore the correct one to use when setting up configuration registers, passing addresses, and most control functions.

The policy is illustrated in Figure 40. Compared to Figure 39 on page 202, the bit ordering is maintained going from the system to the PCI. The PCI byte enables are generated from the inverse of the bottom two system address bits reflecting the different byte significance between the two sides of the interface.

![Figure 40: Match Bit Lane Endian Policy](image)

By matching the bit number in 32 bit words, the bits that the CPU uses will match with the register descriptions in the PCI peripheral data sheet. Similarly, an address written into a control register from a CPU register will have the same meaning. Note that the situation is more complicated for 64-bit accesses (see below), but these are forbidden by the HyperTransport specification for configuration accesses to the HyperTransport space, and forbidden by the bridge for accesses to the PCI configuration registers.

If a byte access is performed the programmer will get the expected result for a big-endian context, i.e. the most significant bits of the register will be at the lowest CPU byte address. Similarly a 16 bit access will be mapped in the way expected for a big-endian machine. A sixty-four bit access performed by the CPU will be split into two thirty two bit accesses, again the data will end up in the positions consistent with a big-endian model.

The problem with this policy is that the byte memory ordering is scrambled since the byte lanes are swapped. This affects bulk data transfer, where the device on the PCI or HyperTransport assumes that the order on the byte lanes matches the memory order with BE#[0] associated with the lowest memory address. Therefore this mapping is unlikely to be useful for devices doing DMA accesses into the system memory. It is useful for access to control registers (where it will save endian swapping in software). The match bits policy can also be used if the peripheral can be configured for big endian DMA operation.
As an example consider accessing control registers with the same layout as the standard PCI header (this layout is used only as a familiar one for the example, the interface does not allow 64 bit accesses in configuration space so this cannot be used directly). Looking at the fields at offsets +0 and +4 with a big endian viewpoint the expected behavior for a 64 bit read should give the data at low memory address in the high (big) end of the register:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:48</td>
<td>Device ID</td>
<td></td>
</tr>
<tr>
<td>47:32</td>
<td>Vendor ID</td>
<td></td>
</tr>
<tr>
<td>31:16</td>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>15:0</td>
<td>Command</td>
<td></td>
</tr>
</tbody>
</table>

This is what a 64 bit read using the Match Bits policy will give. Looking at a PCI 64 bit BAR instead of the control registers (or equivalently a 64 bit DMA address register) shows that 64 bit accesses will not work directly for access to 64 bit addresses in HyperTransport peripherals. A 64 bit BAR would span two 32 bit words, but has the low address bits in the first word and the high address bits in the second word. Looking at this picture with the big endian viewpoint (as above) the 64 bit read will give:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:32</td>
<td>Low address bits</td>
<td></td>
</tr>
<tr>
<td>31:0</td>
<td>High address bits</td>
<td></td>
</tr>
</tbody>
</table>

This is word swapped compared to what would be ideal, but is still consistent with the 64 bit world-view because it was originally defined as two 32 bit quantities.

**VIEWING ENDIAN POLICY AS AN OPTIMIZATION**

One way to view the endian policy selection is as a way to optimize code. This section considers the software viewpoint assuming a portable operating system is being used.

To summarize the issue: The PCI/HyperTransport bus and peripherals are little endian, thus when storing a 32 bit value in a register the least significant byte (i.e. bits [7:0]) are assigned to the lowest memory address. When an access is done from a big endian CPU, which stores a 32 bit value with the most significant byte in the lowest memory address, the data appears swapped. Consider a 32 bit control register in the PCI device that contains the 32 bit value 32'h12345678, on the PCI bus the byte containing '78' will be on the byte lane with the lowest memory address marked with CBE[0] asserted. If the CPU does a read it will place the lowest memory address into the high bits of the register so will see the 32 bit value 32'h78563412.

In most multi-platform operating systems the first cut way of fixing this is to put an endian swap in if required:

```c
#ifdef BIG_ENDIAN /* Need to swap the bytes */
   result = endian_swap(*pci_reg_address);
#else /* on a LITTLE_ENDIAN system can read directly */
   result = *pci_reg_address;
#endif
```

This will work as expected using the Match Bytes access area of the PCI/HyperTransport regions. The Match Bits space allows the hardware to automatically do the swapping. A single physical address bit (bit [29] in the standard memory mapped I/O space) is set to switch from Match Bytes to Match Bits. So the code can become:

```c
#ifdef BIG_ENDIAN /* Get hardware to swap the bytes */
   result = *(pci_reg_address | BYTES_TO_BITS_ADDRESS_BIT);
#else /* on a LITTLE_ENDIAN system can read directly */
   result = *pci_reg_address;
#endif
```
This is an optimization because the endian swap is several instructions compared to the single OR (and in many cases all of the control registers need the swap so the base address of the control block can just be offset once; or if all the control registers are mapped in a single TLB entry and the data accesses in another then the virtual to physical mapping could set the Match Bits address bit for control and not for the data ranges).

The standard address is the Match Bytes since this makes memory order correct and thus causes unoptimized (but endian aware) code to work correctly.

**ACCESSING THE SiByte FROM PCI DEVICES**

The normal PCI Base Address Register (BAR) access scheme is used to accept accesses from the PCI bus and map them to internal accesses. The PCI bus uses 32 bit addresses. To allow PCI masters to do transfers to memory, other devices in the part and peer-to-peer transfers to devices on the HyperTransport a full 40 bit address must be constructed. With the exception of expansion memory space and expansion HyperTransport space the part places everything in the low 4G of the address space (physical address bits [39:32] are zero) allowing a direct map from the 32 bit address on the PCI bus. In addition there is a map table that allows each of sixteen 1 MB regions of PCI space to map to a full 40 bit address anywhere in the internal or HyperTransport space.

The PCI controller has a type 0 (device) configuration header. This allows for up to 6 BARs, 5 are used when the interface is in Host Mode and 3 are used when the interface is in Device Mode. In addition a PCI expansion ROM may be accessed through its special BAR.

Table 123 shows the BARs and the internal addresses that they map to. In Host Mode they default to the values shown and are enabled following reset (if transparent access from PCI is all that is needed then no changes are required by the configuration code). When the interface is run in Device Mode the BARs will be setup by the host (using addresses in the host space). Note that the table shows the addresses. Since the spaces are prefetchable the value in the PCI BAR register will have bit 2 set.

**Table 123: PCI Base Address Register Use**

<table>
<thead>
<tr>
<th>Reg</th>
<th>Offset in Config Header</th>
<th>PCI Size</th>
<th>Default PCI Base Address in Host Mode</th>
<th>PCI Base Address in Device Mode</th>
<th>Internal Base Address</th>
<th>Internal Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAR0</td>
<td>+10</td>
<td>16M</td>
<td>6000_0000 (R/O)</td>
<td>xx00_0000</td>
<td>Table Lookup</td>
<td>Region controlled by map table.</td>
</tr>
<tr>
<td>BAR1</td>
<td>+14</td>
<td>0</td>
<td>0000_0000</td>
<td>0000_0000</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>BAR2</td>
<td>+18</td>
<td>4K</td>
<td>7000_0000 (R/O)</td>
<td>xxxx_x000</td>
<td>00_1002_1xxx</td>
<td>Mailbox CPU 0. Match Bytes mode is used if the system is configured BigEndian.</td>
</tr>
<tr>
<td>BAR3</td>
<td>+1c</td>
<td>4K</td>
<td>7100_0000 (R/O)</td>
<td>xxxx_x000</td>
<td>00_1002_3xxx</td>
<td>Mailbox CPU 1. Match Bytes mode is used if the system is configured BigEndian.</td>
</tr>
<tr>
<td>BAR4</td>
<td>+20</td>
<td>1G</td>
<td>0000_0000 (R/O)</td>
<td>Reserved</td>
<td>00_0000_0000</td>
<td>Memory below PCI space (PCI a[29] indicates endian policy).</td>
</tr>
<tr>
<td>BAR5</td>
<td>+24</td>
<td>2G</td>
<td>8000_0000 (R/O)</td>
<td>Reserved</td>
<td>00_8000_0000</td>
<td>Memory above PCI space (PCI a[29] indicates endian policy).</td>
</tr>
<tr>
<td>ROM</td>
<td>+30</td>
<td>64K</td>
<td>7300_0000 (R/O)</td>
<td>xxxx_0000</td>
<td>00_1FD0_xxxx</td>
<td>Expansion ROM. Match Bytes mode is used if the system is configured BigEndian.</td>
</tr>
</tbody>
</table>
Addresses on the PCI that match in BAR0 are mapped into internal addresses using a mapping table illustrated in Figure 41 on page 206. This allows a PCI device to access any area of in the internal address map, and can be used to ensure PCI devices can only access specified memory areas. The 16 MB region is divided into 1 MB chunks. PCI Address bits [23:20] are used to index into a table which provides address bits [39:20] of the internal address to use and an enable bit. The low bits [19:0] for the internal address are copied directly from the PCI address. If the table entry is enabled then the request on the PCI bus is accepted (DEVSEL# asserted) and the transaction is made to the internal address. If the table entry is not enabled then the bus request will not be accepted (DEVSEL# will not be asserted) and the requester will see an error. There is also a bit in the entry that controls whether the access is sent to the HyperTransport fabric or used internally (this bit overrides the destination implied by the address, so software must take care to set it correctly). If the bit is set to send the request internally and the address is in the HyperTransport range the result is UNPREDICTABLE, but in most cases a target abort error will be returned. If the bit is set to send the access to the HyperTransport and the address is in the internal range then the result depends on the devices on the HyperTransport, in most cases the request will reach the end of chain without being accepted, an NXA error will be returned and be turned in to a target abort, but if there is a host at the other end of the chain then it may unexpectedly reply. If the request is sent in to the part then two more bits in the map entry set the endian policy to be used for the transfer and the state for the level 2 cache allocate on miss (A_L2CA) flag that should accompany the transfer. The mapping table is placed in configuration registers h44 - h80 in the PCI configuration header, however they can only be set by accesses from the ZBbus side of the bridge (even when in Device Mode).

![Figure 41: PCI BAR0 Address Mapping Table](image-url)
BAR2 and BAR3 provide access to the CPU mailbox registers (see Section: “Mailbox Registers” on page 46). The PCI device can read the current value of the mailbox and do writes to set bits in the mailbox, but it is not able to clear the mailbox. The PCI device can therefore raise interrupts to the CPUs, send small messages, and read status, but is prevented from directly clearing any CPU interrupts. This access is provided in both Host and Device Modes (but will most likely be used in Device Mode).

BAR4 is only available in Host Mode. There is an enable/disable bit for this BAR in configuration register h40, it is enabled after system reset. The BAR provides a direct map from the PCI to the bottom 512 MB of the system memory using the match byte lane endian policy, and (by setting a[29]) a map from the PCI to the bottom 512 MB using the match bit lane policy. (This area covers the low memory area, internal devices and some of the generic bus space). The interface will use pre-fetches for any reads in this area.

BAR5 is only available in Host Mode. There is an enable/disable bit for this BAR in configuration register h40, it is enabled after system reset. The BAR provides a direct map from the PCI to the two 512 MB sections of memory 00_8000_0000 - 00_9FFF_FFFF and 00_C000_0000 - 00_DFFF_FFFF using the match byte lane endian policy, and (by setting a[29]) a map from the PCI to the same two areas using the match bit lane policy. (These areas cover the rest of the memory that is addressable with 32 bit addresses, they also cover the cache test space and special access addresses that should not be accessed by normal PCI devices).

The PCI expansion ROM access BAR is mapped to 64 KB of the space that is allocated to the boot ROM at reset time. This allows an external PCI host to access an expansion ROM with no setup required by the CPU (preventing reset ordering problems). The bottom bit of the BAR must be set to enable the region (as required by the PCI specification).

Reads on the PCI do not explicitly indicate a length. To optimize performance the PCI bridge will always prefetch a cache line (i.e. do a read on the ZBbus with all byte enables set) when a read to the part comes in from a PCI device. When memory accesses are made the I/O bridge will do a coherent read and this will work as expected. If an access is made (through BAR4 or BAR5 or the ROM BAR or mapped to through BAR0) to any non-memory space in the part the I/O bridge will do an uncacheable read with all byte enables set. The result of this read will depend on the device accessed: memories on the generic bus will be read correctly, reads of internal registers that are cacheline aligned (i.e. a[5:3]=0) will work as expected, but reads of internal registers that are not cacheline aligned will give UNPREDICTABLE results.

Writes on the PCI do not explicitly indicate a length. The PCI bridge will gather a burst write targeted at the system into a buffer and do a write with the appropriate byte enables set. If the write is to memory space and all the byte enables are set a write-invalidate is sent on the ZBbus to write the data and invalidate old copies that are in any caches. If the write is to memory space and less than the full 32 byte enables are set the I/O bridge will perform a coherent read exclusive to get ownership of the block, the write data will be merged and the block written back. The bridge acts as a full participant in the coherency protocol, so if there is any request for the line it will respond as owner and will provide the data. If the write is to non-memory space an uncacheable write is performed on the ZBbus with the byte enables reflecting the bytes that were written in the PCI transaction. Writes to internal registers or the generic bus will therefore work as expected.

Figure 42 on page 209 shows the memory map that is seen from the PCI bus after reset on with the interface configured in PCI Host Mode.
Once the PCI interface has accepted a transaction that hits in one of the BARs (by asserting P_DEVSEL_L) it will complete or be terminated with a Target Abort. There are four classes of problems that will cause Target Abort:

1. An error is detected on the PCI bus. This could be a parity error, an illegal opcode or illegal byte masks.

2. An error is returned from a read request that has been sent to the ZBbus (this will be logged by the Bus Watcher) or to the HyperTransport interface (this will be logged in the interface CSRs).

3. The request hit in a BAR and targets the HyperTransport interface but the ptp_en bit in the Feature Control register is clear, disabling peer-to-peer transfers from PCI to HT.

4. The request hit in a BAR (in particular BAR0) and was targeted at the ZBbus, but the (mapped) address targets a PCI or HyperTransport space (i.e. one serviced by I/O Bridge 0).

An access that hits in BAR0, but to a disabled map entry, will not be accepted (P_DEVSEL_L will not be asserted). This results in a Master Abort.
Figure 42: Default Host Mode Memory Map from PCI Bus Master
ACCESSING THE SiBYTE FROM HYPERTRANSPORT DEVICES

The mapping from HyperTransport addresses to those in the part is much simpler than from PCI. It is shown in Figure 43.

<table>
<thead>
<tr>
<th>HT Address</th>
<th>SiByte Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF_FFFF_FFFF</td>
<td>Reserved (NxA)</td>
</tr>
<tr>
<td>FD_FEEE00_0100</td>
<td>Configuration Registers</td>
</tr>
<tr>
<td></td>
<td>Device = 0 Function = 0</td>
</tr>
<tr>
<td>FD_FEEE00_0000</td>
<td>Reserved (NxA)</td>
</tr>
<tr>
<td>FD_FEEE00_0000</td>
<td>Interrupt Packet</td>
</tr>
<tr>
<td></td>
<td>Accept and forward to SCD</td>
</tr>
<tr>
<td>FD_F800_0000</td>
<td>Reserved (NxA)</td>
</tr>
<tr>
<td>FD_0000_0000</td>
<td>Bounce Space Map</td>
</tr>
<tr>
<td></td>
<td>and send on HT</td>
</tr>
<tr>
<td>F1_0000_0000</td>
<td>Send on PCI</td>
</tr>
<tr>
<td></td>
<td>using lower 32 address bits</td>
</tr>
<tr>
<td>F0_0000_0000</td>
<td>MAP to 0x_xxxx_xxxx (bit 35=0)</td>
</tr>
<tr>
<td></td>
<td>and send on ZBbus (Match Bits)</td>
</tr>
<tr>
<td>E8_8000_0000</td>
<td>Reserved (NxA)</td>
</tr>
<tr>
<td>E8_4000_0000</td>
<td>MAP to 00_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>and send on ZBbus (Match Bytes)</td>
</tr>
<tr>
<td>E8_0000_0000</td>
<td>MAP to 0x_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>and send on ZBbus (Match Bytes)</td>
</tr>
<tr>
<td>E0_8000_0000</td>
<td>Reserved (NxA)</td>
</tr>
<tr>
<td>E0_6000_0000</td>
<td>Map to 00_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>Send back out on HT</td>
</tr>
<tr>
<td>E0_4100_0000</td>
<td>Map to 00_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>Send on PCI</td>
</tr>
<tr>
<td>E0_4000_0000</td>
<td>MAP to 00_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>send to PCI or HT (southOnLdt bit)</td>
</tr>
<tr>
<td>E0_0000_0000</td>
<td>MAP to 00_xxxx_xxxx</td>
</tr>
<tr>
<td></td>
<td>and send on ZBbus (Match Bytes)</td>
</tr>
<tr>
<td>80_0000_0000</td>
<td>Peer-to-Peer HT transfer</td>
</tr>
<tr>
<td></td>
<td>If srcid nonzero send back on HT</td>
</tr>
<tr>
<td></td>
<td>If srcid is zero return NXA</td>
</tr>
<tr>
<td>01_0000_0000</td>
<td>Send on ZBus</td>
</tr>
<tr>
<td>00_8000_0000</td>
<td>A[38] sets endian policy</td>
</tr>
<tr>
<td>00_6000_0000</td>
<td>Send on ZBus</td>
</tr>
<tr>
<td></td>
<td>A[29] sets endian policy</td>
</tr>
<tr>
<td>00_4100_0000</td>
<td>Send back out on HT</td>
</tr>
<tr>
<td>00_4000_0000</td>
<td>Send on PCI</td>
</tr>
<tr>
<td>00_0000_0000</td>
<td>Send on PCI/HT (southOnLdt bit)</td>
</tr>
<tr>
<td>00_0000_0000</td>
<td>Send on ZBus</td>
</tr>
<tr>
<td></td>
<td>A[29] sets endian policy</td>
</tr>
</tbody>
</table>

Figure 43: Memory Map From HyperTransport Device
The HyperTransport interface always runs as a host bridge, and has a PCI type 1 configuration header. The base and limit registers in the header define the region of the PCI/HyperTransport memory space that is allocated to devices on the HyperTransport fabric (these devices can be accessed by a system that uses 32 bit addresses). The value of the southOnLDT configuration bit determines whether the 00_4000_0000 to 00_40FF_FFFF range is allocated to the HyperTransport or PCI (note that these addresses are not changed when they are forwarded, nor will the COMPAT bit be set if the request is sent back out on the HT). In addition the HyperTransport is allocated the top half of the full 40 bit physical address space.

If the Bus Master Enable bit is clear in the HyperTransport interface bridge header the only requests that will be accepted from the HyperTransport fabric are to the configuration space. All other requests will be discarded or receive an NxA response. If this bit is set the bridge will accept transactions to be forwarded to the ZBbus, PCI bridge and back to the HyperTransport.

Peer to peer operations on the HyperTransport fabric must pass through the host bridge, any incoming addresses that match a HyperTransport region are therefore forwarded to the outgoing link. Requests with srcid=0 to HyperTransport peer-to-peer are responded to with an NXA (non existent address) error since they must have come all the way from a host bridge at the other end of the link and not been accepted by any of the devices on the link.

Requests that match a PCI address, either using standard addresses in the low 32 bit range of the address map or in the special FO_0000_0000 - FO_FFFF_FFFF range, are forwarded to the PCI interface and a PCI cycle is run. A PCI cycle is also run for accesses to the southbridge space if the southOnLDT bit is clear. The data is passed directly between the two interfaces regardless of system endian settings. Only a single HyperTransport-PCI read will be issued at a time. The PCI interface will not respond to requests that it generates, any access forwarded from the HyperTransport fabric that match a PCI address with destination inside the part will result in an error.

Requests that match the ZBbus space will be forwarded through I/O Bridge 0 and run as internal cycles. These requests use one address bit to select the endian mode for the transfer. If the system is big endian then setting the address bit will result in the match bit lane policy and leaving it clear will give the match byte lane policy. If the isochronous bit is set in a HyperTransport request that is sent to the ZBbus, it will have the L2 cacheable bit set in the command. This is particularly useful if the HyperTransport device is writing data that one of the CPUs will need immediate access to. If the address is to the memory range then a cacheable coherent access is made. If the access is a write of smaller than a cache line then the I/O bridge will get the current data and ownership of the block by using a read exclusive command, the new data from the write is merged into the block and it is written back to memory or L2. If the address is not in the memory range then an uncacheable access is used, with the appropriate byte enables set.

If an error is returned from a ZBbus read the MstrAbortMode bit in the Bridge Control Register determines the error reported to the HyperTransport fabric. If the bit is set a HyperTransport error with the NxA bit clear is generated for all ZBbus error types. If the bit is clear, ZBbus Bus Errors or Fatal Errors cause UNPREDICTABLE data marked valid to be returned to the HyperTransport fabric, but other ZBbus errors are still returned as HyperTransport errors with the NxA bit clear. Non-posted writes will never return an error, since they are converted to posted writes internally.

The interface does not accept cycles based on the COMPAT bit being set (even if the southOnLDT flag is clear), it therefore cannot be used to forward from the HyperTransport to a southbridge on the PCI bus.
Force Isochronous Mode Address Range

Allowing the isochronous bit to set the L2 cacheable bit on the ZBbus is useful for new devices that support setting the Isochronous bit during requests, but cannot be used by devices that do not allow control of the Isochronous bit or that are bridged from other buses by a bridge that will not translate requests appropriately. Therefore for revisions RevId 3 or greater of the HT interface there is an additional BAR (the IsocBAR) and mask (the IsocIgnMask) that is used to specify a range of addresses that will be forced to behave as if the Isochronous bit is set. The function is enabled when bit [0] of the IsocBAR is set. The IsocIgnMask is used to indicate which bits should be ignored when the address from a HT request is compared with the IsocBAR. If the other bits match between the received address and the IsocBAR then the Isochronous bit in the request is forced to be set and the L2CA flag will be set on the ZBbus. The Isochronous bit is forced if:

\[(htAddr[35:5] \mid IsocIgnMask[31:1]) == (IsocBAR[31:1] \mid IsocIgnMask[31:1])\]

In addition to allowing the PCI BAR style power-of-two windows this allows for more complicated situations. For example consider the configuration:

IsocBAR = 32'h08120001
IsocIgnMask = 32'h0200FF02

This defines a 1MByte region with base 00_8120_0000 in which the Isochronous bit will be forced for the first 64 bytes in every 4K (since addr[11:6] must be zero and addr[5] is ignored). Bit 25 of the IsocIgnMask is set to ignore address[29] since that is only used to set the endian policy. Assuming buffer alignment this could be used to allow packet headers to be cached and bodies to be written to memory.

Note that bits [39:36] are always ignored in the comparison so that the Ex_xxxx_xxxx address will have the same L2CA behaviour as the 0x_xxxx_xxxx address that it maps to.

ACCESSING THE SiBYTE FROM A SiBYTE ON A DOUBLE HOSTED CHAIN

The interface provides limited support for double-hosted HyperTransport chains. This is primarily intended for interconnection of two BCM1250s, two BCM1125Hs or a BCM1250 to a BCM1125H. The HT configuration registers are accessible from the HyperTransport fabric as device=0 function=0. At system startup one part must be designated the master, and the other the slave (this could be done by using the reset configuration resistors that are allocated for software use, or by having different bootstrap code on the two devices). The master will configure the chain as described in the HyperTransport specification.

The two parts can communicate over the fabric. Since they have the same address map, a mapping must be provided to allow each access to the other's memory space. This is done in a simple manner using a direct map. Accesses from the ZBbus to the region of the address space E0_0000_0000 - EF_FFFF_FFFF will be sent to the HyperTransport fabric (since it is in region N in Figure 37 on page 193). This range will be accepted by the host bridge on the part on the other end of the fabric, and direct mapped into the low address space (00_0000_0000 - 0F_FFFF_FFFF) and a second level decode performed to determine if the request should be forwarded to the ZBbus, the local PCI or sent back out on the HyperTransport. Accesses to these addresses pass over the I/O fabric and therefore outside the coherency domain. Software will need to manage the coherency of any data between the two parts. The space should be mapped either uncacheable or cacheable non-coherent in the CPU, and will not be cached in the L2 cache.
Accesses from the ZBbus of one part to the $F0_{0000 \_0000} - F0_{FFFF \_FFFF}$ region of the address map will be sent over the HyperTransport fabric, accepted by the second part and direct mapped to a PCI access on the second device.

There is a restriction on configuration accesses made from one part to another over the HyperTransport link. Both reads and writes to the configuration registers must be 32 bit operations (a HyperTransport full unmasked double-word), if bigger or smaller operations are performed reads will return UNPREDICTABLE values and writes will leave the configuration register in an UNPREDICTABLE state.

There is a restriction on configuration accesses made from one part to another over the HyperTransport link. Both reads and writes to the configuration registers must be 32 bit operations (a HyperTransport full unmasked double-word), if bigger or smaller operations are performed reads will return UNPREDICTABLE values and writes will leave the configuration register in an UNPREDICTABLE state.

Interface RevId 3 and later support the ActAsSlave function defined in the HyperTransport 1.03 specification, which is of use in some double-hosted configurations.

**HyperTransport Bounce Space**

In a double-ended chain is used with a master host on one end and a slave on the other, peripherals will identify the master device and direct all transactions to it. Anything that matches a memory address will be accepted by the master. The slave has an identical memory map, so direct transactions will never reach it. The Bounce space provides a way for the peripheral to bounce transactions to the slave via the master.

Any address received in the range $F1_{0000 \_0000} - FB_{FFFF \_FFFF}$ will be sent back out on the HyperTransport fabric with the address mapped to $01_{0000 \_0000} - 0B_{FFFF \_FFFF}$ (the top 4 bits are cleared) and any address received in the range $FC_{0000 \_0000} - FC_{FFFF \_FFFF}$ will be sent back out on the HyperTransport fabric with the address mapped to $00_{0000 \_0000} - 00_{FFFF \_FFFF}$ (the top 8 bits are cleared). This allows the device to access the memory space in the slave.

**Performance of the PCI and HyperTransport Interfaces**

The performance of the PCI and HyperTransport interfaces is strongly dependent on the size of the transfer. The best case is when 32 byte cache blocks are moved. If smaller writes are done from the PCI or HyperTransport into the part then the I/O bridge 0 has to read the cache block exclusive (to fetch any modifications that may be in caches), modify it and write it back.

Writes from the ZBbus to either interface are always posted. Reads consist of two stages: the Read Request, which is a non-posted request; and the Read-Data-Return (RDR) which is a response.
ACCESSES FROM THE SiBYTE TO THE PCI OR HYPERTRANSPORT

The data path from the ZBbus to the PCI or HyperTransport is split into three sections. The interface to the ZBbus includes queues of requests and returning data. Inside the I/O bridge there are additional buffers in the routing path to the interface block. Finally, there are additional queues and tables of outstanding requests in the interface blocks.

Figure 44: Buffers Used for Accesses from the ZBbus to PCI and HyperTransport

Figure 44 illustrates these queues. The request queues always maintain transactions in the order that their A-phase happened on the ZBbus, for uncached accesses this matches the program order of the loads and stores. Once the requests are on the HyperTransport fabric or PCI bus the normal PCI ordering rules apply, so writes can pass reads; for uncachable accesses from the SB-1 cores this is not a problem because the CPU will not issue an uncachable write while there are uncachable reads outstanding. The internal connections will maintain the order of reads and the SB-1 CPU supports multiple outstanding uncached reads to allow streaming. There is one case where this can cause problems: in the HyperTransport fabric (due to the ordering of the chain), or PCI with bridges (using delayed reads) reads to different devices may reach their destinations in a different order than they were issued. If the reads have side effects that are signalled on a back-channel between the two devices the second read issued by the CPU may or may not see the effect of the first (this is always a danger with back-channels). If this rare situation is encountered in a system then the program must include a SYNC instruction between the reads to ensure they complete in the expected order.

Writes from the ZBbus are always posted and need no reply, they pass through the request queues and are issued to the I/O bus. Reads also pass through the queues to be issued, but also have associated Read-Data-Return (RDR) passing back up through the bridge to the ZBbus.
The HyperTransport fabric uses split transaction reads and the interface supports 14 reads outstanding on the fabric. When a HyperTransport response arrives it is matched to the associated request and passes back through the RDR path. If a masked read (a read that is not a multiple of aligned 32-bit words) is encountered then no further reads will be accepted by the interface until it has completed (the Idl and ldr instructions can result in reads of 5, 6 or 7 bytes, these will be split into two masked reads that can both be outstanding on the link and must both complete before the RDR is returned).

The PCI interface can hold one read pending and have one read in flight on the bus. If a read is retried excessively (more than the number of times set in the Retry Timeout register described in Table 132 on page 240) the interface will allow any writes or RDRs to pass before retrying the read. If a read exceeds the retry timeout on the second attempt it will be abandoned and a bus error reported.

The restriction on only having one read pending on the PCI bus will limit the performance. Therefore in interface with RevId 3 or greater a prefetch mechanism is added. If prefetching is enabled in the PCI Adaptive Extend Register then a full cache block ZBbus read will cause 2 or 4 cache blocks to be prefetched from the PCI. This is particularly useful when the Data Mover is transferring a large block from the PCI. If a read request from the ZBbus is for a full cacheline at an aligned address and the prefetch is enabled and the address is not greater than (4k page boundary minus 4 cacheline addresses), then a read of 2 or 4 cachelines is issued to the PCI. If the next request from the ZBbus is also a full cacheline read to the next consecutive cacheline address, the prefetched data will be returned. This pattern will repeat until all the prefetched data are returned. If the next request does not meet this criteria, the prefetched data will be flushed and the new request will be sent to the PCI bus. (Note that since the prefetched data could be fetched and discarded it is possible for the prefetching to lower performance.) In the error case where the prefetched data (the additional data that is not used by the initial transaction) is timed out for too many retries, the interface will not reissue the prefetch reads. This may cause the device being accessed to deadlock, to prevent this the retry limit should be set high to ensure only actual errors will be captured.
ACCESS FROM THE HYPERTRANSPORT TO THE SiBYTE

Figure 45 shows the queueing for DMA accesses from the HyperTransport fabric and PCI bus into the part.

![Figure 45: Buffers Used for DMA Accesses from the PCI and HyperTransport](image)

Posted writes from the HyperTransport fabric flow through the queues in the I/O bridge and ZBus interface and are posted on the ZBus. Non-posted writes from the HyperTransport become posted writes internally (since the architecture does not have any internal non-posted writes). An acknowledgment response is always sent on the HyperTransport when the write leaves the HyperTransport interface and is inserted in the I/O bridge queue. Since it has become a posted write it may now pass reads. An error response will never be generated.

Reads from the HyperTransport fabric pass through the queues and are sent onto the ZBus. The I/O bridge interface can have 8 reads outstanding on the ZBus (including those from both PCI and HyperTransport). When all 8 entries in the RDR buffer are in use further reads are blocked from moving into the ZBus request queue. The read-data-return is sent back to the CPU response queue in the HyperTransport interface. Read requests from the HyperTransport that are longer than 32 bytes will result in two reads being issued towards the ZBus.
ACCESSING FROM THE PCI TO THE SiByte

The path for the PCI to access the part is shown in Figure 45 on page 216. It is similar to the HyperTransport flow.

Writes from the PCI are always posted into the part, they flow through the queues shown in the figure.

The PCI interface will accept one outstanding read from the PCI bus. If this read is not serviced after 16 PCI cycles (the debug is programmable as described below) it will be disconnected to allow other PCI devices to make progress (turning it into a Delayed Read Request). Subsequent reads from the PCI bus into the part will be immediately disconnected until the first has been satisfied. PCI reads do not include a length in the request, so a heuristic is used to improve performance. If the read uses a PCI Read or ReadLine command the PCI interface will initially request a single (32 byte) block from the ZBus interface, when this returns it will be used to satisfy up to 32 bytes of the request. If the request is still in progress an additional request will be made to the ZBus interface for the next cache block. If the ReadMultiple PCI command is used the interface will immediately launch two read requests for consecutive cache blocks, and will continue to prefetch blocks as the data is sent to the PCI. The reads share the RDR buffer in the ZBus interface with reads from the HyperTransport fabric.

If a request from the PCI bus requests a burst mode other than Linear Incrementing the interface will disconnect the transfer after a single data phase.

Memory space reads smaller than four bytes will result in the appropriate byte enables being asserted on the PCI bus unless the dis_memrd_be bit is set in the PCI Adaptive Extend register. If this bit is set all memory space reads will have all four byte enables asserted (the additional bytes received from the PCI will be ignored). If the dis_memrd_be bit is clear then any access to the PCI greater than 4 bytes should be an aligned multiple of four bytes. If the dis_memrd_be bit is set then any access may be safely done but the PCI device will always see an aligned multiple of four bytes.

PCI Adaptive Retry

The PCI specification requires that if a request is not completed within 16 cycles the target should terminate the transaction with a retry to free the bus for other devices. It encourages devices that know they have a long latency to disconnect the transaction as early as possible to avoid dead cycles on the bus. On the host the latency of memory reads is hard to calculate because it depends on what other activity is in the system. Typically in a lightly loaded system reads will complete in 200-400 ns, which is well under the PCI timeout. By default the interface will not retry the transaction early, and will wait the full 16 cycles before issuing the retry.

The interface also has an adaptive retry policy, which is enabled by setting the adapt_rety_en bit in the Feature Control Register (Table 133 on page 241). The actual latency of read requests is measured and used to set the number of cycles that the interface waits before signalling the read should retry. Table 124 shows how the retry delay is changed based on the current memory latency. Two consecutive samples must show the same trend in latency before the algorithm changes behavior.

The adaptive retry parameters are spread across the PCI Feature control register and the PCI Adaptive Extend register. The minimum value is a 3 bit field, nominal and maximum are 7 bit fields. This allows the values to be programmed well outside the compliant PCI range, but allows optimal selection to be made in embedded systems.
Table 124: Adaptive Retry Delay

<table>
<thead>
<tr>
<th>Condition</th>
<th>New Delay Before Retry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory latency &gt; max_tar_retry</td>
<td>retry_delay = min_tar_retry</td>
</tr>
<tr>
<td>retry_delay != min_tar_retry AND max_tar_retry &gt; Memory latency &gt; nom_tar_retry</td>
<td>retry_delay = max_tar_retry</td>
</tr>
<tr>
<td>retry_delay == min_tar_retry AND nom_tar_retry &gt; Memory latency &gt; min_tar_retry</td>
<td>retry_delay = nom_tar_retry</td>
</tr>
</tbody>
</table>

Figure 46 illustrates the algorithm. (The suggested delay values are appropriate for a host bridge, which is permitted to wait up to 32 cycles before issuing a retry, compared to only 16 cycles in a device). The retry delay is initialized to the value given in the nom_tar_retry register. This sets the normal retry timeout that should be used and should be set above the delay expected for the memory system. Most reads will be satisfied in this latency (falling in to the Normal Read Latency box in the figure), and the occasional long latency accesses will be disconnected and retried (this behavior is just like the standard case). If the system becomes busier and the latency increases slightly (moving the read latency into the higher than expected load box), to fall between the nominal and maximum values, rather than continually disconnecting transactions just before their data becomes available the adaptive policy will increase the retry delay to the maximum value. This prevents the PCI throughput falling catastrophically when the system experiences a load slightly higher than was expected. However, if the memory latency becomes larger than the maximum (the very high load box) then the retry delay is set to its minimum value causing requests to be rapidly disconnected. Once the overload has been detected the retry delay is left at the minimum until the memory latency returns to its normal range, when the retry delay will also be restored to its normal value.
PEER-TO-PEER ACCESSES

The interface supports peer-to-peer accesses from devices on the PCI bus to devices on the HyperTransport fabric, and from devices on the HyperTransport fabric to devices on the PCI bus. Requests and data being transferred in this way do not travel on the ZBbus, they are directly routed through special buffers in the I/O Bridge 0. In the master bus interface of a peer-to-peer operation the queues are shared with DMA operations (from I/O devices into the ZBbus) and on the slave side the queues are shared with PIO operations (from the ZBbus to I/O devices).

Peer-to-peer operations in I/O space are not supported. Neither the PCI bus nor the HyperTransport bus will accept an incoming I/O space request.

PCI BUS TO HYPERTRANSPORT FABRIC

When the PCI interface is configured in Host Mode, requests on the PCI that fall into the secondary bus memory range specified in the HyperTransport bridge header will be accepted from the PCI and forwarded to the HyperTransport fabric. This forwarding is disabled by default, and must be enabled in the Feature Control register (offset h40 in the PCI configuration header). In addition requests that match BAR0 and have an enabled mapping with the send_lidt bit set are forwarded to the HyperTransport (the send_lidt bit overrides any destination implied by the address). The peer-to-peer bit in the Feature Control register must also be set to enable forwarding through the map table.

If the PCI interface is configured in Device Mode and the peer-to-peer bit is set in the Feature Control register the only requests that will be forwarded to the HyperTransport fabric are those that hit in BAR0 and have an enabled mapping with the send_lidt bit set. This allows the part to act as a non-transparent bridge (i.e. one with separate address spaces on each side and explicit routing between the sides) from the PCI host to the HyperTransport fabric. Note that in device mode the Feature Control register is written by the host on the PCI bus and cannot be accessed from the ZBbus, thus both the host (through Feature Control ptp_en) and device (through BAR0 send_lidt) must agree for the PCI-HT access to be permitted.
Figure 47 shows the queues used in peer-to-peer operations with a PCI master.

Peer-to-peer writes will transfer exactly the number of bytes written and will always be posted to the HyperTransport fabric. They flow in their own channel from the PCI interface to the HyperTransport interface and may pass non-posted reads.

Peer-to-peer reads always assume the destination address is prefetchable (this is required for performance, since the PCI bus does not give a good indication of the transfer size at the start of the transaction). Caution should be used when accessing registers using peer-to-peer operations. The same algorithm is used as for reads to the ZBus, the PCI read is assumed prefetchable, and is a Read or ReadLine is causes single 32 byte request to the HyperTransport fabric while a ReadMultiple will launch two requests.
HYPERTRANSPORT FABRIC TO PCI BUS

Requests received by the HyperTransport interface to any addresses that fall in the space allocated to the PCI memory space (regions A and C in Figure 37 on page 193) or that fall into the special double ended chain mapped address ranges (E0_4000_0000 - E0_7FFF_FFFF and F0_0000_0000 - F0_FFFF_FFFF in Figure 43 on page 210) will be forwarded to the PCI interface. Since the size of the transfer is specified at the start of a HyperTransport request the access to the PCI bus will always be for the exact number of bytes requested.

Figure 48 shows the queues used in peer-to-peer operations with a HyperTransport master.

Peer-to-peer writes will always be posted to the PCI bus. Non-posted write requests will be acknowledged when they leave the HyperTransport interface. Writes flow in their own channel from the HyperTransport interface to the PCI interface and may pass non-posted reads.

Figure 48: Buffers Used for HyperTransport to PCI Peer-to-Peer Accesses

The HyperTransport interface can have two reads outstanding to the PCI interface. If a HyperTransport request is for more than 32 bytes both outstanding reads are needed to service it. Alternatively two smaller requests can proceed together. The buffering and PCI interface will maintain the ordering of the reads. The request path and RDR buffer in the PCI interface are shared with PIO and DMA requests, but there are dedicated return buffers for the peer-to-peer RDR.
The device incorporates a PCI bus arbiter that may be used when the interface is configured as the host bridge. The arbiter supports four external devices in addition to the host. Arbitration is done using a single-ring round robin scheme with the bus parked on the last granted device when there are no new requests. Except for the hotplug situation described below, the arbiter will wait for up to 32 PCI clocks for the master to assert frame before removing the grant (the “broken” master case of section 3.4.1 of the PCI specification), but requests from the master will continue to be responded to and the arbiter does not have a status register that records this being done.

The arbiter includes support for hotplug systems. When the hotplug_en bit is set in the Additional Status and Control Register (at offset +88 in the PCI configuration registers) and P_REQ_L[3] is asserted, once P_GNT_L[3] is granted it will be held in the asserted (granted) state as long as P_REQ_L[3] is asserted, independent of higher priority requests. In addition to enabling hot plug systems, this can be used to solve some arbitration problems in some older southbridge devices.

**PCI INTERRUPTS**

The PCI interface includes 4 active low interrupt lines P_INTA_L-P_INTD_L. When used as a host the interface will normally use all of these as the interrupt inputs from the other PCI devices (the PCI specification describes the recommended way of permuting these as they are connected to the INTA#-INTD# pins of slots or devices). In all cases the four lines are monitored as active low interrupt inputs and the results presented to the interrupt mapper as sources 56-59 (see Table 22: “Interrupt Sources,” on page 52). Any of the inputs that are not used by PCI devices can therefore be used as general active low interrupt inputs.

The P_INTA_L signal can also be driven (as an open collector output) by the interface (note that when it is driven low the input logic will detect that it is low and raise system interrupt 56). This is intended for use in device mode to allow an interrupt to be driven to the host (see Section: “Using the PCI in Device Mode” on page 232). However, the output can still be driven when the interface is configured for host mode so a system that does not need four PCI interrupt inputs may choose to use the P_INTB_L-P_INTD_L for PCI use and use P_INTA_L as a general open collector I/O.

**HYPERTRANSPORT DIFFERENCES FROM REVISION 0.17 SPECIFICATION**

This section lists the main differences between the HyperTransport Specification Revision 0.17 and the interface implemented in the BCM1250 and BCM1125H. Since there was only limited distribution of the 0.17 specification, most readers will find the next section (differences from revision 1.03) more useful.

1. The transmitters drive UNPREDICTABLE data on the CAD and CTL lines during CRC testing mode.
2. Non-zero byte masks for double words which are not sent are ignored.
3. Five additional error conditions are reported: Protocol error detected on the link; Receive FIFO overflow error; Response received with a srctag that has no corresponding request; End-of-chain NXA error issued to the link; and NXA error issued to a request from a peer host bridge (a request with a source tag of zero).
4. The interface has 4 responses to errors. Localized errors result in an error response. The remaining errors can be set to result in sync flooding, raising the HyperTransport fatal error interrupt or raising the HyperTransport non-fatal error interrupt.
5. If the isochronous bit is set in a HyperTransport request that is sent to the ZBbus, it will have the L2 cache allocate bit set in the ZBbus command.
In a dual host topology, if any incoming packet, with an address mapped as to be forwarded to the outgoing link and a srcid = 0, is received by a host it will be NXAed.

The ISA deadlock prevention mechanism, described in section C.1 of the HyperTransport Specification Revision 0.17, is not supported. This mechanism has been removed in Revision 1.0 of the specification. As a result, and as specified in Revision 1.0, no peer-to-peer transactions involving ISA are allowed.

The optional secondary bus reset functionality in the Capability command register is implemented.

The interface only supports an 8 bit CAD width. There is no support for the 2 or 4 bit widths that are described in Revision 1.0.

The ordering between CPU (or any other internal device) writes to the HyperTransport fabric and DMA Read Data Return to the HyperTransport fabric is imposed in the HyperTransport interface. This ordering is not part of the ZBus. In particular the I/O bridge and ZBus behave as if the Response may pass posted Writes bit is always set. If the PCI producer-consumer model is being used read responses should always pass posted writes and this bit should always be clear. The result of not having the ordering rule internally is that if HyperTransport device A polls a flag in memory that the CPU writes after writing HyperTransport device B (which may or may not be the same device), the read response with the flag set may reach device A before the write reaches B. The ordering can be enforced by the CPU doing a PCI or HyperTransport configuration register read between the two writes. This forces the write to device B into the HyperTransport interface (where the ordering rules apply) before the flag gets written in memory. (This also applies for the PCI interface).

Posted writes have the highest priority with regard to transmission to the HyperTransport link. Starvation of non-posted transactions and read responses is prevented by implementing counters in all entries of the interface transmit FIFOs. The outgoing entry counters will get incremented on each transaction issued. When an entry’s counter overflows, its priority is elevated.

The correct ordering of interrupts with respect to writes from the HyperTransport into the BCM1250 is maintained. An interrupt that follows a HyperTransport write will not be raised inside the BCM1250 until the write is visible inside the ZBus coherence boundary.

A non-posted write from the HyperTransport fabric to the ZBus or PCI bus will be acknowledged when the write leaves the HyperTransport interface, and will continue as a posted write. Error responses are never generated for non-posted writes. Therefore, if the HyperTransport device needs to confirm the success of a non-posted write, it must (1) wait for the write tgtDone response, then (2) issue a read to the write address for confirmation. It cannot issue the read without waiting for tgtDone since non-posted transactions may bypass each other.

No peer-to-peer I/O space traffic between the HyperTransport and PCI is supported.

No DMA master is allowed on an ISA bridge on the HyperTransport or PCI buses. It could result in a deadlock.

PCI devices designed for rev2.0 or earlier may not be bridged off the HyperTransport.

The LinkFreq register, described in Revision 1.0 of the Specification, has been added to the HyperTransport Capability registers. This frequency set in this register sets both the transmitter frequency and the maximum frequency the receiver can accept.

Support has been added for frequencies other than those specified in HyperTransport Revision 1.0 by using the sipFrequencyDirect bit in the HyperTransport SRI Command register.

In a dual hosted system determination of the master and slave ends of the chain must be made before link configuration. Neither the HyperTransport Specification nor the BCM1250 hardware specify the method for determining this. System designers are free to implement it in whatever way matches their system. Three possible methods on the BCM1250 are: use one of the software configuration bits latched into the System Configuration Register from the IO_AD[31:26] during reset (see Table 15 on page 43); store the information in the boot ROM; store the information in a configuration EEPROM on the SMBus.
HYPERTRANSPORT DIFFERENCES FROM REVISION 1.03 SPECIFICATION

This section compares the HyperTransport interface with the HyperTransport I/O Link Protocol Specification Rev 1.03 10/10/2001. This is the first specification released by the HyperTransport Consortium, the specification details and section numbers match the 1.02 release made by AMD. Comments are made by section number in the specification. (The sections were reordered in HT 1.04.)

1.1 Terminology

Care is needed with the term Doubleword. In HyperTransport/PCI it is used for four bytes, in MIPS it is used for eight bytes.

2 Signaling

The interface only supports 8 bit links (8 CAD pairs, 1 CLK pair, 1 CTL pair). The optional power management signals are not supported.

3.2.1.4 Command Encoding

The interface uses the Isochronous bit in Wr(sized) and Rd(sized) inbound (dma) commands to set the ZBus L2CA flag to request the access be allocated in the L2 cache on a miss. All memory accesses will be marked cacheable coherent and all others uncached regardless of the coherence bit in the command.

4.1 Topology

The part is always a host. If the interface revision is 3 or greater the ActAsSlave mode is supported for double-hosted chains.

4.1.1 Double-Hosted Chains

The interface supports double-hosted chains. Designation of the master and slave end must be done by software. The Host Hide function is not implemented. The ActAsSlave is implemented in interface RevId 3 and greater, but is not available on earlier versions. Care is therefore needed to understand the dataflows in sharing double-hosted chains.

In a chain with two hosts A and B, if the MAC or Serial port DMA engine on A is doing writes to any device through the HyperTransport then accesses from B to any of the peripherals behind I/O bridge 1 on A can cause deadlock.

4.1.2.2 Host Implementations

The interface has LDT_PWROK and LDT_RESET_L separated from the system reset. A system reset will always cause the HyperTransport to reset until released by software. See Section: “HyperTransport Resets” on page 259 for more details.

4.2 Transactions and UnitID

The interface is a host and therefore always uses UnitID 0. In interface RevId 3 and greater the ActAsSlave function allows the interface to use the UnitID set in the DevNum field on a double-hosted chain.

4.4.1 Sized Writes

In byte writes nonzero byte masks are ignored for doublewords that are not sent.
4.4.2 Broadcast Message

The only broadcast message the interface will generate is an EOI. No broadcast messages are accepted.

4.4.3 Flush, 4.4.4 Fence

The interface will never generate a Flush or Fence command. They will be accepted and Flush will result in a TgtDone return.

4.4.5 Atomic Read-Modify-Write

The Atomic Read-Modify-Write is neither generated nor accepted by the part.

4.5.2 Target Done

Target done is returned when a request has been sent from the HyperTransport interface to the I/O Bridge. Inbound TargetDone responses cause the target done counter to increment (making the completion visible to the processor).

4.7 Virtual Channels

The three standard Virtual Channels are supported. Since all writes within the part are posted, the only writes that will be in the non-posted channel are configuration space writes.

4.9.4 Host Bridges

1. The interface supports the optional double-hosted memory space region as described in Section: “Accessing the SiByte from a SiByte on a Double Hosted Chain” on page 212. Responses from the part always have the Bridge bit set (rather than the specified clear) and a UnitID of zero.
2. Broadcasts are dropped.
3. Directed requests are accepted as described in the Section: “Accessing the SiByte from HyperTransport Devices” on page 210.
4. This rule is not completely enforced. Responses with the Bridge bit set and a UnitID of zero are treated as responses from a host on the far end of the link and compared to the outstanding requests.
5. A SrcTag error is logged in the HyperTransport Error Status register (Table 153) if a response is received with no matching request.

5 Interrupts

The interrupt usage follows the earlier HyperTransport specification which has now been moved to the x86-specific appendix.

5.1 Interrupt Requests

The IntrInfo[55:32] bits are not supported. IntrInfo[31:2] are interpreted in a similar way to the x86-specific model, as described in Section: “HyperTransport Interrupts” on page 48. Interrupt messages can be generated using the special address range described in Section: “HyperTransport End Of Interrupt (EOI) Signaling Space” on page 198.

5.2 End of Interrupt

The interface ignores any EOI messages it receives. EOI messages are generated by software as described in Section: “HyperTransport End Of Interrupt (EOI) Signaling Space” on page 198.
6.1 Upstream I/O Ordering

The interface follows the ordering rules through the HyperTransport Interface into the I/O Bridge.

6.2 Host Ordering Rules

The ordering rules are maintained into the ZBbus domain, which has a different set of rules. The same ordering is used for Cacheable and Non-Cacheable commands, and the rules for these, interrupt messages and responses are maintained correctly. However, because the ZBbus allows responses to pass posted writes care must be taken when using the producer-consumer model as described in point 10 of the previous section.

6.2.1 Host Responses to Nonposted Requests

A non-posted write is acknowledged as the transaction leaves the HyperTransport interface and becomes a posted write internally. See point 13 of the previous section.

6.4 Ordering in Sharing Double-Hosted Chains

The I/O Bridge and HyperTransport interface will always push posted writes ahead of read responses. But this is not the case internally and care must be taken because the ZBbus always allows read responses to pass posted writes. See the Section: "Ordering Rules and Device Drivers" on page 14.

7.1 Configuration Cycle Types

Configuration Cycles are generated as described in Section: “Configuration of PCI and HyperTransport” on page 234.

7.3.1.1 I/O Space Enable, 7.3.1.2 Memory Space Enable

These bits are always set because the address decoders on the ZBbus are always active.

7.3.8 Interrupt Line

The Interrupt Line register is not writeable.

7.3.2.3, 7.3.2.4, 7.3.2.5 Primary Bus Error Status bits

These bits are always clear because errors on the ZBbus side are reported in other ways.

7.4.8.3 ISA Enable

The optional ISA address scrambling is not supported so this bit is always 0.

7.4.8.4 VGA Enable

The optional VGA address forwarding is supported so this bit is R/W. The implementation is described in Section: “The SouthBridge, VGA and Subtractive Decode” on page 195.

7.5 Capability Registers

The interface uses the Host Format from the earlier HyperTransport specifications, these cover the +00, +04 and +08 offsets of the block in Table 29 of the HyperTransport specification. Interface revision 3 uses the format from the 1.03 specification.
7.5.3.3 Host Interface Command Bits

The pass 2 BCM1250 does not implement the fields that are new in the revision 1.02 specification: Device Number, Chain Side, Host Hide, Act as Slave, Drop on Uninit. Interface revision 3 and greater, used for the BCM1125H and later versions of the BCM1250, includes these. The new Host Inbound End of Chain error (Bit 11) functionality is incorporated in the MapNxaError bit in the HyperTransport Error Status Register (Table 153 on page 253).

7.5.4 Link Control Register

The interface does not implement the fields that are new in the revision 1.02 specification: Isocronous Enable, LDTSTOP# Tristate Enable, Extended CTL time. None of the functionality that these control is supported.

7.5.5 Link Configuration Register

This register reflects the fact that the interface only supports 8 bit links and does not support doubleword flow control.

7.5.8 Link Error Register

The error conditions reported in this new register are reported in the HyperTransport Error Status register (Table 153 on page 253). Note that if the bridge is put into the direct frequency selection mode (allowing link frequencies outside the standard) then bit 4 is used for frequency selection.

7.5.9 Link Frequency Capability

The link frequency capability register is only supported on revision 3 and greater of the interface. On revision 2 (BCM1250 pass 2) it would have the value 16'h801F, indicating that frequencies up to 600MHz are supported and there are vendor specific frequencies. On revision 1 (BCM1250 pass 1) it would have the value 16'h800F, indicating that frequencies up to 500MHz are supported and there are vendor specific frequencies.

7.5.10 Feature Capability Register

The link feature capability register is only supported on revision 3 and greater of the interface. On older revisions it would have the value 16'h0004.

7.5.11 Enumeration Scratchpad

This register is not implemented (its address collides with the SRI registers).

7.5.12 Error Handling Register

The functions in this new register are available in the HyperTransport Error Control register (Table 152 on page 252).

7.5.13, 7.5.14 Memory Base/Limit Upper Bits

These are not implemented since the interface constrains PCI style memory operations to the low section of the address map, and has special rules for addresses above the 32-bit region.

7.6 Interrupt Discovery and Configuration Capability Block

This new capability block is not implemented. Generation of interrupt messages is done by software, and is restricted to IntrInfo[55:32]=0 and IntrInfo[31:24]=F8 (i.e. compatibility with revision 1.01 and earlier and the x86 specific interrupt restrictions).
7.7 Address Remapping Capability Block

The interface does not have an address remapping block, the mappings between the ZBbus and HyperTransport addresses are discussed in the earlier sections of this chapter.

8 System Management

The interface does not support any of the legacy x86 system management or special cycles. The optional LDTSTOP# is not supported.

9 Address Map

The internal address map (Table 11 on page 36) reflects the requirements of the HyperTransport map.

Only interrupts with Address[31:24]=F8 are accepted.

10 Error Conditions

The interface detects and reports the errors described in this section. Since the method of detection and reporting were not specified in earlier versions of the HyperTransport Specification the implementation differs.

10.1 Transmission Errors

A CRC error is detected and reported as in the specification. The sync flood control is in the Link Control register as specified. The interrupt enable bits are in the HyperTransport Error Control Register (Table 152 on page 252) rather than the new Error Handling Register in the extended register set.

When a CRC error is detected the expected and received CRC values are logged in the HyperTransport Diagnostic Receive CRC Expected (Table 158 on page 254) and Received (Table 159 on page 254) registers.

10.1.3 Protocol Errors, 10.1.4 Receive Buffer Overflow Errors

Protocol and Overflow errors are reported in the HyperTransport Error Status Register (Table 153 on page 253) rather than the new Link Error Register, and controlled in the HyperTransport Error Control Register (Table 152 on page 252) rather than the new Error Handling Register in the extended register set.

10.1.5 End of Chain Errors

The end of chain error is reported in the EocNxaError flag in the HyperTransport Error Status Register (Table 153 on page 253) and controlled in the HyperTransport Error Control Register (Table 152 on page 252).

The MapNxaError flag is used to report two variants of the issue described in the spec as Host Inbound End Of Chain error. A MapNxaError is flagged when a posted operation is received that is to an address that is not on the HyperTransport link or PCI and does not decode to a legal address within the BCM1250. It is also flagged and the transaction dropped, for a posted operation if the srcid is zero (i.e. it has come from the other host on a double-hosted chain) and the destination is an address on the HyperTransport link. In this second case the transaction has already been rejected by all devices on the link, and sending it back will just lead to the transaction looping forever.

10.1.6 Chain Down Errors

The host interface discards all state on a link RESET# and does not report a chain down error. Responses for in-flight reads will be lost and the requestor may hang.
10.1.7 Response Errors

A response received without a matching source tag (or as a result of a response to a WrSized request) is reported as a SrcTagError in the HyperTransport Error Status Register (Table 153 on page 253) and controlled in the HyperTransport Error Control Register (Table 152 on page 252).

No check is done for a TgtDone response or an incorrect count received for a RdSized request, the requester will receive UNPREDICTABLE data in this case.

The interface does not generate Flush or Atomic Read-Modify-Write cycles.

10.2.1 Error Responses

Error responses are controlled by the Master Abort Mode bit as the specification describes. Bit 5 of Table 146 describes the method used to propagate errors from the HyperTransport fabric to the ZBbus.

10.2.2 Error Interrupts

The error interrupts are passed directly to the interrupt mapper as interrupt numbers 48 and 49.

10.2.2 Error Routing CSRs

The error CSR bits in the implementation are summarised in the Table 125 on page 229 below which follows the format of the Table 50 in the HyperTransport Specification.

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Log Bit</th>
<th>Flood Enable</th>
<th>Fatal Error Enable</th>
<th>Nonfatal Error Enable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>ErrStat/ProtoErr</td>
<td>ErrCtrl/ProtSyncFlood</td>
<td>ErrCtrl/ProtFatalEn</td>
<td>ErrCtrl/ProtNonFatalEn</td>
</tr>
<tr>
<td>Overflow</td>
<td>ErrStat/OvfErr</td>
<td>ErrCtrl/OvfSyncFlood</td>
<td>ErrCtrl/OvfFatalEn</td>
<td>ErrCtrl/OvfNonFatalEn</td>
</tr>
<tr>
<td>Response</td>
<td>ErrStat/SrcTagErr</td>
<td>ErrCtrl/SrcTagSyncFlood</td>
<td>ErrCtrl/SrcTagFatalEn</td>
<td>ErrCtrl/SrcTagNonFatalEn</td>
</tr>
<tr>
<td>CRC</td>
<td>LinkCtrl/CrcErr</td>
<td>LinkCtrl/CrcFloodEn</td>
<td>ErrCtrl/CrcFatalEn</td>
<td>ErrCtrl/CrcNonFatalEn</td>
</tr>
<tr>
<td>SERR</td>
<td>SecStatus/SerrDet</td>
<td>BridgeCtrl/SerrEn</td>
<td>ErrCtrl/SerrFatalEn</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>

11.1 Clocking Mode Definition

The interface supports synchronous and asynchronous mode. Software must configure the interface before setting the SipReady bit, as described in the “System Reset Initialization of the HyperTransport Interface” on page 255.

12 Reset and Initialization

The interface performs the standard link initialization after software has configured the low level parameters and set the SipReady bit.
12.2 Low Level Link Initialization and Table 54 Values of CTL and CAD During Link Initialization Sequence

The transmit side will generate 512 cycles with CTL and CAD as zero before asserting CAD and the receive side will accept 512+M cycles. This matches both the older versions of the spec which need ‘a minimum of 512 cycles’ and can have an arbitrary number of additional bit times, and the newer version ‘512+4N’ where the minimum is still 512 cycles (N=0) but any additional duration of CTL and CAD at zero must be for a multiple of four bit times.

12.3 I/O Fabric Initialization

Since the MIPS architecture does not support non-posted writes the TgtDone counter has been added to allow detection of the completion of configuration writes.

In a Double-hosted chain software is responsible for determining which is the master and slave ends of the fabric and performing appropriate configuration.

12.3.1 Finding the Firmware ROM

The part cannot boot from a ROM over the HyperTransport link.

12.5 Link Frequency Initialization

The interface will normally perform the link frequency change on a link reset, and revert to 200MHz on a cold reset. However, to support other devices that use the HyperTransport specification from before revision 1.0 this behaviour can be prevented by setting the srILdtPLLCompat bit in the SRI Command Register (Table 149 on page 251).

A. Address Remapping

The interface supports the host address map as outlined throughout this chapter, so does not need the address remapping scheme.

B. Ordering Rules

The HyperTransport interface uses the PCI ordering rules.

C. Mapping of Other Protocol Ordering Rules

The interface maps from the ZBbus as described in C.1. Non-posted writes are not part of the MIPS architecture and so will never be generated for memory-mapped I/O.

D. Considerations for Isochronous Traffic

The interface does not use the new isochronous scheme. The isochronous bit from DMA read or write requests is copied into the L2CA flag for the ZBbus transaction. Isochronous requests will therefore cause L2 allocation on a miss.

The part always generates requests with the isochronous bit clear.

E.1, E.2 ISA/LPC Support

There is no special ISA/LPC support. All warnings and restrictions in these sections must be followed.

Note that PCI devices that only support Revision 2.0 or earlier of the PCI Specification may have similar problems to ISA devices since they do not support the full ordering and transaction acceptance rules.
E.3 Subtractive Decode

Subtractive Decode is discussed in the “The SouthBridge, VGA and Subtractive Decode” on page 195.

F.1 Interrupts

The interface follows the x86 interrupt mechanism as outlined in Section: “HyperTransport Interrupts” on page 48. Interrupt messages and EOI are generated by software as described in Section EOI Signalling Space on Section: “HyperTransport End Of Interrupt (EOI) Signaling Space” on page 198.

In logical mode IntrDest[31:2] bits are ignored and the low two bits used to select the processor(s) that will receive the interrupt. The MT[3] bit is ignored so legacy PIC NMI and ExtInt get mapped on to the regular NMI and ExtInt interrupts.

G. CRC Testing Mode

The CRC Testing mode is supported. The data used during the test is UNPREDICTABLE.

H. Doubleword Based Data Buffer Flow Control

The interface does not support the optional doubleword flow control.

ORDERING RULES

The HyperTransport fabric and the PCI bus have a set of ordering rules described in Appendix E of the PCI specification revision 2.2. These differ from the more relaxed ordering rules used on the ZBbus. The PCI/ HyperTransport rules are imposed in the I/O bridge which will allow posted writes to pass reads and will not allow read responses to pass posted writes, allowing the producer-consumer model to be used. Using this model requires some care, see Section: “Ordering Rules and Device Drivers” on page 14.

A deadlock exists in the ordering boundary between the ZBbus and I/O Bridge 1 (in the HyperTransport terms bridge 1 does not have sufficient virtual channels). This is only provoked when writes from I/O Bridge 1 are targeted at the PCI/HT space (the access must have come from one of the MAC or Serial DMA engines) at the same time as any write is in progress from the PCI/HT to any address serviced by Bridge 1 (see Figure 5 on page 9 for the devices that are serviced by Bridge 1) or a read is in progress from I/O Bridge 1 to bridge 0. This deadlock can be avoided by forbidding the Bridge 0 to Bridge 1 writes and the Bridge 1 to Bridge 0 reads.
USING THE PCI IN DEVICE MODE

The part can be used in PCI Device Mode in a couple of different situations. The main one is when it is used as a PCI device on an option card and the second is when the PCI is used to connect several parts in a system.

The BAR0, BAR2, BAR3 and ROM BAR are active in Device Mode. Most accesses by an external host will be done through BAR0 and the mapping table, BAR2 and BAR3 will be used for the host to communicate with the SB-1 CPUs via the mailbox registers and the ROM BAR will typically only be used during initialization. When the part needs to make a memory space access on the PCI it will use the Full Access space, which allows any (32 bit) PCI address to be generated, I/O accesses can be made through the normal I/O space range.

In Device Mode the PCI configuration is mostly done by the host in the system. However the BAR0 Map registers, the SubSysSet register (and the vendor IdSet and ClassRevSet) and the INTA control bit can only be set by configuration accesses from the ZBbus side of the PCI interface and are therefore under control of software on the device.

The address map on the PCI bus is different from the address map within the part, and translation is done through the BAR0 address map. The map table allows control of what areas of the local memory map are accessible to an external PCI master. Therefore it is protected and can only be used with configuration accesses from the ZBbus side of the bridge. Configuration accesses to the map table registers will be accepted from the PCI bus, but the writes will be ignored and reads will return UNPREDICTABLE values. The external master will always see the BAR0 region as a 16 MByte space that needs an address allocation, software running on the part should configure the map table to allow access to appropriate internal resources. The peer-to-peer mappings in the map register give the device some of the characteristics of a non-transparent PCI to HyperTransport bridge.

The SubSysSet register can only be written from the ZBbus side of the bridge. It is used to provide a value in the SubSystem Device and Vendor Id registers seen by external configuration reads. In Device Mode software on the part should write the SubSysSet register with a value that identifies the manufacturer of the option card and its part number. This write must be done early in initialization, before the host needs to read the value. On interface revision 3 and greater the Device/Vendor ID register and the Class/Rev register seen by external configuration reads can also be set using the VendorIdSet and ClassRevSet registers.

The INTA control bit can be used by the device to signal a PCI interrupt on the (bidirectional) P_INTA_L pin. Again, this bit can only be accessed from the ZBbus side and external accesses have UNPREDICTABLE results.

The P_RST_L signal is only an output, and is therefore only appropriate for use when the part is in Host Mode. In Device Mode the pin is not used. There are two options for connecting the PCI reset signal. When the part is used on an option card it should be reset when the PCI bus is reset, therefore the PCI reset should drive the RESET_L input to the part. When the PCI is used to connect multiple parts in a system they may need to be independently reset, in this case the PCI reset line can be connected to one of the GPIO pins used as an interrupt input. Software running on the Device Mode part can then read the state of the PCI reset and can be interrupted to take appropriate action if the PCI bus is unexpectedly reset during normal operation.

Accesses to the PCI configuration registers in device mode are confused by their being shared between accesses from within the part (ZBbus accesses) and accesses originating as Type 0 configuration cycles on the PCI bus (PCI accesses). In addition the interface must ensure that the host doing PCI accesses does not attempt to read the SubSystem information before local software has written the SubSysSet register. The rd_host bit, in the WRITE ONLY Device control register, is used to control the access. This bit is only used in device mode. It defaults to being set which allows read and write ZBbus accesses to the static interface identification, the BAR0 map entries, the SubSysSet register and the SignalINTA register. While the bit is set
any PCI access will be issued with a retry. Software on the part should do initial configuration of the map
registers and set the subsystem information before writing a zero to the rd_host bit and flush the write to
the interface by following it with a dummy read that will return UNPREDICTABLE data. To be compliant with
the PCI specification this must be done within 1 second (0.55 on a 66 MHz bus) of the PCI reset being deasserted.

While the system is running with the rd_host bit clear ZBus writes may be done to the map entries and the
SignalNTA register. Writes to the SubSysSet register will result in the SubSystem ID being updated, but the
results of changing this after device enumeration is system specific: the subsystem information may never be
read again or the operating system may assert an error because static information changes. Reads to any of
the PCI configuration registers will return UNPREDICTABLE data. The rd_host bit may be set to give read
access to the map entries, SubSysSet and SignalNTA, but while the bit is set any read requests from the PCI
bus will be issued retries (read access will probably only be needed for debugging).

Table 131 summarizes access to the PCI configuration registers.

**Table 126: PCI CSR Access Rules**

<table>
<thead>
<tr>
<th>Register</th>
<th>ZBus Read</th>
<th>ZBus Write</th>
<th>PCI Read</th>
<th>PCI Write</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Device/Vendor ID (0)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host Mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valid data</td>
<td>Ignored</td>
<td>Unpredictable</td>
<td></td>
<td>Unpredictable</td>
</tr>
<tr>
<td><strong>Class/Revision (8)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SubSysSet (8C)</td>
<td>Valid data</td>
<td>Write register</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>SignalNTA (90)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface revision 3 or later</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VendorIDSet (9C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassRevSet (A0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>Valid data</td>
<td>Write register</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td><strong>Device mode with rd_host = 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valid data</td>
<td>Ignored</td>
<td>Retried</td>
<td>Unpredictable</td>
<td></td>
</tr>
<tr>
<td><strong>Device/Vendor ID (0)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class/Revision (8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map Entries (44-80)</td>
<td>Valid data</td>
<td>Write register</td>
<td>Retried</td>
<td>Ignored</td>
</tr>
<tr>
<td>SubSysSet (8C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SignalNTA (90)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface revision 3 or later</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VendorIDSet (9C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassRevSet (A0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
<td>Retried</td>
<td>Unpredictable</td>
</tr>
<tr>
<td><strong>Device mode with rd_host = 0</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Valid data</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
<td>Valid data</td>
<td>Ignored</td>
</tr>
<tr>
<td><strong>Device/Vendor ID (0)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class/Revision (8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map Entries (44-80)</td>
<td>Unpredictable</td>
<td>Write register</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
</tr>
<tr>
<td>SubSysSet (8C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SignalNTA (90)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface revision 3 or later</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VendorIDSet (9C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassRevSet (A0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>Unpredictable</td>
<td>Unpredictable</td>
<td>Valid data</td>
<td>Write register</td>
</tr>
</tbody>
</table>
CONFIGURATION OF PCI AND HYPERTRANSPORT

The PCI bus and HyperTransport fabric both require configuration before use. This is normally done by the startup firmware, but may also be done by the operating system as it starts. There is usually a "PCI-BIOS" abstraction layer above the configuration hardware (this is only well specified for the x86 architecture, but most systems have a version of it). Since HyperTransport devices use the same configuration headers as PCI, only minor changes to the configuration software will be needed to support it. In particular accesses to the PCI/ HyperTransport memory or I/O ranges are UNPREDICTABLE until the HyperTransport mem_base, mem_limit, io_base and io_limit registers have been programmed, and configuration accesses to anything other than bus 0 are UNPREDICTABLE until the HyperTransport secondary and subordinate bus numbers have been programmed. (See Section: “Systems That Do Not Use HyperTransport” on page 236 if HyperTransport is not used.)

During configuration the tree of bridged buses (an entire HyperTransport chain is regarded as a single bus) is built along with a list of devices on each bus and their memory size requirements. The buses are allocated numbers, with bus 0 being at the root of the tree. A bridge gets configured with a bus number for its primary interface (the one that is closest to the root of the tree), a bus number for its secondary interface (the other bus it directly connects to) and a subordinate bus number which is the highest numbered bus that is behind the bridge (all buses in the range secondary - subordinate are accessible through the bridge).

Once all the buses are enumerated the configuration code will assign base addresses to all the devices. There are three separate address ranges: for memory mapped I/O, prefetchable memory, and I/O space. In each range, all addresses allocated behind a bridge must be contiguous.

Configuration uses the logical organization of the interfaces shown in Figure 36 on page 191. The PCI bus on the part is bus 0 at the top of the configuration tree and therefore has its configuration space at bus 0, device 0. The HyperTransport bridge configuration is at bus 0, device 1. (The bus number that will be given to the HyperTransport fabric depends on the enumeration software).

The configuration space for PCI and HyperTransport is based at 00_FE00_0000 for match bit lanes (the most useful endian policy for configuration) with a repeat at 00_DE00_0000 using the match byte lane policy. To access a configuration register the address is formed from the register, function, device and bus number as shown in Figure 49. Reads and writes of up to 32 bits can be done. The results of doing a 64 bit access to configuration space are UNDEFINED. Note that in Device Mode the PCI interface will mostly be configured by the host in the system, but the device mode part must still configure the BAR0 address mapping table and the HyperTransport fabric.

<table>
<thead>
<tr>
<th>39</th>
<th>32</th>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>8</th>
<th>7</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>FE</td>
<td>Bus Number</td>
<td>Device Number</td>
<td>Function Number</td>
<td>Register Number</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 49: Configuration Space Address
Accesses to the configuration range with bus 0 device 0 and bus 0 device 1 will be directed to the internal configuration registers for the PCI and HyperTransport interfaces. Once the CPU has performed a series of writes to the internal configuration registers it should read back at least one. This will ensure the writes have taken effect before any other operations will proceed. Although the transfers will be in order on the ZB bus, the data phase will be behind the address. If there is no read separating a configuration write from a subsequent transaction it is possible for the address of the transaction to reach the interface before the data of the configuration write, in this case the address decode for the transaction would be done based on the old value of the configuration register.

Other accesses to bus 0 devices will be sent as Type 0 configuration cycles on the PCI bus, the mapping used for generating IDSEL is to convert the device number into a single one on AD[31:11]. Since the internal PCI and HyperTransport headers are device 0 and 1 the external devices start with AD[13] being set for device 2, AD[14] being set for device 3 and so on up to AD[31] being set for device 20 (note that the internal PCI arbiter only supports four external devices). During PCI configuration cycles the interface will pre-drive the address on the bus for a cycle (as described in section 3.2.2.3.5 of the PCI Specification revision 2.2), this allows a resistor to be used to connect to the IDSEL inputs of PCI devices. A PCI configuration read to a nonexistent device will be terminated with a master abort and set the status flags to indicate this happened, the data returned will be marked valid and will contain 32'hFFFF_FFFF (for normal accesses, a master abort will result in a bus error return). When a PCI configuration read is aborted in this way all other reads that are in flight in the PCI bridge will be terminated with a bus error. Software should therefore ensure that there are no reads in progress through the PCI bridge when it is probing the PCI bus for devices. Note that configuration reads that encounter other PCI errors (e.g. a parity error) will also return 32'hFFFF_FFFF.

If the bus number of an access to the configuration range matches the secondary bus number configured in the HyperTransport Bridge, the request will be forwarded as Type 0 configuration accesses on the HyperTransport fabric. This is done by turning it in to an access to the address range given in the HyperTransport specification for configuration. Accesses that have a bus number greater than the HyperTransport bridge secondary bus and less than or equal to the HyperTransport subordinate bus number are sent as Type 1 configuration accesses on the HyperTransport fabric. Errors generated from configuration accesses to HT will be forwarded to the ZBus rather than the PCI bridge. All other accesses are sent as Type 1 configuration accesses on the PCI bus. This is shown algorithmically in the pseudo-code below.

```plaintext
 endian = zbaddr[29] ? match bit lanes : match byte lanes
 bus = zbaddr[23:16]
 device = zbaddr[15:11]
 func = zbaddr[10:8]
 reg = zbaddr[7:0]
 if (bus == 0)
 { if (dev == 0)
 access PCI config reg
 else if (dev == 1)
 access HyperTransport config reg
 else
 do PCI Type 0 cycle (dev, func, reg)
 }
 else if (bus == HT secondary bus)
 { // Form special HT Type 0 config address
 htaddr = FD_FEO0_0000 + zbaddr[15:0]
 do HT access
 }
 else if (HT secondary bus < bus <= HT subordinate bus)
 { // Form special HT Type 1 config address
 htaddr = FD_FF00_0000 + zbaddr[23:0]
 do HT access
 }
 else
 do PCI Type 1 cycle (bus, dev, func, reg)
```
HyperTransport Target Done Counter

Since the MIPS architecture does not support non-posted writes it is hard for software to tell when HyperTransport configuration writes have completed. To help with this a counter has been implemented in the HyperTransport Additional Status Register. This counter can be cleared by software writing a zero and will increment every time a TgtDone message is returned from the fabric. Thus software can clear the counter, do all the writes to configure a HyperTransport device and then poll the counter until it reaches the number of writes performed. This will ensure the final write has been acknowledged by the device.

SYSTEMS THAT DO NOT USE HYPERTRANSPORT

If a BCM1250 or BCM1125H system does not use the HyperTransport interface the mem_base, mem_limit, io_base and io_limit registers of the HT bridge must still be programmed. This configures a shadow copy of these registers that is by default UNPREDICTABLE. The mem_limit should be set to be lower in address than the mem_base. The io_base upper 16 bits (bits [15:0] of offset h30 in the HyperTransport configuration header) should be set above the 24 bit range used by the I/O space, for example the register can be written with 32'h0000F200 (note that subsequent reads will show bits [15:12] as zero). For configuration accesses to buses other than bus 0 to work the HyperTransport secondary and subordinate bus numbers should be written with zero. Until these registers are programmed accesses to the PCI memory or I/O space will have UNDEFINED results.

CONFIGURATION HEADER DESCRIPTIONS

The following sections describe the configuration headers used by the PCI and HyperTransport interfaces. The PCI interface is bus=0, device=0, function=0 so the header can be accessed from within the part starting at the base address is 00_FE00_0000 (match bits) or 00_DE00_0000 (match bytes). The HyperTransport interface is bus=0, device=1, function=0, with base address 00_FE00_0800 (match bits) or 00_DE00_0800 (match bytes).

The following terms are used in the register descriptions:

- R/O. Read Only. The field contains a static value and should not be written.
- R/W. Read/Write. The field can be modified by software.
- R/C. Read/Clear. The field can be read to give status information. Bits may be cleared by writing a 1 to them, writes of 0 are ignored.

PCI CONFIGURATION HEADER

The PCI configuration header is a standard Type 0 header with extensions to support the address map table. The following table shows the fields of the header, along with the values they contain after a system reset.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Device Id R/O 0001</td>
<td>Vendor Id R/O 166D</td>
</tr>
<tr>
<td>04</td>
<td>Status (see Table 129 on page 239) R/W 02A0</td>
<td>Command (See Table 128 on page 239) R/W 0002</td>
</tr>
</tbody>
</table>

Table 127: PCI Interface Configuration Header (Type 0)
### Table 127: PCI Interface Configuration Header (Type 0) (Cont.)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>Class Code R/O 060000</td>
<td>Class is a host bridge, revision reflects the interface revision code. The revision code is: 1 - for early prototype BCM1250s, 2 - for initial production BCM1250s, 3 - for intial production BCM1125/H and later BCM1250s</td>
</tr>
<tr>
<td>0C</td>
<td>BIST R/O 00,Hdr type R/O 00,LatTimer (Table 130 on page 240) R/W 00, ClineSz (Table 131 on page 240) R/W 00</td>
<td>The bridge uses a Type 0 device header.</td>
</tr>
<tr>
<td>10</td>
<td>BAR 0 - Map Table Host: R/O 6000_0008 Dev: R/W xx00_0008</td>
<td>Hits to this 16 MB BAR will be translated through the mapping table. Bit 3 is set to indicate a prefetchable region.</td>
</tr>
<tr>
<td>14</td>
<td>BAR 1 - Reserved R/O 00000000</td>
<td>This BAR is reserved and will always be read as zero</td>
</tr>
<tr>
<td>18</td>
<td>BAR 2 - mbox 0 Host: R/O 7000_0008 Dev: R/W xxx0_0008</td>
<td>Hits to this 4 KB BAR are translated into accesses to the mailbox set and value register for CPU0.</td>
</tr>
<tr>
<td>1C</td>
<td>BAR 3 - mbox 1 Host: R/O 7100_0008 Dev: R/W xxx0_0008</td>
<td>Hits to this 4 KB BAR are translated into accesses to the mailbox set and value register for CPU1.</td>
</tr>
<tr>
<td>20</td>
<td>BAR 4 - low memory Host: R/O 0000_0008 Dev: R/O 0000_0008 (Reserved)</td>
<td>Hits to this BAR are passed through as accesses to the low 512MB of memory.</td>
</tr>
<tr>
<td>24</td>
<td>BAR 5 - High Memory Host: R/O 8000_0008 Dev: R/O 0000_0008 (Reserved)</td>
<td>Hits to this BAR are passed through as accesses to the upper 2GB of the low 4GB of the address space.</td>
</tr>
<tr>
<td>28</td>
<td>Cardbus CIS R/O 00000000</td>
<td>This register is not used.</td>
</tr>
<tr>
<td>2C</td>
<td>SubSystem Id R/O 0000, SubSys Vendor R/O FFFF</td>
<td>These registers are not used internally since this is a host bridge. If the Configuration registers are read by an external PCI master with the bridge in Device Mode the value read from this register will be the value written from the ZBbus to the SubSysSet register (offset 8C).</td>
</tr>
<tr>
<td>30</td>
<td>ROM Base Address Host: R/W 73000000 Dev: R/W xxx0_0000</td>
<td>Hits to this 64 KB BAR are translated to accesses to the boot rom area of memory.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved R/O 000000</td>
<td>This register is not used.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved R/O 00000000</td>
<td>This register is not used.</td>
</tr>
<tr>
<td>3C</td>
<td>max_lat R/O 00, min_gnt R/O 00, Int Pin R/O 01, Int Line R/W 00</td>
<td>The Max Latency and Min Grant registers are used to specify the device preferences for the latency timer, they default to indicate no special requirement. The IntPin register indicates that this device uses INTA to interrupt in Device Mode. The IntLine register is used by firmware and system software.</td>
</tr>
<tr>
<td>40</td>
<td>FControl (See Table 133 on page 241) R/W 0003, Timeout (See Table 132 on page 240) R/W 8080</td>
<td>This additional register allows configuration of the retry and TRDY timeouts and enable bits for special features.</td>
</tr>
</tbody>
</table>
### Table 127: PCI Interface Configuration Header (Type 0) (Cont.)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>44-80</td>
<td>Map Entry Base Address R/W 00000000</td>
<td>The address mapping table for BAR0, described in Table 134 on page 241.</td>
</tr>
<tr>
<td>84</td>
<td>Error Address R/O xxxxxx</td>
<td>This register records the address of every transaction on the PCI bus. When any of the error bits in the status register are set the address in this register is locked, recording the address of the transaction that experienced the error. The register is unlocked when software clears the error status.</td>
</tr>
<tr>
<td>88</td>
<td>Additional Status and Command (See Table 135 on page 241) R/W 0000000000</td>
<td>This register contains addition command and status bits.</td>
</tr>
<tr>
<td>8C</td>
<td>SubSysSet R/W 0000000000</td>
<td>This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 2C.</td>
</tr>
<tr>
<td>90</td>
<td>SignalIntA R/W 0000000000</td>
<td>Bits 31:1 are R/O 0. Bit 0 can be set from the ZBbus to assert the INTA pin. Writing this bit from the PCI has UNPREDICTABLE results.</td>
</tr>
<tr>
<td>94</td>
<td>ReadHost W/O 0000000001</td>
<td>This register controls register accesses in device mode. See Table 137 on page 242 and “Using the PCI in Device Mode” on page 232.</td>
</tr>
<tr>
<td>98</td>
<td>Adaptive Extend R/W xxxxx00</td>
<td>This register controls performance features. See Table 138 on page 242.</td>
</tr>
<tr>
<td>9C</td>
<td>VendorIdSet R/W 0001166D</td>
<td>This register is only available if the interface Rev Id is 3 or greater. This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 0.</td>
</tr>
<tr>
<td>A0</td>
<td>ClassRevSet R/W 06000003</td>
<td>This register is only available if the interface Rev Id is 3 or greater. This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 4.</td>
</tr>
<tr>
<td>A4</td>
<td>BypassCtrl R/W 00000FFF</td>
<td>This register is only available if the interface Rev Id is 3 or greater. This register controls the number of times a ZBbus initiated write can pass a retrying read before the read is discarded and a fatal error signalled.</td>
</tr>
<tr>
<td>A8-FC</td>
<td>Reserved</td>
<td>Reserved, accesses will have UNPREDICTABLE results.</td>
</tr>
</tbody>
</table>
**Table 128: PCI Command Register - Offset 4 Bits [15:0]**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IoSpaceEn</td>
<td>R/O 1'b0</td>
<td>I/O Space Enable. This bit is always zero. The bridge never accepts I/O space accesses from the PCI bus.</td>
</tr>
<tr>
<td>1</td>
<td>MemSpaceEn</td>
<td>R/W 1'b1</td>
<td>Memory Space Enable. This bit must be set to allow the bridge to accept memory space accesses from the PCI bus and forward them to either ZBus or the HyperTransport bridge. In interface revisions 1 and 2 this bit defaults to 1 to enable these accesses after reset, in interface revision 3 and greater this bit defaults to 0 to match the PCI specification.</td>
</tr>
<tr>
<td></td>
<td>RevId&gt;=3</td>
<td>R/W 1'b0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MasterEn</td>
<td>R/W 1'b0</td>
<td>Bus Master Enable. This bit must be set to allow the bridge to act as a master on the PCI bus. This bit should be set before any PCI accesses are made. Any accesses to the PCI space that are received while this bit is clear will result in UNDEFINED behavior in the interface.</td>
</tr>
<tr>
<td>3</td>
<td>SpecCycEn</td>
<td>R/O 1'b0</td>
<td>The bridge does not accept Special Cycles, so this bit is always zero.</td>
</tr>
<tr>
<td>4</td>
<td>MemWrInvEn</td>
<td>R/W 1'b0</td>
<td>This bit should be set to allow the bridge to generate Memory Write and Invalidate commands. The bridge will only use Write Invalidate if the start address is cacheline aligned and the transfer is at least one cache line long (using the cache line size set in the ClineSz register). If the latency timer expires while a Write Invalidate command is in progress the access will only be terminated at the next cache line boundary.</td>
</tr>
<tr>
<td>5</td>
<td>VgaPalSnpEn</td>
<td>R/O 1'b0</td>
<td>This bit is always zero. The bridge does not snoop VGA palette accesses.</td>
</tr>
<tr>
<td>6</td>
<td>ParErrResp</td>
<td>R/W 1'b0</td>
<td>This bit controls the response to PCI parity errors. If it is set then the MstrDParErr bit is set in the status register and the PCI error interrupt is raised when a parity error is detected. If it is clear then no interrupt is raised. In both cases the DetParErr bit is set in the status register. A parity error resulting from a read will return data to the ZBus with a bus error if this bit is set, and returned as valid data if this bit is clear.</td>
</tr>
<tr>
<td>7</td>
<td>StepCtrl</td>
<td>R/O 1'b0</td>
<td>This bit is always clear. The bridge does not do address/data stepping.</td>
</tr>
<tr>
<td>8</td>
<td>SerrEn</td>
<td>R/W 1'b0</td>
<td>SERR enable. If this bit is set, the bridge will drive the SERR_L pin if it detects a fatal error. If clear the bridge will never drive SERR_L.</td>
</tr>
<tr>
<td>9</td>
<td>FastB2BCap</td>
<td>R/O 1'b0</td>
<td>This bit is always clear. The bridge will never do fast back-to-back cycles to different devices.</td>
</tr>
<tr>
<td></td>
<td>reserved 15:10</td>
<td>R/O 6'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Table 129: PCI Status Register - Offset 4 Bits [31:16]**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>reserved</td>
<td>R/O 4'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>CapList</td>
<td>R/O 1'b0</td>
<td>Always clear. There is no capabilities list.</td>
</tr>
<tr>
<td>5</td>
<td>66MHzCap</td>
<td>R/O 1'b1</td>
<td>This bit is set. The PCI is 66 MHz capable.</td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
<td>R/O 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>FastB2BCap</td>
<td>R/O 1'b1</td>
<td>This bit indicates that the bridge can accept fast back-to-back transactions when the transactions are to different agents.</td>
</tr>
<tr>
<td>8</td>
<td>MstrDParErr</td>
<td>R/C 1'b0</td>
<td>This bit is set if the bridge was a bus master and detected a parity error on read data, or was signalled that a parity error happened on write data. The ParErrEn bit in the command register must be set to allow this bit to be set. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.</td>
</tr>
<tr>
<td>10:9</td>
<td>DevselTiming</td>
<td>R/O 2'b0</td>
<td>The bridge generates medium DEVSEL timing.</td>
</tr>
</tbody>
</table>

---

**Broadcom Corporation**
Table 129: PCI Status Register - Offset 4 Bits [31:16] (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>SigdTgtAbort</td>
<td>R/C 0'b0</td>
<td>This bit is set when the bridge is the target of a PCI transaction and signalled a Target Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.</td>
</tr>
<tr>
<td>12</td>
<td>RcvdTgtAbort</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge was a master for a transaction and received a Target Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.</td>
</tr>
<tr>
<td>13</td>
<td>RcvdMstrAbort</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge was a master for a transaction that was not a Special Cycle and signalled a Master Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.</td>
</tr>
<tr>
<td>14</td>
<td>SigdSerr</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge asserts SERR_L. It will do this when it detects a parity error on an address. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.</td>
</tr>
<tr>
<td>15</td>
<td>DetParErr</td>
<td>R/C 1'b0</td>
<td>This bit will be set whenever the bridge detects a parity error, even if parity error handling is disabled. The parity error is only signalled using the PCI interrupt if the ParErrResp bit is set in the command register. This bit is cleared by software writing a 1.</td>
</tr>
</tbody>
</table>

Table 130: PCI Latency Timer - Offset 0C Bits [15:8]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>LatTime</td>
<td>R/W 8'b0</td>
<td>This contains the value of the Latency Timer for the bridge to use when bus master. If this register is left at its default value of zero the bridge will never do bursts on the PCI. Bits 1:0 are R/O and are always zero.</td>
</tr>
</tbody>
</table>

Table 131: PCI Cache Line Size - Offset 0C Bits [7:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>ClineSz</td>
<td>R/W 8'h0</td>
<td>This register sets the cache line size in DWORDS (32 bit words) used by the bridge for PCI transactions. As a master the bridge bases the PCI command used on how the size of the transfer matches the value in this register: word_count &lt; one cacheline: use Read command word_count &gt;= one cacheline: use Read Line command word_count &gt;= two cache lines: use Read Multiple command. This size is also used to enable the Write Invalidate command.</td>
</tr>
</tbody>
</table>

Table 132: PCI Timeout Register - Offset 40 Bits [15:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>TrdyTimeout</td>
<td>R/W 8'h80</td>
<td>This field sets the maximum number of PCI cycles that the bridge will wait for TRDY to be asserted. If TRDY is deasserted for longer than this the transfer will be aborted. When a transfer is aborted the TrdyErr bit is set in the Additional Status register and the pci_interrupt may be raised. If a read is aborted UNPREDICTABLE data is returned to the ZBus marked with a bus error.</td>
</tr>
<tr>
<td>15:8</td>
<td>RetryTimeout</td>
<td>R/W 8'h80</td>
<td>This field sets the number of times a read will be retried following a disconnect. The read is first attempted this number of times. The bridge will then allow any outstanding writes and read-data-returns to proceed before retrying the read. In interface revisions less than 3 a second attempt is made, in revisions 3 and later the read is retried the number of times specified in the num_piowr_bypass field of the PCI Bypass Control register (Table 139). If the read fails on the final attempt the transfer is aborted, UNPREDICTABLE data marked with a bus error is returned to the ZBus and the RetryErr bit is set in the Additional Status register. A write that is retried will be attempted the number of times specified in this field before being discarded and the RetryErr bit set.</td>
</tr>
</tbody>
</table>
Each map table entry contains the mapping address and flags. There are 16 entries in the table, and it is indexed by bits [23:20] of the PCI address.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>enable</td>
<td>1'b0</td>
<td>This bit must be set to enable the mapping. If it is clear the PCI interface will not assert DEVSEL# to accept the request.</td>
</tr>
<tr>
<td>1</td>
<td>send_ldt</td>
<td>1'b0</td>
<td>If this bit is set then the request will be forwarded to the HyperTransport fabric (regardless of what the address is). If this bit is set the request is sent into the BCM1250.</td>
</tr>
<tr>
<td>2</td>
<td>l2ca</td>
<td>1'b0</td>
<td>If this bit is set then the L2CA flag will be set when an access is made to the ZBus through this entry. This requests that the cache block be allocated in the L2 cache.</td>
</tr>
<tr>
<td>3</td>
<td>endian</td>
<td>1'b0</td>
<td>If the system is configured for big endian operation and this bit is set the match bytes policy will be used, if this bit is clear the match bytes policy is used. If the system is configured little endian this bit is ignored.</td>
</tr>
<tr>
<td>11:4</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>31:12</td>
<td>addr</td>
<td>20'b0</td>
<td>These bits provide bits 39:20 of the internal address. Bits 19:0 are copied from the PCI address.</td>
</tr>
</tbody>
</table>

Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hotplug_en</td>
<td>R/W 1'b0</td>
<td>If this bit is set the hotplug function in the internal PCI arbiter is enabled.</td>
</tr>
<tr>
<td>1</td>
<td>serr_det</td>
<td>R/C 1'b0</td>
<td>Set when some other device on the PCI bus asserts SERR (a PCI error interrupt is also raised). Software must write a 1 to this bit to clear it and remove the interrupt.</td>
</tr>
<tr>
<td>2</td>
<td>TrdyErr</td>
<td>R/C 1'b0</td>
<td>Set when a transfer is aborted because TRDY was not asserted within the TrdyTimeout (See Table 132 on page 240). The TrdyIntEn bit must be set in order for this bit to be set. Software must write a 1 to this bit to clear it and remove the interrupt.</td>
</tr>
</tbody>
</table>
### Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0] (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>RetryErr</td>
<td>R/C 1'b0</td>
<td>Set when a transfer is aborted because a read was retried more than the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RetryTimeout (See Table 132 on page 240). The RetryIntEn bit must be set in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>order for this bit to be set. Software must write a 1 to this bit to clear</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>it and remove the interrupt.</td>
</tr>
<tr>
<td>4</td>
<td>TrdyIntEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set the PCI error interrupt will be raised and the TrdyErr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit set on a TRDY timeout. If this bit is clear no interrupt will be raised</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and the TrdyErr bit will not be set.</td>
</tr>
<tr>
<td>5</td>
<td>RetryIntMask</td>
<td>R/W 1'b0</td>
<td>If this bit is set the PCI error interrupt will be raised and the RetryErr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bit set on a Retry timeout. If this bit is clear no interrupt will be raised</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>and the RetryErr bit will not be set.</td>
</tr>
<tr>
<td>6</td>
<td>DisMulti</td>
<td>R/W 1'b0</td>
<td>This bit should be clear for normal operation.</td>
</tr>
<tr>
<td>31:7</td>
<td>reserved</td>
<td>R/0 25'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 136: PCI INTA Control Register - Offset 90 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>signalINTA</td>
<td>R/W 1'b0</td>
<td>If this bit is set the P_INTA_L line will be asserted. This is intended for use in Device Mode when the device needs to interrupt the host. Note</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>that the input from P_INTA_L is still active, so setting the bit will raise a PCI INTA interrupt to the interrupt mapper. This register should</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>only be accessed from the ZBus side of the PCI interface. Writing to this register from the PCI interface will have UNPREDICTABLE results.</td>
</tr>
<tr>
<td>31:1</td>
<td>reserved</td>
<td>R/0 31'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 137: PCI Read Host Register - Offset 94 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rd_host</td>
<td>W/O 1'b1</td>
<td>This bit controls CPU read access to the PCI configuration registers when the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>interface is in device mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>By default this bit is set, allowing read access to the static PCI registers,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the BAR0 map, the SubSysSet and SignalINTA registers. While this bit is set any</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>read requests from the PCI bus will be issued a retry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When this bit is clear the CPU can still write the registers but reads will</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>return UNPREDICTABLE data.</td>
</tr>
<tr>
<td>31:1</td>
<td>reserved</td>
<td>W/O 31'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 138: PCI Adaptive Extend Register - Offset 98 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>3:1</td>
<td>nom_tar_retry_h</td>
<td>R/W 3'b0</td>
<td>This field is the top three bits of the nom_tar_retry value used by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>adaptive retry algorithm. (See Table 133 PCI Feature Control Register bits 10:7)</td>
</tr>
<tr>
<td>5:4</td>
<td>max_tar_retry_h</td>
<td>R/W 2'b0</td>
<td>This field is the top two bits of the max_tar_retry value used by the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>adaptive retry algorithm. (See Table 133 PCI Feature Control Register bits 15:11)</td>
</tr>
<tr>
<td>6</td>
<td>dis_dmar_iow_dep</td>
<td>1'b0</td>
<td>This bit should be zero for normal operation. If set no checking is performed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for dependencies between DMA read data return and PIO writes.</td>
</tr>
</tbody>
</table>
In Host Mode the header should be configured by the local processor (in many cases the default values will be reasonable and only the map table will need to be initialized), in Device Mode it will be configured by the external host.

The map table allows control of what areas of the internal memory map are accessible to an external PCI master. Therefore it is protected and can only be used with configuration accesses from the ZBbus side of the bridge. Configuration accesses to the map table registers will be accepted from the PCI bus, but the writes will be ignored and reads will return UNPREDICTABLE values. When the interface is run in Device Mode, the external master will always see the BAR0 region as a 16 MByte space that needs an address allocation, software running on the device should configure the map table to allow access to appropriate internal resources. The peer-to-peer mappings in the map register give the part some of the characteristics of a non-transparent PCI to HyperTransport bridge.

The SubSysSet register can only be written from the ZBbus side of the bridge. It is used to provide a value in the SubSystem Device and Vendor Id registers seen by external configuration reads. In Device Mode software on the device should write the SubSysSet register with a value that identifies the manufacturer of the option card and its part number. This write must be done early in initialization, before the host needs to read the value. On interface revision 3 and later the Device/Vendor Id and the Class code that will be read by the external host can also be changed by using the VendorIdSet and ClassRevSet registers.

### Table 138: PCI Adaptive Extend Register - Offset 98 Bits [31:0] (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>dis_memrd_be</td>
<td>1'b0</td>
<td>This bit should be zero for normal operation. If set the byte enables will always be all set for read transactions.</td>
</tr>
<tr>
<td>8</td>
<td>prefetch_en</td>
<td>R/W 1'b0</td>
<td>Interface revision 3 or greater: this bit may be set to enable read prefetching to improve performance of block reads from the ZBbus to the PCI.</td>
</tr>
<tr>
<td>9</td>
<td>prefetch_sz</td>
<td>R/W 1'b0</td>
<td>Interface revision 3 or greater: if this bit is 0 two cachelines are prefetched when the prefetch_en bit is set, if it is 1 then 4 cachelines are prefetched.</td>
</tr>
<tr>
<td>31:10</td>
<td>notimp</td>
<td>22b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 139: PCI Bypass Control Register - RevId >= 3 Offset A8 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:0</td>
<td>num_pior_byp</td>
<td>R/W 12'hFFF</td>
<td>A ZBbus or HT-to-PCI initiated read will be retried a total of num_pior_byp multiplied by the retry_timeout (in the Timeout register). Each retry_timeout there is a chance for one write to bypass the read (this is a workaround for older devices that did not correctly implement the PCI ordering rules).</td>
</tr>
<tr>
<td>31:12</td>
<td>notimp</td>
<td>20b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

In Host Mode the header should be configured by the local processor (in many cases the default values will be reasonable and only the map table will need to be initialized), in Device Mode it will be configured by the external host.
HYPERTRANSPORT CONFIGURATION HEADER

The HyperTransport configuration header is a standard PCI Type 1 (bridge) header, as defined in the PCI-to-
PCI Bridge Architecture Specification revision 1.1, with the extensions required by the HyperTransport
specification and device specific control extensions. The following table shows the fields of the header, along
with the values they contain after a system reset. Values marked R/O are read only.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Device Id R/O 0002</td>
<td>Vendor Id R/O 166D</td>
</tr>
<tr>
<td>04</td>
<td>Status (See Table 142 on page 247) R/W 0010</td>
<td>Command (See Table 141 on page 246) R/W 0003</td>
</tr>
<tr>
<td>08</td>
<td>Class Code R/O 060000</td>
<td>Rev Id R/O xx</td>
</tr>
<tr>
<td>0C</td>
<td>BIST R/O 00</td>
<td>Hdr Type R/O 01</td>
</tr>
<tr>
<td>10</td>
<td>BAR 0 R/O 00000000</td>
<td>Not used.</td>
</tr>
<tr>
<td>14</td>
<td>BAR 1 R/O 00000000</td>
<td>Not used.</td>
</tr>
<tr>
<td>18</td>
<td>SecLatTimer R/O 00</td>
<td>Subord Bus# R/W 00</td>
</tr>
<tr>
<td>1C</td>
<td>Sec Status (See Table 143 on page 247) R/C 0000</td>
<td>I/O Limit R/W x1</td>
</tr>
<tr>
<td>20</td>
<td>Mem Limit R/W xxx0</td>
<td>Mem base R/W xxx0</td>
</tr>
<tr>
<td>24</td>
<td>Prefetch Limit R/O 0000</td>
<td>Prefetch base R/O 0000</td>
</tr>
<tr>
<td>28</td>
<td>Prefetch base upper 32 bits R/O 00000000</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>Prefetch limit upper 32 bits R/O 00000000</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>I/O limit upper 16 bits R/W 0xxx</td>
<td>I/O base upper 16 bits R/W 0xxx</td>
</tr>
</tbody>
</table>
### Table 140: HyperTransport Configuration Header (Type 1) (Cont.)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>Reserved R/O 000000</td>
<td>Cap Ptr R/O 40</td>
</tr>
<tr>
<td>38</td>
<td>ROM base address R/O 0</td>
<td></td>
</tr>
<tr>
<td>3C</td>
<td>Bridge Ctrl (See Table 144 on page 248) R/W 0000</td>
<td>Int Pin R/O 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>HTCmd (See Table 145 on page 248) R/W 2001</td>
<td>Cap Ptr R/O 00</td>
</tr>
<tr>
<td>44</td>
<td>LinkConfig (See Table 147 on page 250) R/O 0000</td>
<td>LinkCtrl (See Table 146 on page 249) R/W 0000</td>
</tr>
<tr>
<td>48</td>
<td>Reserved R/O 0000</td>
<td>LDT Freq (Table 148 on page 250) R/W 00</td>
</tr>
<tr>
<td></td>
<td>RevId &gt;=3 LinkFreqCap R/O 801F</td>
<td></td>
</tr>
<tr>
<td>4C</td>
<td>Reserved R/O Unpredictable</td>
<td>Reserved R/O Unpredictable</td>
</tr>
<tr>
<td></td>
<td>R/O 0000</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>SriCmd (See Table 149 on page 251) R/W 0000</td>
<td>SriRxDen R/W 10</td>
</tr>
<tr>
<td>54</td>
<td>SriTxDen R/W 0000FFFF</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>SriRxDen R/W 0000FFFF</td>
<td></td>
</tr>
<tr>
<td>5C</td>
<td>Reserved R/O Unpredictable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Revid &gt;=3 IsocIgnMask R/W 0000000</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Reserved R/O Unpredictable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Revid &gt;=3 IsocIgnMask R/W 0000000</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Reserved R/O Unpredictable</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>ErrStatus (Table 153 on page 253) R/C 00</td>
<td>ErrCtrl (See Table 152 on page 252) R/W 000000</td>
</tr>
<tr>
<td>6C</td>
<td>Reserved R/O 00</td>
<td>TxCtrl (See Table 154 on page 253) R/W 04</td>
</tr>
</tbody>
</table>
Table 140: HyperTransport Configuration Header (Type 1) (Cont.)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Register Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>70-C4</td>
<td>Reserved R/O Unpredictable</td>
<td>Reserved</td>
</tr>
<tr>
<td>C8</td>
<td>TxBufCountMax (See Table 157 on page 254) R/W 00FFFFFF</td>
<td>Transmit buffer control register.</td>
</tr>
<tr>
<td>CC-D8</td>
<td>Reserved R/O Unpredictable</td>
<td>Reserved</td>
</tr>
<tr>
<td>DC</td>
<td>DiagRxCrcE (See Table 158 on page 254) R/O xxxxxxxx</td>
<td>Expected CRC (diagnostic register).</td>
</tr>
<tr>
<td>E0-EC</td>
<td>Reserved R/O Unpredictable</td>
<td>Reserved</td>
</tr>
<tr>
<td>F0</td>
<td>DiagRxCrcR (See Table 159 on page 254) R/O xxxxxxxx</td>
<td>Received CRC (diagnostic register).</td>
</tr>
<tr>
<td>F4-FC</td>
<td>Reserved R/O Unpredictable</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 141: HyperTransport Bridge Command Register - Offset 4 Bits [15:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IoSpaceEn</td>
<td>R/O 1'b1</td>
<td>I/O Space Enable. This bit is always set. The HyperTransport bridge always forwards I/O space transactions that originate on the ZBbus and match the bridge I/O range. Peer-to-Peer I/O is not supported between PCI and HyperTransport.</td>
</tr>
<tr>
<td>1</td>
<td>MemSpaceEn</td>
<td>R/O 1'b1</td>
<td>Memory Space Enable. This bit is always set. The HyperTransport bridge always forwards memory space transactions that originate on the ZBbus and match the bridge memory range. Peer-to-Peer transfers from the PCI bus will always be accepted by the HyperTransport bridge, but they can be disabled in the PCI bridge by clearing the ptp_en bit in the PCI Feature Control Register (see Table 133 on page 241).</td>
</tr>
<tr>
<td>2</td>
<td>MasterEn</td>
<td>R/W 1'b0</td>
<td>Primary Bus (i.e. ZBbus) Master Enable. This bit controls acceptance of requests from the HyperTransport fabric. If a request has unitid=0 and contains an address that should be sent back on the HyperTransport fabric (a peer-to-peer operation) it must have come the whole length of the fabric from another host bridge and it will always be responded to with an NXA error. 0: Accept only Configuration Cycles and requests that must be sent back out on the HyperTransport fabric. All other posted requests are dropped and non-posted requests return an NXA error. 1: Accept all requests that match an address range in this part.</td>
</tr>
<tr>
<td>3</td>
<td>SpecCycEn</td>
<td>R/O 1'b0</td>
<td>These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.</td>
</tr>
<tr>
<td>4</td>
<td>MemWrInvEn</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VgaPalSnpEn</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ParErrResp</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>WaitCycCtrl</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>SerrEn</td>
<td>R/W 1'b0</td>
<td>SERR enable. This controls the SERR on the primary interface. Since the ZBbus has no equivalent of SERR the setting of this bit is ignored.</td>
</tr>
<tr>
<td>9</td>
<td>FastB2BEn</td>
<td>R/O 1'b0</td>
<td>These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.</td>
</tr>
<tr>
<td>15:10</td>
<td>reserved</td>
<td>R/O 6'b0</td>
<td></td>
</tr>
</tbody>
</table>
**Table 142: HyperTransport Bridge Primary (ZBus) Status Register - Offset 4 Bits [31:16]**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>reserved</td>
<td>R/O 4'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>CapList</td>
<td>R/O 1'b1</td>
<td>Always set. There is a capabilities list pointed to by the pointer register.</td>
</tr>
<tr>
<td>5</td>
<td>66MHzCap</td>
<td>R/O 1'b0</td>
<td>These bits apply to PCI bridges only. This register is the primary interface status register and contains the details for the ZBus side of the bridge. These bits always return zero.</td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>FastB2BCap</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MstrDParErr</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>10:9</td>
<td>DevSelTiming</td>
<td>R/O 2'b0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SigdTgtAbort</td>
<td>R/C 0'b0</td>
<td>This bit is set when the HyperTransport interface returns an error to the ZBus. It is cleared when written with a 1. Note that an error code sent with the data is the ZBus equivalent of a Target Abort.</td>
</tr>
<tr>
<td>12</td>
<td>RcvdTgtAbort</td>
<td>R/O 1'b0</td>
<td>Always clear. Errors from the ZBus are logged by the Bus Watcher in the SCD and errors from peer-to-peer PCI accesses are logged in the PCI bridge.</td>
</tr>
<tr>
<td>13</td>
<td>RcvdMstrAbort</td>
<td>R/O 1'b0</td>
<td>The interface does not directly signal SERR on the ZBus interface, so this bit always returns zero. If a Sync packet is seen on the HyperTransport interface it will not be forwarded but it will be reported in the DetSerr bit in the Secondary Status register.</td>
</tr>
<tr>
<td>14</td>
<td>DetSerr</td>
<td>R/O 1'b0</td>
<td>Always clear. Errors from the ZBus are logged by the Bus Watcher in the SCD.</td>
</tr>
<tr>
<td>15</td>
<td>DetParErr</td>
<td>R/O 1'b0</td>
<td>Always clear. Errors from the ZBus are logged by the Bus Watcher in the SCD.</td>
</tr>
</tbody>
</table>

**Table 143: HyperTransport Bridge Secondary (HT) Status Register - Offset 1C Bits [31:16]**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:0</td>
<td>reserved</td>
<td>R/O 5'b0</td>
<td>These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.</td>
</tr>
<tr>
<td>5</td>
<td>66MHzCap</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>reserved</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>FastB2BCap</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MstrDParErr</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>10:9</td>
<td>DevSelTiming</td>
<td>R/O 2'b0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>SigdTgtAbort</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge issues a non-NXA error response on the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.</td>
</tr>
<tr>
<td>12</td>
<td>RcvdTgtAbort</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge receives a non-NXA error response from the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.</td>
</tr>
<tr>
<td>13</td>
<td>RcvdMstrAbort</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge receives an NXA error response from the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.</td>
</tr>
<tr>
<td>14</td>
<td>DetSerr</td>
<td>R/C 1'b0</td>
<td>This bit is set when the bridge detects SYNC packet flooding of the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.</td>
</tr>
<tr>
<td>15</td>
<td>DetParErr</td>
<td>R/O 5'b0</td>
<td>This bit applies to PCI bridges only. For a HyperTransport bridge it is read-only and always returns a zero.</td>
</tr>
</tbody>
</table>
### Table 144: HyperTransport Bridge Control Register - Offset 3C Bits [31:16]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ParErrRespEn</td>
<td>R/O 1'b0</td>
<td>This bit applies to PCI bridges only. For a HyperTransport bridge it is read only and always returns a zero.</td>
</tr>
<tr>
<td>1</td>
<td>SerrEn</td>
<td>R/W 1'b0</td>
<td>This bit must be set to enable detection of Sync flooding on the HyperTransport link.</td>
</tr>
<tr>
<td>2</td>
<td>IsaEn</td>
<td>R/O 1'b0</td>
<td>This bit is always zero. The HyperTransport bridge does not support modified ISA address forwarding.</td>
</tr>
<tr>
<td>3</td>
<td>VgaEn</td>
<td>R/W 1'b0</td>
<td>This bit controls the routing of VGA address accesses in the compatibility spaces (0A_0000-0B_FFFF in memory space and X3B0-X3BB and X3C0-X3DF in I/O space). If this bit is clear these addresses are sent to the PCI interface and if set they are sent (without the COMPAT bit set) to the HyperTransport fabric.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>MstrAbortMode</td>
<td>R/W 1'b0</td>
<td>Error response behavior. This bit determines the way errors are forwarded through the HyperTransport bridge.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If this bit is set:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any ZBbus error causes a HyperTransport Error response with the NXA bit clear.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any HyperTransport error causes a ZBbus Bus Error.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If this bit is clear:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any ZBbus Bus Error or Fatal Error causes a valid HyperTransport response.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any other ZBbus Error causes a HyperTransport error response with the NXA bit clear.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any HyperTransport error with the NXA bit set causes a ZBbus response with valid data that is all ones.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Any HyperTransport error with the NXA bit clear causes a ZBbus Bus Error response.</td>
</tr>
<tr>
<td>6</td>
<td>SecBusReset</td>
<td>R/W 1'b0</td>
<td>If this bit is set the LDT_RESET_L signal is asserted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the WarmReset bit in the HyperTransport host interface command register is clear, the LDT_PWROK signal is deasserted to cause a cold reset.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When this bit is set, clearing it will bring the fabric out of reset.</td>
</tr>
<tr>
<td>7</td>
<td>FastB2BEn</td>
<td>R/O 1'b0</td>
<td>These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.</td>
</tr>
<tr>
<td>8</td>
<td>PriDiscard</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SecDiscard</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>DiscardStat</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>DiscardSerrEn</td>
<td>R/O 1'b0</td>
<td></td>
</tr>
<tr>
<td>15:12</td>
<td>reserved</td>
<td>R/O 4'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 145: HyperTransport Command Register - Offset 40 Bits [31:16]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>WarmReset</td>
<td>R/W 1'b1</td>
<td>If this bit is set then a warm reset of the fabric is done when the SecBusReset bit is set in the Bridge Control Register. If this bit is clear a cold reset is done. Changing the state of this bit when the SecBusReset bit is asserted results in UNDEFINED behavior.</td>
</tr>
<tr>
<td>1</td>
<td>DoubleEnded</td>
<td>R/W 1'b0</td>
<td>This bit is set by the master host bridge on other side of the chain to indicate that the slave host bridge is successfully configured on a double ended link. This bit is just used by software, it has no affect on hardware.</td>
</tr>
<tr>
<td>6:2</td>
<td>Reserved</td>
<td>R/O 5'b0</td>
<td>On interface RevID 1 and 2 this field is reserved.</td>
</tr>
<tr>
<td></td>
<td>RevID=0</td>
<td>R/W 5'b0</td>
<td>On interface with RevId 3 and greater this field is used to set the Unit ID the part will use if the ActAsSlave bit is set. The default is restored on a cold reset.</td>
</tr>
</tbody>
</table>
Table 145: HyperTransport Command Register - Offset 40 Bits [31:16] (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>On interface RevID 1 and 2 this field is reserved.</td>
</tr>
<tr>
<td></td>
<td>RevID&gt;=3 ChainSide</td>
<td>R/W 1'bx</td>
<td>On interface with RevId 3 and greater this bit reads as a 0 from the ZBus side and 1 from the chain side.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>On interface RevID 1 and 2 this field is reserved.</td>
</tr>
<tr>
<td></td>
<td>RevID&gt;=3 HostHide</td>
<td>R/O 1'b0</td>
<td>The HostHide feature is not supported.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>RevID&gt;=3 ActAsSlave</td>
<td>R/W 1'b0</td>
<td>On interface with RevId 3 and greater if this bit is set the interface acts as a slave and uses the unit ID set in the DevNum field. The default is forced on a cold reset and any change will only take effect on a link warm reset.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>On interface RevID 1 and 2 this field is reserved.</td>
</tr>
<tr>
<td></td>
<td>RevID&gt;=3 InboundEOC</td>
<td>R/O 1'b0</td>
<td>On interface with RevId 3 and greater this bit is not used. The error reporting is compatible with the older version of the interface and differs from 1.03.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td>R/O 1'b0</td>
<td>On interface RevID 1 and 2 this field is reserved.</td>
</tr>
<tr>
<td></td>
<td>RevID&gt;=3 DropOnUninit</td>
<td>R/O 1'b0</td>
<td>The DropOnUninit feature is not supported.</td>
</tr>
<tr>
<td>15:13</td>
<td>CapType</td>
<td>R/O 3'b001</td>
<td>This field indicates that this is a host/secondary bridge capability type.</td>
</tr>
</tbody>
</table>

Table 146: HyperTransport Link Control Register - Offset 44 Bits [15:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>R/O 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>CrcSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If set CRC errors will cause a sync flood and set the Link Failure bit, and will therefore jam the link. If clear a sync flood is not generated and the bit not set. CRC checking is always enabled and errors are always logged in the CRC error bits.</td>
</tr>
<tr>
<td>2</td>
<td>CrcStartTest</td>
<td>R/S 1'b0</td>
<td>If this bit is set the bridge initiates a CRC test sequence on the link (see the HyperTransport specification for details). The bit is cleared by the hardware when the test is complete, software can then check the CRC Error bit to check the status of the test.</td>
</tr>
<tr>
<td>3</td>
<td>CrcForceErr</td>
<td>R/W 1'b0</td>
<td>If this bit is set the bridge generates bad CRCs on the outgoing link.</td>
</tr>
<tr>
<td>4</td>
<td>LinkFail</td>
<td>R/W 1'b0</td>
<td>This bit is set if a link failure is detected. This bit is persistent through warm reset. If this bit is set link synchronization will not be done and the Initialization Complete bit will not get set on either end of the link. If this bit is changed by software it will only have effect after the next warm reset.</td>
</tr>
<tr>
<td>5</td>
<td>InitDone</td>
<td>R/O 1'b0</td>
<td>This bit is set by the bridge to indicate that the low level link initialization has completed successfully.</td>
</tr>
<tr>
<td>6</td>
<td>EOC</td>
<td>R/S 1'b0</td>
<td>This bit may be set by software to indicate this is the last device in a logical chain. If set all received packets are ignored. If the XmitOff bit is clear the bridge will generate NOP packets and good CRCs on its outgoing link. Once set this bit can only be cleared by a fabric reset.</td>
</tr>
</tbody>
</table>
### Table 146: HyperTransport Link Control Register - Offset 44 Bits [15:0] (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>XmitOff</td>
<td>R/S 1'b0</td>
<td>This bit can be set by the software to shut off a transmitter. When set none of the output pins will toggle. If the EOC bit is set on an active link XmitOff should not be set until the transmitter has driven enough NOP packets to fill the receiver's receive FIFOs. Once set this bit can only be cleared by a fabric reset.</td>
</tr>
<tr>
<td>11:8</td>
<td>CrcErr</td>
<td>R/C 4'b0</td>
<td>When a CRC error is received on the incoming link the bridge will set the bit in this field corresponding to the byte lane that had the error. The BCM1250 only supports an 8-bit link, so bit [8] is the only one that will ever be set. Software may clear these bits by writing a 1 to them.</td>
</tr>
<tr>
<td>15:12</td>
<td>reserved</td>
<td>R/O 4'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 147: HyperTransport Link Configuration Register - Offset 44 Bits [31:16]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>MaxIn</td>
<td>R/O 3'b000</td>
<td>This field indicates the BCM1250 only supports an 8 bit input link.</td>
</tr>
<tr>
<td>3</td>
<td>DwFcIn</td>
<td>R/O 1'b0</td>
<td>The bridge is not capable of Doubleword Flow Control.</td>
</tr>
<tr>
<td>6:4</td>
<td>MaxOut</td>
<td>R/O 3'b000</td>
<td>This field indicates the BCM1250 only supports an 8 bit output link.</td>
</tr>
<tr>
<td>7</td>
<td>DwFcOut</td>
<td>R/O 1'b0</td>
<td>The bridge is not capable of Doubleword Flow Control.</td>
</tr>
<tr>
<td>10:8</td>
<td>WidthIn</td>
<td>R/O 3'b000</td>
<td>This field is fixed for an 8 bit link.</td>
</tr>
<tr>
<td>11</td>
<td>DwFcInEn</td>
<td>R/O 1'b0</td>
<td>The bridge is not capable of Doubleword Flow Control.</td>
</tr>
<tr>
<td>14:12</td>
<td>WidthOut</td>
<td>R/O 3'b000</td>
<td>This field is fixed for an 8 bit link.</td>
</tr>
<tr>
<td>15</td>
<td>DwFcOutEn</td>
<td>R/O 1'b0</td>
<td>The bridge is not capable of Doubleword Flow Control.</td>
</tr>
</tbody>
</table>

### Table 148: HyperTransport Link Frequency Register - Offset 48 Bits [15:8]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>LinkFreq</td>
<td>R/W 4'b0000</td>
<td>This register sets the maximum transmitter clock frequency for the link. The BCM1250 uses this value to control the PLL that provides both transmit and receive internal clocks using the CLK100 reference. If the sriLinkFreqDirect bit is clear in the sriCmd register, these bits have the encoding given in the HyperTransport 1.0 standard: 0000: 200 MHz (CLK100 x2) 0001: 300 MHz (CLK100 x3) 0010: 400 MHz (CLK100 x4) 0011: 500 MHz (CLK100 x5) 0100: 600 MHz (CLK100 x6) 0101: Not supported (code for 800 MHz) 0110: Not supported (code for 1000 MHz) Others: UNDEFINED</td>
</tr>
<tr>
<td>4</td>
<td>LinkFreq</td>
<td>R/W 1'b0</td>
<td>If the sriLinkFreqDirect bit is set in the sriCmd register then the [4:0] field is used to set the PLL frequency directly, using the values shown in Table 9 on page 31. The link frequency will be set to CLK100 * LinkFreq[4:0]/2. The HyperTransport link clock frequency must not be set faster than four times the frequency of the I/O bridge 0 clock. If the bridge is set (using the configuration resistor on IO_AD[5]) to CPU clock/4 then the HyperTransport clock frequency must be less than or equal to the CPU clock frequency. If the bridge is set to CPU clock/3 then the HyperTransport link clock must be run at less than or equal to 4/3 of the CPU clock frequency. Note that the sriLinkFreqDirect bit must be set before the frequency code is written to this register for the alternate frequencies to be used.</td>
</tr>
<tr>
<td>7:5</td>
<td>reserved</td>
<td>R/O 3'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 149: HyperTransport SRI Command Register - Offset 50 Bits [31:16]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SipReady</td>
<td>R/W 1'b0</td>
<td>System Initialization Process Ready. This bit should be set by software when the SRI initialization process is complete. The LDT bridge will be held in reset until this bit is set.</td>
</tr>
<tr>
<td>1</td>
<td>SyncPtrCtl</td>
<td>R/W 1'b0</td>
<td>This bit should be set to enable synchronous pointer control. If the interface is synchronous the pointers in the receive FIFO are moved according to a fixed pattern based on the ratio of the load and unload clocks. If the interface is asynchronous the pointers are moved by sampling the incoming receive clock and counting the number of edges.</td>
</tr>
<tr>
<td>2</td>
<td>ReduceSyncZero</td>
<td>R/W 1'b0</td>
<td>If this bit is set only 128 bit times of zeros will be sent during link initialization rather than the standard 512 bit times. Only set during testing.</td>
</tr>
<tr>
<td>3</td>
<td>DisMultTxVld</td>
<td>R/W 1'b0</td>
<td>If clear (default), all 3 virtual channels can send requests simultaneously. If set, only one channel can make a request at a time (as in RevId). This bit is for debug use only.</td>
</tr>
<tr>
<td>8:4</td>
<td>RxMargin</td>
<td>R/W 5'h0</td>
<td>This sets the offset between the receive FIFO load and unload pointers.</td>
</tr>
<tr>
<td>9</td>
<td>sriLdtPLLCompat</td>
<td>R/W 1'b0</td>
<td>This bit should be set for compatibility with 0.17 HyperTransport devices.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>It modifies the behavior for updating the LdtLinkFreq register (the one visible to programmer) and moving it into LdtPLLFreq (the shadow copy used to control the PLL). Older fixed frequency devices need this bit set, so a link cold reset does not alter the link frequency. HyperTransport Rev 1.0 devices reset their link frequency to 200 MHz on a link cold reset. sriLdtPLLCompat set to 1 =&gt; compatible with API AP1011 / SiPackets SP1011 sriLdtPLLCompat set to 0 =&gt; comply with 1.0 spec.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This table summarizes the behavior on reset:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>System reset</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>LdtLinkFreq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200MHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Link cold reset</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Compat == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Comp == 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Link warm reset</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Compat == 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Compat == 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This sets the initial offset between the transmit FIFO unload and load pointers.</td>
</tr>
<tr>
<td>10</td>
<td>exp_endian</td>
<td>R/W 1'b0</td>
<td>Selects endian policy for expansion space. If this bit is clear then all of the expansion space will use the match byte lane policy. If the bit is set then address bit [38] is used to select the endian policy and the address bit is cleared as the request passes through the bridge.</td>
</tr>
<tr>
<td>11</td>
<td>reserved</td>
<td>R/O 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>14:12</td>
<td>TxInitialOffset</td>
<td>R/W 3'b000</td>
<td>This sets the initial offset between the transmit FIFO unload and load pointers.</td>
</tr>
<tr>
<td>15</td>
<td>sriLinkFreqDirect</td>
<td>R/W 1'b0</td>
<td>If this bit is clear the HyperTransport link frequency is set according to the HyperTransport specification revision 1.0. If it is clear a wider range of frequencies can be set. See Table 148 on page 250.</td>
</tr>
</tbody>
</table>

### Table 150: HyperTransport Isochronous BAR - Offset 5C Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>isocEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set then the IsocBAR and IsocIgnMask are used to force inbound transactions within the specified range to behave as if their Isochronous bit is set.</td>
</tr>
<tr>
<td>31:1</td>
<td>isocBase</td>
<td>R/W 31'b0</td>
<td>This register sets the address bits 35:5 that are compared to force the isochoonous bit. See Section: “Force Isochronous Mode Address Range” on page 212.</td>
</tr>
</tbody>
</table>
### Table 151: HyperTransport Isochronous Ignore Mask - Offset 60Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>R/W 1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>31:1</td>
<td>isocMask</td>
<td>R/W 31'b0</td>
<td>This specifies the mask of bits that are ignored when the comparison is done to force the isochronous bit. See Section: “Force Isochronous Mode Address Range” on page 212.</td>
</tr>
</tbody>
</table>

### Table 152: HyperTransport Error Control Register - Offset 68 Bits [23:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ProtFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when a protocol error is detected.</td>
</tr>
<tr>
<td>1</td>
<td>ProtNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when a protocol error is detected.</td>
</tr>
<tr>
<td>2</td>
<td>ProtSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a protocol error will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.</td>
</tr>
<tr>
<td>3</td>
<td>OvfFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when a receive buffer overflow is detected.</td>
</tr>
<tr>
<td>4</td>
<td>OvfNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when a receive buffer overflow is detected.</td>
</tr>
<tr>
<td>5</td>
<td>OvfSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a receive buffer overflow will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.</td>
</tr>
<tr>
<td>6</td>
<td>EocNxaFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when a posted request to a nonexistent address or response that does not match a request is detected. (Non-posted requests will receive an NXA error in this case).</td>
</tr>
<tr>
<td>7</td>
<td>EocNxaNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when a posted request to a nonexistent address or response that does not match a request is detected. (Non-posted requests will receive an NXA error in this case).</td>
</tr>
<tr>
<td>8</td>
<td>EocNxaSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a posted request needing an NXA or a response that does not match a request will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.</td>
</tr>
<tr>
<td>9</td>
<td>CrcFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when a receive CRC error is detected.</td>
</tr>
<tr>
<td>10</td>
<td>CrcNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when a receive CRC error is detected.</td>
</tr>
<tr>
<td>11</td>
<td>SerrFatalEn</td>
<td>R/W 1'b0</td>
<td>The SerrEn bit in the Bridge Control register determines if an SERR will generate an interrupt. If this bit is set a fatal interrupt will be raised, if clear a nonfatal interrupt is used.</td>
</tr>
<tr>
<td>12</td>
<td>SrcTagFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when a source tag error is detected.</td>
</tr>
<tr>
<td>13</td>
<td>SrcTagNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when a source tag error is detected.</td>
</tr>
<tr>
<td>14</td>
<td>SrcTagSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a source tag error will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.</td>
</tr>
<tr>
<td>15</td>
<td>MapNxaFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a fatal interrupt will be raised when an NXA error response is sent to a request with zero source id.</td>
</tr>
<tr>
<td>16</td>
<td>MapNxaNonFatalEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set a nonfatal interrupt will be raised when an NXA error response is sent to a request with zero source id.</td>
</tr>
<tr>
<td>17</td>
<td>MapNxaSyncFloodEn</td>
<td>R/W 1'b0</td>
<td>If this bit is set an NXA error response to a request with zero source id will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.</td>
</tr>
<tr>
<td>23:18</td>
<td>Reserved</td>
<td>R/O 6'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 153: HyperTransport Error Status Register - Offset 68 Bits [31:24]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ProtoErr</td>
<td>R/C 1'b0</td>
<td>This bit will be set if a protocol error is detected on the incoming link. Software may clear this bit by writing a 1 to it. In some circumstances this bit will be set by a HyperTransport link reset.</td>
</tr>
<tr>
<td>1</td>
<td>OvfErr</td>
<td>R/C 1'b0</td>
<td>This bit will be set if the receive FIFO overflows. Software may clear this bit by writing a 1 to it.</td>
</tr>
<tr>
<td>2</td>
<td>EocNxaErr</td>
<td>R/C 1'b0</td>
<td>This bit will be set if a Non Existent Address (NXA) error is returned to a request from the incoming link because the End Of Chain bit (bit 6 in the link control register) is set. Software may clear this bit by writing a 1 to it. This bit will also be set if a request is rejected because the link has not been established (the InitDone bit is clear).</td>
</tr>
<tr>
<td>3</td>
<td>SrcTagErr</td>
<td>R/C 1'b0</td>
<td>This bit will be set if a response packet is received with a source id that does not match any outstanding request. Software may clear this bit by writing a 1 to it.</td>
</tr>
</tbody>
</table>
| 4    | MapNxaError| R/C 1'b0| This bit will be set if a Non Existent Address (NXA) error is returned due to a mapping problem with the request. There are two causes:  
1) A packet is received that does not match one of the valid address ranges in Figure 43 on page 210.  
2) A packet is received that is decoded as having an address on the HyperTransport link and with a zero source id. This indicates that the packet is a peer-to-peer HyperTransport transaction, but it originated in the host bridge at the far end of the link and has therefore not been accepted by any device on the link. This bit is also set (but no error packet sent) if a response is received that has source id of zero and did not come from this host. Software may clear this bit by writing a 1 to it. |

7:5  Reserved        R/O 3'b0  Reserved

### Table 154: HyperTransport SRI Transmit Control Register - Offset 6C Bits [23:16]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>BufRelSpace</td>
<td>R/W 4'b0100</td>
<td>This field sets the minimum number of packets that will be sent between buffer release NOP packets. If the link is idle buffer release packets are sent immediately. If the link is busy buffer release messages are inserted into the packet stream with this minimum spacing so that the bandwidth is not consumed with flow control traffic.</td>
</tr>
<tr>
<td>7:4</td>
<td>Reserved</td>
<td>R/O 4'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 155: HyperTransport SRI Data Buffer Allocation Register - Offset 6C Bits [15:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>NeedResp</td>
<td>R/W 2'b01</td>
<td>This register controls the allocation of the 8 HyperTransport receive data buffers among the 3 virtual channels, to allow performance tuning.</td>
</tr>
<tr>
<td>3:2</td>
<td>NeedNpReq</td>
<td>R/W 2'b01</td>
<td>The &quot;Need&quot; fields indicate the minimum allocation at all times to each channel, minus one. That is, a value of 0 indicates a permanent minimum allocation of 1. The total number of buffers &quot;needed&quot; must be less than or equal to 8 or the behavior of the bridge is UNDEFINED. If the number of &quot;needed&quot; buffers is less than 8 the allocator will dynamically allocate them to the three channels, subject to the limits set in the &quot;Want&quot; fields.</td>
</tr>
<tr>
<td>5:4</td>
<td>NeedPReq</td>
<td>R/W 2'b01</td>
<td>The &quot;Want&quot; fields indicate how many buffers the allocator should try to have released and outstanding to each channel at all times, minus 1.</td>
</tr>
<tr>
<td>7:6</td>
<td>Reserved</td>
<td>R/O 2'b0</td>
<td>Note that these counts must not be set to 2'b11 (i.e. 4 buffers) or the system may behave in UNDEFINED ways.</td>
</tr>
<tr>
<td>9:8</td>
<td>WantResp</td>
<td>R/W 2'b00</td>
<td>In the default (reset) case, there are 2 buffers in each category. Note that these counts must not be set to 2'b11 (i.e. 4 buffers) or the system may behave in UNDEFINED ways.</td>
</tr>
<tr>
<td>11:10</td>
<td>WantNpReq</td>
<td>R/W 2'b01</td>
<td></td>
</tr>
<tr>
<td>13:12</td>
<td>WantPReq</td>
<td>R/W 2'b01</td>
<td></td>
</tr>
<tr>
<td>15:14</td>
<td>Reserved</td>
<td>R/O 2'b00</td>
<td></td>
</tr>
</tbody>
</table>
Table 156: HyperTransport Additional Status Register - Offset 70

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>tgt_done</td>
<td>R/W 8'b0</td>
<td>Target Done Counter. This value is incremented every time a Target Done acknowledgement is received from the fabric. These will happen as a result of the NonPosted writes done in configuration space. Software may write this field, incrementing will continue from the value written.</td>
</tr>
<tr>
<td>31:8</td>
<td>notimp</td>
<td>26'b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

Table 157: HyperTransport SRI Transmit Buffer Count Max Register - Offset C8 Bits [31:0]

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>PCmd</td>
<td>R/W 4'b1111</td>
<td>These fields set the maximum number of each class of buffer the transmitter will use. The classes (Posted Command, Posted Data, NonPosted Command, NonPosted Data, Response and Response Data) match the buffers in the receiver and fields in the NOP flow control packet.</td>
</tr>
<tr>
<td>7:4</td>
<td>Pdata</td>
<td>R/W 4'b1111</td>
<td>_NOP packets received from the other end of the link will increment the number of buffers that the transmitter believes the receiver has available. If the count reaches the limit set in this register the extra credits will be discarded. This allows a general way to throttle traffic in a particular virtual channel on a link.</td>
</tr>
<tr>
<td>11:8</td>
<td>NpCmd</td>
<td>R/W 4'b1111</td>
<td>The maximum counter values should be setup before the sipReady bit is set. The values may be lowered in a running system, but the link must be reset before any increase takes effect.</td>
</tr>
<tr>
<td>15:12</td>
<td>NpData</td>
<td>R/W 4'b1111</td>
<td></td>
</tr>
<tr>
<td>19:16</td>
<td>RCmd</td>
<td>R/W 4'b1111</td>
<td></td>
</tr>
<tr>
<td>23:20</td>
<td>RData</td>
<td>R/W 4'b1111</td>
<td></td>
</tr>
<tr>
<td>31:24</td>
<td>reserved</td>
<td>R/O 8'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 158: HyperTransport Diagnostic Receive CRC Expected - Offset DC

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>RxCrcE</td>
<td>R/O 32'b'x</td>
<td>This contains the expected value of the receive CRC.</td>
</tr>
</tbody>
</table>

Table 159: HyperTransport Diagnostic Receive CRC Received - Offset F0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>RxCrcR</td>
<td>R/O 32'b'x</td>
<td>This register contains the actual value received for the CRC.</td>
</tr>
</tbody>
</table>
SYSTEM RESET INITIALIZATION OF THE HYPERTRANSPORT INTERFACE

There are a number of parameters that must be configured in the HyperTransport interface before the HyperTransport link can come out of reset and the link initialization described in the HyperTransport Specification can start. These registers should be programmed and then the SipReady bit in the SRI Command Register set. Once this bit is set the HyperTransport interface will come out of reset, remove the link reset, and start the link initialization.

Most of the configuration is setting up the HyperTransport receive and transmit FIFO clocks. A high level view of this is shown in Figure 50. Data from the fabric is loaded into the receive FIFO using the clock that was received with the data. It is unloaded using the internal HyperTransport interface clock and passed to the receive logic. Data from the transmit logic is loaded into the transmit FIFO using the internal HyperTransport interface clock and extracted using the link transmit clock. The FIFOs both buffer clock speed differences and provide time for the data to stabilize as it crosses the clock boundary.

![Figure 50: HyperTransport Interface Clocks and FIFOs](image-url)
Configuration Flags in the SriCmd Register

There are three flags in the SriCmd register. Two of these should be set at the start of initialization.

The ReduceSyncZero flag should only be set during debugging; it reduces the number of bit times of zeros during link initialization from the standard 512 to 128.

The SyncPtrCtl should be set if the interface is running in Synchronous Clock Mode (using the same reference clock source as the other end of the link) and clear for Asynchronous Mode. Selection of the correct mode depends on the system design, it will normally be coded in to the boot ROM but could also be set on the software configuration bits in the system control register.

The third flag is the SipReady flag. This should be left clear while the initialization is done. Once all the registers (including the SriCmd register) have been setup this bit should be set.

The SipReady bit is sticky. Once it has been set the only way to clear it is a cold reset. Configuration code should check the state of SipReady to determine if the full initialization sequence or an abbreviated version is needed.

Timing Registers: SriRxDen, SriTxDen, SriRxNum and SriTxNum

The receive timing registers are used when the interface is running in synchronous clocking mode, the transmit registers must always be configured. The registers describe the relationship between the receive clock from the HyperTransport link (fRX) and the internal HyperTransport interface clock (fLDTINT), and the relationship between the internal HyperTransport interface clock and the HyperTransport transmit clock (fTX). Note that the receive and transmit clocks used in this discussion are the actual clocks on the link and therefore are half of the data rate.

The transmit clock frequency is set in the LinkFreq register in the HyperTransport Capability block. The 100 MHz reference clock is multiplied up by the HyperTransport PLL to give the requested frequency. The internal HyperTransport interface clock is always one quarter of the transmit clock frequency: fLDTINT = fTX/4.

The receive clock frequency is set by the device the other end of the link. Data is inserted into the receive FIFO byte wide sent on both edges of the receive clock, it is removed from the FIFO 8 bytes wide clocked on the rising edge of the internal HyperTransport interface clock. The data rate ratio is therefore: fRX/4 * fLDTINT = fRX/fTX.

The SriRxDen and SriRxNum registers encode this receive ratio. The Numerator is a bit pattern in which a 1 indicates that data should be extracted from the receive FIFO. The Denominator sets the number of bits in the Numerator that are used. Two bits from the Numerator pattern are examined each internal cycle, starting from bit zero and ending at bit (Denominator-1), to determine how many 32 bit words should be read out of the FIFO in that cycle.

For example if the receive clock from the link (i.e. the transmit clock set in the LinkFreq register of the device the other end of the link) fRX=300 MHz and the transmit clock (set in the LinkFreq register) fTX=400 MHz then the receive data rate ratio is 3/4. The SriRxNum should therefore be set to 32'b1101 (or some similar pattern with three ones and a zero) and the SriRxDen should be set to 4 to indicate only the bottom four bits of the Numerator pattern are used.

In the common case where the clock frequency is the same at both ends of the link then the default values of SriRxNum = 32'hFFFF and SriRxDen = 16 will work.
The transmitter registers work similarly. Bits in the Numerator indicate that data should be inserted into the transmit FIFO from the internal clock. In the BCM1250 or BCM1125H, the clock ratio is fixed to 1/4. Data is inserted into the FIFO 8 bytes wide on the internal clock and removed byte wide on each edge of the HyperTransport transmit clock. These cancel, so the timing registers must be configured to allow data to be inserted every internal cycle. The defaults of \texttt{SriTxNum} = \texttt{32'hFFFF} and \texttt{SriTxDen} = 16 are suitable values.

**Receive Pointer Margin Control in SriCmd Register**

The receive FIFO is used to separate the link receive clock domain from the internal HyperTransport interface clock domain. To allow for the data to become stable and valid some time must be provided between writing an entry in the FIFO and reading it. This is done by having an offset (or margin) between the load and unload pointers. The initial separation is configured in the \texttt{SriCmd} register.

The value set for the initial margin is computed from the minimum margin that is needed once the interface is running, adjusted for the movement of the load and unload pointers during the initialization sequence. (For implementation reasons the value programmed in the configuration register is twice the basic computation, hence the extra doubling in the discussion below.)

The configuration register must be set to twice the offset that the unload pointer should have from the load pointer when the link is reset. If the pointers did not move during the initialization process this would simply be:

\[
2^{(\text{margin for data to be stable})}
\]

This is negative because it is measured from the load pointer to the unload pointer, which must always be behind. (Since the pointers cycle around the FIFO the offset must be taken mod 8 to get the actual value programmed.) In asynchronous mode the load and unload pointers are stationary during initialization so the correct value can be obtained directly from this computation. In synchronous mode the load and unload pointers are always being moved according to the ratio set in the \texttt{SriRxNum} and \texttt{SriRxDen} registers, so a correction must be applied.

In synchronous mode, the received SYNC pattern indication from the link passes through four synchronization stages clocked by the internal HyperTransport clock on its way in to the link initialization logic. The load pointer for the receive FIFO will advance every two receive clocks (the incoming data is on every edge of the clock, so the four byte FIFO width builds up in two clocks). The minimum advance of the FIFO load pointer during the SYNC synchronization is therefore:

\[
\text{(time for SYNC propagation to initialization logic)/(time per load pointer increment)}
= (4 * \text{internal clock period})/(2 * \text{receive clock period})
= 2^{\text{fRX/fLDTINT}}
\]

The FIFO wraps every eight entries, so this value is taken modulo 8.

The pointer offset must also be adjusted to take into account the possible movement of the unload pointer. The worst case movement of the unload pointer must be used, since that brings it closest to the load pointer. The maximum unload pointer movement can be computed from the \texttt{SriRxNum} value. If there are two adjacent 1s anywhere (including over the wrap from bit \texttt{SriRxDen-1} to 0) in the Numerator then it is possible for two (32 bit) words to be extracted from the FIFO during SYNC. If there are only 10 and 01 patterns for adjacent bits then only one word can be extracted during SYNC.

On the BCM1250 or the BCM1125H, the recommended time allowed for the data to become stable is two cycles.
Putting the three terms together, the value to be programmed for the RxMargin field in synchronous mode is:

\[
RXMargin = 2\times (load\ pointer\ updates - unload\ pointer\ updates - margin\ for\ data\ to\ be\ stable)
\]

\[
= 2\times (2\times \frac{f_{RX}}{f_{LDTINT}} - (1\ or\ 2) - 2)
\]

In asynchronous mode the RxMargin field should be programmed to twice the number of receive clock cycles for the data to settle. The settle time recommended for the BCM1250 or BCM1125H is 7ns, so in asynchronous mode:

\[
RXMargin = 2\times 7\text{ns} \times f_{RX}
\]

**Transmit Pointer Initial Offset in the SriCmd Register**

The transmit FIFO pointers must be offset to take account of the seven stages of synchronization using the HyperTransport transmit clock that are applied to the reset pulse, and to allow time for data settle margin. The minimum number of internal HyperTransport clocks taken for the synchronization time is given by:

\[
\text{minInternalSyncClocks} = \frac{7\times \text{transmitClkPeriod}}{\text{internalClkPeriod}} = \frac{7\times f_{LDTINT}}{f_{TX}} = 7/4 = 1 \text{ (rounding down)}
\]

The minimum advance of the load pointer can be found from the \textit{SriTxNum} value. In this case the Numerator is all 1s so the minimum advance is 2, if the Numerator had 10 or 01 pairs of bits then the minimum advance would be 1.

The recommended data settle margin is 2. This gives:

\[
\text{TxInitialOffset} = \text{minInternalSyncClocks} - \text{minPointerAdvance} - \text{Margin}
\]

\[
= 1 - 2 - 2 = -3 \mod 8 = 5.
\]

**Error Control Register**

The Error Control Register sets the error handling behavior of the HyperTransport interface for both the standard HyperTransport errors and the additional ones reported on the BCM1250 or BCM1125H. The possible responses to errors are: to raise the ldt_fatal_int interrupt, to raise the ldt_nonfatal_int interrupt and to flood the link with SYNC packets. Any of these may be selected as the response to any of the errors. (The CRC error causing SYNC flooding is set in the standard Link Configuration Register).

**Transmit Control Register**

The transmit control register must be configured to set the minimum number of packets that will be sent between NOP packets that are reporting buffer releases to the device on the other end of the link. If the transmitter is idle, the NOP packet will be sent as soon as the buffer is free. If the transmitter is busy and becomes idle the NOP packet will be sent when the link idles. If the transmitter is continually busy the NOP packets will be spaced by the number of packets set in this register.

If the value is set too low then the NOP packets will frequently interrupt transmission of data, so the link utilization will fall. If the value is set too high then buffer releases will not be reported in a timely manner, so the device the other end of the link may throttle its transmissions unnecessarily.
Buffer Control: TxBufCountMax and DataBufAlloc

The Transmit Buffer Count Max register sets the maximum number of buffers the transmitter will use in each of the six buffer classes described in the HyperTransport Specification. The classes (Posted Command, Posted Data, NonPosted Command, NonPosted Data, Response and Response Data) match the buffers in the receiver and fields in the NOP flow control packet.

NOP packets received from the other end of the link will increment the number of buffers that the transmitter believes the receiver has available. If the count reaches the limit set in this register the extra credits will be discarded. This allows a general way to throttle traffic in a particular virtual channel on a link.

The DataBufAlloc register allows allocation of the HyperTransport receive data buffers between the three virtual channels. The minimum number of buffers that must remain allocated to each channel is specified, along with the number of additional buffers that the dynamic buffer allocator should try to make available for that channel.

HyperTransport Resets

There are four reset conditions associated with the HyperTransport. This section describes them and their actions on the HyperTransport link and interface. The four resets are:

1. System cold reset. A system cold reset is caused by the assertion of COLDRES_L or the setting of the system configuration register system_reset bit. This reset causes the subsequent assertion of all other resets detailed here.

2. System warm reset. A system warm reset is caused by the assertion of RESET_L or the setting the system configuration register sb_softres bit and results in the assertion of link reset.

3. HyperTransport link power ok (LDT_PWROK). This open drain output is driven and received by the interface. The HyperTransport link is inactive if this signal is not asserted.

4. HyperTransport link reset (LDT_RESET_L). This open drain output is driven and received by the interface. The HyperTransport link is reset when this signal is asserted, a cold link reset will be performed if LDT_PWROK is deasserted and a warm link reset if LDT_PWROK is asserted.

Following a system cold reset the interface will drive the HyperTransport link controls to ensure LDT_PWROK will remain deasserted and LDT_RESET_L will remain asserted until the SipReady bit has been set in the HyperTransport SRI Command register. Once the bit is set LDT_PWROK will be asserted and 1ms later LDT_RESET_L will be deasserted to start the cold link initialization. These are inputs as well as open drain outputs, so other devices in the chain can hold off starting the link (for example in a double hosted link the parts at each end of the link may take different lengths of time to complete their initialization and set SipReady). Note that writing the SRI configuration registers and setting of the SipReady bit should only be done after a system cold reset, the SipReady bit must not be touched during link resets.

As required by the HyperTransport specification, a deassertion of LDT_PWROK is caused by clearing the warm reset bit in the HyperTransport Command Register and setting the SecBusReset bit in the HyperTransport Bridge Control Register. This is effectively a cold reset on the HyperTransport link and also causes the assertion of the link reset.

As required by the HyperTransport specification, an assertion of link reset is caused by setting the warm reset bit in the HyperTransport Command Register and setting the SecBusReset bit in the HyperTransport Bridge Control Register. This is effectively a HyperTransport link warm reset.
The registers that make up the HyperTransport Bridge header are reset by both system resets and are unaffected by link reset with the following exceptions:

1. The HyperTransport SRI Command Register is reset on system cold reset and persistent across system warm reset.
2. The following bits in the HyperTransport Bridge Secondary Status Register are set to zero on a system cold reset and are persistent through a system warm reset: SigdTgtAbort, RcvdTgtAbort, RcvdMstrAbort, DetSerr.
3. The following bits in the HyperTransport Link Control Register are reset by link reset: InitDone, EOC, XmitOff.
4. The LinkFail bit and CrcErr bits in the HyperTransport Link Control Register are reset to zero on a system cold reset and are persistent through a system warm reset.
5. The HyperTransport Link Frequency Register is cleared by system cold reset and is persistent across system warm reset. Beyond that, its behavior is determined by the sriLdtPLLCompat bit in the HyperTransport SRI Command Register. If the sriLdtPLLCompat bit is set, the Link Frequency implementation becomes backwards compatible with HyperTransport Specification revision 0.17, which does not allow for dynamic frequency negotiation. In this mode, the HyperTransport link must come up at a static frequency and therefore writes to the LinkFrequency Register take effect immediately. It is strongly suggested that this be done prior to setting the SipReady bit in the HyperTransport SRI Command Register and at no other time. If the sriLdtPLLCompat bit is clear, a write to the Link Frequency Register will take effect only on the next link warm reset.
This Page is left blank for notes
Section 9: Ethernet MACs

INTRODUCTION

The BCM1250 includes three Ethernet MACs. The BCM1125 and BCM1125H include two Ethernet MACs. The MACs support the 10 Mbps, 100 Mbps and 1 Gbps Ethernet standards. They can operate in full or half-duplex mode at all speeds. The MII (10/100 Mbps) or GMII (1 Gbps) standard interfaces are used to connect to external PHY (physical interface) chips. The minimum and maximum size of a packet is programmable, giving support for jumbo packets up to 16K-1 Bytes. For half-duplex operation the MAC supports programmable backoff and retransmission following a collision. A set of RMON counters are updated automatically at the end of each packet transmission or reception, these may be read by the CPU at any time to gather interface statistics.

In addition to the standard Ethernet mode, each interface can be put into a bypass mode where the Ethernet protocol processing is disabled and the interface acts as a Packet FIFO. In this the GMII pins are used to provide an 8-bit data path in each direction that can carry packet or unframed data. Selection of Ethernet or 8-bit Packet FIFO is done by software.

The interfaces are identical and independent. The registers and interrupts associated with each are differentiated by appending the interface number to their names. On the BCM1250 the interfaces are _0, _1 and _2. On the BCM1125/H the interfaces are _0 and _1. Accesses made to the address range allocated to a nonexistant MAC may cause all MACs to exhibit UNPREDICTABLE behaviour. The following sections describe one of the MACs.

The MAC interfaces can be replaced by Packet FIFO interfaces that are 16-bits wide in each direction. Pins from MAC E0 and E1 are reassigned to Packet FIFO interface F0 on the BCM1250 or F on the BCM1125/H, and on the BCM1250 pins from all three MACs are reassigned to Packet FIFO interface F1. Selection between Ethernet and Packet FIFO operation is done by software.
**INTERFACE OVERVIEW**

Figure 51 shows the block diagram of a single Ethernet interface. The interface to the system bus is provided through I/O Bridge 1. This allows the CPU to access the interface control registers, the RMON statistical counters and the DMA control interface. It also provides the connection from the DMA engines into the part.

![Ethernet Interface Block Diagram](image)
The DMA controllers are described in Section 7: “DMA” on page 147. On the transmit side there are two DMA channels which are serviced using a weighted round robin algorithm. In the receive direction there are also two channels, the packet header is used to select which will be used for delivery.

The transmit packet flow starts with the packet being DMAed from memory into the transmit FIFO. The FIFO is not intended to store the complete packet, it is there to buffer the line rate from the system bus. The DMA controller will fetch 32 byte cache blocks as required to keep the FIFO filled. The controller can be configured to either fetch one or two blocks at a time. The memory performance will be improved by fetching two blocks at a time since the second is likely to hit the memory page opened by the first, however it lowers the granularity of control over the FIFO. The data unpacker extracts 64 bit double-words from the FIFO and forwards them as a byte stream to the protocol engine. Either the Ethernet MAC engine or the Packet FIFO protocol bypass path is selected. For successful transmission one byte must be sent to the protocol engine every cycle, this is enabled by only starting to send a new packet when enough data has been written in to the FIFO to buffer the DMA latency.

The Ethernet MAC protocol engine formats the packet according to the Ethernet specification, executes the MAC protocol to gain access to the transmission medium and sends the packet to the (external) physical layer interface over the GMII pins. It will also respond to flow control requests and block packet transmission as required. The Packet FIFO protocol engine generates simple framing signals for the external interface, but otherwise acts as a bypass path for data to the GMII pins.

The receive packet flow starts with a byte stream being delivered from the physical layer device over the GMII pins. Bytes are sent either to the Ethernet or Packet FIFO protocol blocks. The protocol engine will validate the packet, strip the MAC layer overhead and forward the data to the packing unit. The received bytes are packed into 64 bit double-words and inserted into the receive FIFO. Again, the FIFO just provides a buffering function to cover the DMA latency. The receive DMA engine will extract 32 byte cache blocks from the FIFO and transfer them into memory. The receive protocol engine can also request flow control, it will do this either under software control or if the DMA engine is running out of buffers for incoming packets.

A set of registers in the control section allows the interface to be configured and status to be read. The MAC must be configured before it is used. After the system has been reset the transmit section, receive section and protocol engine are all held in reset until enabled by software.

There is one main interrupt associated with each interface. It combines the interrupts from the transmit and receive sides of each of the two DMA channels and the interrupt from detection of errors. There is a second interrupt that can be enabled to split the DMA channels. When the second interrupt is enabled it signals events from the channel 1 transmit and receive DMA engines, and the main interrupt only signals events from the channel 0 DMA engines and error conditions.
PROTOCOL ENGINE AND GMII/MII

The Ethernet MAC engine performs all the MAC layer processing needed to comply with the IEEE 802.3 standard at 10 Mbps, 100 Mbps and 1000 Mbps. It interfaces to the receive and transmit FIFOs to move data into and out of the system, and connects to an external physical layer interface (PHY) using the standard MII (media independent interface) connection for 10/100 Mbps operation and the GMII for gigabit operation.

Many of the protocol parameters are programmable. They must be set to the correct values for IEEE 802.3 operation, but may be changed in other applications.

As well as the basic protocol processing during transmission the engine can:

- Automatically retry packets that experience transmission errors during the first few bytes.
- Insert or remove a VLAN tag.
- Append padding to short packets to meet the minimum frame size.
- Overwrite the source address field in the packet with the MAC's own source address.
- Append a CRC to the end of the packet.

During reception the protocol engine can:

- Automatically drop packets that have errors during their first few bytes.
- Filter incoming packets and automatically discard any with a destination address that does not match. Both exact and hash-based matching are available.
- Flag errors and packet type information in the DMA descriptor.

The protocol engine can run the 10 Mbps, 100 Mbps and 1000 Mbps protocol variants and supports full and half-duplex operation at all speeds. Typically the external PHY would support auto-negotiation of the line parameters. During initialization the software will read the negotiated state using the management interface to the PHY and configure the MAC accordingly.

The protocol engine implements flow control. It will back-pressure the link when software explicitly requests or when the number of buffers available to the receive DMA engine falls below a threshold. In half-duplex mode back-pressure is applied by forcing collisions or a fake carrier on the line, in full-duplex mode flow control (pause) frames are sent.
**ETHERNET FRAME FORMAT**

The Ethernet frame format is shown in Figure 52.

![Figure 52: Ethernet Frame Format](image)
The frame is broken up into the following fields:

### Table 160: Ethernet Frame Fields

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFG</td>
<td>Inter-frame Gap. This is the period that the link is idle between frames. The size of the gap used by the MAC is programmable. The ifg 보내의 parameters in the mac_frame_cfg register sets the gap that will be used after reception and transmission of a packet. The ifg_thrsh threshold sets the size of the end of the gap in which the MAC does not check the carrier before transmitting.</td>
</tr>
<tr>
<td>Preamble</td>
<td>The preamble is used to allow other devices to detect the carrier and synchronize with the transmitter. Alternating ones and zeros are sent.</td>
</tr>
<tr>
<td>SFD</td>
<td>The Start of Frame Detect byte is used to mark the start of the active frame. It is a single byte with the value 8'hD5.</td>
</tr>
<tr>
<td>Destination Address</td>
<td>The destination address is the 6 byte unique Ethernet address that this frame is intended for. Packets with the destination address FF-FF-FF-FF-FF-FF are broadcast packets, and those with the least significant bit of the first byte (i.e. the first bit sent on the wire) set are multicast packets. The interface will filter incoming packets based on this field.</td>
</tr>
<tr>
<td>Source Address</td>
<td>The source address is the 6 byte unique Ethernet address of the sender of the frame. During packet transmission the interface can automatically overwrite an old source address with its own.</td>
</tr>
<tr>
<td>Type/Length</td>
<td>The meaning of the type/length field depends on whether the packet is sent in 802.3 format or Ethernet format. The field value is a 16 bit number, where the first byte is the most significant 8 bits and the second byte is the least significant 8 bits. If the value of the field is less than or equal to 1500 then this is an 802.3 format frame and this field gives the length of the data field. If the value is greater than 1535 then this is an Ethernet format frame and this field gives the type. The receive interface can be set to check this length against the actual packet length, and report an error if they differ.</td>
</tr>
<tr>
<td>Data</td>
<td>The data portion of the packet is the (layer 3) data that is being transported from the source to the destination. The standard gives the minimum and maximum sizes of the data as 46 and 1500 bytes (giving minimum and maximum packet sizes of 64 and 1518 bytes). Both can be changed in the mac_frame_cfg register to support jumbo packets.</td>
</tr>
<tr>
<td>CRC</td>
<td>The CRC (also known as Frame Check Sequence or FCS) is a 32 bit cyclic redundancy check calculated over all the bytes in the packet from the first destination byte to the last data byte. It is normally appended to the frame during transmission and checked during reception. The CRC is formed using the CRC-32 polynomial: [ x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1 ] The partial CRC accumulator is initialized to 32'hFFFFFFFF at the start of the packet. The 32 bit CRC is transmitted most significant byte first. The interface can automatically insert the CRC during transmission, this is selected on a per-packet basis in the DMA descriptor. If this is disabled the CRC must be provided as part of the data from the transmit FIFO. The MAC will always check the transmitted CRC and log an error if a packet has been sent with a bad CRC.</td>
</tr>
<tr>
<td>VLAN TAG</td>
<td>The VLAN Tag is not part of the Ethernet standard. It is an extra four byte header that is used when encapsulating other packets into a &quot;virtual LAN&quot; link. The interface can insert, replace or remove VLAN tags from packets during transmission. This is selected on a per-packet basis in the DMA descriptor, if the VLAN tag is altered then automatic CRC generation is also enabled.</td>
</tr>
</tbody>
</table>
PREPENDED HEADER FRAME FORMAT

In devices with system revision indicating PERIPH_REV3 or greater the MAC interface supports a modified frame format that has an additional header prepended before the Ethernet frame. The CRC can be set to cover the prepended header and Ethernet frame or just the Ethernet frame (or a combination). Ignoring the framing (IFG/Preamble/SFD for Ethernet, SOP/EOP marks on first and last bytes for Packet Fifo modes) the prepended header format is shown in Figure 53. This figure shows the CRC offset pointing to part way through the prepended header which is unusual, the standard cases are when the crc_offset is zero (i.e. the CRC covers everything) or the crc_offset is equal to the pkt_offset (i.e. the CRC covers the Ethernet frame and not the prepended header). The crc_offset must be less than or equal to the pkt_offset, if it is greater the behavior of the interface is UNPREDICTABLE.

![Figure 53: Prepended Header Format](image)

The pkt_offset and crc_offset must be a multiple of 8 bytes. An offset of zero indicates there is no offset and the start of the packet is used. The maximum offset is 248 (31*8) bytes. There are separate registers to set the offsets on the transmit and receive sides (and their values may be different). In both cases the CRC offset specifies the first byte of the packet that will be included in the CRC calculation. For the transmit side the tx_pkt_offset is used to locate the Ethernet header for replacement of the source address, and the VLAN tag position for automatic insertion, removal or replacement of the VLAN tag. On reception the rx_pkt_offset is used to specify the start of the Ethernet header for the address filtering and for extraction of the packet type.

The figure also shows the iphdr_offset. This identifies the start of the IP header in the packet for checksum checking (both IPv4 header checksum and TCP checksums are checked) and is discussed in later sections. To allow for networks that have a mix of VLAN and non-VLAN packets the interface can be configured to automatically add 4 bytes to the iphdr_offset when a packet with a VLAN tag is received.
PROTOCOL ENGINE CONFIGURATION

The basic encapsulation parameters are set in the **mac_frame_cfg** register. This allows configuring of the inter-frame gap, maximum backoff time, slot size, minimum frame size and maximum frame size. This must be programmed to the correct values for IEEE 802.3 operation at 10 Mbp/s and 100 Mbp/s. For gigabit operation the slot_size is automatically increased to 512 byte times (rather than 512 bit times) but software must half the IFG values; none of the other parameters should be altered. In networks configured to use jumbo packets the maximum frame size should be increased to match the rest of the network. Parameters should only be altered while the protocol engine is held in reset. Care must be used when altering the values from their standard ones, typically all endpoints on the network must be configured similarly. A detailed description of each of the parameters is given in the register description in Table 180 on page 307.

The effective slot size is adjusted by adding a 10 bit signed offset in bit times to the 512 bit times or 512 byte times selected by the data rate. The offset adjusts the MAC timing to take account of the number of cycles of latency through the PHY and back. Typically the field will be a small positive number.

The lfsr_seed is the only parameter in the **mac_frame_cfg** register that can be altered at any time and that can be altered without violating the 802.3 standard. Whenever it is written the value written sets the lowest eight bits of the linear feedback shift register that generates the pseudo-random backoff following a collision.

The rest of the protocol engine configuration is done in the **mac_cfg** register. For Ethernet operation the bypass sel bit must be clear. The ss_en bit must always be set unless the part is being run in a test mode.

Ethernet packets are recovered into a byte stream before being put into the 64 bit wide receive FIFO, and data is read in 64 bit chunks from the transmit FIFO and converted to a byte stream before it is sent. The system endian configuration is used to correctly order the bytes so that the first byte on the wire is at the lowest memory location.

The flow control portion of the protocol engine must also be programmed with the flow control mechanism and for full-duplex links the number of slot times to request the peer to pause transmission. These are set in the **mac_cfg** register in the fc_cmd and tx_pause_cnt fields.

Control of the movement of data between the transmit and receive FIFOs and the protocol engine is also done in the **mac_cfg** register. These settings are discussed in the transmit and receive sections below.

Errors generated in the interface are signalled in the **mac_status** register. Bits in this register are set when an error is detected and also reflect the interrupt state from the DMA engines. Whenever the register is read the error bits are cleared. There is a mask register **mac_int_mask** associated with the status register. If both the mask and status bits are set for any of the error conditions or DMA channels then the MAC interrupt is raised. Reading the **mac_status** register will clear the error bits, but the DMA channels must also be serviced before the interrupt is removed.

INTERFACE TO PHY

The interface to the PHY is done through the standard GMII/MII interface. This provides an 8 bit path each direction for gigabit operation and a 4 bit path for 100 Mbp/s or 10 Mbp/s operation. The PHY does the appropriate encoding for the link for transmission, and will deserialize the incoming data. The PHY reports carrier sense, collision and code violation to the protocol engine.

In most cases software will interrogate the PHY to discover the link speed and duplex settings. These are obtained either from auto-negotiation or based on the PHY capabilities. The settings must be used to configure the **mac_cfg** register.
TRANSMITTER OPERATION

TRANSMITTER CONFIGURATION

The transmitter clock is accepted as an input from the PHY device for 10/100 Mbit/s operation. However, for 1 Gbit/s operation the data lines are clocked at 125 MHz so the clock is forwarded with the data. In this case the reference clock input is used (on the BCM1250, the REFCK01 pin is for interfaces E0 and E1, and the REFCK2 pin for interface E2; on the BCM1125/H, REFCK0 is for E0 and REFCK1 for E1). The clock source is configured by the MAC speed selection in the mac_cfg register.

The transmit FIFO is a 64 bit wide fifo with 128 entries. Figure 54 shows the thresholds that are used to configure it. When the FIFO has space for data it will signal the DMA engine to request data. The tx_wr_thrsh field in the mac_thrsh_cfg sets the number of empty entries there must be in the FIFO before it will request data. The DMA engine fetches either 32 bytes or 64 bytes at a time, so this value should be set to 4 or 8.

The data from the transmit FIFO is read out by the protocol engine for transmission. Once transmission of a packet has started the DMA engine must ensure that there is always data in the FIFO when the protocol engine needs it. If the FIFO is empty at any time before the end of the packet then an underflow error is reported and transmission fails (in encoded Packet FIFO mode the transmission will not fail, instead cycles with no valid data will be inserted in the link protocol). To reduce the likelihood of the FIFO becoming empty the tx_rd_thrsh threshold can be set to ensure a certain number of entries have been written to the FIFO before transmission starts.

Figure 54: Transmit FIFO Thresholds
In half-duplex operation the Ethernet protocol expects collisions to occur, and relies on them to share access to the transmission medium. These collisions will only occur at the start of packet transmission, and the interface will backoff for a random period before retrying the transmission. The maximum backoff time is exponentially increased (up to a maximum 1024 slot times) each time a collision is encountered. If the collision is detected early in the packet the interface will automatically backoff and retry. If a collision happens late in the transmission then the packet is dropped.

Once transmission has begun entries are retained in the transmit FIFO until tx rl thrsh entries have been transmitted, if an error is detected during this time the packet can be automatically retried. Retaining the start of the packet in this way is enabled by setting the tx_hold_sop_en bit in the mac_cfg register. The errors that will cause automatic retry are also selected in this register.

Table 161 describes all the transmission error conditions, and lists the bit that must be set to enable automatic retry for them.

<table>
<thead>
<tr>
<th>Error</th>
<th>Bit to Set for Automatic Retry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision</td>
<td>retry_en</td>
<td>Collisions near the start of the frame are an expected part of the Ethernet protocol, and should always be retried.</td>
</tr>
<tr>
<td>Excessive Collision</td>
<td>ret_drpreq_en</td>
<td>After 16 attempts to transmit a frame have resulted in a collision the interface will report that excessive collisions have happened. This condition normally indicates a serious problem with the link (although it will occasionally be encountered on a large very busy Ethernet), in most cases the packet should not be automatically retried. Note that this error and late collisions use the same bit to enable automatic retry. Excessive collisions will set the excol_err bit in the mac_status register.</td>
</tr>
<tr>
<td>Late Collision</td>
<td>ret_drpreq_en</td>
<td>The Ethernet protocol sets the minimum packet size so that all collisions will be seen during the transmission of a minimum length packet. Any collision after the minimum length has been sent is a late collision and indicates a serious error. Late collisions can be caused by the Ethernet segment being longer than allowed by the specification, or by some other station on the segment turning on or off or violating the standard. In most cases the packet should not be automatically retried. Note that this error and excessive collisions use the same bit to enable automatic retry. Late collisions will set the ltcol_err bit in the mac_status register.</td>
</tr>
<tr>
<td>Underflow</td>
<td>ret_ufl_en</td>
<td>An underflow error is reported when the transmit protocol engine finds the transmit FIFO empty during transmission of a frame. This will happen if the DMA (or external agent in direct mode) has been unable to write data to the FIFO at the rate of the transmission. The tx rd thrsh threshold can be adjusted to avoid underflows by providing a data buffer in the FIFO to cover the request latency. In normal operation the underflow error may occasionally occur if the DMA engine is locked out from the bus or memory, and this could be automatically retried. If automatic retry is enabled care must be taken to ensure it is not being applied to every packet. If the read threshold is set too low, then all packets will get an underflow shortly after transmission starts (leading to a runt packet being sent). The time they take to be retried could cover the extra time needed to fetch data into the FIFO so they succeed on a second attempt. This situation should be fixed by increasing the read threshold. A FIFO underflow will set the tx undrlf bit in the mac_status register.</td>
</tr>
<tr>
<td>Overflow</td>
<td>None/Cannot be retried</td>
<td>An overflow error is reported when the DMA engine (or external agent in direct access mode) attempts to write to the transmit FIFO when the FIFO is full. The data written is lost. During DMA this error will only be seen if the tx wr thrsh parameter is incorrectly set. A FIFO overflow will set the tx ovfll bit in the mac_status register.</td>
</tr>
</tbody>
</table>
Automatic retry would normally only be enabled for collisions since the other conditions could indicate a more serious problem (once the system is known to work automatic retry could be added for occasional underflows).

The Gigabit Ethernet standard allows bursts of packets to be transmitted without requiring the link be released and reacquired. This is enabled by setting the burst_en configuration bit, which should only be set when the MAC is configured for half-duplex 1000 Mbp/s operation.

**TRANSMIT PATH**

Each MAC has two transmit DMA channels associated with it. If only one is enabled or only one has descriptors available then it is selected for transmission. If both channels are enabled and have packets available then the weights set in the `mac_txd_ctl` register are used to select which is used. The number of packets selected by the weight are sent from one DMA channel then the other is serviced for the number of packets set in its weight; if a channel becomes empty then the other channel immediately starts. (Note that if both channels are enabled and given weights of zero then no packets will be transmitted.) If the packets in each DMA channel are the same length (or on average the same length) then this will share the transmission bandwidth according to the weights. If the packets are on average of very different lengths the weights can be skewed accordingly. The selected DMA channel will write a packet into the transmit fifo, with the start and end specially marked for the protocol engine.

The packet to be transmitted is encapsulated in the Ethernet frame described above. The inter-frame gap, preamble and SFD are automatically prepended before data read from the FIFO. The data is then copied from the FIFO to form the body of the packet. There are six manipulations that the protocol engine can perform, configured per packet by options in the DMA descriptor:

- Replacement of source address. The source address that is in the packet supplied by the DMA engine can be overwritten with the address of this interface that has been set in the `mac_ethernet_addr` register.
- Removal of VLAN tag. The four bytes of VLAN tag can be dropped from the packet, converting a VLAN packet into a regular Ethernet packet.
- Insertion of VLAN tag. The four bytes in the `mac_vlantag` register can be inserted into the packet just after the source address, converting a standard Ethernet packet into a VLAN packet.
- Replacement of VLAN tag. The four bytes in the `mac_vlantag` register overwrite the four bytes in the VLAN tag position in the packet.
- Padding short packets to the minimum frame size.
- Insertion of CRC. The four byte CRC can be automatically replaced in the packet or appended to it. If any of the other modifications are done the CRC will be replaced to ensure the interface uses the correct CRC for the data sent.

During packet transmission the protocol engine always calculates the CRC of the frame. This value will be inserted if the interface is configured to automatically append the CRC. The CRC that is transmitted will always be compared to the CRC that was calculated. If the CRC was automatically generated they will always match. If the CRC was supplied through the transmit FIFO and it does not match the calculated one then a transmit CRC error is reported.

In Packet FIFO mode the only packet modification that can be done by the transmitter is to append the CRC.
**RECEIVER OPERATION**

**RECEIVER CONFIGURATION**

The receive fifo is a 64 bit wide fifo with 64 entries. When the FIFO contains data it will signal the DMA engine to request emptying. The \textit{rx\_rd\_thrsh} field in the \textit{mac\_thrsh\_cfg} register sets the number of valid entries that must be in the FIFO to request emptying. The DMA engine transfers in blocks of 32 bytes so this field should normally be set to four entries. The threshold is overridden when the end of a packet is in the FIFO, since these are unlikely to be correctly aligned.

![Figure 55: Receive FIFO Thresholds](image)

The receiver will hold off informing the DMA engine of the arrival of a new packet until successful reception of the first few bytes. The \textit{rx\_rl\_thrsh} field in the \textit{mac\_thrsh\_cfg} register sets the number of entries that must be written to the FIFO before it signals that data is available. If the end of the packet is received the data is always reported to the DMA engine. If there is any reception error before this point then the packet can be automatically dropped by flushing the data in the FIFO. The error will still be counted in the error statistics. If the \textit{drp\_erpkt\_en} bit in the \textit{mac\_cfg} register is set then all packets with errors will be thrown on the ground in this way. If this bit is clear then more selective bits in the \textit{mac\_cfg} determine whether packets with errors are dropped or delivered with an error status. Regardless of any of these settings, errors that are detected after the \textit{rx\_rl\_thrsh} threshold has passed are always delivered with an error status.

For example, if the \textit{rx\_rl\_thrsh} is set to 8 and the MAC detects a runt packet (shorter than the 64 byte minimum size Ethernet packet) the DMA engine will never be told about the packet and it will be automatically discarded.
Automatic discard can be triggered by any error that happens (FCS error, code error, dribble error, runt error, oversize error or length error) while the FIFO is locked (i.e. less than the number of entries set by rx_rl_thrsh have been written into the receive FIFO after the start of a given packet). Table 162 lists the bits in the `mac_cfg` register that should be set to enable automatic dropping selectively for each of the error types.

**Table 162: Receiver Error Conditions**

<table>
<thead>
<tr>
<th>Error</th>
<th>Bit to set for automatic dropping</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC Error</td>
<td><code>drp_fcserrpkt_en</code></td>
<td>A Frame Check Sequence or CRC error happens when the CRC field in the final four bytes of the received packet does not match the CRC that has been calculated for the data that was received. This error indicates that the data in the packet has been corrupted during transmission. Note that because the automatic packet dropping can only be done early in the packet (set by rx_rl_thrsh) longer packets with a CRC error cannot be dropped and will be received marked with an error.</td>
</tr>
<tr>
<td>Code Error</td>
<td><code>drp_codeerrpkt_en</code></td>
<td>A code error is reported for any error signalled by the PHY device. The details depend on the encoding used and the particular transmission medium. This error will usually be signalled because received data did not match a valid encoding, but it could also be signalled by loss of light in an optical medium, or the clock recovery block losing lock. A single code error can be ignored, but when multiple code errors are seen the PHY status should be checked using the management interface.</td>
</tr>
<tr>
<td>Dribble Error</td>
<td><code>drp_drblerrpkt_en</code></td>
<td>In 10 Mbp/s and 100 Mbp/s operation the data received from the PHY over the GMII interface is smaller than a byte (1 bit per clock for 10 Mbp/s and 4 bits per clock for 100 Mbp/s). However, the length of a valid Ethernet packet is always an integral number of bytes. If a packet is received where this is not the case a dribble error is reported (this error is sometimes called an alignment error). This error can never happen during gigabit operation because the interface from the PHY is 8 bits wide (any similar error would be reported by the PHY as a code error).</td>
</tr>
<tr>
<td>Runt Packet</td>
<td><code>drp_rntpkt_en</code></td>
<td>A runt packet is any packet that is shorter than the minimum packet size. The minimum packet size that is used by the MAC is configurable in the <code>mac_frame_cfg</code> register, for compliance with IEEE 802.3 this size will be set at 64 bytes.</td>
</tr>
<tr>
<td>Oversize Packet</td>
<td><code>drp_oszpkt_en</code></td>
<td>An oversize packet is any packet that is larger than the maximum packet size. The maximum packet size that is used by the MAC is configurable in the <code>mac_frame_cfg</code> register, for compliance with IEEE 802.3 this size will be set at 1518 bytes. If a packet is received that is longer than this size it will be truncated at one byte longer than the maximum frame size and marked with an error. i.e. If the max frame size is set to 1518 bytes and a packet of length 4000 bytes is received from the PHY, the DMA engine will transfer 1519 bytes to memory and in the start of packet DMA descriptor status information software will see the bad packet flag set and a length of 1519 bytes. Note that because the automatic packet dropping can only be done early in the packet (set by rx_rl_thrsh) in most cases oversize packets cannot be dropped and will be received marked with an error.</td>
</tr>
<tr>
<td>Length Error</td>
<td><code>drp_lenerrpkt_en</code></td>
<td>If the packet type/length field is less than 1500 then it holds the length of the data portion of the packet. A length error is signalled if this does not match the number of bytes that were actually received for the data section. Note that because packets may be padded to the minimum frame size a length error is not reported if the packet that is received is longer than the size given in the header.</td>
</tr>
</tbody>
</table>
At the end of the packet the protocol engine can write one additional entry into the FIFO containing status for the packet. This is used by the DMA controller to update the length and flags field in the first DMA descriptor of the packet. Appending of this status word is enabled by setting the ap_stat_en bit in the `mac_cfg` register, this bit must be set when the interface is configured for Ethernet operation and must be clear in Packet FIFO mode.

An underflow error is reported when the receive DMA engine (or external agent in direct mode) reads the receive FIFO when it is empty. The data returned is UNPREDICTABLE and the FIFO pointers will not change. During DMA this error will only be seen if the rx_rd_thrsh parameter is incorrectly set.

An underflow will set the rx_undrfl bit in the `mac_status` register.

An overflow error is reported when the protocol engine attempts to write to the receive FIFO and it is full. This indicates that the DMA engine (or external agent in direct mode) is not emptying the FIFO fast enough. The data written is lost.

A FIFO overflow will set the rx_ovrfl bit in the `mac_status` register.

**RECEIVE PATH**

The protocol engine processes all packets arriving from the PHY, removes the physical layer encapsulation and passes them into the receive FIFO. The receive DMA engine will remove data from the FIFO and transfer it into packet buffers in memory. The receiver is always active, regardless of the full- or half-duplex selection.

The packet is checked for errors during reception, and the CRC is computed and compared with the value in the packet. If an error is detected early in packet reception it will not have started to be DMA’ed and the packet can be dropped. If an error is detected after DMA has started, the DMA will be completed and the error flagged in the status bits written back to the descriptor. The dropping of packets is enabled by holding the first few entries in the receive FIFO and not informing the DMA logic that there is data to extract until a threshold has been reached. If the error happens before the threshold then the FIFO pointer can be restored to the entry that holds the start of the packet, and the packet is dropped.

<table>
<thead>
<tr>
<th>Error</th>
<th>Bit to set for automatic dropping</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Underflow</td>
<td>None</td>
<td>An underflow error is reported when the receive DMA engine (or external agent in direct mode) reads the receive FIFO when it is empty. The data returned is UNPREDICTABLE and the FIFO pointers will not change. During DMA this error will only be seen if the rx_rd_thrsh parameter is incorrectly set. An underflow will set the rx_undrfl bit in the <code>mac_status</code> register.</td>
</tr>
<tr>
<td>Overflow</td>
<td>None</td>
<td>An overflow error is reported when the protocol engine attempts to write to the receive FIFO and it is full. This indicates that the DMA engine (or external agent in direct mode) is not emptying the FIFO fast enough. The data written is lost. A FIFO overflow will set the rx_ovrfl bit in the <code>mac_status</code> register.</td>
</tr>
</tbody>
</table>
DESTINATION ADDRESS FILTERING

The MAC will filter received packets based on their Ethernet destination address. Packets that pass the filter are received, those that do not are never delivered to the DMA engine.

![Receive Address Filter Diagram](image-url)

**Figure 56: Receive Address Filter**
There are five components to the filter.

1. Broadcast packet detection. Broadcast packets are either all accepted or all rejected.

2. Exact match. The incoming packet address is compared to eight addresses (which may be unicast or multicast addresses), if it matches any of them then the packet is accepted, otherwise it is rejected. In parts with system revision PERIPH_REV3 or greater two of the addresses have masks associated with them, if a bit is clear in the mask it is excluded from the comparison. The sense of this section of the filter can be reversed, so that packets will only be accepted if it does not match any of the addresses. All exact match addresses are always enabled, if less than 8 entries are needed the extra ones can be written with copies of one of the accepted addresses, or with 00-00-00-00-00-00 which is not a valid address.

3. Hash match. Nine bits are extracted from the partial CRC of the packet for the six destination address bytes, this provides a hash value. This value is extracted from the standard CRC-32 described in Table 160 on page 268. There are eight 64-bit hash table registers, these are concatenated to form a 512 bit bitmap (map bit 511 is hash_table7 bit 63... map bit 0 is hash_table0 bit 0). The hash value is used as an index into this bitmap, the packet is accepted if the bit is set and rejected if the bit is clear. The sense of the match can be inverted so the packet will only be accepted if the bit is clear.

4. Multicast Match. The filter can be configured to accept all multicast packets. If this is enabled then the only reason to put multicast addresses in the Exact Match filter or enable multicast hash matches is to make use of the match_exact and match_hash flags in the DMA status information.

5. Pause Frame. If the header of the frame indicates that it is a Pause Frame (flow control frame) then it will normally be consumed by the interface and the transmitter flow controlled as requested. In parts where the system revision indicates PERIPH_REV3 or greater if the fwdpause_en control is enabled then the hardware will not act on Pause Frames and the address filter will accept them for reception.

If the destination address of the packet is accepted by any of the three filters or the promiscuous mode bit is set then the packet will be received. If there is no match then the packet is dropped. (Note that the filter will not operate properly if the fifo release threshold rx_rl_thrsh is set smaller than the position required for the packet header, since the threshold has a minimum of two this is only a concern when there is a prepended header.)

The receive address filter is controlled by the mac_adfilter_cfg register. There are a set of enable bits and invert bits to control the filter, as shown in Figure 56 on page 277. The address registers and hash map must also be written.

A packet will be accepted if any of the filters indicate it should be accepted. If the broadcast address FF-FF-FF-FF-FF-FF is put in an exact match address register then broadcast packets will be accepted regardless of the state of the broadcast enable bit. However, setting the hash entry corresponding to the broadcast address will not cause acceptance because a hash match is qualified by the packet being unicast or multicast.

The receive address filter is still applied in Packet FIFO mode. Since the data is not expected to be formatted as an Ethernet frame most applications will need to set the allpkt_en bit to accept all packets.
The value for the hash filter for a particular Ethernet address may be computed using the mchash macro and standard CRC generator given in the following code:

```c
/**
 * eth_hwcrc32(buf,len)
 * Calculate a CRC-32 of the specified bytes. This is used to
 * generate the masks for the hash filter on the receive side.
 * For the hash filter the bits must be swapped to match the hardware
 * Input parameters:
 * buf - buffer
 * len - length of buffer
 * Return value:
 * CRC as held in MAC hardware
 */

#define mchash(mca) ((eth_hwcrc32(mca, ENET_ADDR_LEN) >> 1) & 0x1FF)

static unsigned eth_hwcrc32(unsigned char *databuf,int datalen)
{
    unsigned int idx, crc = 0xFFFFFFFFUL, tmp;

    static unsigned int crctab[] = {
        0x00000000, 0x1db71064, 0x3b6e20c8, 0x26d930ac,
        0x76dc4190, 0x6b6b51f4, 0x4db26158, 0x5005713c,
        0xedb88320, 0xf00f9344, 0xd6d6a3e8, 0xcb61b38c,
        0x9b64c2b0, 0x86d3d2d4, 0xa00ae278, 0xbfbd21c
    };

    for (idx = 0; idx < datalen; idx++) {
        crc ^= *databuf++;
        crc = (crc >> 4) ^ crctab[crc & 0xf];
        crc = (crc >> 4) ^ crctab[crc & 0xf];
    }

   .toLocaleLowerCase
    ** Flip [31:0] to [0:31] to match hardware **
    tmp = 0;
    for (idx=0; idx<32; idx++)
        tmp |= ((crc<<idx)>>31)<<idx;

    return tmp;
}
```
**RECEIVE DMA CHANNEL SELECTION**

If a packet is accepted by the address filter the received DMA channel is selected. This is done by extracting eight bits out of the first 128 bytes received (if the rx_rl_thrsh threshold is set smaller than 128 then the eight bits must be in the rx_rl_thrsh bytes), and using them to lookup a two bit number in a small table formed using the `mac_chup` and `mac_chlo` registers. The offset into the packet is specified in nibbles (half bytes) in the `mac_cfg` register, enabling extraction of the DiffServ code point from the IP v6 traffic class field as well as the (byte aligned) IP v4 TOS field (see Figure 57). If the two bit number is zero the packet will be received on DMA channel zero, otherwise DMA channel 1 is used. Both bits get written in to the packet receive status information.

![Figure 57: Receive Channel Selection](image)

The channel select offset can be used to select any nibble in the first 128 bytes of the packet. For example if selection is to be made based on the low byte of the Ethernet packet type (which is the second byte of the field sent on the wire) the offset should be (decimal) 26.

Figure 58 shows how this offset is obtained. Even offsets refer to bytes in the received packets, odd offsets straddle bytes: the high four bits of the index byte come from the low four bits of the later byte in the frame and the low four bits of the index byte are the high four bits of the earlier byte in the frame. In the example a complete byte from the frame is used so the bottom bit of the offset will be zero. The upper bits are simply the byte offset (starting from 0) of the desired byte which is 13 for the second byte of the length/type field.

![Figure 58: Selecting the Channel Offset](image)
The receive channel selection is used in the same way in Packet FIFO mode.

In devices where the system revision indicates PERIPH_REV3 or greater the 8 bit index into the channel number table can be formed from two four bit fields extracted from the packet. The split_ch_en bit in the mac_cfg register enables this feature. The lower 4 bits of the index are extracted from the packet at the offset specified by the \{rx_ch_sel_msb,rx_ch_sel\} value and the upper 4 bits of the index are extracted from the offset specified by the rx_ch_msn_sel field in the mac_adfilter_cfg register. Both these are nibble offsets as described above. The \texttt{rx.ch.msn.sel} must point later in the packet, if \(\texttt{rx.ch.msn.sel} \leq \texttt{rx.ch.sel}\) the channel selected is UNPREDICTABLE. When \texttt{split.ch.en} is zero the \texttt{rx.ch.msn.sel} value is ignored and the upper 4 bits of the index are the nibble after the \\(\texttt{rx.ch.msn.sel},\texttt{rx.ch.sel}\) offset as described above.

**PACKET TYPE IDENTIFICATION**

The receiver reads the Ethernet packet type field in the inbound packet. Some common packet types are detected and encoded in the status that is written to the descriptor. Four of the types are fixed, the other four are programmable in the \texttt{mac_type_cfg} register. The packet type encodings are shown in Table 163.

<table>
<thead>
<tr>
<th>Ethernet Type Field</th>
<th>Status[57:55] Encoding</th>
<th>Packet Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>16'h0800</td>
<td>3'b000</td>
<td>IP v4</td>
</tr>
<tr>
<td>16'h0806</td>
<td>3'b001</td>
<td>ARP</td>
</tr>
<tr>
<td>&lt;=1500</td>
<td>3'b010</td>
<td>802.3 length</td>
</tr>
<tr>
<td>Other</td>
<td>3'b011</td>
<td>No match</td>
</tr>
<tr>
<td>mac_type_cfg[15:0]</td>
<td>3'b100</td>
<td></td>
</tr>
<tr>
<td>mac_type_cfg[31:16]</td>
<td>3'b101</td>
<td></td>
</tr>
<tr>
<td>mac_type_cfg[47:32]</td>
<td>3'b110</td>
<td></td>
</tr>
<tr>
<td>mac_type_cfg[63:48]</td>
<td>3'b111</td>
<td></td>
</tr>
</tbody>
</table>

Table 163: Ethernet Type Mappings

In Packet FIFO mode the 13th and 14th bytes of the received packet will be checked against the same values and the three bit encoding will be placed in the descriptor status word. In most cases this value should be ignored, but it is possible that the four configurable types could be used to identify special packets.
**IPV4 HEADER CHECKSUM**

The receiver will check packets for an IPv4 header with no options and a correct header checksum. The iphdr_offset field in the **mac_adfilter_cfg** register sets the offset into the received packet that the header starts (this will normally be 15, since the IP header will start after the Ethernet header, but it can be adjusted to allow encapsulation headers to be skipped). Note that this offset is specified with the first byte in the destination address as 1. The first byte of the header must be 8'h45 indicating the IP version number is 4 and the header is the standard 5 (32 bit) words with no options. If this byte matches and the header checksum of the 5 words is correct (per RFC791) then the bad_ip4csbit will be clear in the status written back to the packet DMA descriptor.

If the bad_ip4cs bit is set, the packet may not be an IPv4 packet, it may have header options, or it may have an incorrect checksum. Software must determine which of these is the case. Note that the fast path of good packet, Ethernet type IPv4, IPv4 header with no options and good checksum can be detected with an AND mask and BEQ since all the status bits will be low in this case.

The IPv4 header checksum is only computed in Ethernet mode. In Packet FIFO mode the bad_ip4cs bit will always be set.

**TCP CHECKSUM CHECKING**

The receiver will check packets for a valid TCP checksum. The result of this check is only valid if the IPv4 Header Checksum is correct. The iphdr_offset (described in the previous section) is used to locate the start of the IP header. The pseudo-header containing the source and destination IP addresses, IP protocol type and TCP length (IP length-20) is summed then the checksum continues for the TCP length of the packet (padded with a byte of 0 if required). The bad_tcpcs flag (written back to the b portion of the descriptor) will be set if the checksum is not 16'hFFFF. The same pseudo-header and checksumming algorithm is used for UDP as TCP (the UDP header itself is slightly different but since the header is included in the checksum this does not matter). The checksum is optional in UDP so if the bad_tcpcs flag is set then software must check for a packet without checksum (the field is zero) before rejecting the packet.

The TCP checksum is only computed in Ethernet mode. In Packet FIFO mode the bad_tcpcs flag will be UNPREDICTABLE.

**PACKETS DROPPED BY THE DMA CHANNEL**

A packet will be dropped by the DMA channel for one of three reasons:

1. The channel is disabled.
2. The channel is enabled but has no descriptors to receive the packet into.
3. The Drop bit is set for the channel (for example if one of the DMA channels is used for best effort traffic, that channel may have its drop bit set, so during overload no best effort packets are delivered to the system).

In devices with system revision indicating PERIPH_REV3 or greater the receive channel provides a counter in the **dma_oddpktlost** register that records the number of times a packet was lost and the channel was out of descriptors. The count is incremented every time a packet is received and the channel is out of descriptors, the other state of the channel is not factored in. Thus the count will still be incremented if the channel is disabled and out of descriptors when a packet is received. And it will be incremented if the drop bit is set and the channel is out of descriptors when a packet is received. But the count will not be incremented if there are descriptors available and the packet is dropped because the channel is disabled or the drop bit set.
**FLOW CONTROL**

The MAC supports flow control in both full-duplex and half-duplex modes. In full-duplex mode pause frames are used to request the peer to stop sending for a particular length of time. In half-duplex mode back-pressure is applied by forcing the physical layer into not allowing transmissions.

Flow control can be explicitly invoked by setting the fc_sel bit in the `mac_cfg` register, or it can be requested automatically by the interface when the number of receive DMA buffers falls below the low watermark (see Section: “Descriptor Count Watermarks” on page 157). Flow control is invoked if any of the sources request it.

The flow control method used when the interface is unable to accept data is set by the fc_cmd field in the `mac_cfg` register, as shown in Table 164.

<table>
<thead>
<tr>
<th>fc_cmd</th>
<th>Half-Duplex Back Pressure Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>2'b00</td>
<td>Disabled</td>
</tr>
<tr>
<td>2'b01</td>
<td>Force Collision</td>
</tr>
<tr>
<td>2'b10</td>
<td>False Carrier</td>
</tr>
<tr>
<td>2'b11</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Table 164: Back Pressure Methods in Half-Duplex Operation

If the interface is operating in half-duplex mode and flow control is invoked back pressure will be applied. If the fc_cmd field is set to 2'b01 back pressure is generated by the collision method. The interface forces collisions on the link by transmitting the collision data pattern. If the carrier is not currently being detected, or the protocol engine has not yet received more than the number of bytes set in the ifg_rx field of the `mac_frame_cfg` register then the collision transmission begins immediately. If more than the threshold number of bytes have been received in the current packet the collision will not be forced until after the incoming packet has completed (to avoid a late collision). Packet transmission is still possible when back pressure is applied in this way. If there is an outgoing packet it will contend for access in the usual manner. If the transmitter gains the link then the packet can be sent normally, if it loses then the back pressure machine will cause collisions with the packet that another source is trying to send.

In half-duplex mode if fc_cmd is set to 2'b10 the interface will apply back pressure by causing a false carrier sense. This is done by sending the collision data pattern (i.e. generating a carrier) after only 1/3rd of inter-frame gap time has elapsed since the carrier was last detected. This causes all other sources to sense a carrier and backoff. The carrier must be continually generated to ensure the other stations continue to detect it, so it is not possible to transmit packets while using this method.

If the interface is operating in full-duplex mode, flow control is applied by transmitting a Pause Frame. A frame is created with the Pause Count (the number of slot times that the peer should not transmit) set to the value encoded in the tx_pause_cnt field in the `mac_cfg` register. When a pause is requested the slot times are also counted locally. Towards the end of the period if the interface can still not accept packets an additional pause frame will be sent, requesting another pause in transmission by the peer. If flow control is no longer needed because DMA buffers have been added or the fc_sel bit has cleared, or if flow control is disabled by the fc_cmd field being set to 2'b00, then a Pause Frame is immediately sent with a zero Pause Count to restart the peer.
In full-duplex mode the reception of a valid Pause Frame always suspends transmission at the end of the packet currently being sent, transmission will be resumed after the requested pause time unless another flow control packet is received. Pause frames will be detected if their destination address matches the address in the `mac_ethernet_addr` register or the multicast address 01-80-C2-00-00-01 and the packet type matches the MAC Control type (88_08) and he pause frame opcode (00_01) is found. Normally valid Pause frames will be consumed by the MAC and not delivered to the DMA engine. However in some situations (for example where a system requires that high priority traffic ignore the flow control) it is useful for the Pause Frame passed to the software (which could disable the low priority DMA channel and keep the high priority one active). On devices with the system revision indicating PERIPH_REV3 or greater the fwdpause_en bit can be set in the `mac_adfilter_cfg` register to cause the hardware to pass Pause frames to the DMA engine instead of acting on them.

Hardware response to pause frames is normally done at the MAC level. This ensures that transmission stops as soon as possible. On devices with the system revision PERIPH_REV3 or greater the flow control may be performed in the DMA engines allowing each DMA channel to either obey or ignore the directive. Since there may be packets queued in the transmit fifo (up to 1KB of data between the DMA engine and MAC) there is a longer round-trip for the flow control loop in this case. The ch_base_fc_en bit must be set in the mac_vlantag register to enable channel based flow control. Once this bit has been set if the fc_pause_en bit is set in the dma_config1 register for a channel then it will pause on flow control, otherwise it continues to transmit even when flow control is requested. On devices with the system revision PERIPH_REV3 or greater when the fwd_pause_en bit is clear and hardware is acting on the pause frame the current pause state can be read from the tx_pause_on bit in the mac_status register, this bit will be set if the MAC is still in the pause interval set by the previous pause frame.

There are a number of ways pause frames can be managed. The standard one is to have the MAC act on them and consume them. Alternatively, the DMA engine can act on them and stop only one channel. Or they are passed to software and the hardware will not respond to them. Note that in an entirely software controlled flow control system the cpu_pause_en bits in the dma_config1 register can be used to pause DMA channels, but that the control loop will be much longer than having hardware act on the pause frames. A mix of software and hardware may be best, for example if channel based flow control is being used the DMA engine can be configured to stop both channels on receipt of a pause frame but software can poll the tx_pause_on bit and may choose to release one of the channels.

<table>
<thead>
<tr>
<th>fwdpause_en</th>
<th>ch_base_fc_en</th>
<th>Ch 0 fc_pause_en</th>
<th>Ch 1 fc_pause_en</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Standard operation. The MAC will pause on receipt of a pause frame and will consume the frame.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Ignore pause frames. The pause information is given to the DMA engine but is applied to neither channel.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Pause only channel zero. The pause information is given to the DMA engine but is only applied to channel 0.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Pause only channel one. The pause information is given to the DMA engine but is only applied to channel 1.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Pause both channels. The pause information is given to the DMA engine and is applied to both channels. The Standard operation setting will achieve the same result with a shorter flow control loop time, so this setting is only likely to be used in systems that switch between it and selective channel pause.</td>
</tr>
</tbody>
</table>
If the interface is operating in 8 or 16 bit encoded Packet FIFO mode (see Section: “8-Bit Packet FIFO Operation” on page 293 and Section: “16-Bit Packet FIFO Operation” on page 298) the flow control is invoked by asserting the RXFC signal. In all other Packet FIFO modes flow control is ignored.

**INTERRUPTS**

There are two methods of signalling interrupts from the Ethernet and Packet Fifo interface. The standard method uses a single system interrupt for all DMA and management interrupts, the split method uses a special interrupt for DMA channel 1 and the standard interrupt covers channel 0 and management.

The `mac_status` register contains the status information for a MAC. The low 32 bits contain the receive and transmit status for each of the DMA channels, the upper bits contain error and RMON counter information. There is a mask register `mac_int_mask` associated with the status register. If a bit is set in both the mask and status then an interrupt will be raised. Reads to the `mac_status` register have the side effect of clearing latched interrupt information. The information in the status register may be read with no side effects through the `mac_status_debug` register.

The split_ch1 bit in the `mac_int_mask` register is used to select between the standard and split methods of signalling and clearing interrupts. It defaults to 0 to give the standard behaviour.

**STANDARD INTERRUPT SIGNALING**

In the standard mode the main interrupt for the MAC is raised whenever any status bit is set and the corresponding mask bit is set. Reads from the `mac_status` register will clear all latched bits.

**SPLIT INTERRUPT SIGNALING**

In the split mode the main interrupt for the MAC is raised whenever any channel 0 DMA status bit (from bits 23:16, 7:0) is set and the corresponding mask bit is set, or when any error status bit (from bits 46:40) and corresponding mask bit is set. Reads from the `mac_status` register will clear all latched bits associated with channel 0 or errors. The channel 1 interrupt for the MAC will be raised if any channel 1 DMA status bit (from bits 31:24, 15:8) is set and the corresponding mask bit is set. Reads from the `mac_status1` register will clear all latched bits associated with channel 1.
MANAGEMENT INTERFACE TO PHY

There is a simple serial management interface between the MAC and the PHY device (or devices). It supports up to 32 PHY chips (although some PHY devices attach special meaning to address 0) each of which have 32 16-bit registers.

The CPU runs the serial protocol through reading and writing the `mac_mdio` register.

The management data clock (MDC) is generated by the interface and is an input to the PHY. Data written to the mdc bit in the `mac_mdio` register is put out on the MDC pin.

The management data i/o line (MDIO) can be driven by either the MAC or the PHY. It should have an external 1.5K pull-up resistor to pull the line to a 1 when neither is driving. The state of the pin can always be read on the mdio_in bit in the `mac_mdio` register. The interface can drive the line by clearing the mdio_dir bit and writing the data to output to the mdio_out bit.

The MDIO data is latched by the PHY on the rising edge of MDC, so the mdio_dir and mdio_out bits should never be changed when the mdc bit is changed from a zero to a one. To send data to the PHY the mdio_dir bit should be cleared, a single write to the `mac_mdio` register can be used to set mdc low and put the data on mdio_out. The same data should be written to mdio_out when mdc is set high.

There is a general purpose output pin (GENO) associated with each MAC that is also controlled through the `mac_mdio` register. This pin is not part of the standard G/MII connection.

Each MAC interface has its own set of MDIO/MDC/GENO pins. This allows the MAC drivers to be completely independent with no shared resources, which works well in designs where the MACs are partitioned across the processors with each running its own operating system. In a system running a single operating system (or where all the MACs are controlled from a single driver) the PHY addressing scheme can be used to attach the PHYs for all three MACs to the MDIO/MDC lines of a single MAC. In this case the other MDC lines become available as general outputs, and the other MDIO lines as GPIOs.
Figure 59 shows the flow for the main loop for sending and reading bits.

![Diagram of MDIO Flows]

**Figure 59: MDIO Flows**

- **Write Bits**
  - Single Write to mac_mdio
    - mdc = 0
    - mdio_out = bit to send
    - mdio_dir = 0
  - Wait Half Clock Cycle Time
  - Single Write to mac_mdio
    - mdc = 1
    - mdio_out = bit to send
    - mdio_dir = 0
  - Wait Half Clock Cycle Time
  - More to Send?
    - Yes
    - Single Write to mac_mdio
      - mdc = 0
      - mdio_dir = 1
    - No
    - Single Write to mac_mdio
      - mdc = 0
      - mdio_dir = 1

- **Read Bits**
  - Single Write to mac_mdio
    - mdc = 1
    - mdio_dir = 1
  - Wait Half Clock Cycle Time
  - Single Write to mac_mdio
    - mdc = 0
    - mdio_dir = 1
  - Wait Half Clock Cycle Time
  - Read Bit From mdio_in
  - More To Read?
    - Yes
    - Read Bits
    - No
The management protocol is shown in Table 166. This shows the two sides of the read (the MAC request and the PHY reply) and the full sequence sent by the MAC for a write. Bits are marked Z where the line is not being driven. Bits marked A form the five bit address of the PHY that the MAC is directing the command to (typically the PHY chip will have some input pins to set the address it responds to). The bits marked R form the five bit register address. The bits marked x are the transfer of the 16 bits being read or written. The two addresses and data are sent most significant bit first.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Idle</th>
<th>Start</th>
<th>Code</th>
<th>Device Addr</th>
<th>Register Addr</th>
<th>Turn Around</th>
<th>Data</th>
<th>Idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read (MAC)</td>
<td>Idle</td>
<td>01</td>
<td>10</td>
<td>AAAA</td>
<td>RRRRR</td>
<td>ZZ</td>
<td>ZZZZZ ZZZZZ ZZZZZ ZZ</td>
<td>Idle</td>
</tr>
<tr>
<td>Read (PHY)</td>
<td>Idle</td>
<td>ZZ</td>
<td>ZZ</td>
<td>ZZZZZ</td>
<td>ZZZZZ</td>
<td>Z0</td>
<td>XXXX XXXX XXXX XXXX</td>
<td>Idle</td>
</tr>
<tr>
<td>Write (MAC)</td>
<td>Idle</td>
<td>01</td>
<td>01</td>
<td>AAAA</td>
<td>RRRRR</td>
<td>10</td>
<td>XXXX XXXX XXXX XXXX</td>
<td>Idle</td>
</tr>
<tr>
<td>Write (PHY)</td>
<td>Idle</td>
<td>ZZ</td>
<td>ZZ</td>
<td>ZZZZZ</td>
<td>ZZZZZ</td>
<td>ZZ</td>
<td>ZZZZ ZZZZ ZZZZ ZZZZ</td>
<td>Idle</td>
</tr>
</tbody>
</table>

Some PHY devices can signal interrupts when the link status changes. These may be connected through one of the GPIO pins.

**RMON COUNTERS**

Each MAC has 32 RMON statistics gathering registers. They are all 32 bits wide except the byte counts which are 64 bits wide. Zeros will be read in the top 32 bits if a 64 bit access is made to the 32 bit registers, so software can treat them as all being 64 bit except for computing overflow. The registers may be cleared by software writing 0 to them. The counters must be written when the interface is initialized, until a write has been performed their behavior will be UNPREDICTABLE. A write of any value other than zero will set the counter to that value and it will increment from there. If any counter overflows it will wrap to zero and continue to count. The cntr_ovrfl_err bit is set and the counter number is written to the counter_addr field in the mac_status register. The RMON counters are only updated in Ethernet Mode.

Table 167: RMON Counters

<table>
<thead>
<tr>
<th>Number (offset from 00_1006_0000)</th>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tx Byte Counter</td>
<td>This Counter counts the total number of bytes successfully transmitted by the MAC. All bytes in the frame from the first destination address byte to the last CRC byte are included in the count. Successful transmission of a packet is defined by transmission of a packet without any errors such as collision, underrun or detection of invalid byte valid in control field of FIFO. This is a 64 bit count.</td>
</tr>
<tr>
<td>0 - +4000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - +5000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - +6000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Tx Collision Counter</td>
<td>This Counter counts the total number of collisions experienced by the MAC since the counter was last cleared. The count includes late collisions and will increment by 16 on an excessive collision.</td>
</tr>
<tr>
<td>0 - +4008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - +5008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - +6008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Tx Late Col. Counter</td>
<td>This Counter counts the total number of late collisions experienced by the MAC since the counter was last cleared.</td>
</tr>
<tr>
<td>0 - +4010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - +5010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - +6010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Tx Ex. Col. Counter</td>
<td>This Counter counts the total number of excessive collisions experienced by the MAC since the counter was last cleared. An excessive collision is defined as 16 consecutive collisions during the attempt to transmit a particular packet. The MAC will abort the packet transmission when it suffers excessive collisions.</td>
</tr>
<tr>
<td>0 - +4018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 - +5018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - +6018</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 167: RMON Counters (Cont.)

<table>
<thead>
<tr>
<th>Number</th>
<th>Offset from 00_1006_0000</th>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
</table>
| 4       | _0 +4020  
_1 +5020  
_2 +6020 | Tx FCS Error Counter     | This Counter counts the total number of packets transmitted by the MAC with FCS error. This can only happen when MAC is not appending the CRC to the transmitted packet. The MAC checks the CRC on all transmitted packets and flags all packets with FCS Errors. |
| 5       | _0 +4028  
_1 +5028  
_2 +6028 | Tx Abort Error Counter   | This Counter counts the total number of packets which were aborted during transmission. Transmission can be aborted by a FIFO underrun. |
| 7       | _0 +4038  
_1 +5038  
_2 +6038 | Tx Bad Packet Counter    | This Counter counts the total number of bad packets transmitted by the MAC. Bad packets are those packets which are not successfully transmitted. A packet which experienced late collision or excessive collision or is aborted by MAC during its transmission is considered bad packet. |
| 8       | _0 +4040  
_1 +5040  
_2 +6040 | Tx Good Packet Counter   | This Counter counts the total number of successfully transmitted packets. |
| 9       | _0 +4048  
_1 +5048  
_2 +6048 | Tx Runt Packet Counter   | This Counter counts the total number of runt packets (packets smaller than the minimum frame size) transmitted by the MAC. The minimum frames size for IEEE 802.3 compliance is 64 bytes, if a different minimum frame size has been set in the mac_frame_cfg register then runt packets will be any packet smaller than that number. |
| 10      | _0 +4050  
_1 +5050  
_2 +6050 | Tx Over Size Packet Counter | This Counter counts the total number of oversize packets (packets larger than the maximum frame size) transmitted by the MAC. The maximum frames size for IEEE 802.3 compliance is 1518 bytes, if a different maximum frame size has been set in the mac_frame_cfg register then oversize packets will be any packet larger than that number. |
| 11      | Reserved                 |                          |                                                                             |
| 12      | Reserved                 |                          |                                                                             |
| 13      | Reserved                 |                          |                                                                             |
| 14      | Reserved                 |                          |                                                                             |
| 15      | Reserved                 |                          |                                                                             |
| 16      | _0 +4080  
_1 +5080  
_2 +6080 | Rx Byte Counter          | This Counter counts the total number of bytes received by the MAC regardless of the status of a packet (good or bad, as defined below). This is a 64 bit count. |
| 17      | _0 +4088  
_1 +5088  
_2 +6088 | Rx Multicast Packet Counter | This Counter counts the total number of packets received by the MAC with a multicast destination address (broadcast packets are not counted). |
| 18      | _0 +4090  
_1 +5090  
_2 +6090 | Rx Broadcast Packet Counter | This Counter counts the total number of packets received by MAC with the broadcast destination address. |
| 19      | _0 +4098  
_1 +5098  
_2 +6098 | Rx Bad Packet Counter    | This Counter counts the total number of bad packets received by the MAC. Bad packets are those packets which have at least one of the following errors: FCS Error  
Length Error  
Dribble Error (Only in 10/100 Mbp/s)  
Code Error  
Runt Packet Error  
Oversize Packet Error |

**Broadcom Corporation**
### Table 167: RMON Counters (Cont.)

<table>
<thead>
<tr>
<th>Number (offset from 00_1006_0000)</th>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Rx Good Packet Counter</td>
<td>This Counter counts the total number of good packets received (packets with no errors) by the MAC.</td>
</tr>
<tr>
<td>21</td>
<td>Rx Runt Packet Counter</td>
<td>This Counter counts the total number of runt packets (packet smaller than the minimum frame size) received by the MAC. The minimum frames size for IEEE 802.3 compliance is 64 bytes, if a different minimum frame size has been set in the <code>mac_frame_cfg</code> register then runt packets will be any packet smaller than that number.</td>
</tr>
<tr>
<td>22</td>
<td>Rx Over Size Packet Counter</td>
<td>This Counter counts the total number of oversize packets (packets larger than the maximum frame size) received by the MAC. The maximum frames size for IEEE 802.3 compliance is 1518 bytes, if a different maximum frame size has been set in the <code>mac_frame_cfg</code> register then oversize packets will be any packet larger than that number.</td>
</tr>
<tr>
<td>23</td>
<td>Rx FCS Error Packet Counter</td>
<td>This Counter counts the total number of packets received by the MAC with an FCS Error (i.e. a bad CRC).</td>
</tr>
<tr>
<td>24</td>
<td>Rx Length Error Packet Counter</td>
<td>This Counter counts the total number of packets received by the MAC with Length Error (where the actual number of bytes received did not match the length given in the Ethernet header).</td>
</tr>
<tr>
<td>25</td>
<td>Rx Code Error Packet Counter</td>
<td>This Counter counts the total number of packets received by the MAC that had a Code error signalled by the PHY.</td>
</tr>
<tr>
<td>26</td>
<td>Rx Dribble Error Packet Counter</td>
<td>This Counter counts the total number of packets received by the MAC with a dribble error. A dribble error (also referred to as an alignment error) is when a packet is received that does not contain an integral number of bytes. This counter is not used for 1000 Mbp/s operation.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>
PACKET FIFO INTERFACES

In addition to the standard Ethernet mode, the network interfaces can be put into Packet FIFO mode. In this mode the Ethernet protocol processing is disabled and the pins are used to provide a simple interface with either an 8-bit or 16-bit data path each way. The Packet FIFO interface can be clocked at up to 208 MHz or higher (subject to part speed grade, for full details refer to the hardware data sheet). As can be seen in Figure 51 on page 264 the Packet FIFO engine sits to the side of the Ethernet MAC unit, but most of the network interface logic is common to the Ethernet and Packet FIFO modes. The same DMA controller is used to move data between the packet interface and memory.

There are several modes of Packet FIFO operation, these select the data width and method of framing packets. The minimum packet size supported in the Packet Fifo modes is 10 bytes, using smaller packets than this will have UNPREDICTABLE results. The maximum packet size that will be correctly reported is 16K - 1 bytes (because of the field width in the DMA descriptor) however the interface does not impose a maximum and will continue receiving as long as there are DMA descriptors available.

The 8-bit Packet FIFO modes simply replace the Ethernet MAC with the simpler Packet FIFO logic. The data pins remain the same and the data valid (RXDV, TXEN) and error (RXER, TXER) pins are used to signal framing information. In addition in the encoded mode the collision (COL) and management data (MDIO) pins get used to provide flow control on the link. Switching between Ethernet and any 8-bit Packet mode only affects the interface being switched, so there can be any mix of Ethernet and 8-bit Packet interfaces.

The 16-bit Packet FIFO modes do a larger re-configuration of the pins. The Data Sheet gives the mapping from the Ethernet use to the 16-bit mode uses of the pins. Interface 1 does not support 16-bit modes, the system behavior will be UNDEFINED if it is configured for 16-bit operation. The reuse of pins causes some restrictions on the configurations that can be supported when 16-bit Packet FIFO mode is used. If Interface 0 is put in 16-bit Packet FIFO mode then interface 1 may not be used. If Interface 2 is put in 16-bit Packet FIFO mode then interface 1 may not be used, and interface 0 can only be used in a Packet FIFO mode.

On the BCM1125/H the interfaces may be configured as two 8-bit interfaces or a single 16 bit interface.

Table 168: BCM1125 Ethernet/Fifo Pin Usage

<table>
<thead>
<tr>
<th>E0 pins</th>
<th>E1 pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Interfaces Ethernet or 8-bit Fifo</td>
<td>GMII or 8 bit Fifo E0, GMII or 8-bit Fifo E1</td>
</tr>
<tr>
<td>One 16-bit Packet Fifo</td>
<td>16-bit Packet Fifo F</td>
</tr>
</tbody>
</table>

The valid combinations for a BCM1250 are:

- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in Ethernet mode.
- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in 8-bit Packet mode.
- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in 16-bit Packet mode.
- Interface 0 in 8-bit Packet mode, Interface 1 disabled, Interface 2 in 16-bit Packet mode.
The pin mapping is summarized in the following table:

<table>
<thead>
<tr>
<th>Table 169: BCM1250 Ethernet/Fifo Pin Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E0 pins</strong></td>
</tr>
<tr>
<td>All Interfaces Ethernet or 8-bit Fifo</td>
</tr>
<tr>
<td>One 16-bit Packet Fifo, One Ethernet or 8-bit Fifo</td>
</tr>
<tr>
<td>Two 16-bit Packet Fifos</td>
</tr>
<tr>
<td>One 16-bit Packet Fifo, One 8-bit Packet Fifo</td>
</tr>
</tbody>
</table>

Other than the external data width and framing options (described below) the Packet FIFO modes use the same internal data path. The following rules/differences apply in all modes:

- The speed selection in the `mac_cfg` register must be set for gigabit operation to use the reference input clock to generate the transmit clock.
- The IPv4 header checking is not performed. The iphdr_offset field in the `mac_adfilter_cfg` register is not used and should be set to zero.
- The address filter is not normally useful in Packet FIFO mode and can be configured to accept all packets by setting the allpkt_en bit in the `mac_adfilter_cfg` register.
- The 13th and 14th bytes of the incoming packet are still compared as if they contained an Ethernet type and the DMA status word in bits [57:55] will still be set as described in Section: "Packet Type Identification" on page 281.
- If the bypass_fcs_chk bit is set in the `mac_cfg` register then received packets will have their final four bytes checked for a valid CRC-32 (using the same algorithm as for the Ethernet) and the bit in the receive DMA status word will be set accordingly. If the bypass_fcs_chk bit is clear the status bit will always indicate the CRC check passed.
- The receive channel selection is done in the same way as for the Ethernet mode as described in Section: "Receive DMA Channel Selection" on page 280.
- The only transmit options (see Section: "Data Buffers and Descriptors" on page 147 and Table 107 on page 173) that can be used in the start of packet descriptor are No Modification (4'b0111) and Append CRC (4'b001). All other options will give UNPREDICTABLE results.
- Some modes of the Packet FIFO interface allow packet errors to be signalled. These packets will be marked as bad in the receive DMA status information. The automatic discard of error packets (described in the Section: "Receiver Configuration" on page 274) should be disabled in Packet FIFO modes.
- The RMON counters are not incremented when the interface is in Packet FIFO modes.
Flow Control In Encoded Packet FIFO Modes

Flow control is available in the encoded packet fifo modes. In addition to the DMA descriptor based flow control there is a link level flow control.

The transmitter can be controlled by the COL input in 8 bit mode and the TXFC input in 16 bit mode (note that these are the same physical pin). Since the input is synchronized internally there is some uncertainty in the number of additional bytes that can be transmitted following assertion of the flow control signal. The transmitter will stop:

- 2-4 cycles after TXFC assertion in 16 bit mode no CRC (4-8 bytes)
- 2-6 cycles after TXFC assertion in 16 bit mode + CRC (4-12 bytes)
- 2-4 cycles after TXFC assertion in 8 bit mode no CRC (2-4 bytes)
- 2-8 cycles after TXFC assertion in 8 bit mode + CRC (2-8 bytes)

The receiver will assert its flow control output (MDIO in 8 bit mode, RXFC in 16 bit mode) either because the DMA watermark state requests flow control or to prevent the receive fifo overflowing. The enc_fc_thrsh field in the mac_thrsh_cfg register sets the flow control threshold. Flow control will be requested when there are fewer than this number of doublewords free in the fifo. There is a 1 cycle delay from the number of entries in the fifo falling below the threshold and the external signal being asserted, and there can be up to 10 cycles of data (10 bytes in 8 bit mode, 20 bytes in 16 bit mode) in flight between the receive data pins and the fifo.

As an example consider connecting two parts back-to-back. With the default threshold of 4 double words (32 bytes) in 16 bit mode there can be 22 bytes (2 from the one cycle delay and 20 in flight) added to the receive fifo if the transmitter stopped instantaneously. However, there is additional delay. If CRCs are not being appended by the transmitter an additional 8 bytes could be sent, leaving two bytes margin. But if the transmitter was appending CRC then the additional 12 bytes would cause the receive fifo to overflow, so the threshold must be increased to 5 doublewords.

Note that the direction of the MDIO/RCFC pin is controllable by software using the mdio_dir bit in the mac_mdio register. This must be set for output (which is the default) when receive flow control is being used.

8-BIT PACKET FIFO OPERATION

There are two models for running the Packet FIFO. In the packet model Start of Packet (SOP) and End of Packet (EOP) data are flagged and all data between these marks are counted as a single packet and passed through the DMA engine in the same way an Ethernet packet would be. In this packet based model once a SOP has been seen further SOP signals are ignored until an EOP has been triggered. The second model is to have an unframed data stream, in this case the DMA will run through filling DMA buffers in sequence for as long as there are descriptors available.

The receive filtering options are described in Section: “Packet FIFO Interfaces” on page 291. If an unframed data stream is being received software must ensure that the mac_chup and mac_chlo tables select the same DMA channel at all indices.

Packets transmitted in any of the modes can have a CRC-32 appended by setting the append CRC bit in the transmit packet descriptor. Packets received have their final four bytes checked for a valid CRC-32 and if the bypass_fcs_chk bit is set in the mac_cfg register the status bit in the receive descriptor will be set accordingly (if the bypass_fcs_chk bit is clear the status word will never flag a CRC error). If the CRC-32 is used on the receive side there must be at least seven clock cycles between the end of packet and the start of the next packet. If this inter-frame gap is not provided then the CRC checker will give UNPREDICTABLE results.
There are four formats that are available for the 8 bit Packet FIFO to signal data validity and packet boundaries. In all cases data and control signals are sampled on the rising edge of the clock. On the transmit side the GMII TXEN (enable) and TXER (error) signals are used as two control signals that are used to send out the framing information, and on receive the RXDV (data valid) and RXER (error) pins are used to receive the framing information.

**8-BIT GMII STYLE PACKET FIFO**

The GMII style of Packet FIFO mode is shown in Figure 60 on page 294. The packet data is framed by the TXEN or RXDV signal. The first byte that has it active is marked as the start of a packet and the last byte that has it active is the end of the packet, all bytes between are valid and part of the packet. The TXER or RXER signal can be used to signal an error whenever the frame signal is active.

![Figure 60: 8-bit Packet FIFO GMII Style](image)

The error pin TXER/RXER may be used to signal that a packet has an error. It should be asserted when the error is detected and remain high until the enable signal falls.

<table>
<thead>
<tr>
<th>Table 170: Codes for GMII Packet FIFO Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXEN / RXDV</td>
</tr>
<tr>
<td>Valid Data, Start of Packet</td>
</tr>
<tr>
<td>Valid Data</td>
</tr>
<tr>
<td>Valid Data, End of Packet</td>
</tr>
<tr>
<td>Error</td>
</tr>
</tbody>
</table>
8-BIT ENCODED PACKET FIFO

Encoded mode, shown in Figure 61, uses the control lines to signal the four states that can be associated with the data. This mode should be used for packet data where there can be invalid bytes between the start and end of packet marks. Table shows the encodings used.

![Figure 61: 8-Bit Packet FIFO Encoded Style](image)

Table 171: Codes for 8-Bit Encoded Bypass Mode

<table>
<thead>
<tr>
<th></th>
<th>TXEN / RXDV</th>
<th>TXER / RXER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Data, Start of Packet</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Valid Data</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Valid Data, End of Packet</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Data Not Valid</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In this mode flow control is provided in each direction. Transmit flow control is input on the COL pin (normally the collision sense from the PHY), when this is asserted the interface will stop sending data (and drive the data not valid control code) two rising clock edges later. The interface will assert the MDIO pin (normally the management interface to the PHY) as a receive flow control output if (a) the CPU sets the force flow control configuration bit; (b) the number of descriptors on a receive channel falls below the low watermark and automatic flow control is enabled for that channel; or (c) there are less than 4 64 bit words left free in the receive FIFO. If the device the other end of the link honours the flow control request within a few clock times data will never be lost on the link.

In this mode a transmit FIFO underflow during transmission of a packet causes bubbles to be inserted in the data stream (the data not valid code will appear during a packet) but transmission will not be aborted. (In all other modes a transmit FIFO underrun causes the packet to be aborted).
8-BIT SOP FLAGGED PACKET FIFO

SOP flagged Packet FIFO mode uses one control line as a data valid signal and the other to flag start of packet. The end of packet is one cycle before the SOP or whenever the data goes invalid (this extra EOP is required to push out the last data at the end of a valid sequence).

![Figure 62: 8-Bit Packet FIFO SOP Style](image)

<table>
<thead>
<tr>
<th>TXEN / RXDV</th>
<th>TXER / RXER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid Data, Start of Packet</td>
<td>1</td>
</tr>
<tr>
<td>Valid Data</td>
<td>1</td>
</tr>
<tr>
<td>Valid Data, End of Packet</td>
<td>1</td>
</tr>
<tr>
<td>Data Not Valid</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 172: Codes for 8-Bit SOP Packet FIFO
8-BIT EOP FLAGGED PACKET FIFO

EOP flagged Packet FIFO mode uses one control line as a data valid signal and the other to flag end of packet. The start of packet is one cycle after an end of packet, or the first data after the valid signal has risen. Any SOPs that occur between an SOP and EOP are ignored, so in Figure 63 there is not an SOP flagged in the cycle marked X, one cycle after TXEN/RXDV was low. This mode is good for streaming data that uses only the data valid signal. If the EOP flag is always low the interface will transfer a single never-ending packet for as long as DMA buffers are provided.

![8-Bit Packet FIFO EOP Style](image)

**Table 173: Codes for 8-Bit EOP Bypass Mode**

<table>
<thead>
<tr>
<th>Valid Data, Start of Packet</th>
<th>TXEN / RXDV</th>
<th>TXER / RXER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-&gt;1 at start of cycle</td>
<td>1</td>
<td>1 previous cycle</td>
</tr>
<tr>
<td>Valid Data</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Valid Data, End of Packet</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Data Not Valid</td>
<td>0</td>
<td>anything</td>
</tr>
</tbody>
</table>
16-BIT PACKET FIFO OPERATION

The 16-bit Packet FIFO mode provides an interface that can run bidirectionally at OC-48 data rates. It can externally be converted into a POS-PHY level 3 interface using a simple gearbox PLD or FPGA. Alternatively, it may be connected directly to an ASIC.

On the BCM1250, there are two Packet FIFO interfaces. On the BCM1125/H there is one. Interface F0 (F on the BCM1125/H) replaces the Ethernet MACs E0 and E1, and reuses pins from both. It uses the DMA controller associated with interface E0. On the BCM1250 Packet interface F1 reuses pins from all three Ethernet MACs (E0, E1 and E2) and uses the DMA engine associated with E2. It is possible to run the F0 packet interface along with the E2 Ethernet interface.

The receive filtering options are described in section Section: “Packet FIFO Interfaces” on page 291. If an unframed data stream is being received software must ensure that the mac_chup and mac_chlo tables select the same DMA channel at all indices.

Packets transmitted in any of the Packet FIFO modes can have a CRC-32 appended by setting the append CRC bit in the transmit packet descriptor. Packets received have their final four bytes checked for a valid CRC-32 and if the bypass_fcs_chk bit is set in the mac_cfg register the status bit in the receive descriptor will be set accordingly (if the bypass_fcs_chk bit is clear the status word will never flag a CRC error). If the CRC-32 is used on the receive side there must be at least three clock cycles between the end of packet and the start of the next packet. If this inter-frame gap is not provided then the CRC checker will give UNPREDICTABLE results.

The 16 bit Packet FIFO mode uses three control signals that accompany the data and, in encoded mode, a flow control signal in the reverse direction. The 16 bit wide data path is treated as two bytes where the data on bits [7:0] is earlier in the packet (and therefore will be DMAed to/from a lower memory address) than the data on [15:8]. The last 16 bit half-word of a packet that is an odd number of bytes long will have valid data on bits [7:0] and an UNPREDICTABLE value on bits [15:8].

16-BIT GMII STYLE PACKET FIFO

The 16-bit GMII style of Packet FIFO interface is shown in Figure 64. The packet data is framed by the TXC[0] or RXC[0] signal. The first byte that has it active is marked as the start of a packet and the last byte that has it active is the end of the packet, all bytes between are valid and part of the packet. The TXC[1] or RXC[1] signal can be used to signal an error whenever the frame signal is active, it should be asserted when the error is detected and held high until the frame signal falls.

---

Figure 64: 16-Bit GMII Style Packet FIFO
TXC/RXC[2] is used to indicate the validity of the upper 8-bits of data. It should be the same as TXC/RXC[0] except for the final data in a packet that has an odd number of bytes.

Table 174: Codes for 16-Bit GMII Style Packet FIFO

<table>
<thead>
<tr>
<th>TXC/RXC[2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Not Valid</td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td>Start of Packet, 2 Bytes Valid</td>
</tr>
<tr>
<td>000-&gt;101 at start of cycle</td>
</tr>
<tr>
<td>Middle of Packet, 2 Bytes Valid</td>
</tr>
<tr>
<td>101</td>
</tr>
<tr>
<td>End of Packet, 1 Byte Valid</td>
</tr>
<tr>
<td>001-&gt;000 at end of cycle</td>
</tr>
<tr>
<td>End of Packet, 2 Bytes Valid</td>
</tr>
<tr>
<td>101-&gt;000 at end of cycle</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>111, bit [1] held set until 000</td>
</tr>
</tbody>
</table>

16-BIT ENCODED PACKET FIFO

The 16-bit encoded Packet FIFO mode is shown in Figure 65. The TXC[2:0] signals indicate the status of the data they accompany. The encoding is chosen to allow easy conversion to and from POS-PHY Level 3 signalling, which is a wider slower packet bus. The flow control signal runs in the opposite direction to the data. Figure 65 shows two packets being sent. In the first the sender is unable to supply data for one cycle and removed the valid signal. In the second packet the receiver asserts the flow control signal for a cycle requesting a pause from the sender. The transmitter will see the flow control request on the next rising edge of the clock and will send data for 2-4 cycles before suspending transmission one cycle later. The interface will assert its flow control output when receive channel flow control is invoked either by one of the DMA channels having fewer descriptors than the low watermark or by an explicit processor request.

Figure 65: 16-Bit Encoded Packet FIFO

Table 175: Codes for 16-Bit Encoded Bypass Mode

<table>
<thead>
<tr>
<th>TXC/RXC[2:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Not Valid</td>
</tr>
<tr>
<td>000</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>001</td>
</tr>
<tr>
<td>End of Packet, 1 Byte Valid</td>
</tr>
<tr>
<td>010</td>
</tr>
<tr>
<td>End of Packet With Error</td>
</tr>
<tr>
<td>011</td>
</tr>
<tr>
<td>Reserved</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>
In this mode flow control is provided in each direction. Transmit flow control is input on the TX_FC pin, when this is asserted the interface will stop sending data (and drive the data not valid control code) two rising clock edges later. The part will assert the RX_FC pin as a receive flow control output if (a) the CPU sets the force flow control configuration bit; (b) the number of descriptors on a receive channel falls below the low watermark and automatic flow control is enabled for that channel; or (c) there are less than four 64 bit words left free in the receive FIFO. If the device the other end of the link honours the flow control request within a few clock times data will never be lost on the link.

In this mode a transmit FIFO underflow during transmission of a packet causes bubbles to be inserted in the data stream (the data not valid code will appear during a packet) but transmission will not be aborted. (In all other modes a transmit FIFO underrun causes the packet to be aborted).

**Restrictions When Resetting the Interface**

The MAC and Packet Fifo logic is automatically reset when the part is reset. However, on BCM1250s prior to PERIPH_REV3, correct reset of the logic requires that there be a few rising edges on REFCLK01 (for MACs E0 and E1) and REFCLK2 (for MAC E2) before software releases the interface from reset. This is not normally a problem since the clocks will be free running. If the clocks are not provided then the first access to the registers for the corresponding interface will place the ZBbus in an UNDEFINED state. If an interface is not used there are two options: (1) ensure software never accesses the unused interface registers; (2) provide some edges on the reference clock before software can access the registers. Clearly (2) is preferable, since otherwise a bug in the software (or even a debugger collecting the system state) could cause a hang that would be hard to track down. There is no need to provide a periodic clock. One possible solution would be to connect the unused REFCLK input to the boot ROM chip select (or SCL if the system boots from SMBus) since this is guaranteed to toggle when the part comes out of reset (before any instructions run the SB-1 will have fetched 16 bytes from the boot ROM, thus there will be 4 (for a 32 bit ROM) or 16 (for an 8 bit ROM) edges on the REFCLK even in the unlikely case where the first instruction accesses the MAC registers).

The interface can also be reset by software. Again, for parts prior to PERIPH_REV3 the clock needs to be running for correct operation (note that if the clock was running during the chip reset and the MAC is not being used then it is safe to perform a software reset without a clock). This can be done by setting the (self-clearing) p_reset bit in the mac_enable register. After this bit has been written with a 1, the software should delay for at least six times the clock period of the receive clock then write the bit with a 1 again. Following the second write the interface will be fully reset. (If this sequence is not followed the interface will reset but it is possible for a junk packet to be received when the receiver is enabled again). Before software resets the MAC (or a DMA channel) it should ensure that there is no outstanding DMA activity (failure to do this can cause resources to be lost between the MAC and ZBbus and can degrade performance). The DMA should be stopped by clearing the descriptor count by writing the complement of the current count into the count register (the hardware will add this value and discard the carry so the count will become zero), then reading any of the DMA registers, then waiting for twice the transmission time of a maximum length packet (this ensures any in-progress transmission or reception has completed). Once the DMA has stopped the reset can be safely written.

**Table 175: Codes for 16-Bit Encoded Bypass Mode**

<table>
<thead>
<tr>
<th>TXC/RXC[2:0]</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Start of Packet, 2 Bytes Valid</td>
</tr>
<tr>
<td>110</td>
<td>End of Packet, 2 Bytes Valid</td>
</tr>
<tr>
<td>111</td>
<td>Middle of Packet, 2 Bytes Valid</td>
</tr>
</tbody>
</table>
MAC REGISTERS

Note: All of the MAC registers need to be initialized by software before it will function. The BCM1125/H has MACs 0 and 1. Accesses made to the address range allocated to MAC 2 may cause all MACs to exhibit UNPREDICTABLE behavior.

Table 176: MAC Configuration Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ss_tmode</td>
<td>1'b0</td>
<td>This bit must always be zero for normal operation (Broadcom Use Only test bit).</td>
<td>Ethernet and Packet FIFO modes</td>
</tr>
<tr>
<td>1</td>
<td>tx_hold_sop_en</td>
<td>1'b0</td>
<td>When this bit is set the interface will retain the first few bytes of each transmitted packet in the transmit FIFO. This enables the interface to automatically retry any packet that encounters an error during the initial part of its transmission. The tx_rl_thrsh parameter (in the mac_threshold register) sets the number of 64 bit FIFO entries that are retained, once this number of entries have been successfully transmitted the FIFO is unlocked and no data is retained. For example if tx_hold_sop_en is set and tx_rl_thrsh is set to 8, then if the MAC detects a collision before 8 entries (64 bytes) are read from the transmit FIFO, the FIFO can be restored the start of the packet and the packet can be automatically retried. Automatic retry can be triggered by any error that happens (collision, underrun error, excessive collision error, or late collision) while the FIFO is in locked state. (i.e. less than the number of entries set by tx_rl_thrsh have been read from the transmit FIFO for a given packet.) The retry_en, ret_drpreq_en and ret_ufl_en bits enable automatic retry selectively for each of the error types. If the tx_hold_sop_en bit is clear or the number of entries set by tx_rl_thrsh have been read from the FIFO then the FIFO will be unlocked and automatic retry is not possible. In this case the packet with the error will be dropped.</td>
<td>Ethernet and Packet FIFO modes</td>
</tr>
<tr>
<td>2</td>
<td>retry_en</td>
<td>1'b0</td>
<td>If this bit is set then automatic retry is enabled when collisions are detected (but not late or excessive collisions). The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</td>
<td>Ethernet and Packet FIFO modes</td>
</tr>
<tr>
<td>3</td>
<td>ret_drpreq_en</td>
<td>1'b0</td>
<td>If this bit is set then automatic retry is enabled when late collisions or excessive collisions are detected. The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</td>
<td>Ethernet and Packet FIFO modes</td>
</tr>
<tr>
<td>4</td>
<td>ret_ufl_en</td>
<td>1'b0</td>
<td>If this bit is set then automatic retry is enabled when a transmit FIFO underflow is detected. The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</td>
<td>Ethernet and Packet FIFO modes</td>
</tr>
<tr>
<td>5</td>
<td>burst_en</td>
<td>1'b0</td>
<td>This bit should be set to enable burst operation when the MAC is running in 1000 Mbp/s mode. Burst mode enables transmission of multiple packets without releasing the network between them.</td>
<td>Ethernet mode only</td>
</tr>
</tbody>
</table>
Table 176: MAC Configuration Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:6</td>
<td>tx_pause_cnt</td>
<td>3'b0</td>
<td>Transmit pause count. This field sets the number of slot times pause that will be requested when the interface transmits a pause frame. It is also used to initialize the local pause counter which indicates when the next flow control packet needs to be sent. The encodings are: 00: 512 001: 1K 010: 2K 011: 4K 100: 8K 101: 16K 110: 32K 111: 64K Used in Ethernet mode only.</td>
</tr>
<tr>
<td>16:9</td>
<td>reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>ap_stat_en</td>
<td>1'b0</td>
<td>Append packet status enable. When this bit is set the packet status is written to the receive FIFO following the last entry in the packet. This bit should be set for Ethernet mode and clear for Packet FIFO mode. Used in Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>18</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>drp_errpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then automatic dropping is enabled for all error packets (bits [31:26] are ignored). Used in Ethernet mode only.</td>
</tr>
<tr>
<td>20</td>
<td>drp_fcserrpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then packets with an FCS (CRC) error are automatically dropped. Used in Ethernet mode only.</td>
</tr>
<tr>
<td>21</td>
<td>drp_codeerrpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then packets that are signalled with a code error by the PHY are automatically dropped. Used in Ethernet mode only.</td>
</tr>
<tr>
<td>22</td>
<td>drp_drberrpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then packets with a dribble error are automatically dropped. Used in Ethernet mode only.</td>
</tr>
<tr>
<td>23</td>
<td>drp_rntpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then runt packets (those shorter than the minimum packet size configured in the mac_frame_cfg register) are automatically dropped. Used in Ethernet mode only.</td>
</tr>
<tr>
<td>24</td>
<td>drp_oszpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then oversize packets (those longer than the maximum packet size configured in the mac_frame_cfg register are automatically dropped. (Unless a non-standard protocol is used the maximum packet size will always be larger than the receive FIFO, so this is not likely to happen in practice.) Used in Ethernet mode only.</td>
</tr>
<tr>
<td>25</td>
<td>drp_lenerrpkt_en</td>
<td>1'b0</td>
<td>If this bit is set then packets with a length error are automatically dropped. A length error is signalled when the length/type field of the packet is less than or equal to 1500 (indicating it is being used as the length) and its value does not match the packet length. Used in Ethernet mode only.</td>
</tr>
<tr>
<td>31:26</td>
<td>reserved</td>
<td>6'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

This register is used in both Ethernet and Packet FIFO modes.

This register is used in both Ethernet and Packet FIFO modes.

Busy controller. This controller is used in both Ethernet and Packet FIFO modes.

Bits
Name
Default
Description
32 bypass_sel 1'b0 When this bit is set the MAC is bypassed and Packet FIFO mode is enabled. This provides a simple FIFO interface, backed by the DMA engine. See Section: “8-Bit Packet FIFO Operation” on page 293 for details.

Used in Ethernet mode only.

33 hdx_en 1'b0 When this bit is set the MAC will operate in half-duplex mode, when the bit is clear it will run full-duplex. The MAC supports both full and half-duplex operation at all speeds.

Used in Ethernet mode only.

35:34 speed_sel 2'b0 This field sets the speed the MAC will operate.
- 00: 10 Mbps - the TCLK input should be 2.5 MHz.
- 01: 100 Mbps - the TCLK input should be 25 MHz.
- 10: 1000 Mbps - the transmit clock will be supplied by the interface. The reference clock input should be 125 MHz.
- 11: Reserved

Used in Ethernet mode only. Must be set to 10 for Packet FIFO modes.

36 tx_clk_edge 1'b0 This selects the edge of the transmit clock that is used to send the data. Normally the rising clock edge should be used. The falling edge can be used to support old 10 Mbps PHY chips that require the full hold time specified in the original IEEE specification.
- 0: The rising edge of the clock is the active one.
- 1: The falling edge of the clock is the active one.

Used in Ethernet and Packet FIFO modes.

37 loopback_sel 1'b0 When this bit is set an internal loopback path is enabled, connecting the transmitter and receiver GMII pins.

Used in Ethernet and 8 bit Packet FIFO modes.

38 fast_sync 1'b0 This bit must be set for normal operation. (It defaults to 0 so software should be sure to set it before starting the MAC.)

Used in Ethernet and Packet FIFO modes.

39 ss_en 1'b0 This bit must always be set. (Broadcom Use Only debug bit). (It defaults to 0 so software should be sure to set it before starting the MAC.)

Used in Ethernet and Packet FIFO modes.

41:40 bypass_CFG 2'b0 This field sets the strobe signals that are used on both transmit and receive interfaces in Packet FIFO mode.
- 00: GMII style
- 01: Encoded
- 10: SOP flagged (not valid in 16 bit Packet FIFO mode)
- 11: EOP flagged (not valid in 16 bit Packet FIFO mode)

Used in Packet FIFO modes.

42 bypass_16 1'b0 Set for 16 bit Packet FIFO mode, clear for 8 bit Packet FIFO mode. The interface behavior will become UNDEFINED if this bit is set for MAC 1. Setting this bit for MAC 0 will cause the pins to change from their eight bit E0 and E1 configuration to the 16 bit F0 configuration.
Setting this bit for MAC 2 will cause the pins to change from their eight bit E0, E1 and E2 configuration to the 16 bit F1 configuration.
This bit must be set to reflect the external connection.

Used in Packet FIFO modes.
### Table 176: MAC Configuration Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>bypass_fcs_chk</td>
<td>1'b0</td>
<td>If this bit is set then the interface will perform the FCS check on packets received in Packet FIFO mode. The last four bytes are checked for a valid CRC-32 using the same algorithm as for Ethernet. The status will be recorded in the receive DMA descriptor status field. If this bit is clear no FCS check is done and all packets will be marked as having a correct CRC.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Packet FIFO modes.</td>
</tr>
<tr>
<td>44</td>
<td>rx_ch_sel_msb</td>
<td>1'b0</td>
<td>This is the top bit of rx_ch_sel (see bits 63:57).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>45</td>
<td>split_ch_sel</td>
<td>1'b0</td>
<td>System revision &lt; PERIPH_REV3: Reserved System revision PERIPH_REV3 or later: If this bit is set the 8 bit index used to select the receive channel number is made up from nibbles extracted from two different places in the packet, (rx_ch_sel_msb, rx_ch_sel) specify the offset of the lower 4 bits of the index, rx_ch_msn_sel (in the mac_adfilter register) specify the upper 4 bits. If split_ch_sel is zero then (rx_ch_sel_msb, rx_ch_sel) is used to select the low 4 bits of the index and the upper 4 bits is always the next nibble.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>53:46</td>
<td>bypass_ifg</td>
<td>8'b0</td>
<td>This field gives the number of clock cycles of inter-frame gap that is inserted in Packet FIFO mode. If the transmitter is set to append a CRC then this field has a minimum value of 3 in 16 bit FIFO mode and a minimum value of 7 in 8 bit FIFO mode. If CRCs are never inserted then packets may be transmitted back to back.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Packet FIFO modes.</td>
</tr>
<tr>
<td>54</td>
<td>fc_sel</td>
<td>1'b0</td>
<td>This bit is used by software to force flow control. If it is set then flow control will be asserted on the link (Pause frames in full duplex Ethernet, backpressure by the method encoded in the fc_cmd bits for half duplex Ethernet, or the link level flow control pin RXFC for encoded Packet Fifo mode). Flow control will remain asserted until this bit is cleared. Note that if this bit is set while the mac is disabled the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>56:55</td>
<td>fc_cmd</td>
<td>2'b0</td>
<td>This field sets the flow control style to be used for both software flow control (set by bit 54) and automatic flow control from the receive DMA channels. This field is only used for Half Duplex links. 00: Disabled 01: Enabled using collisions 10: Enabled using false carrier 11: UNPREDICTABLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Ethernet mode only.</td>
</tr>
<tr>
<td>63:57</td>
<td>rx_ch_sel</td>
<td>7'b0</td>
<td>This field along with the rx_ch_sel_msb bit sets the offset into received packets to extract 8 bits to select the receive DMA channel. This offset is in nibbles. If bit [57] is zero then bits [44, 63:58] give the byte of the packet (starting from zero) that will be used to index the channel select table. If bit [57] is one then the top four bits of the byte indexed by [44, 63:58] will be used as the low four bits to index the channel select table, and the lower four bits from the next byte of the packet will be used as the upper four bits to index the channel select table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in Ethernet and Packet FIFO modes.</td>
</tr>
</tbody>
</table>
### Table 177: MAC Enable Registers

This register is used in both Ethernet and Packet FIFO modes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>rxdma_en</td>
<td>2'b0</td>
<td>Receive DMA channel enable. Must be set to enable a channel. Bit 0 corresponds to channel 0, bit 1 to channel 1.</td>
</tr>
<tr>
<td>3:2</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>5:4</td>
<td>txdma_en</td>
<td>2'b0</td>
<td>Transmit DMA channel enable. Must be set to enable a channel. Bit 4 corresponds to channel 0, bit 5 to channel 1.</td>
</tr>
<tr>
<td>7:6</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>p_reset</td>
<td>1'b0</td>
<td>Port reset. Writing a 1 to this bit will reset the entire MAC and Packet FIFO interface as though it had come out of a system reset. All registers are restored to their default values thus disabling and resetting the DMA engines, Ethernet mode logic and Packet FIFO logic. The default for this bit is zero, so it will self-clear as part of the reset sequence.</td>
</tr>
<tr>
<td>9</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>mac_rx_en</td>
<td>1'b0</td>
<td>If this bit is clear the receive Ethernet media access engine will be held in reset. If this bit is set it will run normally.</td>
</tr>
<tr>
<td>11</td>
<td>mac_tx_en</td>
<td>1'b0</td>
<td>If this bit is clear the transmit Ethernet media access engine will be held in reset. If this bit is set it will run normally.</td>
</tr>
<tr>
<td>12</td>
<td>byp_rx_en</td>
<td>1'b0</td>
<td>If this bit is clear the receive Packet FIFO engine will be held in reset. If this bit is set it will run normally.</td>
</tr>
<tr>
<td>13</td>
<td>byp_tx_en</td>
<td>1'b0</td>
<td>If this bit is clear the transmit Packet FIFO engine will be held in reset. If this bit is set it will run normally.</td>
</tr>
<tr>
<td>63:14</td>
<td>reserved</td>
<td>50'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 178: MAC Transmit DMA Control Register

This register is used in both Ethernet and Packet FIFO modes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>weight_ch0</td>
<td>4'b0</td>
<td>Transmit channel 0 round robin weight. Note that if both channels are enabled with weight zero (the default) then no packets will be transmitted.</td>
</tr>
<tr>
<td>7:4</td>
<td>weight_ch1</td>
<td>4'b0</td>
<td>Transmit channel 1 round robin weight. See note above.</td>
</tr>
<tr>
<td>63:8</td>
<td>reserved</td>
<td>56'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 179: MAC FIFO Threshold Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:0</td>
<td>tx_wr_thrsh</td>
<td>7'b0</td>
<td>Transmit FIFO write threshold. Sets the number of free 64 bit entries the transmit FIFO must have before it signals that space is available. For DMA operation this field must be set to 4 or 8 entries depending on the state of the tbx_en bit in the dma_config0 register. See Section: “Transmitter Configuration” on page 271.</td>
</tr>
<tr>
<td>7</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>14:8</td>
<td>tx_rd_thrsh</td>
<td>7'b0</td>
<td>Transmit FIFO read threshold. Sets the number of valid 64 bit entries the transmit FIFO must hold before the MAC will start transmitting the packet. See Section: “Transmitter Configuration” on page 271. Note that the tx_wr_thrsh + tx_rd_thrsh must be less than the size of the fifo or the transmitter behaviour is UNPREDICTABLE (tx_wr_thrsh + tx_rd_thrsh &lt;= 32 for parts with System Revision of 1, tx_wr_thrsh + tx_rd_thrsh &lt;=128 for parts with revision greater than 2).</td>
</tr>
<tr>
<td>15</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>21:16</td>
<td>tx_rl_thrsh</td>
<td>6'b0</td>
<td>Transmit FIFO release count. This sets the number of 64 bit FIFO entries that will be held in the FIFO when the MAC is configured to hold the start of packets. See the tx_hold_sop_en bit in the mac_cfg register. See Section: “Transmitter Configuration” on page 271.</td>
</tr>
<tr>
<td>23:22</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>29:24</td>
<td>reserved</td>
<td>6'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>31:30</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>37:32</td>
<td>rx_rd_thrsh</td>
<td>6'b0</td>
<td>Receive read threshold. This field sets the number of entries that must be in the receive FIFO for it to indicate data is available to be read. For DMA operation this field must be set to 4 entries. See Section: “Receiver Configuration” on page 274.</td>
</tr>
<tr>
<td>39:38</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>45:40</td>
<td>rx_rl_thrsh</td>
<td>6'b0</td>
<td>Receive release threshold. This field sets the number of FIFO entries that must be written at the start of a packet before any of the data is made available for reading. The data is also made available if the packet ends before the threshold is reached. See Section: “Receiver Configuration” on page 274. This field must be greater than 2. If it is zero the MAC receive behavior will be UNPREDICTABLE.</td>
</tr>
<tr>
<td>55:46</td>
<td>reserved</td>
<td>10'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>61:56</td>
<td>enc_fc_thrsh</td>
<td>6'h4</td>
<td>In encoded Packt Fifo modes link level flow control will be requested when the number of free doublewords in the receive fifo falls below this threshold.</td>
</tr>
<tr>
<td>63:62</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
### Table 180: MAC Frame Configuration Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:0</td>
<td>ifg_rx</td>
<td>6'b0</td>
<td>System Revision &lt; PERIPH_REV3: The ifg_rx is not used in the current implementation. The ifg_tx value is used to set both receive to transmit (on half-duplex links) and transmit to transmit gap.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>System Revision PERIPH_REV3 and later: If bit [5] is clear a standard 7 byte preamble is transmitted. If bit [5] is set bits [3:0] set the number of bytes of preamble that will be transmitted ([3:0]=0 will result in one byte of preamble in gigabit mode, no preamble in 10/100 mode). The single SFD byte is always sent after the preamble.</td>
</tr>
<tr>
<td>11:6</td>
<td>ifg_tx</td>
<td>6'b0</td>
<td>ifg_tx determines inter-frame gap in interface cycles that the MAC uses when doing back to back transmissions (or between a reception and transmission in a half-duplex link). When the MAC has completed a packet the ifg counter loaded with the ifg_tx value and counts down. When the ifg counter reaches zero and four more cycles have passed the inter-frame gap time has elapsed and the MAC is free to transmit. If the value written in this register is less than 4 then the gap will be 7 cycles, otherwise the gap will be ifg_tx+4 cycles (See Figure 39). A value of zero gives 68 cycles. For Gigabit operation this is in bytes, and for 10/100 in nibbles. Including the offset of 4 for a standard 12 byte gap this field should be set to 8 for Gigabit operation and 20 for 10/100. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td>17:12</td>
<td>ifg_thrsh</td>
<td>6'b0</td>
<td>This sets the threshold time in interface cycles for checking carrier sense during the inter-frame gap. At the start of the IFG the MAC will monitor the carrier sense and defer if a carrier is detected, the IEEE standard suggests this should last 2/3 of the gap. During the remainder of the IFG the MAC ignores the carrier sense and transmission (potentially with a collision) will begin as usual. The carrier is ignored for the final ifg_thrsh+4 cycles of the IFG. To use the standard 2/3rd:1/3rd split for Gigabit operation the MAC should defer during the first 8 byte times, thus ifg_thrsh+4=4 and this field should be set to zero. For 10/100 operation the 8 byte times become 16 nibbles to defer, thus ifg_thrsh+4=8 and this field should be set to 4. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td>21:18</td>
<td>backoff_sel</td>
<td>4'b0</td>
<td>This field sets the maximum number of slot times the MAC will backoff following a collision. If a collision happens when the MAC is transmitting a packet it will abandon the transmission attempt, and backoff for a pseudo-random period before retrying. The length of the backoff is chosen randomly by a LFSR from a range from zero to a maximum value. The maximum backoff period depends on the number of times the packet has previously collided (to give an exponential backoff) and the value of this field. This field sets the maximum value the backoff count can grow to, regardless of the number of times the packet has collided. The encoding of the backoff time is: backoff_sel = 0 - Backoff disabled, the backoff time will always be zero 1 &lt;= backoff_sel &lt;= 10 - Backoff limit is 0 - (2^n - 1) 11 &lt;= backoff_sel &lt;= 15 - Backoff limit is 0 - (2^10 - 1) This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE. This field should be set to 10 for standard operation.</td>
</tr>
</tbody>
</table>
### Table 180: MAC Frame Configuration Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>29:22</td>
<td>lfsr_seed</td>
<td>8'b0</td>
<td>This field is the partial seed value for the LFSR that generates the random backoff. When the register is written this value is loaded into the 8 lowest bits of the LFSR (the upper 2 bits are loaded with the LFSR output). This value may be changed at any time.</td>
</tr>
<tr>
<td>39:30</td>
<td>slot_offset</td>
<td>10'b0</td>
<td>This field adjusts the slot size for the MAC to match for delays in the PHY. It is a signed value added to the base slot size. For 10/100 operation the slot size is (128 + slot_offset) nibble-clocks, giving the standard 512 bit times when the slot_offset is zero. For 1000 operation the slot size is (512 + slot_offset) byte-clocks, giving the standard 512 byte times when slot_offset is zero. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</td>
</tr>
<tr>
<td>47:40</td>
<td>min_framesz</td>
<td>8'b0</td>
<td>This field sets the minimum frame size in bytes that is used. The MAC will report a runt packet error for any packet that is received that is smaller than this number of bytes (after preamble and SFD have been stripped). For IEEE 802.3 compliance this value should be 64 for all speeds of operation. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE. This size must be greater than 9.</td>
</tr>
<tr>
<td>63:48</td>
<td>max_framesz</td>
<td>16'b0</td>
<td>This field sets the maximum frame size in bytes that is used. If a packet is received that is longer than this (after preamble and SFD have been stripped) the MAC will report an oversize packet error and discard excess bytes. For IEEE 802.3 compliance this value should be 1518 for all speeds of operation. Setting this size above 1518 allows use of “jumbo” packets. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE. This size must be greater than the min_framesz.</td>
</tr>
</tbody>
</table>
Table 181: MAC VLAN Tag Registers

This register is used in both Ethernet and Packet FIFO modes

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mac_vlantag_0</td>
<td>00_1006_4110</td>
<td>VLAN tag. This 32 bit tag is inserted into the packet after the destination and source Ethernet addresses if the packet is marked for VLAN tag insertion. The tag thus becomes bytes 12-15 of the packet. The low byte in this register is the first one put on the wire. Note that for normal VLAN operation the low two bytes should be 16'h8100. Used only in Ethernet mode.</td>
</tr>
<tr>
<td></td>
<td>mac_vlantag_1</td>
<td>00_1006_5110</td>
<td></td>
</tr>
<tr>
<td></td>
<td>mac_vlantag_2</td>
<td>00_1006_6110</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>tag</td>
<td>32'b0</td>
<td></td>
</tr>
</tbody>
</table>

39:32  | tx_pkt_offset | 8'b0 | System Revision PERIPH_REV3 and later only. Sets the offset the Ethernet frame in transmitted packets. Bits 34:32 MUST be 3'b000. Used in both Ethernet and Packet FIFO modes. |

47:40  | tx_crc_offset | 8'b0 | System Revision PERIPH_REV3 and later only. Sets the offset for CRC generation to begin in transmitted packets. Bits 42:40 MUST be 3'b000. Used in both Ethernet and Packet FIFO modes. |

48     | ch_base_fc_en | 1'b0 | System Revision PERIPH_REV3 and later only. If this bit is clear Pause frame flow control is done in the MAC, if set the flow control is done by the DMA engine. Used in both Ethernet and Packet FIFO modes. |

63:49  | notimp        | 32'b0 | Not Implemented.                                                             |

Table 182: MAC Status Registers

This register is used in both Ethernet and Packet FIFO modes

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rx_ch0_eop_count</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>1</td>
<td>rx_ch0_eop_timer</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
<tr>
<td>2</td>
<td>rx_ch0_eop_seen</td>
<td>1'b0</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>3</td>
<td>rx_ch0_hwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched nor cleared by a read of the status register. It always reflects the state at the time the register is read. Therefore it is possible that the bit may become set and cause an interrupt but be cleared by an in-flight write that adds descriptors before the interrupt is taken.</td>
</tr>
<tr>
<td>4</td>
<td>rx_ch0_lwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>5</td>
<td>rx_ch0_dscr</td>
<td>1'b0</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt bit set.</td>
</tr>
</tbody>
</table>
Table 182: MAC Status Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>rx_ch0_derr</td>
<td>1'b0</td>
<td>Set if any error is marked on data returned by a read. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>7</td>
<td>rx_ch0_drop</td>
<td>1'b0</td>
<td>Set if a packet was dropped because there were no descriptors available when the start of packet was received. (If there are descriptors available for the start of packet, but the controller runs out of descriptors during the packet reception then the dscr_err bit will be set in the receive status word for the packet.) The interface will continue dropping data until there are descriptors available when a start of packet is received.</td>
</tr>
<tr>
<td>8</td>
<td>rx_ch1_eop_count</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>9</td>
<td>rx_ch1_eop_timer</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
<tr>
<td>10</td>
<td>rx_ch1_eop_seen</td>
<td>1'b0</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>11</td>
<td>rx_ch1_hwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>12</td>
<td>rx_ch1_lwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>13</td>
<td>rx_ch1_dscr</td>
<td>1'b0</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt bit set.</td>
</tr>
<tr>
<td>14</td>
<td>rx_ch1_derr</td>
<td>1'b0</td>
<td>Set if any error is marked on data returned by a read. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>15</td>
<td>rx_ch1_drop</td>
<td>1'b0</td>
<td>Set if a packet was dropped because there were no descriptors available when the start of packet was received. (If there are descriptors available for the start of packet, but the controller runs out of descriptors during the packet reception then the dscr_err bit will be set in the receive status word for the packet.) The interface will continue dropping data until there are descriptors available when a start of packet is received.</td>
</tr>
<tr>
<td>16</td>
<td>tx_ch0_eop_count</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>17</td>
<td>tx_ch0_eop_timer</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
<tr>
<td>18</td>
<td>tx_ch0_eop_seen</td>
<td>1'b0</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>19</td>
<td>tx_ch0_hwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>20</td>
<td>tx_ch0_lwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>21</td>
<td>tx_ch0_dscr</td>
<td>1'b0</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt bit set.</td>
</tr>
<tr>
<td>22</td>
<td>tx_ch0_derr</td>
<td>1'b0</td>
<td>Set if data marked with an error is returned for any read (descriptor or data buffer). The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>23</td>
<td>tx_ch0_dzero</td>
<td>1'b0</td>
<td>Set if a descriptor has a packet length of zero or the SOP flag bit is not set in the first descriptor of a packet. This bit is also set if the controller runs out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>24</td>
<td>tx_ch1_eop_count</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>25</td>
<td>tx_ch1_eop_timer</td>
<td>1'b0</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
<tr>
<td>Bits</td>
<td>Name</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>26</td>
<td>tx_ch1_eop_seen</td>
<td>1'b0</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>27</td>
<td>tx_ch1_hwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>28</td>
<td>tx_ch1_lwm</td>
<td>1'b0</td>
<td>This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).</td>
</tr>
<tr>
<td>29</td>
<td>tx_ch1_dscr</td>
<td>1'b0</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt bit set.</td>
</tr>
<tr>
<td>30</td>
<td>tx_ch1_derr</td>
<td>1'b0</td>
<td>Set if data marked with an error is returned for any read (descriptor or data buffer). The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>31</td>
<td>tx_ch1_dzero</td>
<td>1'b0</td>
<td>Set if a descriptor has a packet length of zero or the SOP flag bit is not set in the first descriptor of a packet. This bit is also set if the controller runs out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>39:32</td>
<td>Reserved</td>
<td>8'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>rx_undrfl</td>
<td>1'b0</td>
<td>This bit is set by the receive FIFO underflowing. If this happens when DMA is being used then the rx_rd_thsh is incorrectly set.</td>
</tr>
<tr>
<td>41</td>
<td>rx_ovrfl</td>
<td>1'b0</td>
<td>This bit is set by the receive FIFO overflowing. This happens if the DMA (or reader in direct mode) has delayed reading data from the FIFO so there is no space for the next received data.</td>
</tr>
<tr>
<td>42</td>
<td>tx_undrfl</td>
<td>1'b0</td>
<td>This bit is set if the transmit FIFO underflows. This happens if the DMA (or sender in direct mode) has delayed inserting data into the FIFO and it was empty when more data was needed to be transmitted.</td>
</tr>
<tr>
<td>43</td>
<td>tx_ovrfl</td>
<td>1'b0</td>
<td>This bit is set by the transmit FIFO overflowing. This happens if the DMA (or sender in direct mode) writes to the FIFO when it is full. This will only happen for DMA if the tx_wr_thsh threshold is incorrectly set.</td>
</tr>
<tr>
<td>44</td>
<td>ltcol_err</td>
<td>1'b0</td>
<td>This bit will be set when a packet experiences a late collision. A late collision is one that happens after the minimum packet size has been sent. Collisions can never happen in Packet FIFO modes.</td>
</tr>
<tr>
<td>45</td>
<td>excol_err</td>
<td>1'b0</td>
<td>This bit is set if a packet experiences excessive collisions. It is signalled when attempted transmission of the packet collides 16 times. Normally this indicates a hardware problem, but can also be caused by a very busy network or in the unlucky case of the random backoff LFSR being synchronized with another host (normally this will be fixed by the different length of time the two devices take to respond and clear the excessive collision error, but it is also possible to set a different lfsr_seed in the mac_cfg register). Collisions can never happen in Packet FIFO modes.</td>
</tr>
<tr>
<td>46</td>
<td>cntr_ovrfl_err</td>
<td>1'b0</td>
<td>This bit is set if any of the RMON counters overflow. Whenever a counter overflows this bit is set and the counter number is written into the counter_addr field. The RMON counters are not used in Packet FIFO modes.</td>
</tr>
<tr>
<td>51:47</td>
<td>counter_addr</td>
<td>5'b0</td>
<td>This is the number of the most recent RMON counter that overflowed. It is only valid if the cntr_ovrfl_err bit is set. The RMON counters are not used in Packet FIFO modes.</td>
</tr>
</tbody>
</table>
52 tx_pause_on 1'b0 System revision PERIPH_REV3 or greater. This bit is set when the MAC has received a pause frame and the pause time indicated has not expired. It is cleared when the pause time expires (or if a pause frame with zero time is received). No interrupt can be generated from this bit. Pause frames are not used in Packet FIFO modes.

63:53 reserved 11'b0 Reserved

---

### Table 183: MAC Status 1 Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>status</td>
<td>64'h0</td>
<td>Reading this register gives the same result as reading the status register, except that only the latched bits associated with channel 1 transmit and receive DMA are cleared. It is for use in split interrupt mode.</td>
</tr>
</tbody>
</table>

---

### Table 184: MAC Debug Status Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>status</td>
<td>64'b0</td>
<td>Reading this register gives the same result as reading the status register, except the bits are not cleared and none of the other side effects happen. It is intended for debug accesses to the status.</td>
</tr>
</tbody>
</table>
### Table 185: MAC Interrupt Mask Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>46:0</td>
<td>mask</td>
<td>47'b0</td>
<td>Setting a bit in this register enables generation of an interrupt when the corresponding bit is set in the mac_status register.</td>
</tr>
<tr>
<td>47</td>
<td>split_en</td>
<td>1'b0</td>
<td>When this bit is set the channel 1 interrupts will be signalled on the channel 1 interrupt instead of standard interrupt, and reads to the mac_status register will not clear channel 1 state. When this bit is clear all interrupts are signalled on the standard interrupt and reading the mac_status register clears all state.</td>
</tr>
<tr>
<td>63:48</td>
<td>reserved</td>
<td>19'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Table 186: MAC FIFO Pointer Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>status</td>
<td>64'hx</td>
<td>Status for fifo and counts (Broadcom Use Only)</td>
</tr>
</tbody>
</table>

### Table 187: MAC Receive Address Filter Exact Match Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47:0</td>
<td>address</td>
<td>48'bx</td>
<td>The destination address to be exactly matched as part of input packet filtering. The incoming address is always compared to all entries. The low byte in this register corresponds with the first byte of the address to be received.</td>
</tr>
<tr>
<td>63:48</td>
<td>zero</td>
<td>16'b0</td>
<td>Reads as zero, writes ignored.</td>
</tr>
</tbody>
</table>
### Table 188: MAC Receive Address Filter Mask Registers (Only if System Revision >= PERIPH_REV3)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47:0</td>
<td>mask</td>
<td>48'b FFFFF_</td>
<td>The mask qualifies the destination address set in the corresponding mac_addr register. When the incoming address is compared to the address any bits with a mask of zero are excluded from the comparison. The low byte in this register corresponds with the first byte of the address to be received.</td>
</tr>
<tr>
<td>63:48</td>
<td>zero</td>
<td>16'b0</td>
<td>Reads as zero, writes ignored.</td>
</tr>
</tbody>
</table>

### Table 189: MAC Receive Address Filter Hash Match Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>map</td>
<td>64'bx</td>
<td>The hash map bits for a hash based match as part of input packet filtering. Set the bit to zero to not match that hash value, and to one to match on the hash value.</td>
</tr>
</tbody>
</table>

### Table 190: MAC Transmit Source Address Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>47:0</td>
<td>address</td>
<td>48'b0</td>
<td>The source address to be put in outgoing packets. The low byte of this register will be the first byte of the address transmitted. Pause Frames will be accepted on the address configured in this register (in addition to the well known multicast address). The address in this register will be used as the source address when the MAC sends a Pause Frame.</td>
</tr>
<tr>
<td>63:48</td>
<td>zero</td>
<td>16'b0</td>
<td>Reads as zero, writes ignored.</td>
</tr>
</tbody>
</table>
### Table 191: MAC Packet Type Configuration Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>type0</td>
<td>16'b0</td>
<td>If a received packet has this packet type the receive descriptor will be written with type = 3'b100. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.</td>
</tr>
<tr>
<td>31:16</td>
<td>type1</td>
<td>16'b0</td>
<td>If a received packet has this packet type the receive descriptor will be written with type = 3'b101. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.</td>
</tr>
<tr>
<td>47:32</td>
<td>type2</td>
<td>16'b0</td>
<td>If a received packet has this packet type the receive descriptor will be written with type = 3'b110. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.</td>
</tr>
<tr>
<td>63:48</td>
<td>type3</td>
<td>16'b0</td>
<td>If a received packet has this packet type the receive descriptor will be written with type = 3'b111. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.</td>
</tr>
</tbody>
</table>

### Table 192: MAC Receive Address Filter Control Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>allpkt_en</td>
<td>1'b0</td>
<td>When this bit is set all packets will be accepted (promiscuous mode).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>1</td>
<td>ucast_en</td>
<td>1'b0</td>
<td>When this bit is set unicast packets are checked in the hash map.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>2</td>
<td>ucast_inv</td>
<td>1'b0</td>
<td>When this bit is set unicast packets that miss in the hash map will be accepted. If it is clear unicast packets must hit to be accepted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>3</td>
<td>mcast_en</td>
<td>1'b0</td>
<td>When this bit is set multicast packets are checked in the hash map.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>4</td>
<td>mcast_inv</td>
<td>1'b0</td>
<td>When this bit is set multicast packets that miss in the hash map will be accepted. If it is clear multicast packets must hit to be accepted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>5</td>
<td>bcast_en</td>
<td>1'b0</td>
<td>When this bit is set all broadcast packets are accepted. When it is clear all broadcast packets are dropped.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>6</td>
<td>direct_inv</td>
<td>1'b0</td>
<td>When this bit is set packets will be accepted if they do not match any of the entries in the exact address match registers. When this bit is clear packets will be accepted if they match any entry in the direct address match registers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Used in both Ethernet and Packet FIFO modes.</td>
</tr>
</tbody>
</table>
### Table 192: MAC Receive Address Filter Control Registers (Cont.)

This register is used in both Ethernet and Packet FIFO modes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>allm_en</td>
<td>1'b0</td>
<td>When this bit is set all multicast packets are accepted. When it is clear multicast packets will only be accepted if they match in the Exact or Hash filters. Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>15:8</td>
<td>iphdr_offset</td>
<td>8'b0</td>
<td>Byte in the packet (counting from 1) of the IP header for checking the IPv4 header checksum. This is normally 15, but may be different if encapsulation headers are prepended. This offset is also used by the TCP checksum checking. Used only in Ethernet mode.</td>
</tr>
<tr>
<td>23:16</td>
<td>rx_crc_offset</td>
<td>8'b0</td>
<td>System Revision PERIPH_REV3 and later only. Sets the offset for CRC checking to begin in received packets. Bits 18:16 MUST be 3'b000. Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>31:24</td>
<td>rx_pkt_offset</td>
<td>8'b0</td>
<td>System Revision PERIPH_REV3 and later only. Sets the offset for the Ethernet frame in received packets. Bits 26:24 MUST be 3'b000. Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>32</td>
<td>fwdpause_en</td>
<td>1'b0</td>
<td>System Revision PERIPH_REV3 and later only. If this bit is set then Pause Frames will not be processed by the hardware and will be accepted and passed to the DMA engine as though they had passed the address filter. This allows software to do pause frame flow control (for example to only disable the best-effort tx DMA channel when a pause is requested, but allow the priority traffic to continue). Used only in Ethernet mode.</td>
</tr>
<tr>
<td>33</td>
<td>vlan_det_en</td>
<td>1'b0</td>
<td>System Revision PERIPH_REV3 and later only. If this bit is set then 4 is added to the ip_hdr_offset if the type field in the MAC header (i.e. at mac_hdr_off +12 and +13) is the VLAN type 81-00. This allows the correct IP header offset to be used for a network with a mix of VLAN and non-VLAN packets. In addition when the 81-00 type is detected the vlan_pkt bit (written to bit 1 of the dscr_b &quot;options&quot; field i.e. just above the bad_tcpcs bit) will be set and the pkt_type field in the status will come from decoding the two bytes following the VLAN tag (i.e. the packet type from the untagged packet). Used only in Ethernet mode.</td>
</tr>
<tr>
<td>41:34</td>
<td>rx_ch_msn_sel</td>
<td>8'b0</td>
<td>System Revision PERIPH_REV3 and later only. This field specifies the offset of the upper four bits of the channel selection index when the split_ch_sel bit is set in the mac_cfg register. Note that if rx_ch_msn_sel &lt;= (rx_ch_sel_msb,rx_ch_sel) the channel selected is UNPREDICTABLE. Used in both Ethernet and Packet FIFO modes.</td>
</tr>
<tr>
<td>63:42</td>
<td>reserved</td>
<td>22'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>
Table 193: MAC Receive Channel Select Map Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_chlo0_0</td>
<td>00_1006_4300</td>
</tr>
<tr>
<td>mac_chlo0_1</td>
<td>00_1006_4308</td>
</tr>
<tr>
<td>mac_chlo0_2</td>
<td>00_1006_4310</td>
</tr>
<tr>
<td>mac_chlo3_0</td>
<td>00_1006_4318</td>
</tr>
<tr>
<td>mac_chlo3_1</td>
<td>00_1006_5318</td>
</tr>
<tr>
<td>mac_chlo3_2</td>
<td>00_1006_6318</td>
</tr>
<tr>
<td>mac_chlo1_0</td>
<td>00_1006_4308</td>
</tr>
<tr>
<td>mac_chlo1_1</td>
<td>00_1006_5308</td>
</tr>
<tr>
<td>mac_chlo1_2</td>
<td>00_1006_6308</td>
</tr>
<tr>
<td>mac_chlo2_0</td>
<td>00_1006_4308</td>
</tr>
<tr>
<td>mac_chlo2_1</td>
<td>00_1006_5308</td>
</tr>
<tr>
<td>mac_chlo2_2</td>
<td>00_1006_6308</td>
</tr>
</tbody>
</table>

This register is used in both Ethernet and Packet FIFO modes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>map</td>
<td>64'b0</td>
<td>These registers form the table for mapping the 8 bits from a received packet into a two bit channel number. The 256 entry table has bit 0 register 0 as entry 0, and bit 63 register 3 as entry 255. The chup register provides the msb and the chlo register the lsb of the channel number.</td>
</tr>
</tbody>
</table>

Table 194: MAC MII Management Interface Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_mdio_0</td>
<td>00_1006_4428</td>
</tr>
<tr>
<td>mac_mdio_1</td>
<td>00_1006_5428</td>
</tr>
<tr>
<td>mac_mdio_2</td>
<td>00_1006_6428</td>
</tr>
</tbody>
</table>

This register is used in both Ethernet and Packet FIFO modes.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>mdc</td>
<td>1'b0</td>
<td>MII Management interface clock, this bit is copied to the MDC pin.</td>
</tr>
<tr>
<td>1</td>
<td>mdio_dir</td>
<td>1'b0</td>
<td>MII Management Data Direction, when set the mdio pin is an input, when clear the mdio pin is driven from the mdio_out bit. This bit must be clear to allow receive flow control to be output when the interface is configured for encoded packet fifo mode operation.</td>
</tr>
<tr>
<td>2</td>
<td>mdio_out</td>
<td>1'b0</td>
<td>MII Management interface data out, this bit is copied to the MDIO pin if the mdio_dir bit is 0.</td>
</tr>
<tr>
<td>3</td>
<td>genc</td>
<td>1'b0</td>
<td>General output, this bit is copied to the GENO pin.</td>
</tr>
<tr>
<td>4</td>
<td>mdio_in</td>
<td>1'b0</td>
<td>MII Management interface data input, this read only bit gives the value on the mdio pin.</td>
</tr>
<tr>
<td>63:5</td>
<td>reserved</td>
<td>59'b0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

This register is used in both Ethernet and Packet FIFO modes.
This Page is left blank for notes
**Section 10: Serial Interfaces**

**INTRODUCTION**

The part incorporates two identical serial ports that provide full-duplex interfaces to a variety of serial devices. Each port is separately configured, and can be run as an asynchronous serial link from 1200 baud to 5 Mbaud or a synchronous serial link at up to 55 Mbps.

In asynchronous mode the programming model is based on a DUART. Some registers share information between the A and B channels, but the two ports can be independently switched to the synchronous mode.

In synchronous mode the serial interface includes an internal HDLC engine and externally provides a PCM highway style link.

Selection between asynchronous and synchronous interfaces is normally made at reset time using the configuration input on IO_AD[12] (channel A) and IO_AD[14] (channel B). However, the CPU can write the corresponding bits in the `system_cfg` register to change the selection. Modifying this bit will switch the output pins between the asynchronous drivers and the synchronous ones. If the interface is switched between modes software will need to re-initialize the interface and may need to reset the external device.

Each serial port has 8 pins associated with it. In addition if an interface is in synchronous mode it can use one of the GPIO pins as an output (software cannot change the use of this pin, it can only be set by the reset configuration). The following table shows their use in each mode, and for asynchronous mode the correspondence between the pins and input or output registers for channel A and channel B.

![Table 195: Serial Interface Signal Names](image)

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Direction</th>
<th>A</th>
<th>B</th>
<th>Asynchronous Mode</th>
<th>Synchronous Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUT</td>
<td>Output</td>
<td></td>
<td></td>
<td>Transmit data output</td>
<td>Transmit data output</td>
</tr>
<tr>
<td>DIN</td>
<td>Input</td>
<td></td>
<td></td>
<td>Receive data input</td>
<td>Receive data input</td>
</tr>
<tr>
<td>RTS/TSTROBE</td>
<td>Output</td>
<td>op[0]</td>
<td>op[1]</td>
<td>RTS output or general output</td>
<td>Transmit strobe output</td>
</tr>
<tr>
<td>COUT</td>
<td>Output</td>
<td>op[2]</td>
<td>op[3]</td>
<td>General output (e.g. for DTR handshake)</td>
<td>Clock output</td>
</tr>
<tr>
<td>CTS/TCLKIN</td>
<td>Input</td>
<td>ip[0]</td>
<td>ip[1]</td>
<td>CTS input, or general input with transition detector</td>
<td>Transmit clock input</td>
</tr>
<tr>
<td>CIN/RCLKIN</td>
<td>Input</td>
<td>ip[2]</td>
<td>ip[3]</td>
<td>General input with transition detector (e.g. for DSR handshake)</td>
<td>Receive clock input</td>
</tr>
<tr>
<td>TIN</td>
<td>Input</td>
<td>ip[4]</td>
<td>ip[5]</td>
<td>General input (e.g. for DCD)</td>
<td>Transmit enable or synchronization input</td>
</tr>
<tr>
<td>RIN</td>
<td>Input</td>
<td>ip[6]</td>
<td>ip[7]</td>
<td>General input (e.g. for RI)</td>
<td>Receive enable or synchronization input</td>
</tr>
<tr>
<td>RSTROBE</td>
<td>Output</td>
<td></td>
<td></td>
<td>Receive strobe output (if this is not enabled the pins are used for GPIO[0] or GPIO[1])</td>
<td></td>
</tr>
</tbody>
</table>
ASYNCHRONOUS MODE

The asynchronous interface is provided using a DUART. The two channels are separately programmable and each has its own baud rate generator. Each channel has a 16 byte transmit FIFO and a 16 byte receive FIFO. In addition to the data path there are 4 inputs and 2 outputs per channel that can either be used for flow control or are available for general use. The inputs are readable through the input port register (IPR), and the outputs are set through the output port register (OPR). Even numbered I/O lines are associated with channel A, and odd numbered lines with channel B. The UART supports RTS/CTS flow control in hardware, other control must be done in software. When serial port 0 is set in asynchronous mode it is driven from channel A of the DUART, serial port 1 in asynchronous mode is driven from channel B.

BAUD RATE GENERATORS

The baud rate is generated on chip by dividing down from the 100MHz reference clock. Each channel can have a different baud rate, but for a channel the transmit and receive rates must be the same. The baud rate is selected by setting the `duart_clk_sel` register to

\[
(\text{100 MHz/}(\text{baud rate } \times 20)) - 1
\]

Some popular baud rates are shown in the table below.

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Count</th>
<th>Actual</th>
<th>% Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>4095</td>
<td>1220.703</td>
<td>1.72526</td>
</tr>
<tr>
<td>2400</td>
<td>2082</td>
<td>2400.384</td>
<td>0.016003</td>
</tr>
<tr>
<td>4800</td>
<td>1040</td>
<td>4803.074</td>
<td>0.064041</td>
</tr>
<tr>
<td>9600</td>
<td>519</td>
<td>9615.385</td>
<td>0.160256</td>
</tr>
<tr>
<td>19200</td>
<td>259</td>
<td>19230.77</td>
<td>0.160256</td>
</tr>
<tr>
<td>38400</td>
<td>129</td>
<td>38461.54</td>
<td>0.160256</td>
</tr>
<tr>
<td>57600</td>
<td>85</td>
<td>58139.53</td>
<td>0.936693</td>
</tr>
<tr>
<td>115200</td>
<td>42</td>
<td>116279.1</td>
<td>0.936693</td>
</tr>
<tr>
<td>230400</td>
<td>21</td>
<td>227272.7</td>
<td>-1.35732</td>
</tr>
<tr>
<td>500000</td>
<td>9</td>
<td>500000</td>
<td>0</td>
</tr>
<tr>
<td>1000000</td>
<td>4</td>
<td>1000000</td>
<td>0</td>
</tr>
</tbody>
</table>

The baud clock can be output on the Cout pin by setting the appropriate bit in the output port configuration register `duart_opcr`.

The baud clock is also used for the synchronous serial interface when it is configured to use an internal clock. In this case Cout should be configured to pass the clock signal to the external devices.
OPERATION

The two channels of the DUART are identical in terms of configuration and operation. They are controlled by a set of memory mapped registers, in most cases separate registers are provided for channel A and channel B (the register names are identical with either _a or _b indicating the channel); in a few cases the two channels are combined.

Prior to operation a channel must be configured. The duart_mode_reg_1 is used to set the number of data bits in a character (either 7 or 8) and whether parity should be added. The parity bit may be set to give either an even or odd number of ones, or to be fixed as a high (mark) or low (space). The number of stop bits are set in the duart_mode_reg_2. These registers are also used to enable hardware RTS/CTS handshake separately in each direction. The baud rate should be set in the duart_clk_sel register. The mode and clock registers should only be written while the interface is inactive, changing the parameters while the UART is active results in UNPREDICTABLE behavior. Once configured the transmitter and receiver are enabled by writing the duart_cmd register.

The transmitter indicates that it is able to accept data into the transmit FIFO by setting the duart_tx_rdy bit in the duart_status register. For as long as this bit is set, the CPU can write characters to the duart_tx_hold register and they will be inserted in the transmit FIFO. If the transmitter is enabled characters are extracted from the FIFO, have a (low) start bit prepended, parity and (high) stop bits appended and are serialized (least significant bit first) and sent to the DOUT pin.

The transmitter may be disabled or reset by issuing a command to the duart_cmd register. Disabling transmission will cause transmission to stop and the line idle (high) when transmission of the current character is complete, any characters in the FIFO will remain there to be sent when the transmitter is re-enabled. The break command is similar to disable except when transmission of the current character (including stop bits) is complete the line is driven low. The break condition is cleared with the stop-break command, which removes the break synchronously with the transmit bit clock. Resetting the transmitter will cause the current transmission to be aborted and all characters in the FIFO to be discarded, all transmitter state is cleared and it is left in the disabled state. Until the FIFO becomes full characters can continue to be inserted into the FIFO even if the transmitter is disabled (if the disabled state was reached through the reset command any characters in the FIFO when the reset was issued are lost, after this characters can be inserted into the now empty FIFO).

The receiver will detect the (low) start bit on the DIN line, and use it to synchronize the local bit clock. Data bits are then sampled in the middle of the bit time until the data and parity bits have been received. One additional bit is sampled, and an error reported if it is not the expected (high) stop bit. Once a character has been received it is put in to the FIFO along with the frame and parity error flags. If the_FIFO becomes full the character is discarded and the first character to be received that can be put in to the FIFO will have the overrun status flag set (thus the overrun flag indicates that characters were lost before the one that is marked).

There are two error conditions may result if the stop bit was not detected. If the data received is all zeros and a zero is seen in place of the stop bit then the break condition is detected. A zero character is put in the FIFO along with the break-detected flag, the receiver will not resume operation until the break condition has been removed by the line being high for two bit times. If the data received is not all zeros but a zero is seen in place of the stop bit then a frame error is reported, the input data continues to be sampled for half a bit time. During this period, if the input is sampled high it is considered as a late stop bit and the line is monitored for the next start bit. If the input remains low for the full half bit time then it is considered to be an early start bit and the receiver will synchronize at the half bit time point and start assembly of the next character.
When there is a character in the receive FIFO the duart_rx_rdy bit in the duart_status register is set, and the CPU can read it from the duart_rx_hold register. Reading the duart_rx_hold register removes the character from the FIFO. The top four bits of the status register reflect the flags (overrun error, parity error, frame error and break) associated with the character (since reading the character pops it out of the FIFO the flags should be read before the character). If the duart_rx_hold register is read at a time when the duart_rx_rdy bit is not set the result is UNPREDICTABLE.

The receiver may be disabled or reset by issuing a command to the duart_cmd register. Both of these have immediate effect, any character in the process of being received is discarded. Reset will also discard all data in the FIFO, disabling the receiver will retain it and reading from the FIFO can continue. In both cases the receiver must be enabled by writing the command register.

Handshaking using the RTS/CTS protocol is implemented in hardware. It is enabled separately for the transmitter (in duart_mode_reg_2) and the receiver (in duart_mode_reg_1). When enabled the transmitter will monitor the state of the CTS_TCLKIN pin, and will only start sending a character if the signal is low. Once transmission of a character has started it will continue until the character has been sent regardless of the state of CTS_TCLKIN. When enabled the RTS_TSTROBE pin is driven by the receiver. It will be set low if the receiver is enabled and is able to receive a character. On receiving a start bit that will cause the receiver to become full the RTS_TSTROBE line will be deasserted (set high), to indicate that no further characters should be sent.

If hardware handshaking is disabled the RTS_TSTROBE output is controlled from software, and will reflect the inverse of the op[0] bit for channel A or the op[1] bit for channel B. The op bits can be set by writing to the duart_set_opr register and cleared by writing to the duart_clear_opr register. For convenience of systems implementing software flow control the op[0] and op[1] bits (that control RTS_TSTROBE) can also be set and cleared through the duart_out_port register. The CTS_TCLKIN pin can always be read through the duart_in_port register.

The baud rate clock that the transmitter is using can be output on the COUT pin, alternatively the pin can be driven from software as the inverse of op[2] or op[3]. This is set in the output port configuration register duart_opcr.

The CIN_RCLKIN, RIN and TIN pins are provided in the duart_in_port register for software to use either as separate handshake lines or as general inputs. The CTS_TCLKIN and CIN_RCLKIN lines have transition detectors associated with them and the upper four bits of the duart_inport_chng register will latch the fact the line changed since the last time the register was read (the read automatically clears the bits).
INTERRUPTS

The DUART interrupts are provided as system interrupts 8 (for channel A) and 9 (for channel B). The conditions that can cause an interrupt are signalled in the interrupt status register `duart_isr`. This contains the channel A status in the lower four bits and the channel B status in the upper four bits. For convenience the information for each channel is available in the `duart_isr_a` and `duart_isr_b` registers where the status for the channel is always in the lower four bits and the upper bits are zeros. The conditions that can cause an interrupt are: transmitter ready, receiver ready or receiver FIFO full, break detected or removed and change detected on the handshake lines. Corresponding to each bit in the interrupt status register there is a bit in the interrupt mask register `duart_imr`, if a status bit is set and the corresponding mask bit is also set then an interrupt will be raised. Again, for programming convenience, aliases of the lower and upper bits of the mask are provided in the lower four bits of the `duart_imr_a` and `duart_imr_b` registers (these are aliases - internally there is only a single copy of each mask bit).

Figure 66 shows the interrupt generation for a single channel.
The transmit interrupt can be generated either by there being at least one free entry in the fifo (i.e. when the duart_tx_rdy status bit is set) or when the fifo is empty. The tx_irq_sel bit in the duart_mode_reg_1 is used to select between them. The isr status bit is cleared by the write of the duart_rx_hold register that removes the condition (causes the fifo to go full or not-empty depending on the mode).

The receive interrupt can be generated either by there being at least one character in the fifo (i.e. when the duart_rx_rdy status bit is set) or when the fifo is full (or almost full). The rx_irq_sel bit in the duart_mode_reg_1 is used to select between them. The isr status bit is cleared by the read of the duart_rx_hold register that removes the condition (causes the fifo to go empty or not-full depending on the mode).

If the “FIFO full” interrupt source is selected the duart_sig_full field in the duart_full_ctl register can be used to set the number of characters that must be in the receive fifo for the interrupt to be raised. The interrupt is asserted when the fifo contains more than this number of characters. By default this field is 15, causing the interrupt when the fifo is completely full. With this setting software must respond to the interrupt and read from the fifo within one character time to avoid the fifo overflowing and characters being lost. The level at which the interrupt is raised can be lowered to provide more character times of interrupt latency.

If the duart_int_time field in the duart_full_ctl register is non-zero and the uart is set to interrupt on rx full (duart_rx_irq_sel bit = 1) then an interrupt will be raised if the rx fifo is not empty and the link has been idle for duart_int_time*16 bit times. If duart_int_time is zero then there is no timeout. This timeout ensures that characters received when the fifo is below threshold will not suffer long latencies before being serviced.

If either the threshold or timeout mechanism are used the receive interrupt status bit in the duart_isr register will be set based on the threshold or timeout. However the duart_rx_full bit in the duart_status register will continue to reflect the real full state of the fifo. The fifo will continue to accept characters until it is really full regardless of threshold and timeout settings.

The change in break interrupt status is set by the detection of the start of a break and by the detection of the end of a break. It will remain set until the CPU issued a reset break change command to the duart_cmd register.

The input line transition detector interrupt combines the state change information for both input lines of the channel. They are individually enabled in the duart_aux_ctrl registers. The interrupt status is cleared by reading the duart_inport_chng register.

LOOPBACK

There are two loopback modes for diagnostics. They are enabled by setting the duart_chan_mode bits in the duart_mode_reg_2.

In local loopback mode the transmitter data output is internally connected to the receiver data input, so any character transmitted will be received. In this mode the DOUT pin is held idle (high) and the DIN pin is ignored. During local loopback the transmit module will not check the CTS_TCLKIN pin and the receive module will not set the RTS_TSTROBE pin even if their enable bits (duart_tx_cts_ena in duart_mode_reg_2 and duart_rx_rts_ena in duart_mode_reg_1) are set. While the interface is in local loopback mode the receiver will be active even if it has not been enabled. The transmitter must still be enabled for characters to be sent.

In remote loopback mode data received on the DIN pin is re-clocked and transmitted on the DOUT, the received character is not put in to the FIFO and no error checking is done. In remote loopback mode received data is always sent out, the transmit module does not check the CTS_TCLKIN pin even if duart_tx_cts_ena bit is set in the duart_mode_reg_2.
## DUART Registers

### Table 197: DUART Mode Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>duart_bits_per_char</td>
<td>2'b00</td>
<td>0: 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Reserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: 7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: 8</td>
</tr>
<tr>
<td>2</td>
<td>duart_parity_type</td>
<td>1'b0</td>
<td>Parity type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Even/Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Odd/High</td>
</tr>
<tr>
<td>4:3</td>
<td>duart_parity_mode</td>
<td>2'b0</td>
<td>Parity mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Add Parity (even/odd selected by bit 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Add fixed parity bit (low/high selected by bit 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: No Parity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: No Parity</td>
</tr>
<tr>
<td>5</td>
<td>duart_tx_irq_sel</td>
<td>1'b0</td>
<td>Transmitter interrupt select.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Interrupt when the transmitter is ready (there is space in the fifo for at least one character).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Interrupt when the transmit fifo is empty.</td>
</tr>
<tr>
<td>6</td>
<td>duart_rx_irq_sel</td>
<td>1'b0</td>
<td>Receiver Interrupt select.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Interrupt on receiver ready (at least one character in the FIFO).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Interrupt on receiver FIFO full</td>
</tr>
<tr>
<td>7</td>
<td>duart_rx_rts_ena</td>
<td>1'b0</td>
<td>Receiver Request-to-Send Control enable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Channel A: If set the S0_RTS_TSTROBE pin will be used as hardware controlled RTS, if clear the pin will output the inverse of op[0].</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Channel B: If set the S1_RTS_TSTROBE pin will be used as hardware controlled RTS, if clear the pin will output the inverse of op[1].</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 198: DUART Second Mode Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>ignored</td>
<td>3'b000</td>
<td>These bits are ignored.</td>
</tr>
<tr>
<td>3</td>
<td>duart_stop_bit_len</td>
<td>1'b0</td>
<td>Set for 2 Stop bits, clear for 1 stop bit</td>
</tr>
<tr>
<td>4</td>
<td>duart_tx_cts_ena</td>
<td>1'b0</td>
<td>Set to cause the transmitter to check CTS, clear to ignore CTS.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Channel A CTS is on the S0_CTS_TCLKIN pin and is also readable as ip[0].</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Channel B CTS is on the S1_CTS_TCLKIN pin and is also readable as ip[1].</td>
</tr>
<tr>
<td>5</td>
<td>reserved</td>
<td>1'b0</td>
<td>Always 0</td>
</tr>
<tr>
<td>7:6</td>
<td>duart_chan_mode</td>
<td>2'b00</td>
<td>Channel Mode.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0: Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: Normal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: Local loopback</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3: Remote loopback</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
Table 199: DUART Command Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_rx_en</td>
<td>1'b0</td>
<td>Enable receiver. Setting this bit at the same time as duart_rx_dis results in no change in the receiver.</td>
</tr>
<tr>
<td>1</td>
<td>duart_rx_dis</td>
<td>1'b0</td>
<td>Disable receiver immediately (a character being received will be lost) Setting this bit at the same time as duart_rx_en results in no change in the receiver.</td>
</tr>
<tr>
<td>2</td>
<td>duart_tx_en</td>
<td>1'b0</td>
<td>Enable transmitter. Setting this bit at the same time as duart_tx_dis results in no change in the transmitter.</td>
</tr>
<tr>
<td>3</td>
<td>duart_tx_dis</td>
<td>1'b0</td>
<td>Disable transmitter when it next becomes idle (i.e. complete any in progress character) Setting this bit at the same time as duart_tx_en results in no change in the transmitter.</td>
</tr>
<tr>
<td>6:4</td>
<td>duart_misc_cmd</td>
<td>3'b000</td>
<td>0: No action. 1: No action. 2: Reset receiver. 3: Reset transmitter. 4: No action. 5: Reset channel's break-change interrupt. 6: Start break (drive data output line low when the transmitter becomes idle, and prevent further transmission). 7: Stop break (if currently sending a break, drive line high for at least one bit time and resume normal transmission).</td>
</tr>
<tr>
<td>7</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved, write as zero.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

Table 200: DUART Status Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_rx_rdy</td>
<td>1'b0</td>
<td>Receiver ready: at least one character can be read. This bit is set whenever there are characters in the receive FIFO even if the receiver is disabled.</td>
</tr>
<tr>
<td>1</td>
<td>duart_rx_fful</td>
<td>1'b0</td>
<td>Receive FIFO full: if any further characters are received they will be discarded. This signal is asserted when the start bit is detected for the character that will cause the receiver to become full, if handshaking is disabled the CPU has one character-time to read data from the FIFO before characters are lost.</td>
</tr>
<tr>
<td>2</td>
<td>duart_tx_rdy</td>
<td>1'b1</td>
<td>Transmitter ready: there is room for at least one character to be written into the transmitter. This bit is set whenever there is space in the transmit FIFO even if the transmitter is disabled (characters inserted in the FIFO will be transmitted when the transmitter is next enabled).</td>
</tr>
<tr>
<td>3</td>
<td>duart_tx_emt</td>
<td>1'b1</td>
<td>Transmitter empty: there are no characters to send and the transmitter is idle</td>
</tr>
<tr>
<td>4</td>
<td>duart_ovrun_err</td>
<td>1'b0</td>
<td>Overrun error: This flag tags a received character to indicate that characters were lost prior to this one.</td>
</tr>
<tr>
<td>5</td>
<td>duart_parity_err</td>
<td>1'b0</td>
<td>Parity error: This flag tags a received character to indicate that this character was received with incorrect parity</td>
</tr>
</tbody>
</table>
Table 200: DUART Status Registers (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>duart_frm_err</td>
<td>1'b0</td>
<td>Frame error: This flag tags a received character to indicate that this character (including parity bit) was non-zero and did not have a valid stop bit.</td>
</tr>
<tr>
<td>7</td>
<td>duart_rcvd_brk</td>
<td>1'b0</td>
<td>Received break: This flag tags a received character (which will be zero) to indicate that the start of a break was detected.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

Table 201: DUART Baud Rate Clock Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:0</td>
<td>uart_clk_counter</td>
<td>12'h0</td>
<td>Baud rate counter. This register sets the baud rate for both the transmitter and receiver. Set to ((100 \text{ MHz} / (\text{baud rate} \times 20))-1).</td>
</tr>
<tr>
<td>63:12</td>
<td>notimp</td>
<td>52'bX</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

Table 202: DUART Full Interrupt Control Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>duart_sig_full</td>
<td>4'hF</td>
<td>This field sets the threshold for the receive fifo full interrupt. The interrupt is raised when the number of characters in the fifo is greater than the value set. With the default of 15 the interrupt is only raised when the fifo is completely full.</td>
</tr>
<tr>
<td>7:4</td>
<td>duart_int_time</td>
<td>4'h0</td>
<td>If this field is non-zero then the fifo full interrupt is raised if there are any characters in the receive fifo and the receive data line has been idle for duart_int_time \times 16 bit times. If this field is zero the interrupt is never raised by a timeout. The timeout interrupt is cleared by any read of the rx_hold register.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bX</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

Table 203: DUART Received Data Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>rx_data</td>
<td>8'hx</td>
<td>Read Only. Received data. The character is only valid if the duart_rx_rdy bit is set in the status register prior to this register being read, when this register is read the next character is made available and its flag bits will be made available in the duart_status register.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bX</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 204: DUART Transmit Data Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>tx_data</td>
<td>W/O</td>
<td>Write only. Data to transmit. Writes will be ignored if the duart_tx_rdy bit is clear.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 205: DUART Input Port Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_in0_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[0] S0_CTS_TCLKIN.</td>
</tr>
<tr>
<td>1</td>
<td>duart_in1_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[1] S1_CTS_TCLKIN.</td>
</tr>
<tr>
<td>2</td>
<td>duart_in2_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[2] S0_CIN_RCLKIN.</td>
</tr>
<tr>
<td>3</td>
<td>duart_in3_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[3] S1_CIN_RCLKIN.</td>
</tr>
<tr>
<td>4</td>
<td>duart_in4_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[4] S0_TIN.</td>
</tr>
<tr>
<td>5</td>
<td>duart_in5_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[5] S1_TIN.</td>
</tr>
<tr>
<td>6</td>
<td>duart_rin0_pin</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[6] S0_RIN.</td>
</tr>
<tr>
<td>7</td>
<td>duart_rin1_pin</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[7] S1_RIN.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 206: DUART Input Port Change Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>duart_in0_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[0] S0_CTS_TCLKIN.</td>
</tr>
<tr>
<td>7:4</td>
<td>duart_in0_chg</td>
<td>4'b0</td>
<td>Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. The bits are cleared when the register is read.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 207: DUART Debug Access Input Port Change Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>debug</td>
<td></td>
<td>These bits provide the same information as in the duart_inport_chng register, but reads do not have side effects.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 208: DUART Input Port Change Status Register for Channel A

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>duart_in_pin_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[3:0]. These bits match duart_in_port[3:0].</td>
</tr>
<tr>
<td>7:4</td>
<td>duart_in_pin_chng</td>
<td>4'b0</td>
<td>Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. Bits 4 and 6 (the change of state bits for ip[0] and ip[2]) are cleared when the register is read.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 209: DUART Input Port Change Status Register for Channel B

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>duart_in_pin_val</td>
<td>ext</td>
<td>Input pin level (0= Low 1= High) ip[3:0]. These bits match duart_in_port[3:0].</td>
</tr>
<tr>
<td>7:4</td>
<td>duart_in_pin_chng</td>
<td>4'b0</td>
<td>Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. Bits 5 and 7 (the change of state bits for ip[1] and ip[3]) are cleared when the register is read.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 210: DUART Output Port Control Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>1'h0</td>
<td>Not used, always zero.</td>
</tr>
<tr>
<td>1</td>
<td>opc2_sel</td>
<td>1'h0</td>
<td>Controls the S0_COUT pin, when clear the pin is the complement of op[2], when set the pin outputs the 1X baud clock for the channel A transmitter.</td>
</tr>
<tr>
<td>2</td>
<td>reserved</td>
<td>1'h0</td>
<td>Not used, always zero.</td>
</tr>
<tr>
<td>3</td>
<td>opc3_sel</td>
<td>1'h0</td>
<td>Controls the S1_COUT pin, when clear the pin is the complement of op[3], when set the pin outputs the 1X baud clock for the channel B transmitter.</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>4'h0</td>
<td>Not used (no support for output bits 7:4).</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'b'x</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 211: DUART Per Channel Output Control Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>opc_sel</td>
<td>1'b0</td>
<td>Controls Cout pin for the channel. An alternative access path for the corresponding bit in the <strong>duart_opcr</strong>. &lt;br&gt;Channel A: This controls the opc2_sel bit. &lt;br&gt;Channel B: This controls the opc3_sel bit.</td>
</tr>
<tr>
<td>7:2</td>
<td>reserved</td>
<td>6'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 212: DUART Aux Control Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_ip0_chng_ena</td>
<td>1'b0</td>
<td>S0_CTS_TCLKIN change of state enable.</td>
</tr>
<tr>
<td>1</td>
<td>duart_ip1_chng_ena</td>
<td>1'b0</td>
<td>S1_CTS_TCLKIN change of state enable.</td>
</tr>
<tr>
<td>2</td>
<td>duart_ip2_chng_ena</td>
<td>1'b0</td>
<td>S0_CIN_RCLKIN change of state enable.</td>
</tr>
<tr>
<td>3</td>
<td>duart_ip3_chng_ena</td>
<td>1'b0</td>
<td>S1_CIN_RCLKIN change of state enable.</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>4'h0</td>
<td>Unused, always zero.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'b'x</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 213: DUART Per Channel Aux Control Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_cts_chng_ena</td>
<td>1'b0</td>
<td>CTS_TCLKIN change of state enable for the channel: &lt;br&gt;Channel A controls duart_ip0_chng_ena &lt;br&gt;Channel B controls duart_ip1_chng_ena</td>
</tr>
<tr>
<td>1</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>duart_cts_chng_ena</td>
<td>1'b0</td>
<td>CIN_RCLKIN change of state enable for the channel: &lt;br&gt;Channel A controls duart_ip2_chng_ena &lt;br&gt;Channel B controls duart_ip3_chng_ena</td>
</tr>
<tr>
<td>7:3</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'b'x</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
### Table 214: DUART Interrupt Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_isr_tx_a</td>
<td>1'b1</td>
<td>Transmitter Ready</td>
</tr>
<tr>
<td>1</td>
<td>duart_isr_rx_a</td>
<td>1'b0</td>
<td>Receiver Ready / FIFO Full</td>
</tr>
<tr>
<td>2</td>
<td>duart_isr_brk_a</td>
<td>1'b0</td>
<td>Change in Break</td>
</tr>
<tr>
<td>3</td>
<td>duart_isr_in_a</td>
<td>1'b0</td>
<td>Input Port 0,2 changes Status (input port changes for channel A)</td>
</tr>
<tr>
<td>4</td>
<td>duart_isr_tx_b</td>
<td>1'b0</td>
<td>Transmitter Ready</td>
</tr>
<tr>
<td>5</td>
<td>duart_isr_rx_b</td>
<td>1'b0</td>
<td>Receiver Ready / FIFO Full</td>
</tr>
<tr>
<td>6</td>
<td>duart_isr_brk_b</td>
<td>1'b0</td>
<td>Change in Break</td>
</tr>
<tr>
<td>7</td>
<td>duart_isr_in_b</td>
<td>1'b0</td>
<td>Input Port 1,3 changes Status (input port changes for channel B)</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 215: DUART Channel A Only Interrupt Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_isr_tx</td>
<td>1'b1</td>
<td>Transmitter Ready</td>
</tr>
<tr>
<td>1</td>
<td>duart_isr_rx</td>
<td>1'b0</td>
<td>Receiver Ready / FIFO Full</td>
</tr>
<tr>
<td>2</td>
<td>duart_isr_brk</td>
<td>1'b0</td>
<td>Change in Break</td>
</tr>
<tr>
<td>3</td>
<td>duart_isr_in</td>
<td>1'b0</td>
<td>Input Port 0 or 2 Changes Status</td>
</tr>
<tr>
<td>7:4</td>
<td>Reserved</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 216: DUART Channel B Only Interrupt Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_isr_tx</td>
<td>1'b1</td>
<td>Transmitter Ready</td>
</tr>
<tr>
<td>1</td>
<td>duart_isr_rx</td>
<td>1'b0</td>
<td>Receiver Ready / FIFO Full</td>
</tr>
<tr>
<td>2</td>
<td>duart_isr_brk</td>
<td>1'b0</td>
<td>Change in Break</td>
</tr>
<tr>
<td>3</td>
<td>duart_isr_in</td>
<td>1'b0</td>
<td>Input Port 1 or 3 Changes Status</td>
</tr>
<tr>
<td>7:4</td>
<td>Reserved</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
### Table 217: DUART Interrupt Mask Register

**duart_imr - 00_1006_03A0**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_imr tx_a</td>
<td>1'b0</td>
<td>Mask Transmitter ready A</td>
</tr>
<tr>
<td>1</td>
<td>duart_imr rx_a</td>
<td>1'b0</td>
<td>Mask Receiver Ready / FIFO Full A</td>
</tr>
<tr>
<td>2</td>
<td>duart_imr brk_a</td>
<td>1'b0</td>
<td>Mask Change in break A</td>
</tr>
<tr>
<td>3</td>
<td>duart_imr in_a</td>
<td>1'b0</td>
<td>Mask Input port 0,2 change interrupt (input port changes for channel A)</td>
</tr>
<tr>
<td>4</td>
<td>duart_imr tx_b</td>
<td>1'b0</td>
<td>Mask Transmitter Ready B</td>
</tr>
<tr>
<td>5</td>
<td>duart_imr rx_b</td>
<td>1'b0</td>
<td>Mask Receiver Ready / FIFO Full B</td>
</tr>
<tr>
<td>6</td>
<td>duart_imr brk_b</td>
<td>1'b0</td>
<td>Mask Change in Break B</td>
</tr>
<tr>
<td>7</td>
<td>duart_imr in_b</td>
<td>1'b0</td>
<td>Mask Input Port 1,3 Changes Status (input port changes for channel B)</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

Bits in this register must be set to cause an interrupt to be generated when the corresponding bit in the **duart_isr** is set.

### Table 218: DUART Channel A Only Interrupt Mask Register

**duart_imr_a - 00_1006_0330**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_imr tx</td>
<td>1'b0</td>
<td>Mask Transmitter Ready A</td>
</tr>
<tr>
<td>1</td>
<td>duart_imr rx</td>
<td>1'b0</td>
<td>Mask Receiver Ready / FIFO Full A</td>
</tr>
<tr>
<td>2</td>
<td>duart_imr brk</td>
<td>1'b0</td>
<td>Mask Change in break A</td>
</tr>
<tr>
<td>3</td>
<td>duart_imr in</td>
<td>1'b0</td>
<td>Mask Input Port 0,2 Changes Status</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

Convenience register, only writes channel A IMR bits

### Table 219: DUART Channel B Only Interrupt Mask Register

**duart_imr_b - 00_1006_0350**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_imr tx</td>
<td>1'b0</td>
<td>Mask Transmitter Ready</td>
</tr>
<tr>
<td>1</td>
<td>duart_imr rx</td>
<td>1'b0</td>
<td>Mask Receiver Ready / FIFO Full A</td>
</tr>
<tr>
<td>2</td>
<td>duart_imr brk</td>
<td>1'b0</td>
<td>Mask Change in Break</td>
</tr>
<tr>
<td>3</td>
<td>duart_imr in</td>
<td>1'b0</td>
<td>Mask Input Port 1,3 Changes Status</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
### Table 220: DUART Output Port Set Register

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_set_op[0]</td>
<td>W/O</td>
<td>op[0] 1= Set to High 0= No change</td>
</tr>
<tr>
<td>1</td>
<td>duart_set_op[1]</td>
<td>W/O</td>
<td>op[1] 1= Set to High 0= No change</td>
</tr>
<tr>
<td>7:4</td>
<td>reserved</td>
<td>W/O</td>
<td>Reserved (only 4 outputs)</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented</td>
</tr>
</tbody>
</table>

Note: the output pins are the inverse of the op

### Table 221: DUART Output Port Clear Register

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_clr_op[0]</td>
<td>W/O</td>
<td>op[0] 1= Clear to low 0= No change</td>
</tr>
<tr>
<td>1</td>
<td>duart_clr_op[1]</td>
<td>W/O</td>
<td>op[1] 1= Clear to low 0= No change</td>
</tr>
<tr>
<td>7:4</td>
<td>Reserved</td>
<td>W/O</td>
<td>Reserved (only 4 outputs)</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented</td>
</tr>
</tbody>
</table>

Note: the output pins are the inverse of the op

### Table 222: DUART Output Port RTS Register

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>duart_out_pin_set[0]</td>
<td>W/O</td>
<td>0: No change 1: Set output pin S0_RTS_TSTROBE to High (i.e. clear op[0])</td>
</tr>
<tr>
<td>1</td>
<td>duart_out_pin_set[1]</td>
<td>W/O</td>
<td>0: No change 1: Set output pin S1_RTS_TSTROBE to High (i.e. clear op[1])</td>
</tr>
<tr>
<td>2</td>
<td>duart_out_pin_clr[0]</td>
<td>W/O</td>
<td>0: No change 1: Clear output pin S0_RTS_TSTROBE to Low (i.e. set op[0])</td>
</tr>
<tr>
<td>3</td>
<td>duart_out_pin_clr[1]</td>
<td>W/O</td>
<td>0: No change 1: Clear output pin S1_RTS_TSTROBE to Low (i.e. set op[1])</td>
</tr>
<tr>
<td>7:4</td>
<td>Reserved</td>
<td>W/O</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented</td>
</tr>
</tbody>
</table>
SYNCHRONOUS MODE

In synchronous mode, the serial data stream is accompanied by a clock and an optional gating or framing signal. There are two sub-modes, HDLC and transparent. In HDLC sub-mode, frames within the bit stream are recognized and processed according to ISO/IEC-3309 (High-level data link control (HDLC) procedures - Frame structure). In transparent sub-mode, serial data is transmitted and received without modification.

In both sub-modes, the serial stream can be qualified by a gating signal. Typically, such gating would select a subset of the bits at the physical interface for internal processing. The gating signal can be provided externally. Alternatively, it can be generated by a table-driven sequencer that is synchronized to an external framing pulse. Configuration options allow a choice of polarities and delays for the gating or synchronizing signals.

The synchronous serial ports can operate at speeds of 0 to 55 Mbp/s and thus can support up to T3, E3 and OC-1 data rates. In such applications, an external framer or similar interface would typically be used to connect to the serial line and would itself be controlled from the CPU via the generic bus interface (see Section 11: “Generic/Boot Bus” on page 361).

Each port is associated with a DMA channel for transmit and another for receive. In synchronous mode, the port and the pair of DMA channels are collectively called a serial channel.

The interface provides two identical and independent serial channels, 0 and 1. The registers and interrupts associated with each are differentiated by appending _0 or _1 to their names.

FUNCTIONAL OVERVIEW

The serial channels are part of the I/O subsystem on the part and connect to the ZBbus through I/O bridge 1. Figure 67 on page 337 shows a block diagram. Each consists of 3 major functional blocks:

- a pair of built-in DMA controllers with FIFOs
- a protocol engine that can operate in either HDLC or transparent mode
- a programmable line interface

For each serial channel, there are two built-in DMA controllers, one for transmit and one for receive. The DMA controllers interface with the I/O Bridge. They connect to the protocol engine via separate FIFOs for transmit (TxFIFO) and receive (RxFIFO).

Except for the interpretation of link-specific option and status bits, which are summarized below, the serial DMA channels function identically to those provided for the Ethernet MACs. Note that each serial DMA channel supports only a single chain or ring. For a full description of DMA configuration and programming, refer to Section 7: “DMA” on page 147.

The line interface unit controls the signals connecting the part to external serial devices, which might range from codecs to simple line drivers to sophisticated external framers. The line interface is described in the next section.

The protocol engine provides the link-level processing. It maps between DMA’s byte streams and the line interface’s bit streams according to the selected protocol. Section: “Framing Parameters” on page 344 and Section: “HDLC Transmitter” on page 344 describe its operation for the HDLC and transparent protocols respectively.
Figure 67: Synchronous Interface Block Diagram
LINE INTERFACE

The Line Interface controls the flow of bits between the protocol engine and the external pins. In the receive direction it assembles a bit stream and clock for the protocol engine. In the transmit direction it fetches bits from the protocol engine and formats them for the line.

The serial channels share pins with the DUARTs. Each port can independently be configured for synchronous serial operation or for UART operation using reset time configuration resistors or by software (see Section: "Reset" on page 26). When port 0 is configured for synchronous mode, it is connected to channel 0; similarly, port 1 to channel 1.

The 8 pins for each interface are used as follows in synchronous mode.

Table 223: Synchronous Serial Interface Signal Names

<table>
<thead>
<tr>
<th>Pin</th>
<th>Direction</th>
<th>Synchronous serial port Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUT</td>
<td>Output</td>
<td>Transmit data output</td>
</tr>
<tr>
<td>CTS_TCLKIN</td>
<td>Input</td>
<td>Transmit clock input</td>
</tr>
<tr>
<td>TIN</td>
<td>Input</td>
<td>Transmit enable or frame synch input</td>
</tr>
<tr>
<td>RTS_TSTROBE</td>
<td>Output</td>
<td>Transmit strobe output</td>
</tr>
<tr>
<td>DIN</td>
<td>Input</td>
<td>Receive data input</td>
</tr>
<tr>
<td>CIN_RCLKIN</td>
<td>Input</td>
<td>Receive clock input</td>
</tr>
<tr>
<td>RIN</td>
<td>Input</td>
<td>Receive enable or frame synch input</td>
</tr>
<tr>
<td>COUT</td>
<td>Output</td>
<td>Baud rate generator output</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that the configuration must be set in the UART control block, in order for the baud clock to be output on this pin.</td>
</tr>
</tbody>
</table>

In addition, the following GPIO pins can be driven from the serial port when the appropriate reset time configuration resistors select this use (see Section: "Reset" on page 26).

Table 224: Synchronous Serial Interface GPIO Pins

<table>
<thead>
<tr>
<th>Pin</th>
<th>Direction</th>
<th>Channel 0</th>
<th>Channel 1</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSTROBE</td>
<td>Output</td>
<td>GPIO[0]</td>
<td>GPIO[1]</td>
<td>Receive Strobe Out</td>
</tr>
</tbody>
</table>
INPUT LINE INTERFACE

In the receive direction, data on pin DIN is sampled either by an external clock signal, supplied on the CIN_RCLKIN pin, or by the internal baud rate generator. The internal baud clock can be made available externally on the COUT pin by setting the appropriate bit in the duart_opcr register (Table 210 on page 330) in the UART control block. Regardless of source, the polarity of the sampling clock edge is programmable. The clock configuration is done in the ser_clk register.

There are three ways that data bits are qualified to determine if they should be accepted and sent as part of the bit stream to the protocol engine.

1. **Gapped Clock**: If the external device is supplying the clock on CIN_RCLKIN, it can omit clock pulses. Since the BCM1250 never receives a clock pulse this method can always be used to suppress the reception of bits.

2. **External Enable**: Regardless of clock source, the RIN pin can be supplied with an externally generated enable signal. This can qualify the current data bit or may be delayed by 1, 2 or 3 clocks.

3. **Internal Sequencer**: Regardless of clock source, but exclusive with (2), the enable signal can be generated by an internal sequencer, which is itself synchronized to the data stream by a pulse on RIN. The sequencer can also provide a strobe on the RSTROBE output.

The bit stream delivered to the protocol engine consists of the bits sampled during the enabled bit times; clock edges occurring during disabled bit times are suppressed and not seen by the protocol engine.

**Input Using an External Enable**

An external enable signal can be provided on the RIN pin. This can be configured to be active high or low, and indicates valid data when active. The Data (on DIN) and enable (on RIN) are latched on the same edge of the clock. In the simplest case the enable qualifies the data bit that is latched at the same time, but the enable can also be delayed to qualify the data 1, 2 or 3 clocks later.

![Figure 68: Example Reception Using RIN as Active High Enable (sampling on the falling clock edge)](image)

The example in Figure 68 shows data and an active high enable being latched on the falling edge of the clock. The enable becomes inactive during cycle 0. If the enable delay is set to 0 then the data bit at time 0 will be ignored and the others will be accepted. If the enable delay is 3, the data bit at time 3 is skipped, while data bits at times 4, 5 and 6 are accepted (the example does not show the falling edges of cycles -3, -2, and -1 so it is not possible to determine from this figure if the data at bit times 0, 1, or 2 will be accepted).
The enable signal is level sensitive when used to enable the data. In transparent mode (see Section: “Operation in Transparent Mode” on page 348) the enable signal is used to frame the data, this can be selected as being edge (inactive to active) or level based, but only the edge based framing is likely to be useful.

RSTROBE is not used when the interface is used for an external enable. It is best to configure the pin as a GPIO. If the pin is configured as RSTROBE it will remain deasserted.

Input Using the Internal Sequencer

An internal sequencer can be used to qualify received data bits. A user-configured table is used to generate the internal enable signal and an optional external strobe signal. An external framing pulse provided on the RIN pin is used to synchronize traversal of the table with the data stream. The table consists of up to 16 entries, each with the format shown in Table 225.

Table 225: Sequencer Table Entries

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Last</td>
<td>Indicates current entry is the last entry of the table.</td>
</tr>
<tr>
<td>1</td>
<td>Bit/Byte</td>
<td>0: Bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Byte.</td>
</tr>
<tr>
<td>5:2</td>
<td>Count</td>
<td>One less than the number of bits/bytes controlled by this entry</td>
</tr>
<tr>
<td>6</td>
<td>Enable</td>
<td>Enables reception of the current data bit.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Enable</td>
</tr>
<tr>
<td>7</td>
<td>Strobe</td>
<td>Selects the value put on the external strobe pin while this entry is in use. The ser_mode register is used to select if the strobe is active high or low.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Deassert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Assert</td>
</tr>
</tbody>
</table>

Each entry controls the behavior of the line interface for a number of bit times equal to Count+1 if Bit/Byte is 0, or to 8*(Count+1) if Bit/Byte is 1. During those bit times, DIN is sampled and processed on each clock edge if Enable is 1; DIN is ignored and no data is sent to the protocol engine if Enable is 0. If the RSTROBE pin is enabled then it will be driven with the value of Strobe bit in the entry.

The synchronization pulse on RIN is latched on the same clock edge as the data on DIN. The synchronization pulse is delayed by 0, 1, 2 or 3 clocks. The edge_det bit in the ser_mode register selects either the active level or the inactive to active edge of the delayed pulse as the start signal for the sequencer. If the sequencer is currently idle it will reset to map table entry zero when started, and the enable bit in that entry becomes effective immediately. Table entries are thereafter processed in order until encountering an entry with the Last indicator set.

After the last entry of the table is processed, the line interface unit waits for the next assertion of delayed RIN before it restarts with entry 0. During any interval between the end of the table and reassertion of RIN, DIN is not sampled.

Once the table scan has started the start signal is ignored. If an active level or edge is detected on the delayed RIN signal it will be flagged as an rx_sync_error in the ser_status register and it will not affect the sequencer. (This may not be an error on some interfaces, for example if the sync is marked as level sensitive but lasts more than one cycle.)
The example in Figure 69 shows an active high sync pulse. If the delay is 0, map entry 0 determines whether data bit 0 is accepted or rejected. If that entry spans a single bit time, map entry 1 controls the disposition of bit 1. Alternatively if the delay is configured as one and entry 0 spans 14 byte times, it determines the treatment of bits 1 through 112.

![Figure 69: Example Reception Using RIN as Active High Sync (sampling on the falling clock edge)](image)

**OUTPUT LINE INTERFACE**

In the transmit direction, data is driven on pin DOUT either by an external clock signal, supplied on the CTS_TCLKIN pin, or by the internal baud rate generator. The internal clock can be made available externally on the COUT pin if enabled by setting bit 1 (channel 0) or bit 3 (channel 1) in the UART output port configuration register (duart_opcr). The polarity of the transitioning clock edge is programmable. The DOUT pin will be high impedance while the interface is disabled, so a pull-up may be required.

The output bits are optionally gated by means similar to the input:

1 **Gapped Clock**: If the external device is supplying the clock on CTS_TCLKIN, it can omit clock pulses. Since the BCM1250 never receives a clock pulse this method can always be used to suppress the transmission of bits.

2 **External Enable**: Regardless of clock source, the TIN pin can be supplied with an externally generated enable signal. This is latched on the active edge of the clock and disables output 0, 1, 2 or 3 clocks later. The DOUT pin is high impedance (undriven) when disabled.

3 **Internal Sequencer**: Regardless of clock source, but exclusive with (2), the enable signal can be generated by an internal sequencer, which is itself synchronized to the data stream by an edge or level on TIN. The sequencer can also provide a strobe on the TSTROBE output.

The bit stream from the protocol engine is output on the DOUT pin on every enabled clock edge. During cycles where the output is disabled DOUT is tri-stated following the clock edge and will be driven again following an enabled clock edge.

**Output Using an External Enable**

The external enable on the TIN pin can be configured to be active high or low. The signal is delayed by 0 to 3 clock edges. If the delay is zero then the output is immediately enabled and will drive the current output data, provided the enable signal is still asserted on the next active edge of the clock the data will change and new data will be output. If the delay is nonzero the enable is sampled on the same edge of the clock that is used to drive DOUT. TIN must be stable for at least a set-up time prior to that clock edge (in the usual case it will transition on the opposite edge). When output is disabled, DOUT is tri-stated, and the bit stream supplied by the protocol engine does not advance.
The TSTROBE pin is used as a data valid indicator when an external enable is used. The strobe_active bit in the ser_mode register sets the active level that TSTROBE will have when DOUT is valid. If the TIN enable signal is always active (or is only changed between packets) then TSTROBE will frame the packet. The protocol engine will ensure there is at least one bit time of gap between packets.

In the example in Figure 70 TIN is sampled on the rising edge of the clock and DOUT changes as a result of the rising edge of the clock. The delay is set to 1 so the enable is sampled on the rising clock edge. Since the enable is removed at the start of cycle 0 and 2, DOUT is tri-stated during bit times 0 and 2 and driven during bit times 1 and 3. (A similar figure would result if the delay is 3, removal of the enable during cycle 0 and 2 will result in DOUT being tri-stated during bit times 2 and 4 and driven at times 3 and 5).

![Figure 70: Example Transmission Using TIN as Active High Enable (Driving/Sampling on Rising Clock Edge)](image)

**Output Using the Internal Sequencer**

The transmitter has its own serial sequencer and its own table for generating the enable signal. The table is used to generate the internal enable signal and an external strobe signal. An external framing pulse provided on the TIN pin is used to synchronize traversal of the table with the data stream. The table consists of up to 16 entries, each with the same format as the receive table shown in Table 225 on page 340.

Each entry controls the behavior of the line interface for a number of bit times equal to Count+1 if Bit/Byte is 0, or to 8*(Count+1) if Bit/Byte is 1. During those bit times, data is accepted from the protocol engine and sent on DOUT on each clock edge if Enable is 1; DOUT is set high impedance and no data is extracted from the protocol engine if Enable is 0. The TSTROBE pin is driven with the value of Strobe bit in the entry.

The synchronization pulse on TIN is delayed by between 0 and three active edges of the clock. The edge_det bit in the ser_mode register selects either the active level or the inactive to active edge of the delayed pulse as the start signal for the sequencer. If the delay is zero the sequencer is immediately signalled to start (and the data and enable outputs will transition accordingly) the sequencer will start synchronously to the first, second or third clock edge after the synchronization event. If the sequencer is currently idle it will reset to map table entry zero when started, and the enable bit in that entry becomes effective immediately. Table entries are thereafter processed in order until encountering an entry with the Last indicator set.

After the last entry of the table is processed, the line interface unit waits for the next start signal from the delayed TIN before it restarts with entry 0. During any interval between the end of the table and reassertion of TIN, DOUT is high impedance.

Once the table scan has started the start signal is ignored. If an active level or edge is detected on the delayed TIN signal it will be flagged as a tx_sync_error in the ser_status register and it will not affect the sequencer. (This may not be an error on some interfaces, for example if the sync is marked as level sensitive but lasts more than one cycle.)
The example in Figure 71 has the sync pulse set at time 0. The map table will be reset to entry 0 after the configured delay and then determine when DOUT is driven. In the example the delay is one (sync is sampled on the clock edge) and the data is driven for at least the first four bit times.

![Clock Time: TIN/Sync DOUT](image)

**Figure 71: Example Transmission Using TIN as Active High Sync (transition/sampling on rising clock edge)**

**SYNCHRONOUS SERIAL PROTOCOL ENGINE**

The protocol engine converts between the bit streams used by the line interface and the packets transferred by the DMA engines. It has two modes, configured in the `ser_mode` register. In HDLC mode frames are encoded on the bit stream using the HDLC protocol. In transparent mode the bit stream is packed into bytes and the framing is based on the line interface Enable signal.

**OPERATION IN HDLC MODE**

In the HDLC mode the frame structure used by the DMA engines is converted into the HDLC form on the line. The protocol engine can perform the following functions:

- insertion and deletion of HDLC flags
- bit stuffing and de-stuffing
- CRC calculation and checking
- address filtering (optional)
- frame length checking - padding of short frames to a configured minimum frame size (optional)

Table 226 shows the HDLC frame structure (lengths in bytes).

<table>
<thead>
<tr>
<th>Flag (U) (01111110)</th>
<th>Address (1 to 2)</th>
<th>Control/Data (1 to MAX)</th>
<th>CRC (2 or 4)</th>
<th>Flag/Abort (1) (01111110) or (01111111)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bit Stuffed</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 226: HDLC Frame Structure*
The address, control and data fields are of variable length. The lengths and formats of addresses vary among the various link-level protocols that use bit synchronous HDLC-like framing, but lengths of 0, 1 or 2 bytes are typical. A CRC is appended on transmission and is checked on reception. The CRC can be either 16 bits or 32 bits. The CCITT CRC algorithms are used. The generator polynomials are:

\[
\text{CRC-CCITT: } x^{16} + x^{12} + x^5 + 1
\]

\[
\text{CRC-32: } x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1
\]

CRC-32 is also known as CRC-CCITT-32. Both alternatives use a CRC preset of all 1’s and send the resulting CRC in complemented form.

On the physical link, HDLC flags (8'b01111110) delimit frames. A flag must precede and follow the concatenation of the Address, Control, Data and CRC fields of a frame. Bit-stuffing prevents any occurrence of the flag pattern within those fields; the transmitter inserts a 0-bit after any sequence of 5 consecutive 1’s, and the receiver performs corresponding 0 deletion.

When the same flag terminates one frame and begins the next, it is called a shared flag. A configuration option controls the minimum number of additional flags between frames. During idle periods, the transmitter sends either consecutive Flags or will send an idle as zero followed by fifteen 1s and then the output will go high impedance. Consecutive flags are logically equivalent to a single flag and do not delimit frames of length 0. Frames can also be terminated by HDLC Abort or Idle patterns (a zero bit followed by at least 7 ones). Abort and Idle termination are not distinguished by the receiver but are reported as aborted frames.

Bytes are sent and received in order of increasing address according to the system endian mode. Within each byte, except for the CRC, the least significant bit is sent or received first.

**Framing Parameters**

The parameters controlling the format of frames are set in various configuration registers. The `ser_mode` register selects the CRC used, the minimum number of flags between frames, and the pattern sent on an idle line. The `ser_minfrm_sz` and `ser_maxfrm_sz` registers set the minimum and maximum frame size respectively. The minimum should be set to the smallest size permitted for the Address, Control and Data fields prior to bit-stuffing. The maximum should be set to the largest size permitted for the total sizes in bytes of the Address, Control, Data and CRC fields prior to bit-stuffing.

**HDLC Transmitter**

The user defines a frame to be transmitted by constructing the Address, Control and Data fields in a DMA buffer or chain of such buffers. For details of DMA buffers and descriptors, see Section 7: “DMA” on page 147.

For transmit, the DMA option flags shown in Table 227 are supported.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>append CRC</td>
<td>1'b0</td>
<td>If this bit is set the computed CRC will be appended to the packet.</td>
</tr>
<tr>
<td>2</td>
<td>append PAD</td>
<td>1'b0</td>
<td>If this bit is set the packet will be padded to the minimum packet size.</td>
</tr>
<tr>
<td>3</td>
<td>abort</td>
<td>1'b0</td>
<td>If this bit is set the packet will be ended with an abort instead of a standard flag.</td>
</tr>
</tbody>
</table>
In most applications, the protocol engine will compute and append a CRC. If DMA option append_CRC is not set, the user can supply a CRC as part of the DMA buffer.

When the number of empty entries in the TxFIFO exceeds a configurable threshold (ser_tx_wr_thres), the DMA engine will begin to write frame data into the TxFIFO. The start and end of the frame are specially marked for the protocol engine.

Once the number of bytes transferred to the TxFIFO exceeds another configurable threshold (derived from ser_tx_rd_thres) or the end of frame has been written to the TxFIFO, processing of the new frame by the protocol engine can begin.

First the protocol engine ensures that a user-defined number of flags, set in the flag_num field of the ser_mode register, have been sent prior to the opening flag of the new frame. A flag_num of 4'b0 indicates that the closing Flag of one frame can be reused as the opening flag for the next.

The user frame is then read from the TxFIFO into the protocol engine. An opening Flag is automatically prepended and bit-stuffing is performed on the bytes supplied by the DMA.

If the transmit module empties the TxFIFO but the user frame has not completed, an underrun error is reported in the tx_underrun bit in the ser_status register. When this occurs, the transmit module terminates the current frame with an Abort sequence. It then begins to send Flag or Idle depending on the setting of flag_en. The value of ser_tx_rd_thres should be adjusted to minimize the likelihood of such underruns.

During frame transmission, the protocol engine always calculates a CRC, either the CRC-CCITT or CRC-32 as determined by the crc_mode bit in the ser_mode configuration register. The bit-stuffed CRC is inserted before the closing flag if the append_CRC option is set in the DMA descriptor. The transmitted CRC is always compared to the calculated CRC. If the CRC was automatically generated, the two necessarily match. If the CRC was supplied by the user and it does not match the calculated one, a transmit CRC error is reported in the tx_crcerr bit of the ser_status register.

In addition, the protocol engine keeps track of the number of bytes that are sent for the current frame. If the number of bytes provided in the user frame is smaller than ser_min_frm_sz and the append_PAD option is set in the transmit descriptor, the protocol engine automatically adds zero padding to the user frame. When padding occurs, the protocol engine also appends its calculated CRC.

Finally, a trailing Flag is automatically appended to the bit-stuffed frame. If DMA option abort was set, an Abort is sent instead of the final Flag.

The serialized bit stream is passed to the line interface as allowed by the transmit gating signals. When output to that interface is disabled, the serial stream does not advance; no bits are discarded. The output bit stream is not otherwise aligned or correlated with transitions on pin TIN, whether used directly as the enable signal or as synchronization for the serial sequencer.

No status is returned upon completion of DMA, but a serial DMA interrupt may be requested by setting the interrupt or pkt_int option. CRC and underrun errors set the flags tx_crcerr and tx_underrun_error in the ser_status register respectively. These flags are cumulative. If the corresponding bit in the ser_err_mask register is set, they also trigger a serial device interrupt. Reading the ser_status register resets these flags to 0 and clears the interrupt condition.
The transmitter can be temporarily paused (for example as the result of a flow control request) by writing the tx_pause bit in the ser_cmd register. This causes the transmitter to complete sending the current packet and then suspend operation until re-enabled. The next packet will wait in the TxFIFO while the transmitter is paused. The tx_pause_complete flag in the ser_status register will be set to acknowledge completion of a tx_pause command. The flag is set immediately if there is no packet in flight when the command is issued, otherwise it will be set when the end of the current packet is moved from the TxFIFO into the transmit module.

When the transmit module is idle, either Flag (octet synchronous) or Idle (bit synchronous) can be sent onto the channel as configured by flag_en in the ser_mode register.

**HDLC Receiver**

In HDLC mode, frames within the bit stream are self-identifying. The Protocol Engine monitors the input bit stream supplied by the line interface. Frame recognition begins with a Flag not followed by another Flag, an Abort or an Idle.

The receive module first removes any bit stuffing to extract the bytes of the frames delimited by the opening and closing flags.

The first step in processing is frame filtering based on the HDLC address. The bit mask in the ser_addr_mask configuration register selects address bits from within the first two bytes following the opening Flag of each frame. A frame is accepted if its masked address bits match any of the four address registers, ser_usr0_addr through ser_usr3_addr, under the same mask. Otherwise the frame is rejected and not sent to the RxFIFO. Figure 72 shows the address matching logic.

![Figure 72: Frame Address Matching](image-url)
Setting **ser_addr_mask** to all zeros disables address filtering and allows reception of all frames. Link protocols with single-byte addresses should specify masks with all zeros in the most significant byte (i.e. 16’h00ff). When the number of acceptable addresses is less than four, the remaining address registers should be filled with copies of a valid address.

The recovered bytes of the Address, Control, Data and CRC fields of the frame are written, with bit-stuffing removed, into RxFIFO. If the RxFIFO becomes full in the middle of receiving a frame, the receive module discards all except the last word received when the full condition occurs. When space in the RxFIFO subsequently becomes available, this last word held is written into the FIFO with status overrun bit set. Any frames for which an opening Flag is received while the RxFIFO is full are ignored and not reflected in any status bits.

Received frames are moved from the RxFIFO to DMA buffers in memory by the serial DMA engine. The beginning and end of frame are specially marked by the protocol engine. DMA transfer begins when the number of entries in the RxFIFO exceeds a configurable threshold (**ser_rx_rd_thres**) or when an End of Frame is available in the RxFIFO.

The packet is checked for errors during reception. Indications of any errors are passed to the RxFIFO and eventually written back into the DMA descriptor status field.

For each frame, the CRC is computed using CRC-CCITT or CRC-32, as selected by the crc_mode in the **ser_mode** register. The computed CRC is compared with the value in the packet. If the two do not match, crc_error is signaled to the RxFIFO.

A frame that terminates with an Abort or Idle sequence is forwarded to the RxFIFO marked with an abort error flag.

The receive module also checks for a frame with a length (excluding opening and closing Flags) that exceeds the maximum length programmed in the **ser_maxfrm_sz** register. For such a frame, data up to the maximum length allowed is forwarded to the RxFIFO with an indication of a long frame error.

Similarly, the receive module checks for a frame that is shorter than the minimum length programmed in **ser_minfrm_sz** and sends a short frame error when forwarding the data to the RxFIFO. Note that frames of zero length are not recognized as such; a sequence of consecutive flags is equivalent to a single flag for the purpose of delimiting frames.

The receive module also checks for frames that are not-octet aligned, i.e., frames in which the number of bits between flags is not a multiple of 8 after any bit-stuffing is removed. These frames are forwarded to the RxFIFO with octet_error set. The dribble bits are recorded in a final byte that is transferred and included in the reported frame length.
Upon completion of the DMA transfer, any error indications forwarded through the RxFIFO are reported as status bits in the first descriptor for the corresponding frame. Table 228 lists the DMA status bits that are used by the receiver's serial DMA channel.

Table 228: Status Flags for Synchronous Serial Receive Channel

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>55:50</td>
<td>reserved</td>
<td>6'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>crc_error</td>
<td>1'b0</td>
<td>This bit is set if the received packet has a bad CRC.</td>
</tr>
<tr>
<td>57</td>
<td>abort</td>
<td>1'b0</td>
<td>This bit is set if the received packet ended with the abort flag.</td>
</tr>
<tr>
<td>58</td>
<td>octet_error</td>
<td>1'b0</td>
<td>This bit is set if the size of the received packet was not a multiple of 8 bits.</td>
</tr>
<tr>
<td>59</td>
<td>longframe_error</td>
<td>1'b0</td>
<td>This bit is set if the size of the received packet is bigger than the maximum frame size.</td>
</tr>
<tr>
<td>60</td>
<td>shortframe_error</td>
<td>1'b0</td>
<td>This bit is set if the size of the received packet is shorter than the minimum frame size.</td>
</tr>
<tr>
<td>61</td>
<td>overrun_error</td>
<td>1'b0</td>
<td>This bit is set if the received packet overran the FIFO and is therefore invalid.</td>
</tr>
<tr>
<td>62</td>
<td>good</td>
<td>1'b0</td>
<td>This bit is set if the received packet has no errors.</td>
</tr>
<tr>
<td>63</td>
<td>SOP</td>
<td>1'b0</td>
<td>This bit is set to indicate the start of the packet and the other bits are valid. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.</td>
</tr>
</tbody>
</table>

In addition, a cumulative error summary is maintained in the `ser_status` register. The receive errors are `rx_crc_error`, `rx_abort`, `rx_octet_error`, `rx_longframe_error`, `rx_shortframe_error` and `rx_overrun error`. When enabled by the corresponding bit in `ser_err_mask`, any set bit in `ser_status` generates a request for a serial device interrupt. Reading the `ser_status` register sets all bits to zero and clears the interrupt condition.

**OPERATION IN TRANSPARENT MODE**

In transparent mode, bit streams are sent and received without modification. Frames are not self-identifying, but a frame structure can be imposed on the bit stream by external synchronization signals. For both transmit and receive, serialization of each byte can be either least significant bit first or most significant bit first, according to the setting of `msb_first` in the `ser_mode` configuration register. Bytes are transmitted and received in order of increasing address according to the system endian mode.

In transparent mode, the following functions may be performed:

- address matching (optional)
- padding of short frames to a configured minimum frame size (optional)
- CRC calculation and checking (optional)

In transparent mode, the data is framed by implicit start and stop indications at the bit level. The details depend upon configuration of the line interface.

**Note**

The last time-slot specified in the user-defined table should always be enabled.
Transmitter in Transparent Mode

In Transparent mode, the protocol engine does not perform bit-stuffing or Flag/Abort insertion. Thus the DMA abort option is ignored. The append_CRC and append_PAD options remain available.

The TIN signal is used as described in Section: “Output Line Interface” on page 341 either as a level sensitive enable or to synchronize the table driven output sequencer. The transmit module will detect the start of packet indicator from the DMA engine as data comes out of the TxFIFO and align it to the line. If TIN is used as an enable, the edge_det bit in the ser_mode register selects if the start of a packet will be aligned with the next inactive to active edge of enable or the next active level (allowing packets to go our back-to-back).

If TIN is used as a synchronization pulse the start of a packet will be aligned with the first entry of the sequencer table being selected, which can be based on either the edge of the pulse or its level. These two rules ensure that packet boundaries in the DMA stream get aligned with the frame boundaries used on the line. If there are any idle cycles while a new frame is being aligned the DOUT pin will be set high impedance.

In other respects, operation is identical to operation of the transmitter in HDLC mode.

Receiver in Transparent Mode

Transparent mode operates similarly to HDLC mode except that neither Flag detection/deletion nor removal of bit stuffing is performed. CRC checking and address filtering remain available. Note that since there is no way to disable the CRC check, device drivers for protocols that do not have CRCs must ignore the CRC error flag (and therefore cannot use the good packet bit) in the DMA descriptor status information.

Frames are delimited by transitions of the gating signals in each direction. If there are no transitions, transfers make no progress.

If an external enable signal is used, it serves as an envelope delimiting the frame. The initial bit in the frame is the one in the first bit time after transition of the (optionally delayed) enable signal to its active level. The final bit in the frame is the bit preceding the opposite transition.

If the serial sequencer is used, the table is traversed exactly once to delimit the frame. The synchronization pulse (optionally delayed) will start processing bits from entry zero of the table and mark the start of a new frame. The first bit in the received frame will therefore be the first bit that is enabled in the table. The final bit in the frame will be the last bit in the last entry of the table with the Enable bit set, following reception of this bit the sequencer will suspend until the next sync pulse and the packet will be complete. A sync pulse that occurs during traversal of the table will be logged as an rx_sync_error in the ser_error register but is otherwise ignored.

In other respects, operation is identical to operation of the receiver in HDLC mode.
SYNCHRONOUS INTERFACE CONFIGURATION

There are four sections that must be configured before a serial channel can be used. These are: General Control, FIFO control, Protocol Engine/Line Interface Configuration and Address Filtering. In addition, the serial DMA channels must be configured.

At system reset, the transmit and receive modules in the Serial Link Interface are both disabled. They can be individually enabled by writing to the `ser_cmd` register. Note that the `ser_cmd` register is write-only.

Resetting the transmit/receive module will put it into disabled state and flushes the TxFIFO/RxFIFO. However, configuration registers are not restored to their default values. Configuration registers should be written only when the corresponding modules are disabled.

These modules need to be enabled after a reset by writing ones to the Receive/Transmit Enable fields in the `ser_cmd` registers.

DMA CONFIGURATION

Refer to Section 7: “DMA” on page 147 for configuring and initializing the serial DMA channels.

FIFO CONFIGURATION

TxFIFO is a 64 bit wide FIFO with 8 entries. The `ser_tx_wr_thres` register sets the number of empty 64 bit entries that must be in the TxFIFO before it will request DMA data. Since the DMA engine fetches in blocks of 32 bytes, this value must be set to 4 entries.

To reduce the likelihood of TxFIFO becoming empty during transmission of a frame, `ser_tx_rd_thres` should be set to ensure a certain number of entries (8 bytes each) have been written before the Protocol Engine starts.

RxFIFO is a 32 bit wide FIFO with 16 entries. The `ser_rx_rd_thres` register sets the number of valid entries that must be in the FIFO to request emptying by DMA. Since the DMA Engine transfers in blocks of 32 bytes, this value should be set to 8 entries.

TxFIFO and RxFIFO can be enabled or reset using the `ser_cmd` configuration register controls. Threshold values should only be changed when the corresponding FIFO is reset.

PROTOCOL ENGINE CONFIGURATION

The protocol engine is configured in the `ser_mode` register. In addition the receive address mask and filter must be set in the `ser_addr_mask` and `ser_usrN_addr` (N=0,1,2,3) registers. If address filtering is not required the mask should be set to zero and all frames will be received.
LINE INTERFACE CONFIGURATION

The line interface is configured in the `ser_mode` and `ser_line_mode` registers. On reset the clock source defaults to use the internal baud rate clock, ensuring that state is correctly cleared. Since changes to the line mode register can change the clocking of the interface block care must be taken when it is written: all previous configuration requests must have completed before the `ser_line_mode` register is written, and no additional registers should be written until the `ser_line_mode` register write has completed. Any requests that are in progress close in time to when the `ser_line_mode` register is written will have UNPREDICTABLE results. The recommended way to safely change the `ser_line_mode` register is to first read the `ser_mode` register and use the result (or use a SYNC) then write the `ser_line_mode` register and read it back and again use the result.

SYNCHRONOUS SERIAL INTERRUPTS

The DMA interrupts for each channel are combined with status interrupts, and made available in the `ser_status` register. Bits in this register are masked by the `ser_int_mask` register and combined to generate the system interrupts 10 (for channel 0) and 11 (for channel 1). Reading the `ser_status` register will clear all bits in it. For a description of DMA interrupts associated with the serial channels, please refer to Section7: “DMA” on page 147.

To allow debuggers non-intrusive access to the status register, it is also made available through the `ser_status_debug` register, which does not clear the status on read.

SYNCHRONOUS SERIAL LOOPBACK

There are two loopback configurations in the synchronous serial port. In both cases the transmitter output is connected internally to the receiver input. The first configuration relies on external timing pulses and uses TIN and RIN as normal. The second configuration has TIN always active and internally connects RIN to the RTST_STROBE output. These are shown in Figure 73. When the second loopback mode is used the line interface configuration for both tx and rx can be set to the values in Table 229 on page 352 to allow a table driven loopback.

![Figure 73: Synchronous Serial Loopback Connections](image-url)
RMON Counters

The serial interface maintains some counters that are useful for gathering RMON statistics. The counters are all 16 bits wide apart from the byte counters which are 32 bits wide (but occupy two register slots). The registers may be cleared by software writing to them. If any counter overflows it will wrap to zero and continue to count.

Table 230: RMON Counters

<table>
<thead>
<tr>
<th>Number (offset from 00_1006_0000)</th>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tx Byte (low)</td>
<td>Total number of bytes transmitted. (Low 16 bits).</td>
</tr>
<tr>
<td>1</td>
<td>Tx Byte (high)</td>
<td>Total number of bytes transmitted. (High 16 bits).</td>
</tr>
<tr>
<td>2</td>
<td>Rx Byte (low)</td>
<td>Total number of bytes received. (Low 16 bits).</td>
</tr>
<tr>
<td>3</td>
<td>Rx Byte (high)</td>
<td>Total number of bytes received. (High 16 bits).</td>
</tr>
<tr>
<td>4</td>
<td>Tx Underflow</td>
<td>Total number of transmit packets aborted due to FIFO underflow.</td>
</tr>
<tr>
<td>5</td>
<td>Rx Overflow</td>
<td>Total number of receive packets dropped due to overflow.</td>
</tr>
<tr>
<td>6</td>
<td>Rx Error</td>
<td>Total number of packets received with errors (CRC error, abort flag, or Octet error).</td>
</tr>
<tr>
<td>7</td>
<td>Rx Addr Fail</td>
<td>Total number of receive packets dropped due the address not matching.</td>
</tr>
</tbody>
</table>
SYNCHRONOUS SERIAL REGISTER SUMMARY

All configuration and control registers are 16 bits.

Table 231: Serial Mode Configuration Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>crc_mode</td>
<td>1'b0</td>
<td>0: CRC-CCITT. 1: CRC-32.</td>
</tr>
<tr>
<td>1</td>
<td>msb_first</td>
<td>1'b0</td>
<td>0: Transmit/Receive LSB in each byte first. 1: Transmit/Receive MSB in each byte first. This bit is only used in transparent mode.</td>
</tr>
</tbody>
</table>

HDLC Related

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5:2</td>
<td>flag_num</td>
<td>3'b1</td>
<td>Minimum number of Flags sent between Back-to-Back frames if HDLC is enabled. A value of 0 indicates the trailing Flag of the previous frame is used as the Open Flag of the next frame.</td>
</tr>
<tr>
<td>6</td>
<td>flag_en</td>
<td>1'b0</td>
<td>0: Idle is sent when Transmit module is Idle. 1: Flag is sent when Transmit module is Idle.</td>
</tr>
<tr>
<td>7</td>
<td>hdlc_en</td>
<td>1'b1</td>
<td>0: Transparent mode. 1: HDLC encoding.</td>
</tr>
</tbody>
</table>

Loopback

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>loop_strobe</td>
<td>1'b0</td>
<td>Loopback Strobe mode. This configures the Sync/Enable inputs during loopback. 0: TIN and RIN are taken from the external source. 1: TIN is always active and RIN is internally connected to RtsTstrobe.</td>
</tr>
<tr>
<td>9</td>
<td>loop_data</td>
<td>1'b0</td>
<td>Loopback enable. If this bit is set the DIN input be internally connected to the DOUT output to allow loopback testing.</td>
</tr>
<tr>
<td>15:10</td>
<td>reserved</td>
<td>6'bx</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

Table 232: Synchronous Serial Clock Source and Line Interface Mode Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rx_clk_inv</td>
<td>1'b0</td>
<td>This bit controls the inverter on the receive clock. 0: Not inverted (clock on rising edge). 1: Inverted (clock on falling edge).</td>
</tr>
<tr>
<td>1</td>
<td>rx_clk_src</td>
<td>1'b0</td>
<td>Receiver clock source. 0: Internal clock, generated by the baud rate generator. 1: External clock from CiN_RCLKIN pin.</td>
</tr>
<tr>
<td>3:2</td>
<td>rx_sync_delay</td>
<td>2'b0</td>
<td>Number of clock delays between the Frame Sync/Enable and the first bit of the frame.</td>
</tr>
<tr>
<td>4</td>
<td>rx_sync_active</td>
<td>1'b0</td>
<td>0: Sync/Enable is active high. 1: Sync/Enable is active low.</td>
</tr>
<tr>
<td>5</td>
<td>rx_strobe_active</td>
<td>1'b0</td>
<td>0: Strobe in Line Interface is active high. 1: Strobe in Line Interface is active low.</td>
</tr>
</tbody>
</table>
## Table 232: Synchronous Serial Clock Source and Line Interface Mode Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| 6    | rx_edge_det         | 1'b0    | [7:6]These two bits set the interface synchronization method.  
00: External Enable is used using level as frame start.  
01: External Enable is used, using edge as frame start.  
10: Internal table drives enable, sync is level sensitive.  
11: Internal table drives enable, sync is edge sensitive. |
| 7    | rx_table_en         | 1'b0    | This bit controls the inverter on the transmit clock.  
0: Not inverted (clock on rising edge).  
1: Inverted (clock on falling edge). |
| 8    | tx_clk_inv          | 1'b0    | Transmitter clock source.  
0: Internal clock, generated by the baud rate generator.  
1: External clock from RCLKIN pin. |
| 11:10| tx_sync_delay       | 2'b0    | Number of clock delays between the Frame Sync/Enable and the first bit of the frame.         |
| 12   | tx_sync_active      | 1'b0    | 0: Sync/Enable is active high.  
1: Sync/Enable is active low. |
| 13   | tx_strobe_active    | 1'b0    | 0: Strobe in Line Interface is active high.  
1: Strobe in Line Interface is active low. |
| 14   | tx_edge_det         | 1'b0    | [15:14]These two bits set the interface synchronization method.  
00: External Enable is used using level as frame start.  
01: External Enable is used, using edge as frame start.  
10: Internal table drives enable, sync is level sensitive.  
11: Internal table drives enable, sync is edge sensitive. |
| 15   | tx_table_en         | 1'b0    |                                                                                             |
| 63:16| notimp              | 48'bx   | Not implemented.                                                                             |

## Table 233: Synchronous Serial Command Register (Write-only)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rx_en</td>
<td>Receive enable. When the receiver is reset, writing a one enables the receiver.</td>
</tr>
<tr>
<td>1</td>
<td>tx_en</td>
<td>Transmit enable. When the transmitter is reset or paused, writing a one enables the transmitter.</td>
</tr>
</tbody>
</table>
| 2    | rx_reset | Receive reset. Writing a one resets the receiver. This should only be done when the line is  
idle or being restarted, resetting an active interface can result in partial packet reception.  
Writing this bit will only reset the state machines; configuration registers are not returned to their default  
values. |
| 3    | tx_reset | Transmit reset. Writing a one resets the transmitter. This should only be done when the line is  
idle or being restarted, resetting an active interface can result in UNPREDICTABLE data and  
strobe signals being generated. Writing this bit will only reset the state machines; configuration  
registers are not returned to their default values. |
| 4    | reserved | Must be written as 0.                                                                         |
| 5    | tx_pause | Transmit pause. Writing a one causes the transmitter to pause at the end of the next packet.  
If the transmitter has no data it will stop immediately, otherwise it will continue sending until the  
next end of packet has been sent. The transmitter leaves any subsequent packet in the TxFIFO.  
When the transmitter is re-enabled (by writing a 1 to the tx_en bit) the next packet will be fetched  
from the TxFIFO and normal operation will continue. |
| 15:4 | reserved | Must be written as 0.                                                                         |
| 63:16| notimp   | Not implemented.                                                                             |
### Table 234: Serial Write Threshold Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>thrsh</td>
<td>4'b100</td>
<td>Number of free 64 bit entries the TxFIFO must have before it signals the DMA that space is free. This must be set to 4 for normal operation. Writing other values in this register is for Broadcom Use Only.</td>
</tr>
<tr>
<td>15:4</td>
<td>reserved</td>
<td>12'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 235: Serial Transmit Read Threshold Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>thrsh</td>
<td>4'b100</td>
<td>Number of filled 64 bit entries the TxFIFO must have before the protocol engine will start transmitting the frame. The FIFO is only 8 entries, so setting bit [3] will result in UNPREDICTABLE behaviour.</td>
</tr>
<tr>
<td>15:4</td>
<td>reserved</td>
<td>12'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 236: Serial Receive Read Threshold Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>thrsh</td>
<td>4'b1000</td>
<td>Number of valid 32 bit entries the RxFIFO must have before it signals the DMA to read data. This must be set to 8 for normal operation.</td>
</tr>
<tr>
<td>15:4</td>
<td>reserved</td>
<td>12'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 237: Serial Minimum Frame Size Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>size</td>
<td>16'b0</td>
<td>Minimum frame size in bytes.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 238: Serial Maximum Frame Size Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>size</td>
<td>0x100000</td>
<td>Maximum frame size in bytes.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>0x480</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 239: Serial DMA Enable Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rxdma_en</td>
<td>0x0</td>
<td>Receive DMA channel enable. Must be set to enable the channel.</td>
</tr>
<tr>
<td>3:1</td>
<td>reserved</td>
<td>0x3</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>txdma_en</td>
<td>0x0</td>
<td>Transmit DMA channel enable. Must be set to enable the channel.</td>
</tr>
<tr>
<td>7:5</td>
<td>reserved</td>
<td>0x3</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>0x560</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 240: Synchronous Serial Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rx_crcerr</td>
<td>Received frame with CRC error.</td>
</tr>
<tr>
<td>1</td>
<td>rx_abort</td>
<td>Received frame terminated by Abort or Idle.</td>
</tr>
<tr>
<td>2</td>
<td>rx_octet_error</td>
<td>Received frame length not a multiple of 8 bits.</td>
</tr>
<tr>
<td>3</td>
<td>rx_longframe_error</td>
<td>Received frame longer than maximum size.</td>
</tr>
<tr>
<td>4</td>
<td>rx_shortframe_error</td>
<td>Received frame shorter than minimum size.</td>
</tr>
<tr>
<td>5</td>
<td>rx_overrun_error</td>
<td>RxFIFO overrun.</td>
</tr>
<tr>
<td>6</td>
<td>rx_sync_error</td>
<td>Received a sync before the end of the receive table.</td>
</tr>
<tr>
<td>7</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>tx_crcerr</td>
<td>Sent frame with bad user-supplied CRC.</td>
</tr>
<tr>
<td>9</td>
<td>tx_underun_error</td>
<td>TxFIFO underrun.</td>
</tr>
<tr>
<td>10</td>
<td>tx_sync_error</td>
<td>Received a sync before the end of the transmit table.</td>
</tr>
<tr>
<td>11</td>
<td>tx_pause_complete</td>
<td>Set when emptying of the TxFIFO has stopped because of a TX pause command (sets immediately if no packet is in flight when the command is given, otherwise set after the end of packet has been moved from the FIFO to the transmitter).</td>
</tr>
<tr>
<td>15:12</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>rx_eop_count</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>17</td>
<td>rx_eop_timer</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
</tbody>
</table>
### Table 240: Synchronous Serial Status Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>rx_eop_seen</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>19</td>
<td>rx_hwm</td>
<td>Set if the high watermark interrupt is raised.</td>
</tr>
<tr>
<td>20</td>
<td>rx_lwm</td>
<td>Set if the low watermark interrupt is raised.</td>
</tr>
<tr>
<td>21</td>
<td>rx_dscr</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt on packet end command.</td>
</tr>
<tr>
<td>22</td>
<td>rx_derr</td>
<td>Set if the controller ran out of descriptors during a packet reception. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>23</td>
<td>reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>tx_eop_count</td>
<td>Set if the EOP interrupt was raised as a result of the packet count being reached.</td>
</tr>
<tr>
<td>25</td>
<td>tx_eop_timer</td>
<td>Set if the EOP interrupt was raised as a result of the packet timer triggered.</td>
</tr>
<tr>
<td>26</td>
<td>tx_eop_seen</td>
<td>Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).</td>
</tr>
<tr>
<td>27</td>
<td>tx_hwm</td>
<td>Set if the high watermark interrupt is raised.</td>
</tr>
<tr>
<td>28</td>
<td>tx_lwm</td>
<td>Set if the low watermark interrupt is raised.</td>
</tr>
<tr>
<td>29</td>
<td>tx_dscr</td>
<td>Set if the interrupt is triggered by a descriptor with the interrupt on packet end command.</td>
</tr>
<tr>
<td>30</td>
<td>tx_derr</td>
<td>Set if the controller ran out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>31</td>
<td>tx_dzero</td>
<td>Set if a descriptor has a packet length of zero. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.</td>
</tr>
<tr>
<td>63:32</td>
<td>notimp</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

**ser_status_0** - 00_1006_0588  
**ser_status_1** - 00_1006_0988  
**READ ONLY, Read Clears**

### Table 241: Serial Status Debug Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>status</td>
<td>32'b0</td>
<td>Reading this register gives the same value as reading the ser_status register, but does not have the side effect of clearing the register.</td>
</tr>
<tr>
<td>63:32</td>
<td>notimp</td>
<td>32'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

**ser_status_debug_0** - 00_1006_05A8  
**ser_status_debug_1** - 00_1006_09A8

### Table 242: Serial Interrupt Mask Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>mask</td>
<td>32'b0</td>
<td>Setting a bit in this register enables generation of an interrupt when the corresponding status bit is set in the ser_status register.</td>
</tr>
<tr>
<td>63:32</td>
<td>notimp</td>
<td>32'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

**ser_int_mask_0** - 00_1006_0590  
**ser_int_mask_1** - 00_1006_0990
### Table 243: Serial Address Mask Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>mask</td>
<td>16'b0</td>
<td>Each one bit selects the address bits in the received frame to be matched with corresponding bits in the address match registers. Bits 7:0 correspond to the first byte of the frame; bits 15:8 to the second. Set this to zero for no address filtering; set bits 15:8 to zero for single-byte address filtering.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 244: Serial Address Match Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>addr</td>
<td>16'b0</td>
<td>The destination address to be matched in all bit positions selected by the <code>ser_addr_mask</code> register. Bits 7:0 correspond to the first byte of the frame; bits 15:8 to the second.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 245: Sequencer Table Entries

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Last</td>
<td>Indicates current entry is the last entry of the table.</td>
</tr>
<tr>
<td>1</td>
<td>Bit/Byte</td>
<td>Selects the multiplier for Count to give the number of bit times described by this table entry. 0: Bit 1: Byte</td>
</tr>
<tr>
<td>5:2</td>
<td>Count</td>
<td>Number minus 1 of bits/bytes controlled by this entry.</td>
</tr>
<tr>
<td>6</td>
<td>Enable</td>
<td>Enable the serial line interface for the number of bit times described by this entry. For receive, ignore DIN during disabled bit times. For transmit, tristate DOUT during disabled bit times. 0: Disable 1: Enable</td>
</tr>
<tr>
<td>7</td>
<td>Strobe</td>
<td>Enable the output strobe for the number of bit times described in this entry. 0: Deassert 1: Assert</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 246: Serial RMON Counters

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>count</td>
<td>16'b0</td>
<td>Counter. Any write will zero the count.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

**Counters:***

- `ser_tx_byte_lo_0` - `00_1006_05C0`
- `ser_tx_byte_hi_0` - `00_1006_05C8`
- `ser_rx_byte_lo_0` - `00_1006_05D0`
- `ser_rx_byte_hi_0` - `00_1006_05D8`
- `ser_tx_underrun_0` - `00_1006_05E0`
- `ser_rx_overflow_0` - `00_1006_05E8`
- `ser_rx_errors_0` - `00_1006_05F0`
- `ser_rx_badaddr_0` - `00_1006_05F8`
This Page is left blank for notes
Section 11: Generic/Boot Bus

INTRODUCTION

The generic bus is used to attach the boot ROM and a variety of simple peripherals. Eight regions of memory are defined, each has its own chip select line, data width and set of timing parameters. All accesses to the generic bus range are accepted by the bus controller, a secondary decode determines which chip select region is used or raises an error. At reset time the generic bus address/data lines are read by the part to set static configuration options, so weak pull-up or pull-down resistors should be put on the board (see Section: “Reset” on page 26 for details of the configuration bits).

OVERVIEW

The generic bus is allocated a 767 MB block of the address space, from 00_1009_0000 to 00_3FFF_FFFF. Accesses to this address space cause transfers to take place on the generic bus pins. Eight regions are defined within the range by setting the base address and size of each region. Each region has an external active low chip select line, and a set of registers that set the transfer width and timing.

Six of the eight chip select regions are entirely free for individual devices. One (IO_CS_L[0]) is used for the boot Flash/ROM, and one (IO_CS_L[6]) is used for the PCMCIA device when it is enabled but is otherwise free. Each region can be configured to use either a 32 bit multiplexed address/data bus or an 8 bit wide data bus and 24 bit wide address bus.

After reset, region 0 is configured to map 4MB starting at the physical address location of the MIPS processor reset exception routine, 00_1FC0_0000 and ending at 00_1FFF_FFFF. Selection of a 32 bit multiplexed or 8 bit non-multiplexed bus for the boot region is made using the reset-time configuration options. An additional reset option diverts accesses made to region 0 to the SMBus interface to allow boot code to be fetched from an SMBus EEPROM (see Section: “Booting using an SMBus EEPROM” on page 412). The address range initially assigned to region 0 also covers the physical address range (00_1FD0_0000 to 00_1FD0_FFFF) that external PCI devices can access through the Expansion ROM BAR. The same ROM could therefore hold the boot code and PCI Expansion code.

The data width of each region that is configured for multiplexed address/data can be specified as 8-bit, 16-bit or 32-bit. When an access is made to a narrower region the generic bus interface logic will automatically perform repeated reads or writes and can transfer up to a cache line of data as a result of a single request.

The generic interface is big endian, thus an 8-bit device connects to the upper byte IO_AD[31:24], a 16-bit device connects to the upper word IO_AD[31:16], and a 32-bit device connects to the entire IO_AD[31:0]. In non-multiplexed mode the data pins connect to the upper byte of the IO_AD[31:24] and the address is provided on the lower three bytes IO_AD[23:0]. The PCMCIA bus and other little endian devices should have D[7:0] connected to IO_AD[31:24] and D[15:8] connected to IO_AD[23:16] to keep the even addressed byte lane lined up correctly (see Section12: “PCMCIA Control Interface” on page 383 for full details on connecting PCMCIA cards). An alternative way of putting this is that the upper byte IO_AD[31:24] is always the byte at the lowest memory address. This is true regardless of the internal system endian; the generic bus interface logic takes care of any swapping that is required. In alternate 8 bit multiplexed mode the data is transferred on IO_AD[7:0], allowing direct connection to peripherals that use an 8 bit multiplexed connection.
Table 247 shows the byte lane correspondences for the generic bus. These are discussed in more detail in the sections that follow.

**Table 247: Byte Lanes for the Generic Bus**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplex: ALE active / 8 bit device</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplex: ALE inactive / 16 bit big endian device</td>
<td>D[15:8]</td>
<td>D[7:0]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplex: ALE inactive / 8 bit PCMCIA</td>
<td>D[7:0] CE1#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplex: ALE inactive / 16 bit PCMCIA</td>
<td>D[7:0] CE1#</td>
<td>D[15:8] CE2#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplex: ALE inactive / 16 bit little endian devices</td>
<td>D[7:0]</td>
<td>D[15:8]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplex: Offsets i.e. A[1:0]</td>
<td>+0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
</tr>
<tr>
<td>Byte enable: Active low while ALE active (Valid for both reads and writes)</td>
<td>IO_AD[31]</td>
<td>IO_AD[30]</td>
<td>IO_AD[29]</td>
<td>IO_AD[28]</td>
</tr>
</tbody>
</table>

**CONFIGURING A CHIP SELECT REGION**

The properties of a chip select region on the generic bus are configured by writing a set of registers. There are eight sets, denoted by having _0 to _7 appended to the register names used in this discussion.

**ADDRESS RANGE**

The address range allocated to the generic bus is partitioned by configuring the base address and size of each region. The base address of the region always has bits [39:30] and [15:0] as zero, bits [29:16] are set in the `io_ext_start_addr` register. This allows the start address to be set on any 64KB boundary below 00_4000_0000. If the start address is set below the base address of the generic bus range (00_1009_0000) the region is disabled. The region start address is subtracted from addresses as they pass through the bus interface, so each device sees a contiguous block starting from zero.

The size of the region is set in the `io_ext_mult_size` register. The size of the region is 64KB * (`io_ext_mult_size`+1). Since only 12 bits of the register are valid, the minimum region size is 64KB, and the maximum size is 256MB. An alternative way of thinking about this is that the last address in the region has bits [39:30] all zero, bits [29:16] equal to `io_ext_start_addr + io_ext_mult_size` and bits [15:0] all ones. If the last address of the region is set above 00_3FFF_FFFF some of the region will not be accessible.
CACHEABLE ACCESS BLOCKING

To provide protection against programming errors and prevent side effects from accidental speculative accesses through kseg0 and xkphys (see section “CPU Speculative Execution” on page 16) cacheable reads may be blocked from being driven on the generic bus. This is done by setting the blk_cache bit in the io_ext_start_addr register. If this bit is set and a cacheable access is done to the region then the access will be blocked. This bit is clear by default and should be left clear for memory areas like the boot ROM that are likely to be cached for performance. To avoid causing the CPU to take an imprecise error as the result of a bad speculative access, a data error is not raised when a read is blocked. The access is just logged.

When a cacheable access is blocked:

1. No cycle is run on the generic bus.
2. For a read UNPREDICTABLE data is returned, marked with the valid data return code.
3. The io_cacheable_blk bit in the io_interrupt_status register is set (but no interrupt is raised). This bit is cleared by a read of the register.
4. If the log is not locked (i.e. the io_error interrupt is deasserted) the address of the access gets logged in io_interrupt_addr0 and io_interrupt_addr1.

GENERIC BUS PARITY

A reset time system configuration option enables parity on the data portion of a transfer on the generic bus (not the address). This is provided on the IO_ADP[3:0] pins, which are available as GPIO pins if generic bus parity is disabled. If parity is enabled for the system then each region can be configured in the io_ext_cfg register to have even, odd or no parity check.

On read transactions from parity generating devices, the parity bit of each byte of the incoming data is latched at the same time as the data, and checked against the parity computed for the byte. If an error is detected, then the address and region of the error are logged, the io_rd_par_int bit is set in the io_interrupt_status register, the generic bus error interrupt is raised and the data is passed to the system bus marked with a bus error. The data must be passed to the system bus to terminate the read transaction, the error flag will prevent its use. The Bus Watcher in the SCD (see section “Bus Watcher” on page 64) will note the error flag and increment the bus_io_error count. If the data is returning to one of the CPUs then the bus error exception will be raised (thus the CPU could be told about the error three times: the bus error exception, the bus watcher error interrupt and the interrupt from the generic bus controller).

When parity is enabled for a region it will be generated for the data in writes and will be driven on the parity pins with the same timing that the data has. If the device sees a parity error on writes it is device dependent how it responds and how it reports the error. Table 247 on page 362 shows how the IO_ADP pins match with the byte lanes.

Data with an uncorrectable error from the system bus will not be seen by the generic bus interface. The Bus Watcher will log and report the error.
BUS WIDTH

The data width for a region is set in the io_ext_cfg register. If a region is in multiplexed address/data mode it can be set to have a data bus 8 or 16 or 32 bits wide. Regions using non-multiplexed mode must always be configured for 8 bit widths. The interface will pack or unpack data as required. The generic bus is always ordered in a big endian manner, so the lowest memory location is on IO_AD[31:24]. If the region is configured as 16 bit wide the odd addresses will be on IO_AD[23:16], for 32 bit regions IO_AD[15:8] and IO_AD[7:0] are added to carry the bytes with a[1:0] equal to 2 and 3.

Writes to the generic bus can be 1-8 bytes direct from the CPU, 1-32 bytes from an uncached-accelerated merge from the CPU or Data Mover, or 32 bytes when a cache line (which must be cacheable non-coherent) is moved. The interface will order these and align them so that they are written on the generic bus in ascending address order (this is useful when writing to FIFOs on generic bus devices).

Reads from the generic bus will similarly be broken down into requests that match the width of the region. Again the requests for a single read will be made in ascending address order.

If a multiplexed address/data bus is used then the upper four bits IO_AD[31:28] are used during the address phase as byte lane enables. These active low signals are used when doing byte accesses to 16 or 32 bit regions, and for half-word accesses to 32 bit regions. The mapping from the byte enables to the byte lanes is shown in Table 247 on page 362.

GENERIC BUS TIMING

The timing of the generic bus is configurable for each of the chip select regions. When multiple devices are connected, care must be taken to ensure that the timing for one device does not confuse other devices. The chip select signals and idle time between cycles will normally be sufficient to ensure this.

Internally all the timing is done with reference to the 100 MHz clock (this clock is also used by the SMBus and serial interfaces). This clock will be output on the IO_CLK100 pin if enabled by the reset configuration resistor on IO_AD[23]. In most cases the clock is not needed externally and the interface is run as an asynchronous one using the computed timings given in the Hardware Data Sheet.

There are two modes for timing cycles. In fixed cycle mode the timing of the access is entirely controlled by the generic bus interface based on the parameters in the configuration registers. In acknowledgement based mode a ready/busy signal from the device controls the length of the access. The mode and active sense of the ready/busy signal is programmed in the io_ext_cfg register.

Table 248 on page 365 shows the parameters that can be set to control the cycle time. The next four sections have timing diagrams that show how these parameters control the cycle. Each timing diagram includes the IO_AD timing for both multiplexed and non-multiplexed operation. On reset the parameters for the IO_CS_L[0] region are suitable for a slow EPROM or flash memory (the parameters for the other regions do not get useful defaults). The parameters are programmed into the io_ext_time_cfg0 and io_ext_time_cfg_1 registers and are expressed in cycles of the 100 MHz reference clock (i.e. multiples of 10ns).
In the fixed mode, the cycle time is:

\[
\text{ale\_width} + \text{ale\_to\_cs} + \text{cs\_width} + 1 + \text{idle\_cycle}
\]

In the acknowledgement mode the minimum cycle time is:

\[
\text{ale\_width} + \text{ale\_to\_cs} + \text{cs\_width} + \text{rdy\_smple} + \text{oe\_to\_cs} + 1 + \text{idle\_cycle}
\]

---

<table>
<thead>
<tr>
<th>Name</th>
<th>Range (cycle = 10ns)</th>
<th>IO CS[0] Reset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ale_width</td>
<td>1 - 7 cycles</td>
<td>4</td>
<td>Width of the address latch enable pulse. This signal is asserted for this number of cycles to indicate the start of the cycle, the address and byte enables are output at the same time this signal asserts and remain stable after IO_ALE deasserts (i.e. IO_ALE is suitable for use as the latch input of a flow through latch, or the clock input of a falling-edge clocked flipflop).</td>
</tr>
<tr>
<td>ale_to_cs</td>
<td>1 - 3 cycles</td>
<td>2</td>
<td>Delay from IO_ALE being deasserted until assertion of the IO_CS_L line for the region addressed.</td>
</tr>
<tr>
<td>cs_width</td>
<td>1 - 31 cycles</td>
<td>24</td>
<td>The width of the chip select assertion in fixed cycle mode. In acknowledgement mode the IO_RDY line will not be sampled until this number of cycles have passed, but the chip select remains asserted until the IO_RDY signal has been asserted and rdy_smple+oe_to_cs cycles have passed.</td>
</tr>
<tr>
<td>rdy_smple</td>
<td>0 - 7 cycles</td>
<td>1</td>
<td>In acknowledgement mode this is the number of cycles from IO_RDY asserting to IO_WR_L or IO_OE_L deasserting (and data being latched in a read access). The chip select remains asserted oe_to_cs additional cycles.</td>
</tr>
<tr>
<td>idle_cycle</td>
<td>1 - 15 cycles</td>
<td>6</td>
<td>The number of cycles that the bus should be idle before the next IO_ALE. Note that the cycle does not end until one cycle after the IO_CS_L line has deasserted.</td>
</tr>
<tr>
<td>ale_to_wr</td>
<td>1 - 7 cycles</td>
<td>7</td>
<td>The number of cycles from IO_ALE deassertion to IO_WR_L assertion in a write cycle.</td>
</tr>
<tr>
<td>wr_width</td>
<td>1 - 15 cycles</td>
<td>7</td>
<td>In fixed mode this is the number of cycles that IO_WR_L is asserted in a write cycle. In acknowledgement mode this value is ignored and the IO_WR_L strobe deasserts rdy_smple cycles after the acknowledgement.</td>
</tr>
<tr>
<td>cs_to_oe</td>
<td>0 - 3 cycles</td>
<td>0</td>
<td>The number of cycles from IO_CS_L assertion to IO_OE_L assertion is a read cycle.</td>
</tr>
<tr>
<td>oe_to_cs</td>
<td>0 - 3 cycles</td>
<td>0</td>
<td>In fixed timing mode this sets the number of cycles before IO_CS_L deassertion that IO_OE_L deasserts in a read cycle.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In acknowledgement mode this sets the number of cycles between the strobe (IO_WR_L or IO_OE_L) deasserting and IO_CS_L deasserting.</td>
</tr>
<tr>
<td>io_timeout</td>
<td>1 - 255 us</td>
<td>8</td>
<td>The number of microseconds to wait for the device to assert IO_RDY in acknowledgement mode before terminating the access and signalling a timeout bus error.</td>
</tr>
<tr>
<td></td>
<td>(in io_ext_cfg)</td>
<td></td>
<td>If this parameter is set to zero the access will never be terminated with a timeout, so will hang if an acknowledgement is never provided.</td>
</tr>
<tr>
<td>burst_width</td>
<td>0 - 3 cycles</td>
<td>2&quot;b0</td>
<td>Sets the number of cycles gap between a transfer in a burst transaction. The gap is two cycles larger than the value set in this field.</td>
</tr>
</tbody>
</table>
The reference point for the IO_WR_L write strobe is the deassertion of IO_ALE. It is the responsibility of software to ensure that the assertion timing (and width in fixed cycles) places the strobe within the cycle timing. It is required that

\[ \text{ale_to_wr} \geq \text{ale_to_cs} \]
\[ (\text{cs_width} + \text{ale_to_cs}) \geq (\text{wr_width} + \text{ale_to_wr}) \]

Similarly, in acknowledgement mode the IO_OE_L and IO_WR_L timing is expected to assert the pulses before the IO_RDY line is checked. It is required that

\[ \text{cs_width} \geq \text{cs_to_oe} \]
\[ \text{cs_width} \geq (\text{ale_to_wr} - \text{ale_to_cs}) \]

Failure to meet these assumptions will result in UNDEFINED behavior.

**FIXED CYCLE READ ACCESS**

A fixed read cycle has the simplest timing. The address is put out at the start of the cycle along with IO_ALE. In the non-multiplexed version the address remains stable until the end of the cycle and the data lines are high impedance until the device drives them. In the multiplexed form the address stays valid until IO_CS_L asserts and the bus is turned around for data transfer. The read strobe IO_OE_L is asserted with a delay (which may be zero) from IO_CS_L and will deassert a fixed number of cycles (which may be zero) before IO_CS_L. The write strobe IO_WR_L remains deasserted for the entire cycle.
**FIXED CYCLE WRITE ACCESS**

The fixed cycle write is the same as the fixed cycle read, except IO_OE_L remains deasserted and the ale_to_wr and wr_width parameters are used to set the assertion of the IO_WR_L signal. The data is output on the IO_AD lines with the assertion of chip select.

![Fixed Cycle Write Access Diagram](image)
ACKNOWLEDGEMENT READ ACCESS

The start of an acknowledgement access is the same as a fixed cycle, however after IO_CS_L is asserted and the cs_width interval has passed the IO_RDY line is monitored. The IO_RDY line is treated as an asynchronous signal, and therefore synchronized internally to the 100 MHz reference clock (if it fails to meet the rdy_setup time it may not be sampled until the next cycle) so it should be asserted for a minimum of 20 ns. Once IO_RDY has been asserted (to match the rdy_active control bit) and the rdy_smpl delay has passed the data will be sampled and IO_OE_L deasserted. Zero to three cycles later, set by the oe_to_cs parameter, IO_CS_L will be deasserted. If the device does not signal that it is ready within the timeout period then the read will be aborted, the timeout error status set, the io_error_int interrupt will be raised and UNPREDICTABLE data flagged with a bus error will be returned.

In the case where the peripheral is using the IO_CLK100 and can meet the setup and hold time for IO_RDY the synchronizer may be bypassed. There will therefore be no delay in the acknowledgement reaching the generic bus state machine and it will take effect immediately. For a read if the rdy_smpl parameter is set to zero and the synchronizer is bypassed the data is latched by the same IO_CLK100 edge that the IO_RDY is setup to and the IO_OE_L will deassert following that clock edge (by the clock-to-out delay).
ACKNOWLEDGEMENT WRITE ACCESS

The acknowledgement cycle write is the same as the read, except IO_OE_L remains deasserted and the ale_to_wr parameter is used to set the assertion of the IO_WR_L signal. The IO_WR_L signal is deasserted rdy_smple cycles after the ready signal is sampled. The oe_to_cs parameter is used to set the number of cycles IO_CS_L remains asserted after IO_WR_L is deasserted. If the device does not signal that it is ready within the timeout period then the write will be aborted, the timeout error status set and the io_error_int interrupt will be raised.

Figure 77: Acknowledge Write Access
Burst Mode

A chip select region on the generic bus can be set in a burst mode. This allows faster throughput by only doing a single address cycle at the start of the burst. However, there are some restrictions when in burst mode.

The burst operation is based on the standard cycle outlined in the previous sections. The start of the transaction is the same with the address asserted with IO_ALE. The chip select and strobe (IO_OE_L or IO_WR_L) are asserted and data transferred as normal. When the strobe deasserts (again with the normal timing) either there are no more transfers in the transaction and the access ends in the usual way, or the chip select remains asserted through an idle period of 2 or more cycles before the strobe reasserts for the next data.

This is shown in Figure 78 and Figure 79 and summarized in Table 249.
The additional burst_width parameter is used to specify the duration of the gap between transfers in a burst, to this is added 2 cycles. Thus the fastest sequence for a cache line burst to a region configured for 32 bit bus width and with burst_width=0 and the other parameters at their minimum values is 5-3-3-3-3-3-3 (the initial 5 being ale_width, ale_to_cs and the first 3 cycle access), giving 32 bytes in 26 cycles. Factoring in request and response buffering in the I/O bridge and generic interface, for multiple cache line transfers both reads and writes should achieve a little over 100 MBytes/s (reads will be slightly slower on parts with system revision < PERIPH_REV3).

The burst mode is enabled by setting the io_burst_en bit in the io_ext_cfg register. If this is done then all transactions to the region must be smaller than or a multiple of the data width of the region and any uncached accelerated transactions (merged writes from the CPU) must have contiguous byte enables. If this is not done then extra locations will be accessed during the transfer so the result could be UNPREDICTABLE. The byte enables put out with the address are valid only for the first data transfer on the generic bus.

In burst mode the acknowledgement mode may be used. For the fastest burst the rdy_smpl must be set to 0 and the cs_width equal to cs_to_oe. In addition the IO_RDY signal should be made synchronous. This allows the interface to detect the IO_RDY signal (and the data for reads) during the first cycle the strobe is asserted. In the read case the ready and data are sampled and the IO_OE_L will be deasserted after one cycle. The ready signal should be held asserted for the remainder of the cycle to ensure the fastest burst (or can be deasserted to delay the remaining accesses). The chip select width can still be used to ignore the IO_RDY signal, but this only applies to the first transfer in the burst (i.e. ready is ignored from the chip select assertion for the cs_width but then must be valid for the rest of the cycle). If the IO_RDY detection is asynchronous the internal detection can be as much as two cycles behind the external signal; this does not pose a problem if the ready indication is only used to delay the first transfer in the burst and remains asserted through the rest of the burst, but if the IO_RDY is going to be used for each transfer in the burst then burst_width will need to be at

<table>
<thead>
<tr>
<th>Table 249: Burst Cycle Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ref</strong></td>
</tr>
<tr>
<td>t0</td>
</tr>
<tr>
<td>t1</td>
</tr>
<tr>
<td>t2</td>
</tr>
<tr>
<td>t3</td>
</tr>
<tr>
<td>Repeat next two lines while there is more data</td>
</tr>
<tr>
<td>t4</td>
</tr>
<tr>
<td>t3</td>
</tr>
<tr>
<td>End the cycle when there is no more data</td>
</tr>
<tr>
<td>t5</td>
</tr>
<tr>
<td>t6</td>
</tr>
</tbody>
</table>

The burst mode is enabled by setting the io_burst_en bit in the io_ext_cfg register. If this is done then all transactions to the region must be smaller than or a multiple of the data width of the region and any uncached accelerated transactions (merged writes from the CPU) must have contiguous byte enables. If this is not done then extra locations will be accessed during the transfer so the result could be UNPREDICTABLE. The byte enables put out with the address are valid only for the first data transfer on the generic bus.

In burst mode the acknowledgement mode may be used. For the fastest burst the rdy_smpl must be set to 0 and the cs_width equal to cs_to_oe. In addition the IO_RDY signal should be made synchronous. This allows the interface to detect the IO_RDY signal (and the data for reads) during the first cycle the strobe is asserted. In the read case the ready and data are sampled and the IO_OE_L will be deasserted after one cycle. The ready signal should be held asserted for the remainder of the cycle to ensure the fastest burst (or can be deasserted to delay the remaining accesses). The chip select width can still be used to ignore the IO_RDY signal, but this only applies to the first transfer in the burst (i.e. ready is ignored from the chip select assertion for the cs_width but then must be valid for the rest of the cycle). If the IO_RDY detection is asynchronous the internal detection can be as much as two cycles behind the external signal; this does not pose a problem if the ready indication is only used to delay the first transfer in the burst and remains asserted through the rest of the burst, but if the IO_RDY is going to be used for each transfer in the burst then burst_width will need to be at
least 1 (or more if the ready deasserts longer after the strobe). The rdy_smpl value may be used to widen the strobe if required.

If the burst mode is selected with a non-multiplexed bus the address will not increment through the burst, it will remain at the start address of the burst for the entire access.

**EARLY CHIP SELECT**

The chip select normally asserts ale_to_cs cycles after the IO_ALE deasserts. Some peripherals will only take note of the IO_ALE when their chip select is asserted. The early_cs bit in the io_ext_time_cfg0 register can be set to support them. It causes the external chip select signal to be asserted at the same time as IO_ALE and remain asserted through the ale_to_cs delay and the normal chip select assertion time. Internally the normal chip select timing is used for control of the cycle.

**BOOT ROM SUPPORT**

After reset, chip select region 0 is configured to map 4MB starting at the physical address location of the MIPS processor reset exception routine, 00_1FC0_0000. The timing parameters for the region are set for a slow (40 ns) address latch connected to a slow ROM with 240 ns access time from chip select and output enable. The other configuration parameters are set based on the boot_mode configuration resistors as described in Table 250.

*Table 250: Generic Bus Configuration for Each Boot Mode*

<table>
<thead>
<tr>
<th>Boot Mode</th>
<th>Generic Configuration</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Width = 32 bit / Fixed timing / Multiplexed A/D / No Parity</td>
<td>This gives the highest performance ROM connection.</td>
</tr>
<tr>
<td>01</td>
<td>Width = 8 bit / Fixed timing / NonMultiplexed A/D / No Parity</td>
<td>If address bits [27:24] and the byte enables (on bits [31:28]) are not needed, this will work with an 8 bit ROM connected for a multiplexed A/D bus.</td>
</tr>
<tr>
<td>10</td>
<td>Width = 32 bit / Acknowledgement mode / Timeout disabled / Access small EEPROM using SMBus</td>
<td>These two boot modes use the SMBus boot path described in section Section: “Booting Using an SMBus EEPROM” on page 412.</td>
</tr>
<tr>
<td>11</td>
<td>Width = 32 bit / Acknowledgement mode / Timeout disabled/ Access large EEPROM using SMBus</td>
<td></td>
</tr>
</tbody>
</table>

The SMBus boot modes still use the generic bus module. The SMBus interface 0 protocol engine is placed under the control of the generic bus controller. Rather than run a generic bus cycle to fetch 32 bits of data, a request is made to the SMBus controller. The SMBus interface can be returned to normal software control by clearing the boot_mode[1] bit (bit[18] in the system_cfg register), once this is done the behavior of chip select region 0 becomes UNDEFINED until the io_ext_cfg_0, io_ext_time_CFG0_0 and io_ext_time_CFG1_0 registers have been written.
**GENERIC BUS ERRORS**

There are five error conditions that will raise the io_bus_int interrupt. When the interrupt signal is raised the associated address, data and parity are stored and the interrupt cause recorded. The causes are:

1. A data parity error is detected on an access to a generic bus region.
2. An access is performed to an address in the generic bus range that does not match any of the configured regions.
3. An access is performed to an address that matches multiple regions.
4. A region is configured in acknowledgement mode and the external device does not signal ready within the time limit set in the timeout register.
5. A cacheable coherent access is made to the generic bus and in the response phase of the access an agent asserts both R_SHD and R_EXC to indicate it has a tag error and cannot resolve the ownership.

Error cases (2), (3), and (5) are detected early so that no external access is made.

After an error has been signalled accesses to the generic bus may continue. However, if another error occurs before the previous interrupt was serviced by the CPU, it will be ignored. The interrupt will be cleared only when the `io_interrupt_status` register is read (a cacheable read of the register will not clear the interrupt). To ensure consistent data, software should therefore read the address, data and parity registers before this status register.

In the acknowledgement mode, if the access times out because the device does not signal ready the current transaction will be aborted (in the case of burst mode, the rest of the burst will not be completed). If the access was a read then data marked with a bus error will be returned. The next request will then be serviced. When the timeout occurs, the latency from the timeout being detected on the external bus to the bus error being returned is UNPREDICTABLE, and could be as high as 64K cycles of the IO_CLK100.

**DRIVE STRENGTH CONTROL**

The output drive strength and internal slew rate may be set for all the general 3.3V outputs. This covers the generic bus, the serial ports, the SMBus ports, the Ethernet/packet FIFO interfaces and the GPIO pins. The drive strength effectively sets the number of transistors that will be used to switch the output. The drivers can be varied in 2mA steps from a nominal source/sink current of 2mA up to 8mA for low strength drivers and from 6mA to 12mA for high strength drivers. The slew rate can be varied from fast (recommended default) to slow in four steps. The drivers are grouped by function for setting the drive strength, and into larger groups for setting the slew rate. The parameters are described in more detail in the Hardware Data Sheet.

The drive strength registers for all of the general 3.3V outputs are included in the generic bus configuration register region of the address space.
### GENERIC BUS REGISTERS

**Table 251: Generic Bus Region Configuration Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rdy_active</td>
<td>1'b0</td>
<td>When low the IO_RDY is active low (i.e. the device lets the line go low when it is ready), when high IO_RDY is active high.</td>
</tr>
<tr>
<td>1</td>
<td>io_ena_rdy</td>
<td>1'b0</td>
<td>When high, the interface uses acknowledgement based access. (See also bit 0)</td>
</tr>
<tr>
<td>3:2</td>
<td>io_width_sel</td>
<td>2'h0</td>
<td>Select the data width size. 2'b00: 1 byte width (default for _0 bootmode 01) 2'b01: 2 byte width 2'b10: Alternate 8 bit width (mux on D[7:0]) 2'b11: 4 byte width (default for cs0 bootmode 00, 10, 11)</td>
</tr>
<tr>
<td>4</td>
<td>io_parity_ena</td>
<td>1'b0</td>
<td>When high, enable parity check.</td>
</tr>
<tr>
<td>5</td>
<td>io_burst_en</td>
<td>1'b0</td>
<td>When high the interface will use the burst mode to run each transaction.</td>
</tr>
<tr>
<td>6</td>
<td>io_parity_type</td>
<td>1'b0</td>
<td>When low even parity is used, when high odd parity is used.</td>
</tr>
<tr>
<td>7</td>
<td>io_nonmux</td>
<td>1'b0</td>
<td>When high the bus is used in non-multiplexed mode with AD[31:24] being 8 bits of data and AD[23:0] being 24 bits of address. If the io_width_sel is not set to 2'b00 (for 8 bits) then bus operation is UNPREDICTABLE. For _0, the default is 1 in bootmode 01 and 0 in other bootmodes.</td>
</tr>
<tr>
<td>15:8</td>
<td>io_timeout</td>
<td>8'h8</td>
<td>Time out value. Unit is 1 us. If this field is zero the access will never timeout.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

**Table 252: Generic Bus Region Size Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11:0</td>
<td>io_mult_size</td>
<td>12'h0</td>
<td>Size of the memory region in the multiple of 64KB. The region size is (io_mult_size + 1)’64KB</td>
</tr>
<tr>
<td>15:12</td>
<td>reserved</td>
<td>4'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 253: Generic Bus Region Base Address Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:0</td>
<td>io_start_addr</td>
<td>14'h0</td>
<td>Bits [29:16] of start address of segment. If this address is outside the generic bus range then the region is disabled.</td>
</tr>
<tr>
<td>14</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>io_blk_cache</td>
<td>1'b0</td>
<td>If this bit is set then cacheable accesses to this region will be blocked.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented</td>
</tr>
</tbody>
</table>

A write to any bit causes all bits to be written.

### Table 254: Generic Bus Region Timing 0 Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>io_ale_width</td>
<td>3'h4</td>
<td>Width of IO_ALE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3'h1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>early_cs</td>
<td>1'b0</td>
<td>If this bit is set the external chip select is asserted during the ale_width and ale_to_cs gap as well as at the normal time. Note that this bit only affects the external signal, the timing follows the internal chip select asserting in the normal cycle.</td>
</tr>
<tr>
<td>5:4</td>
<td>io_ale_to_cs</td>
<td>2'h2</td>
<td>Chip select assertion after deassertion of IO_ALE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2'h1</td>
<td></td>
</tr>
<tr>
<td>7:6</td>
<td>burst_width</td>
<td>2'b0</td>
<td>Sets the gap between data transfers during a burst. The actual gap is burst_width + 2.</td>
</tr>
<tr>
<td>12:8</td>
<td>io_cs_width</td>
<td>5'h18</td>
<td>Width of the chip select asserted. If the interface is in acknowledgement mode this is the number of cycles from IO_CS_L being asserted before IO_RDY will start to be checked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5'h1</td>
<td></td>
</tr>
<tr>
<td>15:13</td>
<td>io_rdy_smple</td>
<td>3'h1</td>
<td>Number of clock cycles after the assertion of the acknowledgement is returned that the data is sampled into the interface. This is ignored if the interface is in fixed cycle mode.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented</td>
</tr>
</tbody>
</table>
Table 255: Generic Bus Region Timing 1 Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:0</td>
<td>io_ale_to_write</td>
<td>_0 3'h7</td>
<td>Assertion of write strobe after the deassertion of ALE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>rdy_sync</td>
<td>1'b0</td>
<td>If this bit is clear the IO_RDY input is an asynchronous input and will be internally synchronized so there is no need for the peripheral to meet the setup and hold specification on the signal. If this bit is set the synchronizer is bypassed, so the acknowledgement is detected earlier by the generic bus state machine, but the peripheral must meet the setup and hold time specification.</td>
</tr>
<tr>
<td>7:4</td>
<td>io_write_width</td>
<td>_0 4'h7</td>
<td>Width of the write strobe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11:8</td>
<td>io_idle_cycle</td>
<td>_0 4'h6</td>
<td>Number of idle cycles between back_to_back operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:12</td>
<td>io_oe_to_cs</td>
<td>2'h0</td>
<td>Number of cycles IO_OE_L deasserts before IO_CS_L deasserts. In acknowledgement mode this parameter is also the number of cycles between IO_WR_L deasserts and IO_CS_L deasserts.</td>
</tr>
<tr>
<td>15:14</td>
<td>io_cs_to_oe</td>
<td>2'h0</td>
<td>Number of cycles between IO_CS_L assertion and IO_OE_L assertion. This parameter must be less than io_cs_width.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

A write to any bit causes all bits to be written.

Table 256: Generic Bus Interrupt Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>io_cs_err_int</td>
<td>8'h0</td>
<td>Indicates which of the 8 chip select regions have an error resulting in an interrupt. This field is only valid when one or more of bits [9], [10], [13], or [14] are set.</td>
</tr>
<tr>
<td>8</td>
<td>reserved</td>
<td>1'b0</td>
<td>Not used, reads as zero.</td>
</tr>
<tr>
<td>9</td>
<td>io_rd_par_int</td>
<td>1'b0</td>
<td>When high, indicates parity error on read data from a parity enabled device. The address of the 32 bit word accessed will be put in the address log, the data in the appropriate part of the data log, and the parity in the appropriate part of the parity log.</td>
</tr>
<tr>
<td>10</td>
<td>io_timeout_int</td>
<td>1'b0</td>
<td>When high, indicates timeout has occurred on one of the IO blocks. The address that was being accessed is put in the address log.</td>
</tr>
<tr>
<td>11</td>
<td>io_ill_addr_int</td>
<td>1'b0</td>
<td>When high, indicates an address referenced did not match any region. The address that was being accessed is put in the address log.</td>
</tr>
<tr>
<td>12</td>
<td>io_mult_cs_int</td>
<td>1'b0</td>
<td>When high, indicates multiple chip selects selected based on the address accessed. The address that was being accessed is put in the address log.</td>
</tr>
</tbody>
</table>
### Table 256: Generic Bus Interrupt Status Register (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>io_cacheable_blk</td>
<td>1'b0</td>
<td>When high, indicates that a cacheable access was blocked from a region that has the io_blk_cache bit set in its io_ext_start_addr register. The log contains the address of the blocked access.</td>
</tr>
<tr>
<td>14</td>
<td>io_coh_err</td>
<td>1'b0</td>
<td>When high indicates that a cacheable coherent access was made to the generic bus address space, but an agent indicated that it has a tag error and was unable to resolve ownership. The log contains the address of the failing access.</td>
</tr>
<tr>
<td>15</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 257: Generic Bus Error Data Register 0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_data0</td>
<td>16'h0</td>
<td>This register contains bits 15:0 of the data containing the parity error or captured when the timeout expired.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 258: Generic Bus Error Data Register 1

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_data1</td>
<td>16'h0</td>
<td>This register contains bits 31:16 of the data containing the parity error or captured when the timeout expired.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 259: Generic Bus Error Data Register 2

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_data2</td>
<td>16'h0</td>
<td>This register contains bits 47:32 of the data containing the parity error or captured when the timeout expired.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 260: Generic Bus Error Data Register 3

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_data3</td>
<td>16'h0</td>
<td>This register contains bits 63:48 of the data containing the parity error or captured when the timeout expired.</td>
</tr>
</tbody>
</table>
### Table 260: Generic Bus Error Data Register 3 (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48’bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 261: Generic Bus Error Address Register 0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_addr0</td>
<td>16’h0</td>
<td>This register contains bits 15:0 of the address causing the interrupt.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48’bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 262: Generic Bus Error Address Register 1

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>io_int_addr1</td>
<td>16’h0</td>
<td>This register contains bits 31:16 of the address causing the interrupt.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48’bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 263: Generic Bus Error Parity Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>io_int_parity</td>
<td>4’h0</td>
<td>This register contains the parity of the data that generated the error.</td>
</tr>
<tr>
<td>63:4</td>
<td>notimp</td>
<td>60’bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 264: Output Drive Control Register 0

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>io_slew0</td>
<td>2'b11</td>
<td>Slew0 control for I/O groups A, B, C, D, E, F, G, H, J, K.</td>
</tr>
<tr>
<td>3:2</td>
<td>io_drv_A</td>
<td>2'b01 8mA</td>
<td>Group A drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_AD[31:0].</td>
</tr>
<tr>
<td>5:4</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>7:6</td>
<td>io_drv_B</td>
<td>2'b01 8mA</td>
<td>Group B drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_CS[7:0].</td>
</tr>
<tr>
<td>9:8</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>11:10</td>
<td>io_drv_C</td>
<td>2'b01 8mA</td>
<td>Group C drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_WR_L, IO_OE_L, IO_ALE, IO_CLK100.</td>
</tr>
</tbody>
</table>
### Table 264: Output Drive Control Register 0 (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:12</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>15:14</td>
<td>io_drv_D</td>
<td>2'b01</td>
<td>8mA Group D drive strength control. High drive 6/8/10/12 mA. Uses Slew0. PC_EN3V PC_EV5V PC_ENVPP GPIO[15:12].</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 265: Output Drive Control Register 1

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>3:2</td>
<td>io_drv_E</td>
<td>2'b01</td>
<td>8mA Group E drive strength control. High drive 6/8/10/12 mA. Uses Slew0. GPIO[11:6], GPIO[1:0].</td>
</tr>
<tr>
<td>5:4</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>7:6</td>
<td>io_drv_F</td>
<td>2'b01</td>
<td>8mA Group F drive strength control. High drive 6/8/10/12 mA. Uses Slew0. GPIO[5:2]/IO_ADP[3:0].</td>
</tr>
<tr>
<td>9:8</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>11:10</td>
<td>io_drv_G</td>
<td>2'b01</td>
<td>4mA Group G drive strength control. Low drive 2/4/6/8mA. Uses Slew0. SDA0 SCL0.</td>
</tr>
<tr>
<td>13:12</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>15:14</td>
<td>io_drv_H</td>
<td>2'b01</td>
<td>4mA Group H drive strength control. Low drive 2/4/6/8mA. Uses Slew0. SDA1 SCL1.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 266: Output Drive Control Register 2

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>3:2</td>
<td>io_drv_J</td>
<td>2'b01</td>
<td>Group J drive strength control. Low drive 2/4/6/8mA. Uses Slew0. S0_DOUT S0_TOUT S0_COUT.</td>
</tr>
<tr>
<td>5:4</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>7:6</td>
<td>io_drv_K</td>
<td>2'b01</td>
<td>Group K drive strength control. Low drive 2/4/6/8mA. Uses Slew0. S1_DOUT S1_TOUT S1_COUT.</td>
</tr>
<tr>
<td>9:8</td>
<td>io_slew1</td>
<td>2'b11</td>
<td>Slew1 control for I/O groups L and Q.</td>
</tr>
<tr>
<td>11:10</td>
<td>io_drv_L</td>
<td>2'b11</td>
<td>Group L drive strength control. High drive 6/8/10/12mA. Uses Slew1. E0_MDIO (F0_RXFC) E0_MDC (F1_TXD0) E0_GENO (F0_TXC2) E1_MDIO (F1_TXD2) E1_MDC (F1_TXD1) E1_GENO (F1_TXD6) E2_MDIO (F1_RXFC) E2_MDC (F1_TXD7) E2_GENO (F1_TXC2).</td>
</tr>
<tr>
<td>13:12</td>
<td>io_slew2</td>
<td>2'b11</td>
<td>Slew2 control for I/O groups M and P.</td>
</tr>
<tr>
<td>15:14</td>
<td>io_drv_M</td>
<td>2'b11</td>
<td>Group M drive strength control. High drive 6/8/10/12mA. Uses Slew2. E0_TXEN (F0_TXC0) E0_TXER (F0_TXC1) E0_TXD[7:0] (F0_TXD[7:0]) E1_TXEN (F1_TXD4) E1_TXER (F1_TXD5) E1_TXD[7:0] (F0_TXD[15:8]).</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 267: Output Drive Control Register 3

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>io_slew3</td>
<td>2'b11</td>
<td>Slew3 control for I/O groups N and R.</td>
</tr>
<tr>
<td>3:2</td>
<td>io_drv_N</td>
<td>2'b11</td>
<td>Group N drive strength control. High drive 6/8/10/12mA. Uses Slew3. E0_TCLKO (F0_TCLKO).</td>
</tr>
<tr>
<td>5:4</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>7:6</td>
<td>io_drv_P</td>
<td>2'b11</td>
<td>Group P drive strength control. High drive 6/8/10/12mA. Uses Slew2. E1_TCLKO (F1_TXD3).</td>
</tr>
<tr>
<td>9:8</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>11:10</td>
<td>io_drv_Q</td>
<td>2'b11</td>
<td>Group Q drive strength control. High drive 6/8/10/12mA. Uses Slew1. E2_TXEN (F1_TXC0) E2_TXER (F1_TXC1) E2_TXD[7:0] (F1_TXD[15:8]).</td>
</tr>
<tr>
<td>13:12</td>
<td>reserved</td>
<td>2'b11</td>
<td>Reserved</td>
</tr>
<tr>
<td>15:14</td>
<td>io_drv_R</td>
<td>2'b11</td>
<td>Group R drive strength control. High drive 6/8/10/12mA. Uses Slew3. E2_TCLKO (F2_TCLKO).</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
This Page is left blank for notes
Section 12: PCMCIA Control Interface

INTRODUCTION

The part provides limited support for using PCMCIA memory cards attached to the generic bus. The interface allows card insertion and removal to be detected, provides and removes power from the card, and allows reading and writing of memory and attribute space on the card. The interface does not directly support I/O cards.

The standard was renamed in 1995, so PCMCIA cards are more correctly known as 16-bit PC Cards. Since the interface discussed in this section refers only to the cards defined in the PCMCIA Standard Release 2.1 the more popular name PCMCIA is used in this document.

The PCMCIA interface connects as chip select region 6 on the generic bus (see Section: “Configuring a Chip Select Region” on page 362). The generic bus configuration registers are used to set the address mapping and access timing for the card. The additional control logic provides:

- The two card enable signals (CE1# and CE2#) required by a card with a 16 bit data path.
- Detection of card insertion and removal (CD1# and CD2#), the VCC voltage requested by the card (VS1# and VS2#), card READY, and write protect state (WP).
- Software control of the PCMCIA REG# signal to select between accessing regular memory and attribute memory, and the card RESET signal.
- Outputs to control a supply of VCC and VPP to the card. VCC can either be controlled from software or automatically. Power is automatically removed when the card is removed.
- Interrupts raised to report card status changes.

CONNECTING A PCMCIA SLOT

The PCMCIA interface uses ten of the GPIO pins to provide the additional support signals. In addition there are three dedicated pins that provide the card power enables (see Section: “Using the Power Outputs” on page 392 for information on using these pins as general outputs if PCMCIA is not used). The reset-time configuration resistor on generic address bit IO_AD[16] must pull high to convert the GPIO[15:6] pins to PCMCIA use. For proper operation the gpio_input_invert register bits [15:6] must be left as zeros (their default state) to disable the input inverter on the PCMCIA pins. Operation of the PCMCIA card is UNDEFINED if any of these bits are set, and if the voltage sense inputs are inverted the wrong VCC voltage could be supplied to the card.
Figure 80 shows the connections between the interface and the memory only PCMCIA slot. Chip select region 6 of the generic bus is configured in multiplexed mode and set for a 16 bit wide device. To allow hot-swap the card signals must be isolated from the generic bus. For the 26 address bits the buffering can be done by the address latch. For the data, a bidirectional buffer is needed.

The databus connections for the PCMCIA card reflect the little endian nature of the PCMCIA bus and the big endian byte lane assignment on the generic bus. So data[7:0] of the PCMCIA slot should connect via an isolation buffer to IO_AD[31:24] and data[15:8] of the PCMCIA slot should connect through a buffer to IO_AD[23:16]. This allows use of both 8 bit and 16 bit cards.
Since the interface inputs are 3.3V only, care must be taken to buffer inputs from the card if 5V cards need to be supported. A FET switch or translating buffer can be used to provide the voltage conversion from card outputs and to provide power-down isolation for card inputs. The card detect and voltage sense lines are only statically pulled down or not connected on the card, they do not need buffers but should be pulled up to 3.3V on the main board.

The PCMCIA card uses two card enable signals (CE1# and CE2#) to indicate which byte lane is being used in a transfer. The PCMCIA control logic generates these from IO_CE_L[6] based on the size of the transfer requested by the CPU and the data width that the cs6 region has been configured for. If the interface is 2 bytes wide then the bottom address bit is not used by the card and the card enable lines select which byte lane is used during byte accesses. If the interface is only 1 byte wide then PC_CE1_L is always used with the lowest address bit, and PC_CE2_L is never asserted.

<table>
<thead>
<tr>
<th>io_width_sel configuration</th>
<th>Access Size</th>
<th>PC_CE1_L (data on IO_AD[31:24])</th>
<th>PC_CE2_L (data on IO_AD[23:16])</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 - 1 bytes</td>
<td>any</td>
<td>IO_CS_L[6]</td>
<td>1</td>
</tr>
<tr>
<td>01 - 2 bytes</td>
<td>even byte</td>
<td>IO_CS_L[6]</td>
<td>1</td>
</tr>
<tr>
<td>01 - 2 bytes</td>
<td>odd byte</td>
<td>1</td>
<td>IO_CS_L[6]</td>
</tr>
</tbody>
</table>

The PC_REG_L signal is used to select between accesses to attribute memory (REG# low on the card) and regular memory. It is entirely under software control. The PC_RESET signal is asserted to reset the card, while the card is in place it is controlled by software but the signal (and corresponding bit in the pcmcia_cfg configuration register) will be set high when there is no card detected.

The WP (write protect) and READY status signals from the card are monitored and made available to software. Each of these signals has a transition detector and can raise the pcmcia_interrupt when they change. These signals have a 60ns glitch filter.

The VS1# and VS2# voltage sense signals are monitored and made available to software, they have a 60ns glitch filter. These lines are static signals from the card, and are either not connected or passively pulled low on the card. They indicate the VCC that the socket requests when it is first powered up.

The PCMCIA card detect lines (CD1# and CD2#) are monitored by the hardware. A card is only considered inserted when these two signals are both low (physically these are near the two ends of the card socket and have shorter pins so they are the last pins to make contact). These signals have a 1 ms glitch filter. The (internal) card detect signal is the NOR of the filtered version of these signals. The pcmcia_interrupt can be raised when the card detect status changes.

There are three dedicated outputs used to enable power to the card. Two of these (PC_EN3V and PC_EN5V) control VCC, and the third (PC_ENVPP) controls the VPP programming voltage. The power enable signals are forced to be deasserted when the two card detect signals indicate a card is removed or not present. This provides the auto power down feature when a card is extracted.
Software can directly control the card power using three bits in the `pcmcia_cfg` register, each bit corresponds to one output. These signals should be connected to an external power switch that provides the card VCC. When either CD1# or CD2# is set these bits are forced to zero and writes are ignored (i.e. software is unable to supply power if there is no card, the `pwr_ctrl` bit in the `pcmcia_cfg` register can be set to keep this behaviour when PCMCIA mode is disabled). When a card is inserted the card detect status change interrupt is used to alert the software. The state of the VS1# and VS2# signals is then read and software sets the appropriate configuration bits to power up the card. If the card is subsequently removed the power is automatically turned off (as described above) and the bits in the `pcmcia_cfg` register are again cleared to zero to prevent immediate application of power when a new card is inserted (this must be done to avoid providing the wrong voltage to the new card).

Alternatively, VCC power management can be done entirely in hardware. This is configured by setting the `apron_en` bit in the `pcmcia_cfg` register. When this bit is set, the hardware detects the VS1# and VS2# signals and will enable the requested power to the card.

The following table summarizes the VCC power control options. This table includes the options for controlling the power enable output bits when the PCMCIA mode is not selected at reset time (this is discussed below in Section: “Using the Power Outputs” on page 392).

<table>
<thead>
<tr>
<th>Mode Control Bits</th>
<th>Input signals from PCMCIA card</th>
<th>Software Control Bits in pcmcia_cfg</th>
<th>Card State</th>
<th>Power Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 X 1 X X X X X X</td>
<td>Not inserted</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 1 X X X X X X</td>
<td>Not inserted</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 X X 0 0 0</td>
<td>Inserted off</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 X X 0 0 1</td>
<td>Inserted sw set 5V</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 X X 0 1 0</td>
<td>Inserted sw set 3.3V</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 X X 0 1 1</td>
<td>Inserted sw set both power matrix decodes</td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 1 1 1 X X</td>
<td>5V requested and supplied</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 0 1 1 X X</td>
<td>3.3V requested and supplied</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 1 0 1 X X</td>
<td>low-voltage X.XV requested none supplied</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 0 0 1 X X</td>
<td>X.X/3.3/5V requested 3.3V supplied</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mode Control Bits</th>
<th>Input signals from PCMCIA card</th>
<th>Software Control Bits in pcmcia_cfg</th>
<th>Card State</th>
<th>Power Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 X X X X X X</td>
<td>Non-PCMCIA cd used forced off</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 X X X X X X X</td>
<td>Non-PCMCIA cd used forced off</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X X X 0 0</td>
<td>Non-PCMCIA cd used off</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X X X 0 1</td>
<td>Non-PCMCIA cd used sw set</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The VPP power is supplied under software control, but is always turned off when no card is inserted. Note that the hardware does not check the write protect state, and will supply VPP to a write protected card. This is summarized in the table below, which again includes details of use of the output pin when PCMCIA mode is not selected.

### Table 269: PCMCIA 3.3V and 5V VCC Power Enable Truth Table (Cont.)

<table>
<thead>
<tr>
<th>Mode Control Bits</th>
<th>Input signals from PCMCIA card</th>
<th>Software Control Bits in pcmcia_cfg</th>
<th>Card State</th>
<th>Power Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset AD[16]</td>
<td>ppcmcia_cfg</td>
<td>ppcmcia_cfg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD1#</td>
<td>CD2#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VS1#</td>
<td>VS2#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apron</td>
<td>cfg3v</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cfg5v</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X X X 1 0</td>
<td>Non-PCMCIA cd used</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X X X 1 1</td>
<td>Non-PCMCIA cd used</td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X X 0 0</td>
<td>Non-PCMCIA off</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X X 0 1</td>
<td>Non-PCMCIA sw set</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X X 1 0</td>
<td>Non-PCMCIA sw set</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X X 1 1</td>
<td>Non-PCMCIA sw set</td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note** In the case of both PC_EN5V and PC_EN3V being asserted (which can only be done under software control) the PCMCIA power switch will determine what voltage is actually applied.

The VPP power is supplied under software control, but is always turned off when no card is inserted. Note that the hardware does not check the write protect state, and will supply VPP to a write protected card. This is summarized in the table below, which again includes details of use of the output pin when PCMCIA mode is not selected.

### Table 270: PCMCIA VPP Power Enable Truth Table

<table>
<thead>
<tr>
<th>Mode Control Bits</th>
<th>Input signals from PCMCIA card</th>
<th>Software Control</th>
<th>Card State</th>
<th>Power Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset AD[16]</td>
<td>ppcmcia_cfg</td>
<td>ppcmcia_cfg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD1#</td>
<td>CD2#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VS1#</td>
<td>VS2#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>apron</td>
<td>wp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ppcmcia_cfg cfgvpp bit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 1 X X X X</td>
<td>Not Inserted</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X X 1 X X</td>
<td>Not Inserted</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 1 0</td>
<td>Write Protected, vpp disabled</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 1 1</td>
<td>Write Protected, vpp enabled</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 0 0</td>
<td>Writable, vpp disabled</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 X 0 0 0 1</td>
<td>Writable, vpp enabled</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1 X X X</td>
<td>Non-PCMCIA, cd used, forced off</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
OTHER PCMCIA SIGNALS

The Memory Only PCMCIA slot includes two signals for indicating the status of a battery on the card (BVD1 and BVD2). If these are required they can be connected via a level translating buffer to GPIO pins configured as inputs.

An additional signal WAIT# is used on some cards to indicate that it needs to extend the access time to a transaction. This can be connected through a buffer (to protect the interface input from 5V PCMCIA cards) to the generic bus IO_RDY input. Additional logic may be required to combine the signal with the ready or busy signals from other devices if other generic bus chip select regions use the acknowledgement based mode. If the WAIT# line is connected in this way and a card that does not support WAIT# is inserted, the interface can be switched to fixed timing mode to ignore the IO_RDY signal and allow the card to be used.

The I/O and Memory PCMCIA interface changes the use of some of the signals and adds some others.

- The READY pin becomes IREQ#, an active low interrupt signal. This will work with the standard connection to PC_READY, and the PCMCIA interrupt can be raised whenever the pin changes state regardless of it indicating the card is ready, or is interrupting.
- The WP pin becomes IOIS16#. This is used by the card to inform the host for a particular address if a 16 bit access is possible. The generic bus does not support this behavior, so for cards which need this (indicated in the TPCE_IO field of the CISTPL_CFTABLE_ENTRY configuration tuple) either the interface must be configured for 8 bit only operations or a software restriction made to ensure only 8 bit accesses are done to registers that are 8 bits. The PCMCIA specification indicates that on any card that has a mixture of 8 and 16 bit registers byte registers that are at an odd address can be read on the D[15:8] byte lane when CE2# is asserted, matching Table 270 on page 387.
- The IORD# and IOWR# signals have been added. These match OE# and WE# but for accesses to I/O addresses. One method of generating these is with external logic that drives IO_OE_L and IO_WE_L to either (OE#, WE#) or (IORD#, IOWR#) based on an address bit.
- The PCMCIA card does not share the generic bus region with other peripherals, so the input port acknowledgement signal INPACK# is not needed.
- BVD2 is replaced by SPKR#, which can be used to drive a beep speaker. This would go to some other block than the BCM1250 or BCM1125/H.
- BVD1 is replaced by STSCHNG# which is used to indicate the card status has changed and the card pin replacement register should be read to get the values that would otherwise be reported on the READY, WP, BVD1 or BVD2 pins.

### Table 270: PCMCIA VPP Power Enable Truth Table (Cont.)

<table>
<thead>
<tr>
<th>Mode Control Bits</th>
<th>Input signals from PCMCIA card</th>
<th>Software Control</th>
<th>Card State</th>
<th>Power Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset AD[16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCMCIA mode enable</td>
<td>ppcmcia_cfg</td>
<td>pwr_ctl bit</td>
<td>CD1#</td>
<td>CD2#</td>
</tr>
<tr>
<td>0 1 X 1 X X</td>
<td>Non-PCMCIA, cd used, forced off</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X 0</td>
<td>Non-PCMCIA, cd used, sw turned off</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 X 1</td>
<td>Non-PCMCIA, cd used, sw turned on</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X 0</td>
<td>Non-PCMCIA, sw turned off</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 X X X 1</td>
<td>Non-PCMCIA, sw turned on</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CompactFlash and CF+ cards have a similar interface to the PCMCIA cards. They also support a TrueIDE mode, which cannot be directly connected, but they are required to be able to work in the other modes so this should not be a problem. If only the memory mode of CF/CF+ cards is required then the interface is the same as the memory only PCMCIA interface presented above. However, CF/CF+ cards must be able to operate at 3.3V so the voltage level translators can be avoided (in this case software will have to reject any cards that require 5V for full operation).

**USING THE PCMCIA CARD**

The generic bus address window scheme is used to map between the system address space and the PCMCIA flash memory address space. The PCMCIA cards are accessed through region 6 of the generic bus address space. The usual `io_start_addr` and `io_multi_size` registers are configured to set the base address that the card appears in the system address space and its size. The base address of the region will be mapped to address zero in the PCMCIA space, so PCMCIA card addresses can be computed by adding this offset. In addition the `pcmcia_reg` bit within the `pcmcia_cfg` register must be set according to whether the flash card's attribute or common memory is being accessed. This bit defaults to the low state which selects the common memory.

The card timing is configured using the generic bus timing registers. Since these default to being appropriate for the boot ROM (see Section: “Generic Bus Timing” on page 364) software should configure the timing in region 6 for PCMCIA access at system startup.

The first access to a card after power on should be an attribute memory read. The `pcmcia_cfg` register bit should be set to allow this. The transfer is 16 bits but only the even byte contains valid data because attributes are stored in even byte locations only. The initial attribute memory accesses should be used to extract details from the Card Information Structure (CIS). These details include the card identification and the timing parameters for the card (which will generally have a faster access time than the default). Software should update the parameters for the generic bus region 6. Once the CIS has been read the `pcmcia_cfg` register bit can be cleared to allow access to the memory portion of the card.

The PCMCIA controller monitors status change events and generates an interrupt. The interrupt is cleared by reading the `pcmcia_status` register. An interrupt mask is provided so specific status change events can be masked out. The PCMCIA controller supports 3 card conditions or events:

- Card insertion or removal.
- Card READY status change.
- Card WP write protect state change.

Software can perform a card reset by setting the reset bit high inside the `pcmcia_cfg` control register. The flash card remains in reset until the reset bit is set low. The reset signal is set when the system powers up and when no card is detected.
EXAMPLE PCMCIA TIMINGS

A flash card might have the access timings given in Table 271.

Table 271: Example Flash Card AC Specs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Name</th>
<th>Min (ns)</th>
<th>Max (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Cycle Time</td>
<td>tRC</td>
<td>150</td>
<td>-</td>
</tr>
<tr>
<td>Address Access Time</td>
<td>tAcca</td>
<td>-</td>
<td>150</td>
</tr>
<tr>
<td>Card Enable Access Time</td>
<td>tAcce</td>
<td>-</td>
<td>150</td>
</tr>
<tr>
<td>Output Enable Access Time</td>
<td>tAccoe</td>
<td>-</td>
<td>80</td>
</tr>
<tr>
<td>Output Disable Time</td>
<td>tDis</td>
<td>-</td>
<td>50</td>
</tr>
<tr>
<td>Write Cycle Time</td>
<td>tWC</td>
<td>150</td>
<td>-</td>
</tr>
<tr>
<td>Write Pulse Width</td>
<td>tWP</td>
<td>75</td>
<td>-</td>
</tr>
<tr>
<td>Address Setup Time to WE</td>
<td>tSUa</td>
<td>20</td>
<td>-</td>
</tr>
<tr>
<td>Card Enable Setup Time to WE</td>
<td>tSUcewe</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Address Setup Time to WE end</td>
<td>tSUaweh</td>
<td>140</td>
<td>-</td>
</tr>
<tr>
<td>Data Setup Time to WE end</td>
<td>tSUd</td>
<td>50</td>
<td>-</td>
</tr>
<tr>
<td>Data Hold time from WE end</td>
<td>tHd</td>
<td>20</td>
<td>-</td>
</tr>
<tr>
<td>Card Enable to WE end</td>
<td>tSUceweh</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Card Enable hold from WE end</td>
<td>tHce</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>Write Recover time</td>
<td>Trec</td>
<td>20</td>
<td>-</td>
</tr>
</tbody>
</table>

If such a card were connected as shown in Figure 80 on page 384 possible generic bus timing parameters for this card are given in Table 272 on page 392. The timing diagram that results for read and write accesses is shown in Figure 81 on page 391. Note that the values used for the card timing and the propagation delays of the buffers are chosen for illustration only and do not necessarily match any real devices.
Figure 81: Example Flash Card Timing Diagram
Selection of most of the parameters is straightforward. For reads the chip select width controls the cycle and must be selected to be larger than the access time. In this case it is the $t\text{Acca}$(max) that controls this. The chip select enables the address buffer, so the address only becomes valid after the output enable delay of the buffer (5ns in this case).

The output disable time ($t\text{Dis}$) sets the number of idle cycles. Assuming the next cycle is a write (i.e. the data buffer will drive on the next IO_CS[6]) there are two cycles or 20ns belonging to the next cycle that can be part of $t\text{Dis}$ (the cycle with IO_ALE asserted and the ale_to_cs delay), thus 30ns or three cycles are needed in the current access ($t\text{Dis}$(part) on the diagram). There is always one idle cycle so idle_cycle must be set to 2 to give the extra two idle cycles.

There are many parameters related to the write pulse. The controlling ones are the write width ($t\text{WP}$) which directly requires the wr_width to be 8, and the address to write deassertion setup time ($t\text{SUaweh}$). Fortunately once these are set the data hold time is satisfied with the same chip select width as for the read.

### Table 272: Example Generic Bus Timing parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ale_width</td>
<td>1</td>
</tr>
<tr>
<td>ale_to_cs</td>
<td>1</td>
</tr>
<tr>
<td>cs_width</td>
<td>17</td>
</tr>
<tr>
<td>cs_to_oe</td>
<td>0</td>
</tr>
<tr>
<td>oe_to_cs</td>
<td>0</td>
</tr>
<tr>
<td>ale_to_wr</td>
<td>8</td>
</tr>
<tr>
<td>wr_width</td>
<td>8</td>
</tr>
<tr>
<td>idle_cycle</td>
<td>2</td>
</tr>
</tbody>
</table>

#### USING THE POWER OUTPUTS

If the PCMCIA mode is not enabled by the reset time configuration resistor then the three power control lines can be controlled by software directly through the `pcmcia_cfg` register.

If the `pcmcia_cfg_pwr_ctl` bit is set then the state of the GPIO[12] and GPIO[13] pins are used in the same way they would be (as CD1# and CD2#) in PCMCIA mode. They must be low to allow software to set the power control output lines, and the bits in the `pcmcia_cfg` bits will be cleared if either goes high. This may be used in systems other than PCMCIA that need to implement hot-swap logic. The pins remain GPIO pins, if they are configured as outputs then the value being driven will be read back from the pin and used to control the power logic.

Table 269 on page 386 and Table 270 on page 387 show the logic of the power control lines when PCMCIA mode is disabled.
### PCMCIA Controller Registers

The PCMCIA controller is enabled at reset time by a selection resistor on the generic address bus. The state can be read from the system configuration register. The other PCMCIA control registers are:

**Table 273: PCMCIA Configuration Register**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>pcmciacfgreg</td>
<td>1'b0</td>
<td>When high, selects attribute memory. The PC_REG_L pin is the inverse of this bit.</td>
</tr>
<tr>
<td>1</td>
<td>pcmciacfg3v_en</td>
<td>1'b0</td>
<td>When high, 3 volt supply enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the EN3V output.</td>
</tr>
<tr>
<td>2</td>
<td>pcmciacfg5v_en</td>
<td>1'b0</td>
<td>When high, 5 volt supply enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the EN5V output.</td>
</tr>
<tr>
<td>3</td>
<td>pcmciacfgvpp_en</td>
<td>1'b0</td>
<td>When high, programming voltage enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the ENVPP output.</td>
</tr>
<tr>
<td>4</td>
<td>pcmciacfgreset</td>
<td>1'b1</td>
<td>When high, apply reset to the pcmcia card. This bit is set when the system is reset or whenever no card is detected.</td>
</tr>
<tr>
<td>5</td>
<td>pcmciaapron_en</td>
<td>1'b0</td>
<td>When high, enable auto power on.</td>
</tr>
<tr>
<td>6</td>
<td>pcmciacfgcd_mask</td>
<td>1'b1</td>
<td>When high, disable Card detect interrupt. When low an interrupt is raised whenever the cd1_L and cd2_L (gpio[13:12]) lines either become both zero or stop being both zero. (The PCMCIA mode setting has no effect on this interrupt).</td>
</tr>
<tr>
<td>7</td>
<td>pcmciacfgwp_mask</td>
<td>1'b1</td>
<td>When high, disable Write protect interrupt. When low an interrupt is raised whenever a change is detected on the WP (gpio[11]) line (The PCMCIA mode setting has no effect on this interrupt).</td>
</tr>
<tr>
<td>8</td>
<td>pcmciacfgrdy_mask</td>
<td>1'b1</td>
<td>When high, disable Card ready interrupt. When low an interrupt is raised whenever a change is detected on the READY (gpio[9]) line (The PCMCIA mode setting has no effect on this interrupt).</td>
</tr>
<tr>
<td>9</td>
<td>pcmciacfgpwr_ctl</td>
<td>1'b0</td>
<td>This bit is only used when the PCMCIA function is not selected at reset time. When high, the PCMCIA card detect inputs are used in driving the power select pins when PCMCIA mode is not selected.</td>
</tr>
<tr>
<td>15:10</td>
<td>reserved</td>
<td>7'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
Table 274: PCMCIA Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>pcmcia_status_cd1</td>
<td>ext</td>
<td>This bit reflects the state of the card detect 1 signal (PC_CD1_L) from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCMCIA card. When both PC_CD1_L and PC_CD2_L are low the card is fully</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>inserted.</td>
</tr>
<tr>
<td>1</td>
<td>pcmcia_status_cd2</td>
<td>ext</td>
<td>This bit reflects the state of the card detect 1 signal (PC_CD1_L) from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCMCIA card. When both PC_CD1_L and PC_CD2_L are low the card is fully</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>inserted.</td>
</tr>
<tr>
<td>2</td>
<td>pcmcia_status_vs1</td>
<td>ext</td>
<td>This bit reflects the state of the voltage sense 1 signal (PC_VS1_L) from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the PCMCIA card.</td>
</tr>
<tr>
<td>3</td>
<td>pcmcia_status_vs2</td>
<td>ext</td>
<td>This bit reflects the state of the voltage sense 2 signal (PC_VS2_L) from</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the PCMCIA card.</td>
</tr>
<tr>
<td>4</td>
<td>pcmcia_status_wp</td>
<td>ext</td>
<td>This bit reflects the state of the write protect signal (PC_WP) from the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PCMCIA card. When high, write protect is enabled.</td>
</tr>
<tr>
<td>5</td>
<td>pcmcia_status_rdy</td>
<td>ext</td>
<td>This bit reflects the state of the ready signal (PC_READY) from the PCMCIA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>card. When high, indicates the card is ready for access.</td>
</tr>
<tr>
<td>6</td>
<td>pcmcia_status_3v_en</td>
<td>1'b0</td>
<td>When high, indicates 3 volt VCC is enabled. This bit reflects the value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>driven on the PC_EN3V signal.</td>
</tr>
<tr>
<td>7</td>
<td>pcmcia_status_5v_en</td>
<td>1'b0</td>
<td>When high, indicates 5 volt VCC is enabled. This bit reflects the value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>driven on the PC_EN5V signal.</td>
</tr>
<tr>
<td>8</td>
<td>pcmcia_cd_change</td>
<td>1'b0</td>
<td>This bit is set when either of the card detect signals change. It is cleared</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>by a read. If this bit is set an interrupt will be raised if it is not masked</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the pcmcia_cfg register.</td>
</tr>
<tr>
<td>9</td>
<td>pcmcia_wp_change</td>
<td>1'b0</td>
<td>This bit is set when the write protect signal changes. It is cleared by a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>read. If this bit is set an interrupt will be raised if it is not masked in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the pcmcia_cfg register.</td>
</tr>
<tr>
<td>10</td>
<td>pcmcia_rdy_change</td>
<td>1'b0</td>
<td>This bit is set when card ready signal changes. It is cleared by a read. If</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>this bit is set an interrupt will be raised if it is not masked in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pcmcia_cfg register.</td>
</tr>
<tr>
<td>15:11</td>
<td>reserved</td>
<td>6'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
Section 13: GPIO

INTRODUCTION

The part has a number of pins that are available for general use as inputs, outputs or interrupt inputs. These pins are controlled entirely by software. In addition, there are a number of pins allocated to other peripherals that may be used as general pins if the peripheral is not required.

THE GPIO PINS

The GPIO pins can be configured for use as either inputs or outputs, and can be set to raise an interrupt. A single GPIO pin is shown in Figure 82.

![Figure 82: Single GPIO Pin Diagram]

The `gpio_direction` register sets the direction of each pin individually, bits in it should be set to enable the output buffer. When used as outputs (shown in the bottom of the figure) the CPU sets and clears bits in the output latch by writing 1s to the `gpio_pin_set` or `gpio_pin_clr` registers, and the pin will change state. The output register retains its state even when the line is configured as an input, so if an "open collector" output is required the output register only needs to be set low once and the direction of the pin can be changed from output (to pull low) to input (to float).

The input path is always active. An optional inverter is enabled by setting the corresponding bit in the `gpio_input_invert` register. The 100MHz reference clock is used to synchronize the signal to the internal logic and to provide filtering against glitches. By default the line has to change state for about 60ns before being recognized (a 60ns glitch filter), but the filter may be increased to provide a 1 ms glitch filter. The state of the gpio pin (after inversion and the glitch filtering) is readable from the `gpio_read` register.
GPIO lines can be enabled as interrupts in pairs. For each pair, both can have their interrupts disabled, both can be level interrupts, both can be edge sensitive interrupts or one can be level and the other an edge interrupt. This is selected in the `gpio_int_type` register. When a level interrupt is selected the input after inversion and glitch filtering is directly passed to the interrupt mapper in the SCD (see Section: “The Full Interrupt Mapper” on page 50) where it must be high to signal an interrupt. If the edge detector is enabled, the signal to the SCD is asserted when a rising edge is detected on the output of the glitch filter, the signal is cleared when the corresponding bit is set in a write to the `gpio_clr_edge` register. A negative edge can be detected by inverting the input. Masking of the GPIO interrupt lines is also done in the interrupt mapper. The interrupt function is not affected by the direction of the line, if the GPIO pin is set as an output then the interrupt will be raised whenever the output is set appropriately (after the glitch filtering delay).

Some of the GPIO pins are used by the PCMCIA controller and some to provide parity on the generic bus. These functions are set by reset time configuration resistors on the generic bus AD lines. When used for PCMCIA operation the input inverters must be disabled by leaving `gpio_input_invert` bits [15:6] in their default state as zeros. The GPIO signals are summarized in the table below.

<table>
<thead>
<tr>
<th>GPIO Bit</th>
<th>Alternate Name</th>
<th>Alternate Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>S1:0_RSTROBE</td>
<td>Synchronous serial output strobes.</td>
</tr>
<tr>
<td>6</td>
<td>PC_CE1_L</td>
<td>PCMCIA Card enable output.</td>
</tr>
<tr>
<td>7</td>
<td>PC_CE2_L</td>
<td>PCMCIA Card enable output.</td>
</tr>
<tr>
<td>8</td>
<td>PC_RESET</td>
<td>PCMCIA card reset output.</td>
</tr>
<tr>
<td>9</td>
<td>PC_READY</td>
<td>PCMCIA ready input.</td>
</tr>
<tr>
<td>10</td>
<td>PC_REG_L</td>
<td>PCMCIA regular/attribute select output.</td>
</tr>
<tr>
<td>11</td>
<td>PC_WP</td>
<td>PCMCIA write protected input.</td>
</tr>
<tr>
<td>12</td>
<td>PC_CD1_L</td>
<td>PCMCIA card detect input (the 1ms glitch filter is always used).</td>
</tr>
<tr>
<td>13</td>
<td>PC_CD2_L</td>
<td>PCMCIA card detect input (the 1ms glitch filter is always used).</td>
</tr>
<tr>
<td>14</td>
<td>PC_VS1_L</td>
<td>PCMCIA voltage sense input (the 60 ns glitch filter is always used).</td>
</tr>
<tr>
<td>15</td>
<td>PC_VS2_L</td>
<td>PCMCIA voltage sense input (the 60 ns glitch filter is always used).</td>
</tr>
</tbody>
</table>

*Table 275: GPIO Pins and Alternate Uses*
## GPIO Registers

There are eight GPIO registers.

### Table 276: GPIO Edge Clear Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>edge_clr</td>
<td>16'h0</td>
<td>Writing a 1 in a bit position in this register will clear the corresponding edge detector.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 277: GPIO Interrupt Type Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>int_type0</td>
<td>2'b0</td>
<td>The two bit fields set the interrupt type for each pair of pins.</td>
</tr>
<tr>
<td>3:2</td>
<td>int_type2</td>
<td>2'b0</td>
<td>00: Disable both lines. The interrupts will never be raised.</td>
</tr>
<tr>
<td>5:4</td>
<td>int_type4</td>
<td>2'b0</td>
<td>01: Both GPIO pins are edge sensitive interrupts.</td>
</tr>
<tr>
<td>7:6</td>
<td>int_type6</td>
<td>2'b0</td>
<td>10: Both GPIO pins are level sensitive interrupts.</td>
</tr>
<tr>
<td>9:8</td>
<td>int_type8</td>
<td>2'b0</td>
<td>11: The even numbered pin is an edge sensitive interrupt.</td>
</tr>
<tr>
<td>11:10</td>
<td>int_type10</td>
<td>2'b0</td>
<td>The odd numbered pin is a level sensitive interrupt.</td>
</tr>
<tr>
<td>13:12</td>
<td>int_type12</td>
<td>2'b0</td>
<td></td>
</tr>
<tr>
<td>15:14</td>
<td>int_type14</td>
<td>2'b0</td>
<td></td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

### Table 278: GPIO Read Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>gpio</td>
<td>16'h0</td>
<td>Reads give value on each pin, after the optional inverter and glitch filtering.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 279: GPIO Input Invert Control Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>invert</td>
<td>16'b0</td>
<td>Each bit controls the input inverter for the corresponding GPIO pin. If a bit is set then the value used internally and read in the gpio_read register will be the inverse of the value on the pin.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 280: GPIO Glitch Filter Select Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>gpio_glitch</td>
<td>2'h0</td>
<td>When high the input from the pin has a 1ms glitch filter, when low a 60 ns glitch filter.</td>
</tr>
<tr>
<td>5:2</td>
<td>gpio_glitch</td>
<td>4'h0</td>
<td>When high the input from the pin has a 1ms glitch filter, when low a 60 ns glitch filter. If generic bus parity is enabled these bits are ignored.</td>
</tr>
<tr>
<td>15:6</td>
<td>gpio_glitch</td>
<td>10'h0</td>
<td>When high the input from the pin has a 1ms glitch filter, when low a 60 ns glitch filter. In PCMCIA mode these bits are ignored.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 281: GPIO Direction Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:0</td>
<td>gpio_ddr</td>
<td>2'h0</td>
<td>High for output driven from the output latch, low for input. If the synchronous serial interface RstrobeOut output is enabled this register is ignored and the pin will be an output.</td>
</tr>
<tr>
<td>5:2</td>
<td>gpio_ddr</td>
<td>4'h0</td>
<td>High for output, low for input. When generic bus parity is enabled these bits are ignored.</td>
</tr>
<tr>
<td>15:6</td>
<td>gpio_ddr</td>
<td>10'h0</td>
<td>High for output, low for input. When PCMCIA is enabled these bits are ignored.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 282: GPIO Pin Clear Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>pin_clr</td>
<td>16'h0</td>
<td>Writing a 1 to a bit position will clear that bit in the output latch.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 283: GPIO Pin Set Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>pin_set</td>
<td>16'h0</td>
<td>Writing a 1 to a bit position will set that bit in the output latch.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
OTHER PINS THAT CAN BE USED

The part has many built in peripherals, and in any particular application it is possible that some of them are not used. Pins on some of the peripherals can be used to provide additional software controlled input, output or interrupt lines. This section describes some of these.

SERIAL PORTS

If a serial port is unused most of its pins can be reused. The port should be configured in asynchronous mode. All of the input pins (except for the data input pin) can be read through the duart_in_port register. The output handshake pins can be used by configuring them for software control. In an extreme case it would be possible to use the data out pin by enabling the transmitter and either letting the line idle (by sending nothing) to give a high output, or sending a break to give a low output.

The following table summarizes the pins that can be directly used.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Direction</th>
<th>Channel A access (n=0)</th>
<th>Channel B access (n=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn_RTS_TSTROBE</td>
<td>Output</td>
<td>op[0]</td>
<td>op[1]</td>
</tr>
<tr>
<td>Sn_CTS_TCLKIN</td>
<td>Input transition detect</td>
<td>ip[0]</td>
<td>ip[1]</td>
</tr>
</tbody>
</table>

See Section: “Operation” on page 322 for how to configure the UART and control the I/O lines.

PCI

If the PCI bus is not used the PCI interrupt lines can be used as active low level interrupt inputs, as can the SERR and PERR signals. If the PCI is in device mode the P_INTB_L, P_INTC_L and P_INTD_L lines can be used as active low level interrupt lines.

MACS

The MAC management interface provides the only pins that can easily be used from an unused MAC interface. The MDIO line provides input and output, and the MDC line is output only. The GENO pin from a MAC is available as a general output unless the interface is in 16 bit bypass mode. The FIFO mode of the MAC may be used for bulk data DMA transfers, but there is no easy way to use it for programmed I/O.

PCMCIA POWER CONTROL PINS

The PCMCIA power control pins are three dedicated outputs that are forced low when the part is reset. If PCMCIA support is not required then these lines can be used as general outputs by setting the PCMCIA power logic for software power control and writing the power control bits. This is described in Section: “Using the Power Outputs” on page 392.
This Page is left blank for notes
This Page is left blank for notes
Section 14: Serial Configuration Interface

INTRODUCTION

There are two serial configuration interfaces based on the SMBus. The interfaces provide hardware assistance for simple read and write of slave devices with the part as the bus master. The hardware assistance provides a subset of the SMBus version 1.1 specification published by the Smart Battery System Implementers Forum. The bus clock is programmable, it should be set in the 10-100kHz range for conformance operation, but can go higher. In particular many devices can use a 400kHz clock.

SMBus OVERVIEW

There are a large number of devices that can be configured using a two wire serial interface with one clock wire and one data wire. The interface to these devices is basically the same, but there are differences in timing and electrical specifications. The serial configuration interface is based on version 1.1 of the System Management Bus (SMBus) standard from the Smart Battery System Implementers Forum (SBS-IF), but has been generalized to support other devices. This interface only implements a subset, so is not fully compliant with the SMBus 1.1 specification. This section provides a brief introduction to the interface, for full details see the SMBus specification that may be found on the SMBus web site at http://www.smbus.org and the related documents.

Devices on the serial configuration bus may be masters or slaves (or both). The interface is always a master. Masters can issue commands, provide the clock and control the transfer. Slaves respond to commands from masters and use the supplied clock. Each slave has a unique 7 bit address (or a few addresses) that the master uses to select it. Transactions always start with the master sending a byte containing the address of the slave and a single bit that indicates if the master is reading from the slave or writing to it. Following this one or more bytes of data are transferred. The SMBus adds a protocol layer above this basic transport.

TRANSPORT PROTOCOL

Devices connected to the serial configuration bus use open drain outputs on both the clock and data lines. Pull-up resistors restore the lines high when no device is driving. The inputs should not be pulled above 3.3V.

Both the data (SDA) and clock (SCL) lines are high when idle. In normal operation SDA will only change while SCL is low. SDA changing with SCL high is used to signal control flags. Figure 83 shows the three conditions for bit transfer on the SMBus:

Start

A Start condition is used to indicate the beginning of a transaction. It consists of SDA changing from high (idle) to low while SCL remains high (idle).

Data bit

A data bit is transferred by putting it on the SDA line with the SCL clock low and pulsing the clock high and then low with SDA stable. Data bytes are sent with the most significant bit first.
**Stop**

A stop condition signals the end of a transaction. It consists of SDA making a transition from low to high (idle) while SCL is high (idle).

![SMBus Signaling Start, Data Transfer and Stop](image)

**Figure 83: SMBus Signaling Start, Data Transfer and Stop**

A master will initiate a transfer by sending a start condition. This causes all slaves to listen to the next byte which includes the 7 bit address and a single read/write bit. Following this the master sets SDA idle and pulses SCL once more. If a slave matches the address it will acknowledge by holding SDA low. Bytes are then transferred in the direction marked by the read/write bit, with the receiver (slave during a write, master during a read) acknowledging after each. Bytes continue to be transferred until the slave does not acknowledge a byte or the master decides all bytes have been transferred. The transaction is then ended by the master signalling a stop, which returns both lines to idle.

There may be multiple masters on the bus. Most of the time they will see the start and stop framing of other masters to avoid collision, however if two masters start at the same time they may collide. In this case the first time their SDA signals differ the master that drives the line to zero wins arbitration and the other must back off. When forced to back off the interface will retry after it sees a stop condition, or if SCL and SDA are both high for 50 us (this is defined as an idle condition).

The timing specification for slave devices tends to vary with frequency. To comply with both the standard timings at 100 kHz and typical timings at 400 kHz the interface produces a SCL clock signal with a 9:7 ratio of time low:time high.

The slave device is permitted to slow the clock by stretching the low period. If a slave has stretched the clock for 25 ms then an error has occurred and the master will abort the transfer.
TRANSPORT PROTOCOL RESET

A system reset during an SMBus transaction can leave the bus in an unusable state. The reset will cause the master to release the clock line and SCL will therefore go high. But many slaves do not detect the system reset, and if a slave is being read it is not permitted to change the state of the data line while the clock is high. Following a reset it is possible for a slave to be driving the SDA low while SCL is high. All masters will backoff from using the bus since it looks as though it is in use. A START or STOP cannot be sent because the slave is forcing the SDA low.

The interface will attempt to clear this bad state. If SDA is detected low when the interface is released from reset it will issue clock pulses on SCL until SDA goes high. In the absence of protocol errors, at most 9 clock pulses are needed to ensure that any slave that believes a read is in progress will complete transfer of a byte (of 0s) and see a NACK, causing it to end the transaction. If SDA goes high before the slave believes the transaction is finished the interface will stop providing clock pulses but the bus will be left in a state where a new master can initiate a new transaction. The START of the next transaction will ensure all slave state machines are resynchronized to the command stream. If a genuine transaction is in progress under the control of another master the part's attempt to clear state will at worst result in clock cycle stretching and should not interfere with the transaction.

During the reset procedure accesses to the control registers are held off. This will at most last for the first 90us after reset so it is unlikely to be noticed in practice.

SMBUS PROTOCOL

The SMBus protocol defines a number of transfer types between the host and peripherals. The interface implements all of these except for process call and block transfers. The latest revision of the SMBus includes Packet Error Code (PEC) checking, this appends an 8-bit CRC (using the polynomial $x^8 + x^2 + x + 1$) at the end of the message. The CRC is calculated across all bytes in the transfer (address, mode and data). The receiver can verify correctness of the CRC and issues a Not Acknowledge if it is incorrect.

The interface allows use of PEC by optionally adding a byte at the end of a transfer, but the hardware will neither generate nor check the PEC byte. Software should calculate the correct value to be sent with writes and must load this into the smb_pec register before starting the transfer. If the CRC check fails at the slave device it will NACK the PEC byte and an error will be reported in the smb_status register. When the interface receives PEC it is always acting as the transaction master, so the hardware will always NACK the byte and signal a stop to terminate the transfer. Software should then check the smb_pec register and can retry the transfer if it detects an error.

The supported SMBus commands are listed in Table 285, along with the parameters they use.

<table>
<thead>
<tr>
<th>Transfer Type (smb_tt)</th>
<th>Command (smb_cmd)</th>
<th>1st Byte (smb_data[7:0])</th>
<th>2nd Byte (smb_data[15:8])</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Command (110)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Send a single bit (in the R/W bit position) to the slave.</td>
</tr>
</tbody>
</table>

In the figures below S = start, P = stop, A = ACK, A = NACK, Wr (write) = 0, Rd (read) = 1
Bits driven by the slave are shaded

<table>
<thead>
<tr>
<th>S</th>
<th>Slave Address</th>
<th>Data</th>
<th>AP</th>
</tr>
</thead>
</table>

During the reset procedure accesses to the control registers are held off. This will at most last for the first 90us after reset so it is unlikely to be noticed in practice.
If the device that is addressed is not an SMBus device, then it will most likely use a different protocol. In this case a mapping will need to be done from the SMBus commands to those that the device supports. The Send Byte and Receive byte commands perform the fundamental transfers, but in many cases the other commands can be used (for example many memory devices can use the Read Word command to set the address pointer and read back two bytes). Since it is not interpreted by the hardware the PEC byte can be used as an additional byte in the transfer.

### Table 285: Supported SMBus Transfer Types (Cont.)

<table>
<thead>
<tr>
<th>Transfer Type (smb_tt)</th>
<th>Command (smb_cmd)</th>
<th>1st Byte (smb_data[7:0])</th>
<th>2nd Byte (smb_data[15:8])</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send Byte (000)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Send a single byte to the slave.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A P</td>
</tr>
<tr>
<td>Receive Byte (101)</td>
<td>No</td>
<td>No</td>
<td>Data</td>
<td>Read a single byte from the slave using a simple read with no associated command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Rd A Data Byte A P</td>
</tr>
<tr>
<td>Write Byte (001)</td>
<td>Yes</td>
<td>Data</td>
<td>No</td>
<td>Send a command byte and a single data byte to the slave.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A Data Byte A P</td>
</tr>
<tr>
<td>Read Byte (011)</td>
<td>Yes</td>
<td>Data</td>
<td>No</td>
<td>Send a command to the slave and read back a byte.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A S Slave Address Rd A Data Byte A P</td>
</tr>
<tr>
<td>Write Word (010)</td>
<td>Yes</td>
<td>LSB</td>
<td>MSB</td>
<td>Send a command and two bytes to the slave.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A Data Byte Low A Data Byte High A P</td>
</tr>
<tr>
<td>Read Word (100)</td>
<td>Yes</td>
<td>LSB</td>
<td>MSB</td>
<td>Send a command to the slave and read two bytes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A S Slave Address Rd A Data Byte Low A Data Byte High A P</td>
</tr>
<tr>
<td>EEPROM Read (111)</td>
<td>Yes</td>
<td>LSB</td>
<td>MSB</td>
<td>Send a command and one byte to the slave and read four bytes back. This matches a 32 bit read from a &gt;16 kbit EEPROM.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Wr A Command Code A Data Byte Low A Data Byte High A P</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>7</td>
<td>S Slave Address Rd A Data Byte Low A Data Byte High A Xtra Byte Low A Xtra Byte High A P</td>
</tr>
</tbody>
</table>
EXTENDED PROTOCOL

The Extended protocol mode of the interface supports a much wider range of devices by allowing transactions other than those specified in SMBus. Up to seven bytes can be written to a device, and up to seven bytes read.

An extended mode transaction consists of a command/address being written to the device followed by either data being written to the device or a repeated start and data being read from the device.

A write transaction consists of writing a command/address, with the format selected from Table 286, followed by write data, selected from Table 287. If PEC is enabled then an additional byte from the smb_pec register will be sent before the STOP. (The interface places no interpretation on the bytes, the names command/address, data and PEC just refer to conventional use of these bytes.)

A read transaction consists of writing the command/address, as shown in Table 286, followed by a repeated START and reading data, as shown in Table 288. If PEC is enabled then an additional byte will be read into the smb_pec register before the NACK and STOP. The command/address may be null, in which case the START shown in Table 288 will be a standard START not a repeated one.

The extended mode is invoked by setting the extend bit in the smb_start register, this changes the interpretation of some of the other fields. Thus SMBus and extended transactions can be freely mixed.

Table 286: Command/Address Options

<table>
<thead>
<tr>
<th>smb_start[12:11]</th>
<th>Address Type</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Just Read</td>
<td>This is for reads that do not have an address phase. The Start (and slave address) that is part of the read becomes a regular Start rather than a Repeated Start.</td>
</tr>
<tr>
<td>01</td>
<td>Slave Address</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>Byte Address</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>16 bit Address</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 287: Write Data Options

<table>
<thead>
<tr>
<th>smb_start [10:8]</th>
<th>Data Format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1 Byte</td>
<td>8</td>
</tr>
<tr>
<td>001</td>
<td>2 Bytes</td>
<td>8</td>
</tr>
<tr>
<td>010</td>
<td>3 Bytes</td>
<td>8</td>
</tr>
<tr>
<td>011</td>
<td>4 Bytes</td>
<td>8</td>
</tr>
</tbody>
</table>

Broadcom Corporation
### Table 287: Write Data Options (Cont.)

<table>
<thead>
<tr>
<th>smb_start [10:8]</th>
<th>Data Format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>No Data</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>101</td>
<td>Reserved</td>
<td>Reserved encodings. UNPREDICTABLE operation.</td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 288: Read Data Options

<table>
<thead>
<tr>
<th>smb_start [10:8]</th>
<th>Data Format</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1 Byte</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>001</td>
<td>2 Bytes</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>010</td>
<td>3 Bytes</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>011</td>
<td>4 Bytes</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>100</td>
<td>No Data</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
<tr>
<td>101</td>
<td>5 Bytes</td>
<td><img src="" alt="Table Entry" /></td>
</tr>
</tbody>
</table>
**PROGRAMMING MODEL**

**USING SMBUS PROTOCOLS**

A transfer is configured by setting data values in the SMBus registers and writing the transfer type to the `smb_start` register. This starts the operation. If the remote device fails to acknowledge when expected to then the transfer is abandoned with an error. If the interface loses arbitration or detects over-long slave stretching then the transfer is aborted. The interface will retry an aborted transfer 15 times before abandoning it with an arbitration error.

Reading from a slave on the serial bus involves these steps:

1. Wait until the busy bit in the `smb_status` register is clear. The interface will ignore accesses while the bit is set.
2. For a Read Byte (but not a Receive Byte) transfer specify the command to be sent by writing to the `smb_cmd` register.
3. Specify the device address and the transfer type by writing to the `smb_start` register. This will cause the transfer to begin. The busy bit in the `smb_status` register will be set to a one as long as the transfer is in progress.
4. When the transaction finishes the busy bit will be cleared. This can either be detected by polling the `smb_status` or by enabling the `smb_finish` interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction.
5. Read the data from the `smb_data` register. If one byte was requested, only the lsb field will be valid.
6. If PEC is enabled the `smb_pec` register should be read and checked.

Writing to a slave on the serial bus involves these steps:

1. Wait until the busy bit in the `smb_status` register is clear. The interface will ignore accesses while the bit is set.
2. Specify the command to be sent as the first byte by writing it in the `smb_cmd` register.
3. If data is being sent write it in the `smb_data` register.
4. If PEC is being used the CRC must be calculated by the software and written to the `smb_pec` register.
5. Specify the device address and the transfer type by writing to the `smb_start` register. This will cause the transfer to begin. The busy bit in the `smb_status` register will be set to a one as long as the transfer is in progress.
6 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the smb_status register or by enabling the smb_finish interrupt. The error bit indicates that an acknowledgement was not received from the slave device at some point during the transaction. (It is important to check the data sheet for the slave device, some will always terminate a transaction by not sending an acknowledgement).

A quick command can be sent with these steps:

1 Wait until the busy bit in the smb_status register is clear. The interface will ignore accesses while the bit is set.

2 Specify the device address, the command bit (which is sent as the R/W bit in a Quick Command) and the transfer type by writing to the smb_start register. This will cause the transfer to begin. The busy bit in the smb_status register will be set to a one as long as the transfer is in progress.

3 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the smb_status register or by enabling the smb_finish interrupt. The error bit indicates that an acknowledgement was not received from the slave device.

An EEPROM read is a write with a command and one data byte followed (without a stop) by a read of four data bytes (and optional PEC). Typically the command and data byte will contain the address to be read. The command sequence reflects this:

1 Wait until the busy bit in the smb_status register is clear. The interface will ignore accesses while the bit is set.

2 Specify the command (high address bits) to be sent as the first byte by writing it in the smb_cmd register.

3 Specify the data (low address bits) to be sent by writing it in the smb_data register.

4 Specify the device address (on some EEPROMs the low three bits of the device address provide an additional three address bits) and the transfer type by writing to the smb_start register. This will cause the transfer to begin. The busy bit in the smb_status register will be set to a one as long as the transfer is in progress.

5 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the smb_status register or by enabling the smb_finish interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction or as the address was written.

6 Read the data from the smb_data (from address and address+1) and the smb_xtra (from address+2 and address+3) registers.

7 Most EEPROMs do not support PEC. Using standard parts if PEC is enabled the smb_pec register will contain a fifth byte of data (from address+4).

The SMBus specification includes a Process Call command and Block Read and Write commands for transferring up to 32 bytes of data. The interface does not support these operations.
USING EXTENDED PROTOCOLS

A transfer is configured by setting data values in the SMBus registers and writing the transfer type with the extend bit set to the smb_start register. This starts the operation. If the remote device fails to acknowledge when expected, then the transfer is abandoned with an error. If the part loses arbitration or detects over-long slave stretching then the transfer is aborted. The interface will retry an aborted transfer 15 times before abandoning it with an arbitration error.

Reading from a slave on the serial bus involves these steps:

1. Wait until the busy bit in the smb_status register is clear. The interface will ignore accesses while the bit is set.
2. If the read is preceded by an address/command (other than just a Slave Address) write the smb_cmd register with the 8 or 16 bit data to be sent.
3. Specify the device address and the transfer type by writing to the smb_start register. This will cause the transfer to begin. The busy bit in the smb_status register will be set to a one as long as the transfer is in progress.
4. When the transaction finishes the busy bit will be cleared. This can either be detected by polling the smb_status or by enabling the smb_finish interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction.
5. Read the data from the smb_cmd, smb_data and smb_xtra registers.
6. If PEC is enabled the smb_pec register should be read.

Writing to a slave on the serial bus involves these steps:

1. Wait until the busy bit in the smb_status register is clear. The interface will ignore accesses while the bit is set.
2. If an address/command is needed write 8 or 16 bits to the smb_cmd register.
3. Write the data to the smb_data and smb_xtra registers.
4. If PEC is enabled an extra data byte should be written to the smb_pec register.
5. Specify the device address and the transfer type by writing to the smb_start register. This will cause the transfer to begin. The busy bit in the smb_status register will be set to a one as long as the transfer is in progress.

The SMBus interface is accessed through a number of registers in the I/O portion of the memory map. The two interfaces are identical, register names in this discussion should have _0 or _1 appended to indicate the interface. There are two interrupts generated from each interface. Both interrupts are maskable through the control register. The smb_finish interrupt is generated when the interface completes an operation and the smb_busy status bit changes from one to zero, the interrupt is cleared by a read from the status register. The smb_error interrupt is generated when an error condition occurs, both the interrupt and status bit are cleared by writing a 1 to the error bit in the status register.

Prior to using the configuration interface the clock speed should be set. This is derived from the 100 MHz reference clock as configured in the smb_freq register. The two standard frequencies are 100 kHz and 400 kHz, but a much wider range can be programmed. Since the setup and hold time parameters vary between different slave devices it may be necessary to run the interface slightly slow to ensure reliable operation.
DIRECT ACCESS

The interface allows the hardware assist to be disabled giving direct ("bit-banged") access to the SDA and SCL lines. The CPU executes the whole protocol in software. This is useful for connecting to devices that deviate from the standard.

Direct access is enabled through the `smb_control` register. Two bits in the `smb_control` register are used to set the SDA and SCL lines to drive low or go high impedance, and two bits in the `smb_status` register reflect the current state of the lines. The output from the SMB bit clock is provided in the status register since it may be useful for timing transfers.

BOOTING USING AN SMBus EEPROM

The part can be configured to boot using an EEPROM connected to SMBus interface 0. When configured this way accesses to the generic bus chip select 0 space are converted into SMBus accesses using either the EEPROM Read or Read Word protocols. Since there is a large protocol overhead and the SMBus will be running at the default 100 kHz, accessing the bootstrap code in this way is slow. Typically, a small primary bootstrap program would be run from the EEPROM (which can also store the ethernet ids for the part), this would do sufficient initialization to load the system code from a master device.

The boot_type is configured using resistors on the generic bus pins `IO_AD[18:17]` as described in Section: “Reset” on page 26. The value on these is readable in the System Configuration register (Table 15 on page 43). While bit 18 is set in this register the SMBus 0 interface will only be used for servicing accesses to generic bus chip select 0.

While the SMBus interface is being used for booting any access to its control registers have UNPREDICTABLE results. Once booting is complete software can clear bit 18 in the System Configuration register to allow normal SMBus use. This will return the chip select 0 space to using the generic bus to service accesses. Following SMBus use software must set the chip select 0 timing parameters (even if it is setting the default values) before any access is made to the chip select 0 space or the results will be UNPREDICTABLE. When the system is reset by any method that causes the configuration bits to be sampled (See Table 8 on page 27), bit 18 in the System Configuration register will be set again, and the SMBus will be used for booting. If the watchdog timer is reset then the SCD will not be reset and the SMBus will not be reselected. In this configuration either the SMBus interface must be dedicated to be used by the generic chip select 0, or a ROM must be put on the generic bus to service the watchdog resets.

The boot mode works with EEPROMs that have the standard interface and use SMBus address 1010xxx. The low bits from address that the CPU issues to the generic bus (that would normally be put out on `IO_AD`) is sent to the SMBus interface, which runs an addressed cycle to the EEPROM and returns 32 bits of data that would normally be placed in the `smb_data` and `smb_xtra` registers. Two protocols are in use for these EEPROMS:

EEPROMS <= 16k bit

If the boot_type is configured as 2'b10 the part will boot from a small EEPROM (up to 2K bytes) accessed using a modified Read Word protocol that fetches four bytes:
- The upper 4 bits of the SMBus device address are set to 4'b1010.
- The low three bits of the SMBus device address are set to bits [10:8] of the address being accessed.
- The Command byte is set to bits [7:0] of the address being accessed.
EEPROMS > 16k bit (boot type = 2'b11)

If the boot_type is configured as 2'b11 the part will boot from a large EEPROM (bigger than 2K bytes) accessed using the EEPROM Read protocol.
The upper 4 bits of the SMBus device address are set to 4'b1010.
The low three bits of the SMBus device address are set to bits [18:16] of the address being accessed.
The Command byte is set to bits [15:8] of the address being accessed.
The Data byte is set to bits [7:0] of the address being accessed.

The data that returns from the SMBus interface is aligned and assembled to satisfy the request by the generic bus logic as if it had been returned by a 32 bit wide ROM attached to the generic bus. Note that if the SMBus transaction fails (for example the interface loses arbitration 15 times) then there is no way to recover and the boot process will hang (in this situation the CPU graduation timer will eventually trigger a Machine Check exception, but since the exception vector is also directed to the boot rom it will also be unable to make progress).

SWITCHING FROM SMBUS MODE

Care must be taken when switching out of the SMBus boot mode. There must be no accesses in progress to the chip select 0 region when the switch is made or the system will behave in UNDEFINED ways (for example a response could be lost). Instruction fetches are a particular concern and careful location of instructions is required to avoid them causing a problem. Two cases will be described as examples, in both cases the system is initially running from SMBus boot space in the generic chip select 0 region. (These examples are selected to provide simple illustration of the problem and are not representative of real systems.)

The first case is when the SMBus boot bit is cleared and the system continues running from a device in the chip select 0 generic bus region. This could be done when the SMBus code is used to configure the region with a different timing from the default.

```
// Fetching from cs0 region from SMBus
// t1 points to generic CS0 config registers
// t3 points to SCD registers
addr:  // this is on a cache block boundary
   sh  t5, io_ext_time_cfg0(t1) // set timing registers
   sh  t6, io_ext_time_cfg1(t1)
   sync
   sync
addr+16:
   sh  t4, io_ext_cfg(t1) // set config
   sd  t7, system_cfg(t3) // remove SMBus boot
   sync
   sync
addr+32:
   // Fetched from cs0 on the generic bus
```
In this example the chip select 0 timing registers are configured in one instruction bundle. The sync instructions serve to pad out the bundle and prevent the next four instructions being fetched until the stores have been issued (the SB-1 will only issue an uncacheable instruction fetch when the previous instructions have graduated). The second group of four instructions write to the configuration to complete the setup and to the system_cfg register to disable SMBus booting. Again the sync instructions are used to ensure these writes have issued. This is sufficient to ensure the mode has changed and the next instruction fetch will be serviced from the generic bus.

The second example is switching to the chip select 1 region at the same address. This can be done to move to a generic bus device while leaving the SMBus mode enabled to allow a soft reset to revert back to SMBus mode. Again care must be taken during the switch-over.

Here the timing is set for the chip select 1 region in the first instruction bundle. The second group sets the size of the chip select 1 region, changes the base of chip select 0 to move it out of the way and configures the base of chip select 1 to 00_1FC0_0000. It is important that the second group of instructions are together since they change the target of the next instruction fetch. Again the sync instruction is used to ensure that the stores have completed before the CPU will issue the next instruction fetch.

If the chip select region 1 were located at a different address then there would be no need to move the chip select region 0 base address. In this case the two CS1 stores (of t5 and t7) would be the first two instructions in the bundle, the third instruction would be the branch to the new address range and the sync would be in the branch delay slot. Having the branch third and the sync in the delay slot will ensure that the stores are completed (due to the sync and the SB-1 not fetching more instructions until the current ones graduate) and the next instruction fetch comes from the chip select 1 space (because of the jump and the fact that its delay slot is satisfied from the current instruction bundle).
The SMBus registers are mapped into the internal I/O section of the memory space. They all occupy a 64 bit wide slot, although in most cases only one or two bytes are implemented. When a register is written its entire contents must be written, writes that are smaller than the implemented width will result in the register value becoming UNPREDICTABLE.

**Table 289: SMBus Clock Frequency Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:0</td>
<td>smb_freq_div</td>
<td>13'h7d</td>
<td>The 100 MHz sys_clock is divided by this x 8 to determine the frequency of the serial clock. The default value is for 100 kHz. The minimum SMBus clock is 10 kHz which is generated using a value of 1250. For 400 kHz the value for this field is 31.</td>
</tr>
<tr>
<td>63:13</td>
<td>notimp</td>
<td>51'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

**Table 290: SMBus Command Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>smb_cmd</td>
<td>8'h0</td>
<td>Write: Low byte of command to be sent following address. Read: First byte received in type 5 or 6 extended mode read.</td>
</tr>
<tr>
<td>15:8</td>
<td>smb_cmdh</td>
<td>8'h0</td>
<td>Write: High byte of command to send in extended mode. Read: 2nd Byte received in type 6 extended mode read.</td>
</tr>
<tr>
<td>63:15</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>

**Table 291: SMBus Control Registers**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>smb_mk</td>
<td>1'b0</td>
<td>When high, enables interrupt if error is high.</td>
</tr>
<tr>
<td>1</td>
<td>smb_finish_en</td>
<td>1'b0</td>
<td>When high, an interrupt will be generated when busy transitions from high to low. The interrupt is cleared by reading the smb_status register.</td>
</tr>
<tr>
<td>3:2</td>
<td>reserved</td>
<td>6'h0</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>smb_data_out</td>
<td>1'b0</td>
<td>Driven to data pin in direct mode if smb_data_dir is set.</td>
</tr>
<tr>
<td>5</td>
<td>smb_data_dir</td>
<td>1'b0</td>
<td>Data direction in direct mode, set for output clear to turn driver off.</td>
</tr>
<tr>
<td>6</td>
<td>smb_clk_out</td>
<td>1'b0</td>
<td>Driven to clock pin in direct mode</td>
</tr>
<tr>
<td>7</td>
<td>smb_direct</td>
<td>1'b0</td>
<td>Set to enable direct control of clock and data lines (from bits 6:4).</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 292: SMBus Status Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>smb_busy</td>
<td>1'b0</td>
<td>When high, indicates the serial interface is busy; when low, it is not busy.</td>
</tr>
<tr>
<td>1</td>
<td>smb_error</td>
<td>1'b0</td>
<td>When high, indicates an error has occurred. If the smb_mk bit in the smb_control register has a value of 1'b1, an interrupt will be generated. The interrupt and this status bit are cleared by writing this bit with a 1.</td>
</tr>
<tr>
<td>2</td>
<td>error_type</td>
<td>1'h0</td>
<td>When bit 1 is high this bit indicates the error that occurred. This bit is cleared if an expected acknowledgement is not seen and set if the transfer has failed after 15 retries.</td>
</tr>
<tr>
<td>5</td>
<td>smb_scl_in</td>
<td>1'bx</td>
<td>Reserved. Will be used to provide value on SCL pin in later revisions of the part.</td>
</tr>
<tr>
<td>4:3</td>
<td>reserved</td>
<td>2'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>smb_ref</td>
<td>1'b0</td>
<td>In direct mode (smb_direct set in the control register) this bit shows the output of the serial clock generator (that would be driven to the clock pin in normal mode). Software may use this as a reference clock. If direct mode is not enabled this bit is zero.</td>
</tr>
<tr>
<td>7</td>
<td>smb_data_in</td>
<td>1'b0</td>
<td>In direct mode (smb_direct set in the control register) this bit shows the value on the data pin, otherwise it is zero.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 293: SMBus Data Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>smb_lb</td>
<td>8'h0</td>
<td>Least significant data byte that was read (or will be written) across the serial bus.</td>
</tr>
<tr>
<td>15:8</td>
<td>smb_mb</td>
<td>8'h0</td>
<td>Most significant data byte that was read (or will be written) across the serial bus.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 294: SMBus Extra Data Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>smb_lb</td>
<td>8'h0</td>
<td>Third data byte that was read in an EEPROM read transfer.</td>
</tr>
<tr>
<td>15:8</td>
<td>smb_mb</td>
<td>8'h0</td>
<td>Fourth data byte that was read in an EEPROM read transfer.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>
### Table 295: SMBus Packet Error Check Registers

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>pec</td>
<td>8'h0</td>
<td>This register holds the PEC information that is sent or received for a command that has the smb_pec bit set. The hardware does not generate or check the CRC, software must do that.</td>
</tr>
<tr>
<td>63:8</td>
<td>notimp</td>
<td>56'bx</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

### Table 296: SMBus Start and Command Registers SMBus Mode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:0</td>
<td>smb_addr</td>
<td>7'h0</td>
<td>The serial interface address. These 7 bits are used to form the address byte that is sent across the serial interface.</td>
</tr>
<tr>
<td>7</td>
<td>smb_qdata</td>
<td>1'b0</td>
<td>Bit of data to send as R/W bit in a Quick Command. Ignored, write as zero for all other commands.</td>
</tr>
<tr>
<td>10:8</td>
<td>smb_tt</td>
<td>3'h0</td>
<td>Value Transfer Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>000 1 byte write (address, command).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>001 2 byte write (address, command, write least significant field of smb_data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>010 3 byte write (address, command, write both fields of smb_data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>011 Command and 1 byte read (address, command, address, read byte into LB field of smb_data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>100 Command and 2 byte read (address, command, address, read byte into LB field of smb_data, read byte into HB field of smb_data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>101 1 byte read (address, read byte into LB field of smb_data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>110 Quick Command (address with one bit of data).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>111 EEPROM Read. Reads 32 bits into smb_data and smb_xtra from a standard EEPROM &gt;16 kbit that needs 2 address bytes.</td>
</tr>
<tr>
<td>13:11</td>
<td>reserved</td>
<td>3'h0</td>
<td>reserved, set to zero</td>
</tr>
<tr>
<td>14</td>
<td>extend</td>
<td>1'b0</td>
<td>This bit should be set to zero for SMBus protocol mode.</td>
</tr>
<tr>
<td>15</td>
<td>smb_pec</td>
<td>1'b0</td>
<td>Set to modify the transfer type to use Packet Error Checking as defined in the SMBus 1.1 specification. Zero for regular operation.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48'bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
### Table 297: SMBus Start and Command Registers Extended Mode

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6:0</td>
<td>smb_addr</td>
<td>7’h0</td>
<td>The serial interface address. These 7 bits are used to form the address byte that is sent across the serial interface.</td>
</tr>
<tr>
<td>7</td>
<td>smb_qdata</td>
<td>1’b0</td>
<td>Bit of data to send as R/W bit in a Quick Command. Ignored, write as zero for all other commands.</td>
</tr>
<tr>
<td>10:8</td>
<td>smb_datafmt</td>
<td>3’h0</td>
<td>Data Format</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Data Transfer</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>Data[7:0]</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td></td>
<td>Data[7:0], Data[15:8]</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td>Data[7:0], Data[15:8], Xtra[7:0]</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td></td>
<td>Data[7:0], Data[15:8], Xtra[7:0], Xtra[15:8]</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td>No data.</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>Only valid for read transactions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cmd[7:0], Data[7:0], Data[15:8], Xtra[7:0],Xtra[15:8]</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td>Only valid for read transactions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cmd[7:0], Cmd[15:8], Data[7:0], Data[15:8], Xtra[7:0], Xtra[15:8]</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12:11</td>
<td>smb_afmt</td>
<td>2’h0</td>
<td>Address/Command Format</td>
</tr>
<tr>
<td></td>
<td>Value</td>
<td>Address/Command Transfer</td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>Only valid for read transactions: No Address.</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>Slave Address Only</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Slave Address then Cmd[7:0]</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Slave Address then Cmd[7:0], Cmd[15:0]</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>smb_dir</td>
<td>1’b0</td>
<td>Set to 0 for a write transaction, 1 for a read transaction.</td>
</tr>
<tr>
<td>14</td>
<td>extend</td>
<td>1’b1</td>
<td>This bit should be set to one for extended protocol mode.</td>
</tr>
<tr>
<td>15</td>
<td>smb_pec</td>
<td>1’b0</td>
<td>Set to modify the transfer type to use Packet Error Checking as defined in the SMBus 1.1 specification. Zero for regular operation.</td>
</tr>
<tr>
<td>63:16</td>
<td>notimp</td>
<td>48’bx</td>
<td>Not Implemented.</td>
</tr>
</tbody>
</table>
Section 15: JTAG and Debug

INTRODUCTION

The part includes extensive debugging and performance monitoring features. These can be accessed by the CPU(s) or through the JTAG port. The JTAG interface supports the required subset of MIPS EJTAG 2.5 and additions to allow both processors to be debugged. The debug and bus trace unit in the SCD can also be controlled through the JTAG port.

A JTAG probe may be connected to the part for accessing both the boundary scan and debug functions. There is a standard connector for the MIPS EJTAG interface. The application note 'BCM1250 EJTAG Connector' #00-117 from Corelis Inc. (www.corelis.com) includes a recommendation for the connector and terminations for the BCM1250 and BCM1125/H.

TAP CONTROLLER

The SB1250 implements a IEEE 1149.1 compliant Test Access Port (TAP) controller. The TMS pin is used to control the tap state machine. If TRST_L is asserted then the TAP controller state machine enters the Test-Logic-Reset state.

Table 298: JTAG Signals

<table>
<thead>
<tr>
<th>JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDI</td>
<td>Test data in. Data is shifted serially in to the part through this input.</td>
</tr>
<tr>
<td>TDO</td>
<td>Test data out. Data is shifted serially out of the part through this output.</td>
</tr>
<tr>
<td>TMS</td>
<td>Test Mode select. TAP state machine control signal. The value captured on the rising edge of TCK determines the state transition.</td>
</tr>
<tr>
<td>TCK</td>
<td>Test clock. The JTAG interface can be clocked at up to 10 MHz.</td>
</tr>
<tr>
<td>TRST_L</td>
<td>Test Reset. Asserting this signal forces the TAP state machine in to the Test-Logic-Reset state.</td>
</tr>
</tbody>
</table>

Figure 84 on page 422 shows the state machine used by the JTAG interface. This is the standard JTAG state machine. TRST_L can be used to reset the state machine to the Test-Logic-Reset state, which will also be entered if TMS is asserted for five TCK rising edges. The state machine advances each TCK rising edge as directed by the TMS level. The states are as defined in the JTAG specification. The IR scan branch is used to scan instructions into the Instruction register described in Section: “Instruction Register” on page 423. The DR branch scans bits into and out of whatever data scan chain has been selected by an instruction.

When TRST_L is asserted or the TAP enters the Test-Logic-Reset state, the instruction register is loaded with the IDCODE instruction, any EJTAGBOOT indication is removed and the EJTAG Control register is cleared.

The COLDRES_L input is also used to reset the TAP, this ensures that the scan chains and boundary scan drivers remain in a safe state as the power comes up. A side effect is that the TAP is held in reset and none of the JTAG machinery can be used while COLDRES_L is asserted. If testing is to be done using the JTAG interface with the BCM1250 held in reset then RESET_L should be used to keep the system in reset while allowing the JTAG port to be active. The standard EJTAG connector allows the probe to control RESET_L.
Figure 84: JTAG TAP State Machine
**INSTRUCTION REGISTER**

The TAP implements a 6 bit instruction register. The instructions are shown in Table 299. The required BYPASS, SAMPLE/PRELOAD, and EXTEST instructions are implemented, along with the optional IDCODE, INTEST and CLAMP instructions. 25 additional BCM1250 specific instructions have been implemented. The instruction register is scanned in and out LSB first. The value scanned out of the instruction register has a 1 in the LSB and zeros in all other bits.

**Table 299: JTAG Instructions**

<table>
<thead>
<tr>
<th>IR Value</th>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>EXTEST</td>
<td>Selects the boundary scan register for EXTEST. This is used to drive the external pins for testing board connectivity.</td>
</tr>
<tr>
<td>0x01</td>
<td>IDCODE</td>
<td>Selects Chip Identification Data Register.</td>
</tr>
<tr>
<td>0x02</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x03</td>
<td>IMPCODE</td>
<td>Selects the Implementation Code Register.</td>
</tr>
<tr>
<td>0x04-0x07</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td>ADDRESS</td>
<td>Selects the EJTAG Address Register.</td>
</tr>
<tr>
<td>0x09</td>
<td>DATA</td>
<td>Selects the EJTAG Data Register.</td>
</tr>
<tr>
<td>0x0A</td>
<td>CONTROL</td>
<td>Selects the EJTAG Control Register.</td>
</tr>
<tr>
<td>0x0B</td>
<td>EJTAGALL</td>
<td>Selects the EJTAG Address, Data and Control Registers.</td>
</tr>
<tr>
<td>0x0C</td>
<td>EJTAGBOOT</td>
<td>Force the CPUs to boot in debug mode.</td>
</tr>
<tr>
<td>0x0D</td>
<td>NORMALBOOT</td>
<td>Force the CPUs to boot in the normal mode.</td>
</tr>
<tr>
<td>0x0E-0x1F</td>
<td>EJTAG reserved</td>
<td>Reserved for EJTAG.</td>
</tr>
<tr>
<td>0x20</td>
<td>SYSCTRL</td>
<td>Selects System Control and Status Register.</td>
</tr>
<tr>
<td>0x21</td>
<td>TRACE</td>
<td>Selects Trace Register for Scan Out.</td>
</tr>
<tr>
<td>0x22</td>
<td>PERF</td>
<td>Selects Performance counters for Scan Out.</td>
</tr>
<tr>
<td>0x23</td>
<td>TRACECTRL</td>
<td>Selects the Trace Control Registers for Scan Out/in.</td>
</tr>
<tr>
<td>0x24</td>
<td>WAFERID</td>
<td>Selects the Wafer ID for Scan Out.</td>
</tr>
<tr>
<td>0x25</td>
<td>PROCESSMON</td>
<td>Selects the Ring Oscillator Register for Scan In/Out.</td>
</tr>
<tr>
<td>0x26</td>
<td>CPU0OSC</td>
<td>Broadcom Use Only. Selects CPU0 Observability Chain.</td>
</tr>
<tr>
<td>0x27</td>
<td>CPU0DSC</td>
<td>Broadcom Use Only. Selects CPU0 Debug Chain.</td>
</tr>
<tr>
<td>0x28</td>
<td>CPU0TSC</td>
<td>Broadcom Use Only. Selects CPU0 Test Chain.</td>
</tr>
<tr>
<td>0x29</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x2A</td>
<td>CPU1OSC</td>
<td>Broadcom Use Only. Selects CPU1 Observability Chain.</td>
</tr>
<tr>
<td>0x2B</td>
<td>CPU1DSC</td>
<td>Broadcom Use Only. Selects CPU1 Debug Chain.</td>
</tr>
<tr>
<td>0x2C</td>
<td>CPU1TSC</td>
<td>Broadcom Use Only. Selects CPU1 Test Chain.</td>
</tr>
<tr>
<td>0x2D</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x2E</td>
<td>SCANIOB0</td>
<td>Broadcom Use Only. Selects IOB0 Chain.</td>
</tr>
<tr>
<td>0x2F</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x30</td>
<td>SCANIOB1</td>
<td>Broadcom Use Only. Selects IOB1 Chain.</td>
</tr>
<tr>
<td>0x31</td>
<td>reserved</td>
<td></td>
</tr>
<tr>
<td>0x32</td>
<td>SCANL2C</td>
<td>Broadcom Use Only. Selects L2C Chain.</td>
</tr>
</tbody>
</table>
### BYPASS Instruction

The required BYPASS instruction allows the processor to remain in a functional mode and selects the Bypass register to be connected between TDI and TDO. The BYPASS instruction allows serial data to be transferred through the processor from TDI to TDO without affecting its operation. The bit code of this instruction is defined to be all ones by the IEEE 1149.1 standard. All unused instructions default to the BYPASS instruction.

### IDCODE Instruction

The IDCODE instruction selects the Device Identification (ID) register to be connected between TDI and TDO. The Device ID register is a 32-bit shift register containing the 11-bit manufacturer ID (11'h150), the 16-bit part number (this matches the part number in the system_revision register) and the 4-bit version number. The idcode register is scanned out LSB first and is shown in Table 300.

The version number is 4'h1 for prototype BCM1250, 4'h2-4'h8 or 4'h9 for initial production BCM1250 and 4'h9 for later production BCM1250 (stepping C0). The version number is 4'h1 for initial production BCM1125/H.

### WAFERID Instruction

The WAIFERID instruction selects the WaferId register to be connected between TDI and TDO. The WaferId register is a 32-bit shift register containing a manufacturing lot code number. The register is scanned out LSB last.
first and is shown in Table 301. This register can also be read in the top bits of the system_revision register (see Table 14 on page 43).

### Table 301: JTAG Wafer ID Register

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lot (swap bits 26-27, 20-21, 18-19)</th>
<th>Wafer</th>
<th>Broadcom Use Information</th>
</tr>
</thead>
</table>

**IMPCODE Instruction**

This instruction selects the Implementation register for output, which is always 32 bits. The impcode for the BCM1250 and BCM1125/H is 32'h21404001 indicating EJTAG 2.5, R4k style CP0, DINT supported, ASID size is 8 bits, no MIPS16 support, no EJTAG DMA, MIPS64 support. This code is defined by revision 2.5 of the MIPS EJTAG spec. (Older versions of the parts incorrectly report 32'h20814001.)

**ADDRESS Instruction**

This instruction is used to cause the Address Register to be connected between TDI and TDO. The EJTAG Probe shifts 77 bits through the TDI pin into the Address Register and shifts out the captured address via the TDO pin. The Address Register is described in Section: “Address Register” on page 438.

**DATA Instruction**

This instruction is used to cause the Data Register to be connected between TDI and TDO. The EJTAG Probe shifts 277 bits of data through the TDI pin into the Data Register as the captured data is shifted out via the TDO pin. The Data Register is described in Section: “Data Register” on page 438.

**CONTROL Instruction**

This instruction is used to select the EJTAG Control register to be connected between TDI and TDO. The EJTAG Probe shifts 12 bits of data into the EJTAG Control register through the TDI pin, and shifts out the current value via TDO. The Control Register is described in Section: “EJTAG Control Register” on page 439. See the description of Probe access in Section: “Probe Accesses to the ZBbus” on page 437.

**EJTAGALL Instruction**

This instruction is used to select the concatenation of the Address Register, the Data Register, and the EJTAG Control register between TDI and TDO. The total chain is 366 bits. It can be used to accelerate scanning out of requests, since there is no need for instructions to be used to switch between the scan chains.

**EJTAGBOOT Instruction**

When the EJTAGBOOT instruction is given and the Update-IR state is left, the PrTrap0 and PrTrap1 bits are set in the EJTAG control register to assert the DBBOOT signal to both CPUs. The ProbEn bit is set to allow CPU accesses to the JTAG space.
The DBBOOT signal will force the CPU to use the alternate debug vector when it enters debug mode. When the CPUs are next reset, they will sample their DBBOOT signals and if it is asserted they will enter Debug Mode and start execution from the alternative debug vector (physical 00_1000_0480). The CPUs will therefore enter debug mode and execute code from the JTAG probe, without fetching or executing any instructions from the normal boot vector. This can be used to download code into a system which has no code in ROM.

The EJTAGBOOT indication is effective until the NORMALBOOT instruction is given, TRST_L asserted or a rising edge of TCK occurs when TAP controller is in the Test-Logic-Reset state. The PrTrap0 and PrTrap1 bits may be cleared by scanning in to the EJTAG Control Register and will read back as zeros, but this is not sufficient to clear the EJTAGBOOT state. The Bypass register is selected when the EJTAGBOOT instruction is given.

**NORMALBOOT Instruction**

When the NORMALBOOT instruction is given and Update-IR state is left, then the DBBOOT signal is deasserted and the PrTrap0, PrTrap1 and ProbEn bits in the EJTAG Control register are set to 0. Since DBBOOT is deasserted the CPUs will start at the standard Reset vector the next time they are reset. The Bypass register is selected when the NORMALBOOT instruction is given.

**SCAN Instructions (0x26 - 0x38)**

When these instructions are given the corresponding scan chain is selected. When the SCANALL instruction is given then all of the scan chains are selected in series in order TDI to MC to SCD to L2C to IOB1 to IOB0 to CPU1 to CPU0 to TDO. These scan chains are for Broadcom Use Only. Scanning inappropriate values into the chains can put the BCM1250 in an UNDEFINED state that requires both a system and a JTAG reset to clear.

**SYSCTRL Instruction**

The system control and status register is used to control the part for testing and debug purposes. This register is defined by Table 15 on page 43. The System Control Scanchain, summarized in Table 15, includes this register and additional control bits.
### Table 302: System Control Scan Chain

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>reserved</td>
<td>ext</td>
<td>Configuration bit for IO_AD[0].</td>
</tr>
<tr>
<td>1</td>
<td>reserved</td>
<td>ext</td>
<td>Configuration bit for IO_AD[1]. Indicates source for IO_CLK100. This must not be changed while the BCM1250 is operating.</td>
</tr>
<tr>
<td>2</td>
<td>ldt_minrstcnt</td>
<td>ext</td>
<td>Configuration bit for IO_AD[2]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>3</td>
<td>ldt_bypass_pll</td>
<td>ext</td>
<td>Configuration bit for IO_AD[3]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>4</td>
<td>pci_test</td>
<td>ext</td>
<td>Configuration bit for IO_AD[4]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
<tr>
<td>5</td>
<td>iob0_div</td>
<td>ext</td>
<td>Configuration bit for IO_AD[5] that controls the clock divider for I/O Bridge 0. 0: I0B0 runs at CPU clock/4, for use with fast CPU clocks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: I0B0 runs at CPU clock/3, for use with slow CPU clocks.</td>
</tr>
<tr>
<td>6</td>
<td>iob1_div</td>
<td>ext</td>
<td>Configuration bit for IO_AD[6] that controls the clock divider for I/O Bridge 1. 0: I0B1 runs at CPU clock/3, for use with fast CPU clocks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: I0B1 runs at CPU clock/2, for use with slow CPU clocks.</td>
</tr>
<tr>
<td>11:7</td>
<td>pll_div</td>
<td>ext</td>
<td>Configuration bits for IO_AD[11:7] that select the PLL Divide ratio. These bits must not be changed while the BCM1250 is operating.</td>
</tr>
<tr>
<td>12</td>
<td>ser0_enable</td>
<td>ext</td>
<td>Configuration bit for IO_AD[12]. 0: Serial interface 0 is in asynchronous (uart) mode. 1: Serial interface 0 is in synchronous mode.</td>
</tr>
<tr>
<td>13</td>
<td>ser0_rstb_en</td>
<td>ext</td>
<td>Configuration bit for IO_AD[13] that allocates GPIO[0] pin to the synchronous serial interface.</td>
</tr>
<tr>
<td>14</td>
<td>ser1_enable</td>
<td>ext</td>
<td>Configuration bit for IO_AD[14]. 0: Serial interface 1 is in asynchronous (uart) mode. 1: Serial interface 1 is in synchronous mode.</td>
</tr>
<tr>
<td>16</td>
<td>pcmcia_enable</td>
<td>ext</td>
<td>Configuration bit for IO_AD[16] that configures the PCMCIA mode.</td>
</tr>
<tr>
<td>18:17</td>
<td>boot_mode</td>
<td>ext</td>
<td>Configuration bit for IO_AD[18:17] that configures the boot mode. 00: 32 bit generic bus ROM (multiplexed). 01: 8 bit generic bus ROM (non-multiplexed).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10: SMBus EEPROM &lt;= 16 kbit (read word protocol). 11: SMBus EEPROM &gt; 16kbit (eeprom read word protocol).</td>
</tr>
<tr>
<td>19</td>
<td>pci_host</td>
<td>ext</td>
<td>Configuration bit for IO_AD[19], that configures the PCI interface to be host or device mode.</td>
</tr>
<tr>
<td>20</td>
<td>pci_arbiter</td>
<td>ext</td>
<td>Configuration bit for IO_AD[20], that configures the PCI interface to use an internal or external arbiter. (If the PCI is set in device mode the resistor must be set for an external arbiter)</td>
</tr>
<tr>
<td>21</td>
<td>southOnLDT</td>
<td>ext</td>
<td>Configuration bit for IO_AD[21], that configures the southbridge to be on the HyperTransport fabric or PCI bus.</td>
</tr>
<tr>
<td>22</td>
<td>big_endian</td>
<td>ext</td>
<td>Configuration bit for IO_AD[22], that configures the system to be big or little endian.</td>
</tr>
<tr>
<td>23</td>
<td>genclk_en</td>
<td>ext</td>
<td>Configuration bit for IO_AD[23], that enables output of the generic bus clock on IO_CLK100. If this bit is zero then the IO_CLK100 will be held in a high impedance state.</td>
</tr>
<tr>
<td>24</td>
<td>ldt_test_en</td>
<td>ext</td>
<td>Configuration bit for IO_AD[24]. Broadcom Use Only. This must be zero for normal operation.</td>
</tr>
</tbody>
</table>
### Table 302: System Control Scan Chain (Cont.)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:26</td>
<td>config</td>
<td>ext</td>
<td>Configuration bit for IO_AD[31:26]. These configuration bits are available for interpretation by software.</td>
</tr>
<tr>
<td>32</td>
<td>clkstop</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only. Takes effect on UpdateDR.</td>
</tr>
<tr>
<td>33</td>
<td>clkstep</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>41:34</td>
<td>clkcount</td>
<td>8'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>42</td>
<td>pllbypass</td>
<td>1'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>44:43</td>
<td>pll_iref</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>46:45</td>
<td>pll_vco</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>48:47</td>
<td>pll_vreg</td>
<td>2'b0</td>
<td>Writable via JTAG only. Broadcom use only.</td>
</tr>
<tr>
<td>49</td>
<td>mem_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the memory controller is held in reset.</td>
</tr>
<tr>
<td>50</td>
<td>l2c_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the level 2 cache is held in reset.</td>
</tr>
<tr>
<td>51</td>
<td>io_reset_0</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the I/O bridge to the PCI and HyperTransport fabric is held in reset.</td>
</tr>
<tr>
<td>52</td>
<td>io_reset_1</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the I/O bridge to the slow speed devices and generic bus is held in reset.</td>
</tr>
<tr>
<td>53</td>
<td>scd_reset</td>
<td>1'b0</td>
<td>Writable via JTAG only. When set the SCD is held in reset.</td>
</tr>
<tr>
<td>54</td>
<td>cpu_reset_0</td>
<td>1'b0</td>
<td>When set CPU 0 will be held in reset.</td>
</tr>
<tr>
<td>55</td>
<td>cpu_reset_1</td>
<td>1'b1</td>
<td>When set CPU 1 will be held in reset. This bit is set on a BCM1250 reset, causing the processor to remain in reset until released under software control.</td>
</tr>
<tr>
<td>56</td>
<td>unicpu0</td>
<td>1'b0</td>
<td>Set to indicate uniprocessor using physical processor 0. (This bit will always be set on the BCM1125/H.)</td>
</tr>
<tr>
<td>57</td>
<td>unicpu1</td>
<td>1'b0</td>
<td>Set to indicate uniprocessor using physical processor 1.</td>
</tr>
<tr>
<td>58</td>
<td>sb_softres</td>
<td>1'b0</td>
<td>When a write changes this bit from a 0 to a 1 a soft reset will be performed. This will reset of whole chip except for this register. Note that once it is set the bit must be cleared before writing a 1 will again cause a soft reset.</td>
</tr>
<tr>
<td>59</td>
<td>ext_reset</td>
<td>1'b0</td>
<td>When set the RESETOUT_L pin will be asserted.</td>
</tr>
<tr>
<td>60</td>
<td>system_reset</td>
<td>1'b0</td>
<td>When written with a 1 a full system reset will be performed (thus setting this bit back to zero).</td>
</tr>
<tr>
<td>61</td>
<td>misr_mode</td>
<td>1'b0</td>
<td>Broadcom Use Only. Set MISR mode (1=misr 0=capture).</td>
</tr>
<tr>
<td>62</td>
<td>scd_misr_reset</td>
<td>1'b0</td>
<td>Broadcom Use Only. Reset SCD MISRs on 0-&gt;1 transition.</td>
</tr>
<tr>
<td>63</td>
<td>reserved</td>
<td>1'b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>95:64</td>
<td>stp_cnt</td>
<td>32'b0</td>
<td>Broadcom Use Only. Stop counter set to zero for normal operation.</td>
</tr>
<tr>
<td>96</td>
<td>pllstop</td>
<td>1'b0</td>
<td>Broadcom Use Only. Set to zero for normal operation.</td>
</tr>
<tr>
<td>97</td>
<td>stop_stre</td>
<td>1'b0</td>
<td>Broadcom Use Only. Set to zero for normal operation.</td>
</tr>
<tr>
<td>98</td>
<td>start_trc</td>
<td>1'b0</td>
<td>Broadcom Use Only. Set for trace, clear for reset.</td>
</tr>
<tr>
<td>99</td>
<td>stopping</td>
<td>1'b0</td>
<td>Broadcom Use Only. Read Only.</td>
</tr>
<tr>
<td>100</td>
<td>ss_done</td>
<td>1'b0</td>
<td>Broadcom Use Only. Read Only.</td>
</tr>
<tr>
<td>101</td>
<td>zbser_ard</td>
<td>1'b0</td>
<td>Broadcom Use Only. Serialize A-R-D. Set to zero for normal operation.</td>
</tr>
<tr>
<td>102</td>
<td>zbser_ar</td>
<td>1'b0</td>
<td>Broadcom Use Only. Serialize A-R. Set to zero for normal operation.</td>
</tr>
</tbody>
</table>
On the BCM1250 Bits 104:0 of this register are selected in between TDI and TDO, on the BCM1125/H there are two additional Broadcom Use Only bits so bits 106:0 are selected. The scan order is from LSB to MSB. The configuration bits may be written by the JTAG probe while the part is held in reset using RESET_L provided the cold reset delay (Tcr) described in the Hardware Data Sheet has elapsed since COLDRES_L was released. Most of the other bits in this scanchain are for Broadcom Use Only, incorrect settings will cause the part to behave in UNDEFINED ways and could require a full cold reset to restore deterministic behavior. The CPU uniprocessor bits (in the resets field) are normally set by only on uniprocessor parts. However if one of these bits is set in the control register and then the Broadcom-soft-reset bit is set, following the reset the BCM1250 will behave as a uniprocessor. This can be used to disable one of the processors, it will not be clocked and will enter a low power state.

Broadcom soft reset occurs only when a 1 is scanned into this register, and the TAP controller enters the Update-DR state. This reset is just like a normal reset except that the contents of the system control register are not reset. This allows testing of various modes.

TRACEx Instruction

When the TRACE instruction is set, the trace_read register is selected between TDI and TDO. This is a read only register. It is scanned out LSB first. The trace buffer should be frozen and the startread bit set prior to reading out. This is a 64 bit register, and gets the next 64 bits of trace data every time the Capture-DR state is entered. It performs just as if the data had been read out using address 00_1002_0a08. Following a capture the first scan of the register returns UNPREDICTABLE data and the next will return the first valid bits. After the entire contents of the trace buffer have been read, it will return all zeros.

PERF Instruction

The PERF instruction selects the performance registers for scan in and out. Note that the value scanned in will overwrite the counters. The performance chain is scanned out LSB first and is made up of the performance registers as shown in Table 303.

**Table 303: Performance Counter Scan Chain**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>33:0</td>
<td>perf_cnt_cfg[33:0]</td>
<td>Performance counter configuration (See Table 31 on page 61).</td>
</tr>
<tr>
<td>74:34</td>
<td>perf_cnt_0[40:0]</td>
<td>Performance counters (See Table 32 on page 62).</td>
</tr>
<tr>
<td>115:75</td>
<td>perf_cnt_1[40:0]</td>
<td></td>
</tr>
<tr>
<td>156:116</td>
<td>perf_cnt_2[40:0]</td>
<td></td>
</tr>
<tr>
<td>197:157</td>
<td>perf_cnt_3[40:0]</td>
<td></td>
</tr>
</tbody>
</table>

---

**Table 302: System Control Scan Chain (Cont.)**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>103:104</td>
<td>str_mode</td>
<td>2'b0</td>
<td>Broadcom Use Only, sets stretch mode for PLL. Set to zero for normal operation.</td>
</tr>
</tbody>
</table>
TRACECTRL and TRACECURCNT Instructions

The trace control registers are selected for both scan in and out using the TRACECTRL instruction. The registers are connected between TDI and TDO and are scanned out LSB first. The registers are scanned out in the order shown in Table 304, starting with trace_event_0[0] and ending with trace_sequence_7[24].

Writing values in the trace control scan chain has the same effect as writing the registers from the CPU. In particular when the event count field is written both the current and maximum count get written.

The TRACECURCNT instruction allows the current event counts to be read without forcing an update. The read-only Trace Current Count Scan Chain has the order shown in Table 304.

**Table 304: Trace Control Scan Chain**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>trace_event_0[31:0]</td>
<td>Trace Event Registers (See Table 46 on page 72).</td>
</tr>
<tr>
<td>63:32</td>
<td>trace_event_1[31:0]</td>
<td></td>
</tr>
<tr>
<td>95:64</td>
<td>trace_event_2[31:0]</td>
<td></td>
</tr>
<tr>
<td>127:96</td>
<td>trace_event_3[31:0]</td>
<td></td>
</tr>
<tr>
<td>152:128</td>
<td>trace_sequence_0[24:0]</td>
<td>Trace sequence Registers (See Table 47 on page 74).</td>
</tr>
<tr>
<td>177:153</td>
<td>trace_sequence_1[24:0]</td>
<td></td>
</tr>
<tr>
<td>202:178</td>
<td>trace_sequence_2[24:0]</td>
<td></td>
</tr>
<tr>
<td>227:203</td>
<td>trace_sequence_3[24:0]</td>
<td></td>
</tr>
<tr>
<td>259:228</td>
<td>trace_event_4[31:0]</td>
<td>Trace Event Registers (See Table 46 on page 72).</td>
</tr>
<tr>
<td>291:260</td>
<td>trace_event_5[31:0]</td>
<td></td>
</tr>
<tr>
<td>323:292</td>
<td>trace_event_6[31:0]</td>
<td></td>
</tr>
<tr>
<td>355:324</td>
<td>trace_event_7[31:0]</td>
<td></td>
</tr>
<tr>
<td>380:356</td>
<td>trace_sequence_4[24:0]</td>
<td>Trace sequence Registers (See Table 47 on page 74).</td>
</tr>
<tr>
<td>405:381</td>
<td>trace_sequence_5[24:0]</td>
<td></td>
</tr>
<tr>
<td>430:406</td>
<td>trace_sequence_6[24:0]</td>
<td></td>
</tr>
<tr>
<td>455:431</td>
<td>trace_sequence_7[24:0]</td>
<td></td>
</tr>
</tbody>
</table>

**Table 305: Trace Current Count Scan Chain**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7:0</td>
<td>trace_event_cnt_0[7:0]</td>
<td>Current count for Trace Event Registers (See Table 46 on page 72).</td>
</tr>
<tr>
<td>15:8</td>
<td>trace_event_cnt_1[7:0]</td>
<td></td>
</tr>
<tr>
<td>23:16</td>
<td>trace_event_cnt_2[7:0]</td>
<td></td>
</tr>
<tr>
<td>31:24</td>
<td>trace_event_cnt_3[7:0]</td>
<td></td>
</tr>
<tr>
<td>39:32</td>
<td>trace_event_cnt_4[7:0]</td>
<td></td>
</tr>
<tr>
<td>47:40</td>
<td>trace_event_cnt_5[7:0]</td>
<td></td>
</tr>
<tr>
<td>55:48</td>
<td>trace_event_cnt_6[7:0]</td>
<td></td>
</tr>
<tr>
<td>63:56</td>
<td>trace_event_cnt_7[7:0]</td>
<td></td>
</tr>
</tbody>
</table>
**PROCESSMON Instruction**

The PROCESSMON instruction selects the ring oscillator register for scan in and out. When the instruction is selected, it enables the ring oscillator pointed to by 9:8 of the data register. When this instruction is selected, the counter only counts while DEBUG_L pin is asserted. Ring oscillators will run at less than 1 GHz. This is for Broadcom Use Only.

**Table 306: Ring Oscillator Scan Chain**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:0</td>
<td>count</td>
<td>Selected counter value.</td>
</tr>
<tr>
<td>18:16</td>
<td>sel</td>
<td>Oscillator select.</td>
</tr>
<tr>
<td>19</td>
<td>enable</td>
<td>High to enable counter</td>
</tr>
</tbody>
</table>

**BOUNDARY SCAN REGISTER**

Each pin on the part with the exception of COLDRES_L, RESET_L, TMS, TDO, TDI, TRST_L and TCK, can be controlled by the boundary scan register. The boundary scan register is selected between TDI and TDO when INTEST or EXTEST instructions are set in the instruction register. When the TAP controller traverses the Capture-DR state, the inputs are sampled into the shadow register.

During the Shift-DR state of the TAP controller the contents of the boundary scan shadow register are shifted in between TDI and TDO. When the Update-DR state is traversed, the contents of the BSR gets the contents of the shadow register. This allows the user to scan in new contents while the old contents of the BSR are still being held at the outputs. During EXTEST, outputs are driven by the BSR instead of the normal internal signals. During INTEST the inputs to the part are driven by the BSR instead of the pins.
Figure 85 shows an example of the boundary scan register and the shadow register for most of the pins on the part. All the pins apart from HyperTransport use this input/output pin building block. It consists of three BC_1 scan cells, one for the output, one for the output enable and one for the input. The Mode signal causes the JTAG to drive both the INTEST and EXTEST data.
To use the boundary scan and EXTEST to test board connectivity the following steps should be performed:

1. The TAP controllers of both parts under test are set to the EXTEST instruction.
2. The desired pattern on the output pins is shifted into the BSR shadow register.
3. The Update-DR state is traversed.
4. The Capture-DR state of the target part is traversed.
5. The contents of the shadow register in the target part is scanned out and then compared to expected results.

To use boundary scan and INTEST to run test vectors against the part the following steps should be performed:

1. The instruction register for the part is set to the INTEST instruction.
2. The desired test pattern is scanned into the shadow register.
3. The Update-DR state is traversed.
4. The Capture-DR state is traversed.
5. The results from the vector are scanned out from the shadow register. At the same time the next test pattern can be scanned in.

The differential signals on the HyperTransport interface are scanned a little differently. All of these are unidirectional, but the signals are also differential and double data rate. The data rate is doubled and the differential is formed in the output pads, the differential is received and the data rate halved in the input pads.

Figure 86 shows the output cell. In normal operation two bits of data are presented, one for transmission when the clock is high the other when the clock is low. The inverted version of these bits drives the negative output of the differential pair. The scan function uses two BC_1 cells, one drives each of the pins, but rather than the pin data they monitor the single data rate version of the data. Thus when performing EXTEST the two pins are separate (although they should be driven as inverses to guarantee correct reception), but when scanning out vectors the data for both edges of the clock can be captured.

![Figure 86: JTAG HyperTransport Output Boundary Scan Block](image-url)
The HyperTransport input block, shown in Figure 87, is similar. It uses two BC_1 cells to monitor the two pins. However, these will always be inverses when a correct differential signal is received. The outputs of the BC_1 cells can be used to drive the single data rate version of the data into the part simplifying INTEST vector injection.

![Figure 87: JTAG HyperTransport Input Boundary Scan Block](image)

The order of the input/output pins are shown in the BSDL file for the part.

**BSRMODE - HOLDING BOUNDARY SCAN ACTIVE**

Using the normal JTAG definition it is not possible to keep the boundary scan register active (driving pins or into the part) and scan other internal registers, because the boundary scan must be made inactive when a non-BSR instruction is scanned in to the instruction register. The large amount of internal state in the part makes it useful to be able to do this so the BSRMODE instruction has been added. If the BSRMODE instruction is used after EXTEST, CLAMP or INTEST then the BSR mode is maintained (keeping the signals driven from the BSR) even if another JTAG instruction is used. This allows any number of other chains to be scanned while the BSR values are held. If EXTEST, CLAMP or INTEST is followed by any instruction other than BSRMODE the BSR mode is removed (per the standard JTAG behaviour). After BSRMODE has been used its state should be cleared by performing one of the BSR instructions followed by a non-BSR instruction.
**PROCESSOR AND PROBE ACCESS**

The JTAG probe can act as both a master and a slave on the ZBbus. As a master, the probe can initiate any bus transaction. As a slave, the probe responds to accesses in the EJTAG memory range 00_1000_0000 to 00_1001_FFFF. (This address range will be accessed by the CPU using the alternative debug vector at VA FFFF_FFFF_B000_0480.) At any time the probe can either be a master or a slave, however it is possible for an external debugger to switch between the two modes to support both processor accesses to the probe and probe accesses to memory.

Probe accesses are controlled by three JTAG registers: the Address Register, the Data Register and the Control Register. These are described in the subsequent sections. JTAG commands are provided for scanning these registers individually or as a concatenated chain.

There are two Control Register bits that set the behavior of the probe agent. The ProbEn bit enables accesses to the EJTAG range and the MaSl bit selects if the probe is a master or a slave. Table 307 shows the behavior for the possible cases.

<table>
<thead>
<tr>
<th>ProbEn</th>
<th>MaSl</th>
<th>CPU Access to 00_1000_0000 - 00_1001_FFFF</th>
<th>Probe Access to ZBbus</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Probe memory disabled. Read returns zeros, writes are discarded.</td>
<td>Master mode disabled.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Probe memory disabled. Read returns zeros, writes are discarded.</td>
<td>Master mode active. The probe can run ZBbus cycles.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Probe memory active. Reads and writes are serviced by the probe. The EJTAG control register should be scanned to detect CPU requests.</td>
<td>Master mode disabled.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Probe accesses delayed. Reads and writes are blocked and will be serviced when MaSl is set to zero.</td>
<td>Master mode active. The probe can run ZBbus cycles.</td>
</tr>
</tbody>
</table>
An external debugger connected to the JTAG probe can use both master and slave accesses to communicate with the part. An example flow is shown in Figure 88. Normally the debugger will scan out the Control Register and service CPU requests. When it needs to run an access (for example to read memory) it will scan in the Control Register with the MaSl bit set. A check needs to be done for CPU accesses that may have arrived while the mode was being changed, once they are serviced the interface is ready for the master access.

Figure 88: Example JTAG Probe Flowchart
PROCESSOR ACCESSES TO THE JTAG SPACE

When the JTAG probe is configured for slave accesses, an access to the JTAG range (00_1000_0000 to 00_1001_FFFF) is responded to by the debugger. If the CPU is set to use the Alternate Debug Vector then a debug exception will cause it to fetch instructions from 00_1000_0480.

When the CPU does a read from JTAG memory, the JTAG unit latches the address, bus command, cache attributes and the byte enables from the request into the scannable Address Register. The JTAG unit then sets the PrAcc bit to 1. The debugger software will be scanning out the EJTAG Control Register, and will see the PrAcc bit set. It will then scan out the Address Register, decode A_CMD as a read and fetch the data in the debugger memory, and then scan the data, properly aligned, into the Data Register. It will then scan in the control register with the PrAcc bit cleared. The JTAG unit will then terminate the bus transaction with a data cycle on the bus, containing the data, dcode and modified flag from the Data Register.

When the CPU does a write to JTAG memory, the JTAG unit latches the address, bus command, cache attributes and the byte enables from the request into the scannable Address Register and the data, dcode, responder id and modified flag into the Data Register. The JTAG unit then sets the PrAcc bit to 1. The debugger software will be scanning out the EJTAG Control Register, and will see the PrAcc bit set. It will then scan out the Address Register, decode A_CMD as a write, scan the Data Register, and write the data in the debugger memory. It will then scan in the Control Register with the PrAcc bit cleared causing the JTAG unit to free up the buffers and allow more transactions.

The CPU can map the JTAG space cacheable non-coherent or uncacheable. It must not be mapped cacheable coherent, or the behavior of the system becomes UNDEFINED.

PROBE ACCESSES TO THE ZBBUS

When the MaSl bit is set in the EJTAG control register, the probe can master the ZBBus and access any area of system memory. A write to system memory is done by scanning into the Address Register the address, byte enables, command and cache attribute bits. The data, dcode and modified flag are scanned in to the Data Register. The four responder id bits that are scanned into the Data Register are ignored, the SCD agent ID is always used for a request from the JTAG unit. When the PbAcc and PW bits are set in the EJTAG Control Register the write will be initiated, and the PbAcc bit will be cleared when it has completed.

A read from system memory is done by scanning into the Address Register the address, byte enables, command and cache attribute bits. The access is started by scanning a 1 into the PbAcc bit, and a 0 into the PW bit. The EJTAG unit then makes a read request on the bus and when the transaction has finished, the data, dcode, responder id and modified flag will be returned in the Data Register, and the PbAcc bit will be cleared. The probe should keep scanning out the Control Register until it sees that the PbAcc bit has been cleared. It then can scan out the Data Register.

Note that if ProbeEn is 1, a master access should not be made to an address serviced by the SCD, since these will deadlock.
**ADDRESS REGISTER**

The Address Register is 77 bits long and contains the address bits, byte enables, bus command and cache attributes. To maintain coherency the cache attributes must be set correctly for the address so snoop responses are correct (in some situations the debugger may want to use different attributes but it must do so with care, for example a cacheable non-coherent read to a memory address can be used to force data to come from the L2 cache or memory, never a processor L1 cache, and an uncacheable read from a memory address will force the data to come from memory). The address is for a 32 byte cache block, the byte enables complete selection within the block. Table 308 shows the Address Register, which is scanned starting with the LSB.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>34:0</td>
<td>A_AD[39:5] Address</td>
</tr>
<tr>
<td>66:35</td>
<td>A_BE[31:0] Byte Enables. Indicate which byte lanes are valid in the Data Register, see Table 5 on page 24 for the mapping from byte enable to byte address.</td>
</tr>
<tr>
<td>71:70</td>
<td>A_L1CA[1:0] L1 Cacheability attribute. See Table 4 on page 23.</td>
</tr>
<tr>
<td>72</td>
<td>A_L2CA L2 Cache Allocate request.</td>
</tr>
<tr>
<td>76:73</td>
<td>A_ID[9:6] Requester agent ID (see Table 1 on page 20). These bits are only valid when the probe is the slave, in master mode the SCD agent ID will be used for the transaction.</td>
</tr>
</tbody>
</table>

**DATA REGISTER**

The Data Register contains 256 data bits, the status/error code, the ID of the agent that put the data on the ZBbus, the modified flag, and (on reads) the response phase information for the transaction. The register is shown in Table 309, and is scanned from LSB to MSB. The mapping from the byte enables in the address register to bits in this register is fixed; however, the byte address associated with each byte lane depends on the system endian mode as shown in Table 5 on page 24.

<table>
<thead>
<tr>
<th>Bits</th>
<th>ZBbus signal (see Table 2 on page 20)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>255:0</td>
<td>D_DA[255:0] Data. Only the byte lanes that had byte enables set contain valid data.</td>
<td></td>
</tr>
<tr>
<td>258:256</td>
<td>D_CODE[2:0] Data status/error code (see Table 6 on page 24).</td>
<td></td>
</tr>
<tr>
<td>259</td>
<td>D_MOD Set to indicate the data is modified (dirty).</td>
<td></td>
</tr>
<tr>
<td>263:260</td>
<td>D_RSP[3:0] Id of Agent putting the data on the ZBbus. This is a read only field, bits scanned in will be ignored.</td>
<td></td>
</tr>
<tr>
<td>264</td>
<td>R_L2HIT Response indicating block is in the L2 cache (read only, only valid for probe master reads).</td>
<td></td>
</tr>
<tr>
<td>270:265</td>
<td>R_EXC[5:0] Response indicating block is exclusive in an agent (read only, only valid for probe master reads).</td>
<td></td>
</tr>
<tr>
<td>276:271</td>
<td>R_SHD[5:0] Response indicating block is shared by an agent (read only, only valid for probe master reads).</td>
<td></td>
</tr>
</tbody>
</table>
**EJTAG CONTROL REGISTER**

This is a 12 bit register to control the various operations of the debug support modules. This register is selected by shifting in the CONTROL instruction. Bits in the EJTAG Control Register can be set/cleared by shifting in data; status is read by shifting out this register. This EJTAG Control Register can only be accessed by the TAP interface.

**Table 310: EJTAG Control Register**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>Use</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DM0</td>
<td>Debug Mode 0. This bit indicates the value of the EDEN signal from CPU0. When this bit is set, it indicates that CPU0 is signalling a debug event (see SB-1 CPU User Manual). This bit is read only, and any writes to it are ignored.</td>
<td>R</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>DM1</td>
<td>Debug Mode 1. This bit indicates the value of the EDEN signal from CPU1. When this bit is set, it indicates that CPU1 is signalling a debug event (see SB-1 CPU User Manual). This bit is read only, and any writes to it are ignored. On the BCM1125/H this bit is not used and reads are UNPREDICTABLE.</td>
<td>R</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>EJTAG Break0</td>
<td>Setting this bit to 1 causes the DINT pin to CPU0 to be set. This allows the probe to signal a debug interrupt to CPU0. The bit must be cleared to remove the interrupt.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>EJTAG Break1</td>
<td>Setting this bit to 1 causes the DINT pin to CPU1 to be set. This allows the probe to signal a debug interrupt to CPU1. The bit must be cleared to remove the interrupt. On the BCM1125/H this bit is not used.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>PrTrap0</td>
<td>Probe Trap0. Setting this bit to 1 forces the DBBOOT signal to CPU0 to 1. This allows the probe to force CPU0 into debug mode upon the next deassertion of reset. This bit is set on the Update-IR state of the EJTAGBOOT instruction. It is reset on the Update-IR state of the NORMALBOOT instruction or by asserting the TRST_L pin or by a rising edge of TCK when the TAP controller is in the Test-Logic-Reset state. This bit can also be set or cleared by scanning a 1 or 0 into this bit.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>PrTrap1</td>
<td>Probe Trap1. This bit performs the same way as PrTrap0, except it does so for CPU1. On the BCM1125/H this bit is not used.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ProbEn</td>
<td>Probe Enable. Setting this bit to 1 will indicate that EJTAG memory is handled by the probe, so processor accesses are answered. 0: The probe does not handle EJTAG memory transactions. Accesses to the EJTAG memory space always returns 0. 1: The probe does handle EJTAG memory transactions. When this bit is set the MaSl bit is used to determine if the probe is acting as bus master or slave. See discussion in Section: “Processor and Probe Access” on page 435. Note that setting the PrTrap bits without this bit set causes an invalid state. The EJTAGBOOT instruction will set this bit to 1. The NORMALBOOT instruction will clear this bit. This bit is also cleared by asserting TRST_L or by a rising edge of TCK while in the Test-Logic-Reset State.</td>
<td>R/W</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 310: EJTAG Control Register (Cont.)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
<th>Use</th>
<th>Reset Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>PrAcc</td>
<td>Processor Access. Read value of this bit indicates if a Processor Access (PA) to the EJTAG memory is pending: 0: No pending processor access. 1: Pending processor access. The probe control software must clear this bit to 0 to indicate the end of the PA. Write of 1 is ignored.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>PW</td>
<td>Processor Access Write. This bit is not used.</td>
<td>R/W</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>PbAcc</td>
<td>Probe Initiated Transaction. When set this bit indicates a probe initiated transaction is in progress. The probe sets this bit to indicate to the JTAG unit that the probe is initiating a transaction. The JTAG unit clears this bit when the transaction has been completed. This bit can only be set if the ProbEn bit is cleared. Probe initiated transactions can occur only if the MaSI bit is set. 0: No pending probe access. 1: Pending probe access. If the probe writes a zero to this bit it is ignored. The only way the bit can be set is by the probe writing a 1, the only way the bit can be cleared is by the JTAG unit clearing it.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>MaSI</td>
<td>Probe Master/Slave. When set this bit indicates the probe can initiate a transaction, if ProbEn is set then CPU accesses will be blocked until this bit is clear. When clear CPU transactions will be accepted if ProbEn is set. 0: CPU accesses accepted. 1: Probe can initiate a bus transaction.</td>
<td>R/W</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>ClkStop</td>
<td>Clock stop flag. This bit is for Broadcom Use Only.</td>
<td>R/O</td>
<td>0</td>
</tr>
</tbody>
</table>
DIFFERENCES FROM EJTAG 2.5 (Feb. 22, 2000) SPECIFICATION

The BCM1250 has some differences from the EJTAG 2.5 due to support for the dual processor configuration and system level debug access. A summary, with reference to the section numbers in the EJTAG 2.5 (Feb. 22, 2000) specification:

2.2.2
The BCM1250 has no dseg (A kseg1 address is used for probe serviced accesses)

2.2.2.1 N/A

2.2.2.2 N/A

2.2.3.2
The random register continues to run in debug mode

2.3.2
Alternate Vector location is FFFF_FFFF_B000_0480
- Set by SW in EDEBUG register
- Set by DBBOOT signal from JTAG TAP (in EJTAG control register)

2.3.5/2.3.6
Hardware breakpoints/watchpoints can be done in extended debug mode using the regular Watch registers not special ones.

2.3.7
Watch register exceptions are precise for both data and instructions

2.6.3
Processor reset is done through System Config register not EJTAG Control

2.6.4
Rocc is not supported.

3
There is no DCR (dseg is not supported)

4
There are no special debug hardware breakpoints. In extended debug mode the CP0 Watch registers can be used to provide one hardware data break and one hardware instruction breakpoint.

5
A single TAP is shared by the two processors and the system logic.

5.5.3
The data register is 277 bits (contains ZBbus data + info)
5.5.4

The address register is 77 bits (contains ZBbus address + cmd)

5.5.5

Rocc - Not implemented
Ps  - Not implemented, size in address register
Doze - Not implemented
Halt - Not implemented
PerRst - Use System Config register for resets
PRnW - Not used
PrAcc - Supported
PrRst - Use System Config register for resets
ProbEn - Supported (works with additional MaSI bit to set direction)
ProbTrap - Two Copies, one per CPU. Also causes EJTAG boot on reset
EjtagBrk - Two Copies, one per CPU, level sensitive probe must clear
DM - Two copies. But shows CPU EDEN signal rather than DM

5.6.3

Rocc is not supported

5.6.4

Processor access is through physical address 00_1000_0000 - 00_1001_FFFF and uses ZBbus request

There is an additional mode for allowing the probe to initiate bus requests to memory or memory mapped I/O.

7.1.2

External interrupt possible through DEBUG_L pin but the driver must be open collector or open drain.

7.1.2

The probe RST signal can be connected to RESET_L.
Section 16: Reference

Internal Register Addresses by Function

This section lists the registers and address assignments, per ZBus agent. In the electronic version of the document the Table/Page column provides a hyperlink to the table that defines the register.

Note: The following table details the specific addresses and names of each register. Each register is 1 byte, 2 bytes, 4 bytes or 8 bytes. Registers can be read or written as double-word (8 bytes), word (4 bytes), 2 byte or signal byte access. Reads that are wider than the defined width or the register will return UNPREDICTABLE data in the bits that are not defined. The address used to access the register will need to be adjusted if the system is running in Big Endian mode. See Section: “Internal Registers” on page 11 for details. Configuration registers should be mapped in uncacheable space, so transactions will never be wider than 8 bytes (and will never span more than an single 8 byte aligned range).

Reading registers marked ‘Write Only’ will give UNPREDICTABLE results. Writes to ‘Read Only’ registers will be ignored.

Table 311: Internal Register Addresses by Function

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_config_0</td>
<td>00_1005_1100</td>
<td>72/135</td>
<td>Channel 0 attributes.</td>
</tr>
<tr>
<td>mc_dramcmd_0</td>
<td>00_1005_1120</td>
<td>75/139</td>
<td>Channel 0 SDRAM command.</td>
</tr>
<tr>
<td>mc_drammode_0</td>
<td>00_1005_1140</td>
<td>76/139</td>
<td>Channel 0 SDRAM mode.</td>
</tr>
<tr>
<td>mc_timing1_0</td>
<td>00_1005_1160</td>
<td>77/140</td>
<td>Channel 0 SDRAM timing 1.</td>
</tr>
<tr>
<td>mc_timing2_0</td>
<td>00_1005_1180</td>
<td>78/141</td>
<td>Channel 0 SDRAM timing 2.</td>
</tr>
<tr>
<td>mc_cs_start_0</td>
<td>00_1005_11a0</td>
<td>79/141</td>
<td>Channel 0 CS[3:0] start address.</td>
</tr>
<tr>
<td>mc_cs_end_0</td>
<td>00_1005_11c0</td>
<td>80/141</td>
<td>Channel 0 CS[3:0] end+1 address.</td>
</tr>
<tr>
<td>mc_interleave_0</td>
<td>00_1005_11e0</td>
<td>81/142</td>
<td>Channel 0 interleaved CS position.</td>
</tr>
<tr>
<td>mc_cs0_row_0</td>
<td>00_1005_1200</td>
<td>82/142</td>
<td>Channel 0 CS0 row address bits.</td>
</tr>
<tr>
<td>mc_cs0_col_0</td>
<td>00_1005_1220</td>
<td>83/142</td>
<td>Channel 0 CS0 column address bits.</td>
</tr>
<tr>
<td>mc_cs0_ba_0</td>
<td>00_1005_1240</td>
<td>84/143</td>
<td>Channel 0 CS0 bank select.</td>
</tr>
<tr>
<td>mc_cs1_row_0</td>
<td>00_1005_1260</td>
<td>82/142</td>
<td>Channel 0 CS1 row address bits.</td>
</tr>
<tr>
<td>mc_cs1_col_0</td>
<td>00_1005_1280</td>
<td>83/142</td>
<td>Channel 0 CS1 column address bits.</td>
</tr>
<tr>
<td>mc_cs1_ba_0</td>
<td>00_1005_12a0</td>
<td>84/143</td>
<td>Channel 0 CS1 DRAM bank select.</td>
</tr>
<tr>
<td>mc_cs2_row_0</td>
<td>00_1005_12c0</td>
<td>82/142</td>
<td>Channel 0 CS2 row address bits.</td>
</tr>
<tr>
<td>mc_cs2_col_0</td>
<td>00_1005_12e0</td>
<td>83/142</td>
<td>Channel 0 CS2 column address bits.</td>
</tr>
<tr>
<td>mc_cs2_ba_0</td>
<td>00_1005_1300</td>
<td>84/143</td>
<td>Channel 0 CS2 DRAM bank select.</td>
</tr>
<tr>
<td>mc_cs3_row_0</td>
<td>00_1005_1320</td>
<td>82/142</td>
<td>Channel 0 CS3 row address bits.</td>
</tr>
<tr>
<td>mc_cs3_col_0</td>
<td>00_1005_1340</td>
<td>83/142</td>
<td>Channel 0 CS3 column address bits.</td>
</tr>
<tr>
<td>mc_cs3_ba_0</td>
<td>00_1005_1360</td>
<td>83/142</td>
<td>Channel 0 CS3 DRAM bank select.</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_cs_attr_0</td>
<td>00_1005_1380</td>
<td>85/143</td>
<td>Channel 0 CS[3:0] attribute.</td>
</tr>
<tr>
<td>mc_test_data_0</td>
<td>00_1005_1400</td>
<td>86/144</td>
<td>Channel 0 ECC error set data bits.</td>
</tr>
<tr>
<td>mc_test_ecc_0</td>
<td>00_1005_1420</td>
<td>87/144</td>
<td>Channel 0 ECC error set ecc bits.</td>
</tr>
<tr>
<td>mc_clock_cfg_0</td>
<td>00_1005_1500</td>
<td>74/138</td>
<td>Channel 0 memory clock ratio.</td>
</tr>
<tr>
<td>mc_1</td>
<td></td>
<td></td>
<td>Memory controller channel 1 configuration (at memory controller 0 + 1000).</td>
</tr>
</tbody>
</table>

### L2 Controller

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>l2_read_address</td>
<td>00_1004_0018</td>
<td>56/100</td>
<td>Read only. Last address/tag in a read (for testing)</td>
</tr>
<tr>
<td>l2_ecc_address</td>
<td>00_1004_0038</td>
<td>56/100</td>
<td>Read only. Last address with ecc error (correctable or not).</td>
</tr>
<tr>
<td>l2_misc_value</td>
<td>00_1004_0058</td>
<td>57/100</td>
<td>Read only. PERIPH_REV3 and later. Value of L2 hidden registers.</td>
</tr>
<tr>
<td>l2_way_disable</td>
<td>00_1004_1x00</td>
<td>/91</td>
<td>Accesses made to this range of addresses will write the value x to l2_wayen[3:0] register. If l2_wayen[i] is clear Way i is removed from the L2 replacement algorithm. See Section: “Using the L2 Cache as Memory” on page 91</td>
</tr>
<tr>
<td>l2_cache_disable</td>
<td>00_1004_2x00</td>
<td>/94</td>
<td>Accesses made to this range of addresses will write the value x to l2_cache_disable[3:0] register. See Section: “Reduced Cache Size” on page 94.</td>
</tr>
<tr>
<td>l2_misc_config</td>
<td>00_1004_3x00</td>
<td>/99</td>
<td>Accesses made to this range of addresses will write the value x to l2_misc_config[3:0] register. See Section: “Cache Configuration Register” on page 99.</td>
</tr>
</tbody>
</table>

### PCI Host Bridge

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 00 PCI header</td>
<td>00_FB00_0000</td>
<td>127/236</td>
<td>Configuration space bus=0, dev=0.</td>
</tr>
<tr>
<td></td>
<td>00_FB00_00FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### HyperTransport Host Bridge

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 01 PCI header</td>
<td>00_FB00_0800</td>
<td>140/244</td>
<td>Configuration space bus=0, dev=1.</td>
</tr>
<tr>
<td></td>
<td>00_FB00_08FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Ethernet DMA and MACs

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_tx_byte_0</td>
<td>00_1006_4000</td>
<td>167/288</td>
<td>RMON transmit byte counter.</td>
</tr>
<tr>
<td>mac_collisions_0</td>
<td>00_1006_4008</td>
<td>167/288</td>
<td>RMON total collisions.</td>
</tr>
<tr>
<td>mac_late_col_0</td>
<td>00_1006_4010</td>
<td>167/288</td>
<td>RMON late collisions.</td>
</tr>
<tr>
<td>mac_ex_col_0</td>
<td>00_1006_4018</td>
<td>167/288</td>
<td>RMON excessive collisions.</td>
</tr>
<tr>
<td>mac_fcs_error_0</td>
<td>00_1006_4020</td>
<td>167/288</td>
<td>RMON packets Tx with bad FCS.</td>
</tr>
<tr>
<td>mac_tx_abort_0</td>
<td>00_1006_4028</td>
<td>167/288</td>
<td>RMON transmit packets aborted.</td>
</tr>
<tr>
<td>mac_tx_bad_0</td>
<td>00_1006_4038</td>
<td>167/288</td>
<td>RMON total of bad Tx packets.</td>
</tr>
<tr>
<td>mac_tx_good_0</td>
<td>00_1006_4040</td>
<td>167/288</td>
<td>RMON total successfully transmitted packets.</td>
</tr>
<tr>
<td>mac_tx_runt_0</td>
<td>00_1006_4048</td>
<td>167/288</td>
<td>RMON total runt packets transmitted.</td>
</tr>
<tr>
<td>mac_tx_oversize_0</td>
<td>00_1006_4050</td>
<td>167/288</td>
<td>RMON total oversize packets transmitted.</td>
</tr>
</tbody>
</table>
Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_rx_bytes_0</td>
<td>00_1006_4080</td>
<td>167/288</td>
<td>RMON total received bytes.</td>
</tr>
<tr>
<td>mac_rx_mcast_0</td>
<td>00_1006_4088</td>
<td>167/288</td>
<td>RMON total received multicast packets.</td>
</tr>
<tr>
<td>mac_rx_bcast_0</td>
<td>00_1006_4090</td>
<td>167/288</td>
<td>RMON total received broadcast packets.</td>
</tr>
<tr>
<td>mac_rx_bad_0</td>
<td>00_1006_4098</td>
<td>167/288</td>
<td>RMON total received bad packets.</td>
</tr>
<tr>
<td>mac_rx_good_0</td>
<td>00_1006_40A0</td>
<td>167/288</td>
<td>RMON total received good packets.</td>
</tr>
<tr>
<td>mac_rx_runt_0</td>
<td>00_1006_40A8</td>
<td>167/288</td>
<td>RMON total runt packets received.</td>
</tr>
<tr>
<td>mac_rx_oversize_0</td>
<td>00_1006_40B0</td>
<td>167/288</td>
<td>RMON total oversize packets received.</td>
</tr>
<tr>
<td>mac_rx_fcs_error_0</td>
<td>00_1006_40B8</td>
<td>167/288</td>
<td>RMON total packets received with bad FCS.</td>
</tr>
<tr>
<td>mac_rx_length_error_0</td>
<td>00_1006_40C0</td>
<td>167/288</td>
<td>RMON total packets received with length error.</td>
</tr>
<tr>
<td>mac_rx_code_error_0</td>
<td>00_1006_40C8</td>
<td>167/288</td>
<td>RMON total packets received with code error.</td>
</tr>
<tr>
<td>mac_rx_align_error_0</td>
<td>00_1006_40D0</td>
<td>167/288</td>
<td>RMON total packets received with align error.</td>
</tr>
<tr>
<td>mac_cfg_0</td>
<td>00_1006_4100</td>
<td>176/301</td>
<td>Ethernet interface configuration register.</td>
</tr>
<tr>
<td>mac_thrsh_cfg_0</td>
<td>00_1006_4108</td>
<td>179/306</td>
<td>Ethernet interface FIFO threshold configuration register.</td>
</tr>
<tr>
<td>mac_vlantag_0</td>
<td>00_1006_4110</td>
<td>181/309</td>
<td>VLAN tag for insertion into packets on transmit</td>
</tr>
<tr>
<td>mac_frame_cfg_0</td>
<td>00_1006_4118</td>
<td>180/307</td>
<td>Ethernet interface MAC frame configuration.</td>
</tr>
<tr>
<td>mac_rx_fifo_ptrs_0</td>
<td>00_1006_4120</td>
<td>186/313</td>
<td>MAC FIFO pointers (Debug, Read Only).</td>
</tr>
<tr>
<td>mac_tx_fifo_ptrs_0</td>
<td>00_1006_4128</td>
<td>186/313</td>
<td>MAC FIFO pointers. (Debug, Read Only.)</td>
</tr>
<tr>
<td>mac_adfilter_cfg_0</td>
<td>00_1006_4200</td>
<td>192/315</td>
<td>MAC receive address filter configuration register.</td>
</tr>
<tr>
<td>mac_ethernet_addr_0</td>
<td>00_1006_4208</td>
<td>190/314</td>
<td>MAC source ethernet address for insertion during transmission.</td>
</tr>
<tr>
<td>mac_type_cfg_0</td>
<td>00_1006_4210</td>
<td>191/315</td>
<td>MAC packet type table for receive packet comparisons.</td>
</tr>
<tr>
<td>mac_hash0_0</td>
<td>00_1006_4240</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash1_0</td>
<td>00_1006_4248</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash2_0</td>
<td>00_1006_4250</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash3_0</td>
<td>00_1006_4258</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash4_0</td>
<td>00_1006_4260</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash5_0</td>
<td>00_1006_4268</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash6_0</td>
<td>00_1006_4270</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash7_0</td>
<td>00_1006_4278</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_addr0_0</td>
<td>00_1006_4280</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_admask0_0</td>
<td>00_1006_4288</td>
<td>188/314</td>
<td>Address filter exact match mask register (PERIPH_REV3).</td>
</tr>
<tr>
<td>mac_addr1_0</td>
<td>00_1006_4290</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_admask1_0</td>
<td>00_1006_4298</td>
<td>187/313</td>
<td>Address filter exact match mask register (PERIPH_REV3).</td>
</tr>
<tr>
<td>mac_addr2_0</td>
<td>00_1006_42A0</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr3_0</td>
<td>00_1006_42A8</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr4_0</td>
<td>00_1006_42B0</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
</tbody>
</table>

Broadcom Corporation
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_addr5_0</td>
<td>00_1006_42A8</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr6_0</td>
<td>00_1006_42B0</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr7_0</td>
<td>00_1006_42B8</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_chup0_0</td>
<td>00_1006_4300</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup1_0</td>
<td>00_1006_4308</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup2_0</td>
<td>00_1006_4310</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup3_0</td>
<td>00_1006_4318</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chlo0_0</td>
<td>00_1006_4320</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo1_0</td>
<td>00_1006_4328</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo2_0</td>
<td>00_1006_4330</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo3_0</td>
<td>00_1006_4338</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_enable_0</td>
<td>00_1006_4400</td>
<td>177/305</td>
<td>MAC enable register.</td>
</tr>
<tr>
<td>mac_status_0</td>
<td>00_1006_4408</td>
<td>182/309</td>
<td>MAC status/error register (Read Only, read clears).</td>
</tr>
<tr>
<td>mac_status1_0</td>
<td>00_1006_4430</td>
<td>183/312</td>
<td>MAC status/error register (Read Only, read clears ch 1).</td>
</tr>
<tr>
<td>mac_int_mask_0</td>
<td>00_1006_4410</td>
<td>185/313</td>
<td>MAC interrupt mask register.</td>
</tr>
<tr>
<td>dmaasic_addr_mac_0</td>
<td>00_1006_4418</td>
<td>94/166</td>
<td>ASIC mode base address.</td>
</tr>
<tr>
<td>mac_txd_ctl_0</td>
<td>00_1006_4420</td>
<td>178/305</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>mac_mdio_0</td>
<td>00_1006_4428</td>
<td>194/317</td>
<td>MDIO pin control register.</td>
</tr>
<tr>
<td>mac_debug_status_0</td>
<td>00_1006_4448</td>
<td>184/312</td>
<td>MAC status/error register (Debug, Read Only, no side effects).</td>
</tr>
<tr>
<td>dma_config0_mac_0_rx_ch_0</td>
<td>00_1006_4800</td>
<td>91/163</td>
<td>DMA config 0 register.</td>
</tr>
<tr>
<td>dma_config1_mac_0_rx_ch_0</td>
<td>00_1006_4808</td>
<td>92/164</td>
<td>DMA config 1 register.</td>
</tr>
<tr>
<td>dma_dscr_base_mac_0_rx_ch_0</td>
<td>00_1006_4810</td>
<td>93/166</td>
<td>DMA descriptor base register.</td>
</tr>
<tr>
<td>dma_dscr_cnt_0_rx_ch_0</td>
<td>00_1006_4818</td>
<td>95/166</td>
<td>DMA descriptor count.</td>
</tr>
<tr>
<td>dma_dscr_a_mac_0_rx_ch_0</td>
<td>00_1006_4820</td>
<td>96/167</td>
<td>DMA current descriptor A.</td>
</tr>
<tr>
<td>dma_dscr_b_mac_0_rx_ch_0</td>
<td>00_1006_4828</td>
<td>97/167</td>
<td>DMA current descriptor B.</td>
</tr>
<tr>
<td>dma_cur_dscr_addr_mac_0_rx_ch_0</td>
<td>00_1006_4830</td>
<td>98/167</td>
<td>DMA current descriptor address.</td>
</tr>
<tr>
<td>dma_oodpktlost_mac_0_rx_ch_0</td>
<td>00_1006_4838</td>
<td>99/168</td>
<td>DMA packet lost counter. (PERIPH_REV3)</td>
</tr>
<tr>
<td>dma_mac_0_rx_ch_1</td>
<td>00_1006_4900</td>
<td>1000</td>
<td>Rx channel 1 at Rx channel 0 + 100.</td>
</tr>
<tr>
<td>dma_mac_0_tx_ch_0</td>
<td>00_1006_4a00</td>
<td>1400</td>
<td>Tx channel 0 at Rx channel 0 + 400.</td>
</tr>
<tr>
<td>dma_mac_0_tx_ch_1</td>
<td>00_1006_4d00</td>
<td>1500</td>
<td>Tx channel 1 at Rx channel 0 + 500.</td>
</tr>
<tr>
<td>mac_1</td>
<td>00_1006_5000</td>
<td>1900</td>
<td>MAC 1 registers (+1000 from mac_0 registers).</td>
</tr>
<tr>
<td>mac_2</td>
<td>00_1006_6000</td>
<td>2000</td>
<td>MAC 2 registers (+2000 from mac_0 registers).</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/ Page</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Serial Ports</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>duart_mode_reg_1a</td>
<td>00_1006_0100</td>
<td>197/326</td>
<td>Mode register 1 port A MR1A.</td>
</tr>
<tr>
<td>duart_mode_reg_2a</td>
<td>00_1006_0110</td>
<td>198/326</td>
<td>Mode register 2 port A MR2A.</td>
</tr>
<tr>
<td>duart_status_a</td>
<td>00_1006_0120</td>
<td>200/327</td>
<td>Status register port A (Read Only).</td>
</tr>
<tr>
<td>duart_clk_sel_a</td>
<td>00_1006_0130</td>
<td>201/328</td>
<td>Clock select register port A.</td>
</tr>
<tr>
<td>duart_full_ctl_a</td>
<td>00_1006_0140</td>
<td>202/328</td>
<td>Full control port A.</td>
</tr>
<tr>
<td>duart_cmd_a</td>
<td>00_1006_0150</td>
<td>199/327</td>
<td>Command register port A.</td>
</tr>
<tr>
<td>duart_rx_hold_reg_a</td>
<td>00_1006_0160</td>
<td>203/328</td>
<td>RX holding register port A (Read Only, read pops character from FIFO).</td>
</tr>
<tr>
<td>duart_tx_hold_reg_a</td>
<td>00_1006_0170</td>
<td>204/329</td>
<td>TX holding register port A (Write Only).</td>
</tr>
<tr>
<td>duart_opcr_a</td>
<td>00_1006_0180</td>
<td>211/331</td>
<td>Output port control register alias that only alters port A bits.</td>
</tr>
<tr>
<td>duart_aux_ctrl_a</td>
<td>00_1006_0190</td>
<td>213/331</td>
<td>Aux control register alias that only alters port A bits.</td>
</tr>
<tr>
<td>duart_mode_reg_1b</td>
<td>00_1006_0200</td>
<td>197/326</td>
<td>Mode register 1 port B.</td>
</tr>
<tr>
<td>duart_mode_reg_2b</td>
<td>00_1006_0210</td>
<td>198/326</td>
<td>Mode register 2 port B.</td>
</tr>
<tr>
<td>duart_status_b</td>
<td>00_1006_0220</td>
<td>200/327</td>
<td>Status register port B (Read Only).</td>
</tr>
<tr>
<td>duart_clk_sel_b</td>
<td>00_1006_0230</td>
<td>201/328</td>
<td>Clock select register port B.</td>
</tr>
<tr>
<td>duart_full_ctl_b</td>
<td>00_1006_0240</td>
<td>202/328</td>
<td>Full control port B.</td>
</tr>
<tr>
<td>duart_cmd_b</td>
<td>00_1006_0250</td>
<td>199/327</td>
<td>Command register port B.</td>
</tr>
<tr>
<td>duart_rx_hold_reg_b</td>
<td>00_1006_0260</td>
<td>203/328</td>
<td>RX holding register port B (Read Only, read pops character from FIFO).</td>
</tr>
<tr>
<td>duart_tx_hold_reg_b</td>
<td>00_1006_0270</td>
<td>204/329</td>
<td>TX holding register port B (Write Only).</td>
</tr>
<tr>
<td>duart_opcr_b</td>
<td>00_1006_0280</td>
<td>211/331</td>
<td>Output port control register alias that only alters port B bits.</td>
</tr>
<tr>
<td>duart_aux_ctrl_b</td>
<td>00_1006_0290</td>
<td>213/331</td>
<td>Aux control register alias that only alters port B bits.</td>
</tr>
<tr>
<td>duart_inport_chng</td>
<td>00_1006_0300</td>
<td>206/329</td>
<td>Input port change register (Read Only, read clears).</td>
</tr>
<tr>
<td>duart_aux_cntrl</td>
<td>00_1006_0310</td>
<td>212/331</td>
<td>Aux control register.</td>
</tr>
<tr>
<td>duart_isr_a</td>
<td>00_1006_0320</td>
<td>215/332</td>
<td>Interrupt status register port A (Read Only).</td>
</tr>
<tr>
<td>duart_imr_a</td>
<td>00_1006_0330</td>
<td>218/333</td>
<td>Interrupt mask register port A.</td>
</tr>
<tr>
<td>duart_isr_b</td>
<td>00_1006_0340</td>
<td>216/332</td>
<td>Interrupt status register port B (Read Only).</td>
</tr>
<tr>
<td>duart_imr_b</td>
<td>00_1006_0350</td>
<td>219/333</td>
<td>Interrupt mask register port B.</td>
</tr>
<tr>
<td>duart_out_port</td>
<td>00_1006_0360</td>
<td>222/334</td>
<td>Output port register (Write Only).</td>
</tr>
<tr>
<td>duart_opcr</td>
<td>00_1006_0370</td>
<td>210/330</td>
<td>Output port control.</td>
</tr>
<tr>
<td>duart_in_port</td>
<td>00_1006_0380</td>
<td>205/329</td>
<td>Input port state (Read Only).</td>
</tr>
<tr>
<td>duart_isr</td>
<td>00_1006_0390</td>
<td>214/332</td>
<td>Full interrupt status (Read Only).</td>
</tr>
<tr>
<td>duart_imr</td>
<td>00_1006_03a0</td>
<td>217/333</td>
<td>Full interrupt mask.</td>
</tr>
<tr>
<td>duart_set_opr</td>
<td>00_1006_03b0</td>
<td>220/334</td>
<td>Output port register bit set (Write Only).</td>
</tr>
<tr>
<td>duart_clear_opr</td>
<td>00_1006_03c0</td>
<td>221/334</td>
<td>Output port register bit clear (Write Only).</td>
</tr>
<tr>
<td>duart_inport_chng_a</td>
<td>00_1006_03d0</td>
<td>208/330</td>
<td>Input port change register for channel A (Read Only, read clears channel A change state)</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duart_inport_chng_b</td>
<td>00_1006_03E0</td>
<td>209/330</td>
<td>Input port change register for channel B (Read Only, read clears channel B change state)</td>
</tr>
<tr>
<td>duart_inport_chng_debug</td>
<td>00_1006_03F0</td>
<td>207/330</td>
<td>Alias of input port change register with no read side effects. (Read Only)</td>
</tr>
<tr>
<td>dma_config0_ser_0_rx</td>
<td>00_1006_0400</td>
<td>91/163</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_0_rx</td>
<td>00_1006_0408</td>
<td>92/164</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_0_rx</td>
<td>00_1006_0410</td>
<td>93/166</td>
<td>Receive DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_cnt_ser_0_rx</td>
<td>00_1006_0418</td>
<td>95/166</td>
<td>Receive DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_0_rx</td>
<td>00_1006_0420</td>
<td>96/167</td>
<td>Receive DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_0_rx</td>
<td>00_1006_0428</td>
<td>97/167</td>
<td>Receive DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_0_rx</td>
<td>00_1006_0430</td>
<td>98/167</td>
<td>Receive DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>dma_config0_ser_0_tx</td>
<td>00_1006_0480</td>
<td>91/163</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_0_tx</td>
<td>00_1006_0488</td>
<td>92/164</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_0_tx</td>
<td>00_1006_0490</td>
<td>93/166</td>
<td>Transmit DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_cnt_ser_0_tx</td>
<td>00_1006_0498</td>
<td>95/166</td>
<td>Transmit DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_0_tx</td>
<td>00_1006_04A0</td>
<td>96/167</td>
<td>Transmit DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_0_tx</td>
<td>00_1006_04AB</td>
<td>97/167</td>
<td>Transmit DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_0_tx</td>
<td>00_1006_04B0</td>
<td>98/167</td>
<td>Transmit DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>ser_mode_0</td>
<td>00_1006_0500</td>
<td>231/353</td>
<td>Mode select.</td>
</tr>
<tr>
<td>ser_minfrm_sz_0</td>
<td>00_1006_0508</td>
<td>237/355</td>
<td>Min frame size.</td>
</tr>
<tr>
<td>ser_maxfrm_sz_0</td>
<td>00_1006_0510</td>
<td>238/356</td>
<td>Max frame size.</td>
</tr>
<tr>
<td>ser_addr_mask_0</td>
<td>00_1006_0518</td>
<td>243/358</td>
<td>Address mask.</td>
</tr>
<tr>
<td>ser_usr0_addr_0</td>
<td>00_1006_0520</td>
<td>244/358</td>
<td>Match address 0.</td>
</tr>
<tr>
<td>ser_usr1_addr_0</td>
<td>00_1006_0528</td>
<td>244/358</td>
<td>Match address 1.</td>
</tr>
<tr>
<td>ser_usr2_addr_0</td>
<td>00_1006_0530</td>
<td>244/358</td>
<td>Match address 2.</td>
</tr>
<tr>
<td>ser_usr3_addr_0</td>
<td>00_1006_0538</td>
<td>244/358</td>
<td>Match address 3.</td>
</tr>
<tr>
<td>ser_cmd_0</td>
<td>00_1006_0540</td>
<td>233/354</td>
<td>Command</td>
</tr>
<tr>
<td>ser_tx_rd_thresh_0</td>
<td>00_1006_0560</td>
<td>235/355</td>
<td>Transmit FIFO read threshold.</td>
</tr>
<tr>
<td>ser_tx_wr_thresh_0</td>
<td>00_1006_0568</td>
<td>234/355</td>
<td>Transmit FIFO write threshold.</td>
</tr>
<tr>
<td>ser_rx_rd_thresh_0</td>
<td>00_1006_0570</td>
<td>236/355</td>
<td>Receive FIFO read threshold.</td>
</tr>
<tr>
<td>ser_line_mode_0</td>
<td>00_1006_0578</td>
<td>232/353</td>
<td>Line Interface configuration register.</td>
</tr>
<tr>
<td>ser_dma_enable_0</td>
<td>00_1006_0580</td>
<td>239/356</td>
<td>DMA channel enable register.</td>
</tr>
<tr>
<td>ser_status_0</td>
<td>00_1006_0588</td>
<td>240/356</td>
<td>Serial interface and DMA status (Read Only, read clears).</td>
</tr>
<tr>
<td>ser_int_mask_0</td>
<td>00_1006_0590</td>
<td>242/357</td>
<td>Interrupt mask.</td>
</tr>
<tr>
<td>dma_asic_addr_ser_0</td>
<td>00_1006_0598</td>
<td>94/166</td>
<td>ASIC mode address.</td>
</tr>
<tr>
<td>ser_debug_status_0</td>
<td>00_1006_05A0</td>
<td>241/357</td>
<td>Serial interface and DMA status (Read Only, no side effects).</td>
</tr>
<tr>
<td>sertx_byte_lo_0</td>
<td>00_1006_05C0</td>
<td>246/359</td>
<td>Serial interface transmit byte count (low 16 bits).</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ser_tx_byte_hi_0</td>
<td>00_1006_05C8 246/359</td>
<td>Serial interface transmit byte count (high 16 bits).</td>
<td></td>
</tr>
<tr>
<td>ser_rx_byte_lo_0</td>
<td>00_1006_05D0 246/359</td>
<td>Serial interface receive byte count (low 16 bits).</td>
<td></td>
</tr>
<tr>
<td>ser_rx_byte_hi_0</td>
<td>00_1006_05D8 246/359</td>
<td>Serial interface receive byte count (high 16 bits).</td>
<td></td>
</tr>
<tr>
<td>ser_tx_underrun_0</td>
<td>00_1006_05E0 246/359</td>
<td>Serial interface transmit underrun count.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_overflow_0</td>
<td>00_1006_05E8 246/359</td>
<td>Serial interface receive overflow count.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_errors_0</td>
<td>00_1006_05F0 246/359</td>
<td>Serial interface receive error packet count.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_badaddr_0</td>
<td>00_1006_05F8 246/359</td>
<td>Serial interface receive address mismatch count.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table0_0</td>
<td>00_1006_0600 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table1_0</td>
<td>00_1006_0608 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table2_0</td>
<td>00_1006_0610 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table3_0</td>
<td>00_1006_0618 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table4_0</td>
<td>00_1006_0620 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table5_0</td>
<td>00_1006_0628 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table6_0</td>
<td>00_1006_0630 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table7_0</td>
<td>00_1006_0638 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table8_0</td>
<td>00_1006_0640 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table9_0</td>
<td>00_1006_0648 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table10_0</td>
<td>00_1006_0650 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table11_0</td>
<td>00_1006_0658 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table12_0</td>
<td>00_1006_0660 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table13_0</td>
<td>00_1006_0668 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table14_0</td>
<td>00_1006_0670 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_rx_table15_0</td>
<td>00_1006_0678 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table0_0</td>
<td>00_1006_0700 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table1_0</td>
<td>00_1006_0708 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table2_0</td>
<td>00_1006_0710 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table3_0</td>
<td>00_1006_0718 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table4_0</td>
<td>00_1006_0720 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table5_0</td>
<td>00_1006_0728 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table6_0</td>
<td>00_1006_0730 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table7_0</td>
<td>00_1006_0738 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table8_0</td>
<td>00_1006_0740 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table9_0</td>
<td>00_1006_0748 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table10_0</td>
<td>00_1006_0750 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table11_0</td>
<td>00_1006_0758 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table12_0</td>
<td>00_1006_0760 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>ser_tx_table13_0</td>
<td>00_1006_0768 245/358</td>
<td>Sequence table.</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ser_tx_table14_0</td>
<td>00_1006_0770</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table15_0</td>
<td>00_1006_0778</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>dma_config0_ser_1_rx</td>
<td>00_1006_0800</td>
<td>91/163</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_1_rx</td>
<td>00_1006_0808</td>
<td>92/164</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_1_rx</td>
<td>00_1006_0810</td>
<td>93/166</td>
<td>Receive DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_count_ser_1_rx</td>
<td>00_1006_0818</td>
<td>95/166</td>
<td>Receive DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_1_rx</td>
<td>00_1006_0820</td>
<td>96/167</td>
<td>Receive DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_1_rx</td>
<td>00_1006_0828</td>
<td>97/167</td>
<td>Receive DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_1_rx</td>
<td>00_1006_0830</td>
<td>98/167</td>
<td>Receive DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>dma_config0_ser_1_tx</td>
<td>00_1006_0880</td>
<td>91/163</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_1_tx</td>
<td>00_1006_0888</td>
<td>92/164</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_1_tx</td>
<td>00_1006_0890</td>
<td>93/166</td>
<td>Transmit DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_count_ser_1_tx</td>
<td>00_1006_0898</td>
<td>95/166</td>
<td>Transmit DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_1_tx</td>
<td>00_1006_08A0</td>
<td>96/167</td>
<td>Transmit DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_1_tx</td>
<td>00_1006_08A8</td>
<td>97/167</td>
<td>Transmit DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_1_tx</td>
<td>00_1006_08B0</td>
<td>98/167</td>
<td>Transmit DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>ser_mode_1</td>
<td>00_1006_0900</td>
<td>231/353</td>
<td>Mode select.</td>
</tr>
<tr>
<td>ser_minfrm_sz_1</td>
<td>00_1006_0908</td>
<td>237/355</td>
<td>Min frame size.</td>
</tr>
<tr>
<td>ser_maxfrm_sz_1</td>
<td>00_1006_0910</td>
<td>238/356</td>
<td>Max frame size.</td>
</tr>
<tr>
<td>ser_addr_mask_1</td>
<td>00_1006_0918</td>
<td>243/358</td>
<td>Address mask.</td>
</tr>
<tr>
<td>ser_usr0_addr_1</td>
<td>00_1006_0920</td>
<td>244/358</td>
<td>Match address 0.</td>
</tr>
<tr>
<td>ser_usr1_addr_1</td>
<td>00_1006_0928</td>
<td>244/358</td>
<td>Match address 1.</td>
</tr>
<tr>
<td>ser_usr2_addr_1</td>
<td>00_1006_0930</td>
<td>244/358</td>
<td>Match address 2.</td>
</tr>
<tr>
<td>ser_usr3_addr_1</td>
<td>00_1006_0938</td>
<td>244/358</td>
<td>Match address 3.</td>
</tr>
<tr>
<td>ser_cmd_1</td>
<td>00_1006_0940</td>
<td>233/354</td>
<td>Command</td>
</tr>
<tr>
<td>ser_tx_rd_thrss_1</td>
<td>00_1006_0960</td>
<td>235/355</td>
<td>Transmit FIFO read threshold.</td>
</tr>
<tr>
<td>ser_tx_wr_thrss_1</td>
<td>00_1006_0968</td>
<td>234/355</td>
<td>Transmit FIFO write threshold.</td>
</tr>
<tr>
<td>ser_rx_rd_thrss_1</td>
<td>00_1006_0970</td>
<td>236/355</td>
<td>Receive FIFO read threshold.</td>
</tr>
<tr>
<td>ser_line_mode_1</td>
<td>00_1006_0978</td>
<td>232/353</td>
<td>Line Interface configuration register.</td>
</tr>
<tr>
<td>ser_dma_enable_1</td>
<td>00_1006_0980</td>
<td>239/356</td>
<td>DMA channel enable register.</td>
</tr>
<tr>
<td>ser_status_1</td>
<td>00_1006_0988</td>
<td>240/356</td>
<td>Serial interface and DMA status (Read Only, read clears).</td>
</tr>
<tr>
<td>ser_int_mask_1</td>
<td>00_1006_0990</td>
<td>242/357</td>
<td>Interrupt mask.</td>
</tr>
<tr>
<td>dma_asic_addr_ser_1</td>
<td>00_1006_0998</td>
<td>94/166</td>
<td>ASIC mode address.</td>
</tr>
<tr>
<td>ser_debug_status_1</td>
<td>00_1006_09A8</td>
<td>241/357</td>
<td>Serial interface and DMA status (Read Only, no side effects).</td>
</tr>
<tr>
<td>ser_tx_byte_lo_1</td>
<td>00_1006_09C0</td>
<td>246/359</td>
<td>Serial interface transmit byte count (low 16 bits).</td>
</tr>
<tr>
<td>ser_tx_byte_hi_1</td>
<td>00_1006_09C8</td>
<td>246/359</td>
<td>Serial interface transmit byte count (high 16 bits).</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ser_rx_byte_lo_1</td>
<td>00_1006_09D0</td>
<td>246/359</td>
<td>Serial interface receive byte count (low 16 bits).</td>
</tr>
<tr>
<td>ser_rx_byte_hi_1</td>
<td>00_1006_09D8</td>
<td>246/359</td>
<td>Serial interface receive byte count (high 16 bits).</td>
</tr>
<tr>
<td>ser_tx_underun_1</td>
<td>00_1006_09E0</td>
<td>246/359</td>
<td>Serial interface transmit underrun count.</td>
</tr>
<tr>
<td>ser_rx_overflow_1</td>
<td>00_1006_09E8</td>
<td>246/359</td>
<td>Serial interface receive overflow count.</td>
</tr>
<tr>
<td>ser_rx_errors_1</td>
<td>00_1006_09F0</td>
<td>246/359</td>
<td>Serial interface receive error packet count.</td>
</tr>
<tr>
<td>ser_rx_badaddr_1</td>
<td>00_1006_09F8</td>
<td>246/359</td>
<td>Serial interface receive address mismatch count.</td>
</tr>
<tr>
<td>ser_rx_table0_1</td>
<td>00_1006_0A00</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table1_1</td>
<td>00_1006_0A08</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table2_1</td>
<td>00_1006_0A10</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table3_1</td>
<td>00_1006_0A18</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table4_1</td>
<td>00_1006_0A20</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table5_1</td>
<td>00_1006_0A28</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table6_1</td>
<td>00_1006_0A30</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table7_1</td>
<td>00_1006_0A38</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table8_1</td>
<td>00_1006_0A40</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table9_1</td>
<td>00_1006_0A48</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table10_1</td>
<td>00_1006_0A50</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table11_1</td>
<td>00_1006_0A58</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table12_1</td>
<td>00_1006_0A60</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table13_1</td>
<td>00_1006_0A68</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table14_1</td>
<td>00_1006_0A70</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table15_1</td>
<td>00_1006_0A78</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table0_1</td>
<td>00_1006_0B00</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table1_1</td>
<td>00_1006_0B08</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table2_1</td>
<td>00_1006_0B10</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table3_1</td>
<td>00_1006_0B18</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table4_1</td>
<td>00_1006_0B20</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table5_1</td>
<td>00_1006_0B28</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table6_1</td>
<td>00_1006_0B30</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table7_1</td>
<td>00_1006_0B38</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table8_1</td>
<td>00_1006_0B40</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table9_1</td>
<td>00_1006_0B48</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table10_1</td>
<td>00_1006_0B50</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table11_1</td>
<td>00_1006_0B58</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table12_1</td>
<td>00_1006_0B60</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table13_1</td>
<td>00_1006_0B68</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table14_1</td>
<td>00_1006_0B70</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>ser_tx_table15_1</td>
<td>00_1006_0B78</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>io_ext_cfg_0</td>
<td>00_1006_1000</td>
<td>251/374</td>
<td>cs0 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_1</td>
<td>00_1006_1008</td>
<td>251/374</td>
<td>cs1 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_2</td>
<td>00_1006_1010</td>
<td>251/374</td>
<td>cs2 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_3</td>
<td>00_1006_1018</td>
<td>251/374</td>
<td>cs3 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_4</td>
<td>00_1006_1020</td>
<td>251/374</td>
<td>cs4 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_5</td>
<td>00_1006_1028</td>
<td>251/374</td>
<td>cs5 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_6</td>
<td>00_1006_1030</td>
<td>251/374</td>
<td>cs6 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_7</td>
<td>00_1006_1038</td>
<td>251/374</td>
<td>cs7 interface configuration.</td>
</tr>
<tr>
<td>io_ext_mult_size_0</td>
<td>00_1006_1100</td>
<td>252/374</td>
<td>cs0 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_1</td>
<td>00_1006_1108</td>
<td>252/374</td>
<td>cs1 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_2</td>
<td>00_1006_1110</td>
<td>252/374</td>
<td>cs2 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_3</td>
<td>00_1006_1118</td>
<td>252/374</td>
<td>cs3 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_4</td>
<td>00_1006_1120</td>
<td>252/374</td>
<td>cs4 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_5</td>
<td>00_1006_1128</td>
<td>252/374</td>
<td>cs5 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_6</td>
<td>00_1006_1130</td>
<td>252/374</td>
<td>cs6 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_7</td>
<td>00_1006_1138</td>
<td>252/374</td>
<td>cs7 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_start_addr_0</td>
<td>00_1006_1200</td>
<td>253/375</td>
<td>cs0 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_1</td>
<td>00_1006_1208</td>
<td>253/375</td>
<td>cs1 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_2</td>
<td>00_1006_1210</td>
<td>253/375</td>
<td>cs2 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_3</td>
<td>00_1006_1218</td>
<td>253/375</td>
<td>cs3 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_4</td>
<td>00_1006_1220</td>
<td>253/375</td>
<td>cs4 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_5</td>
<td>00_1006_1228</td>
<td>253/375</td>
<td>cs5 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_6</td>
<td>00_1006_1230</td>
<td>253/375</td>
<td>cs6 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_7</td>
<td>00_1006_1238</td>
<td>253/375</td>
<td>cs7 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_time_cfg0_0</td>
<td>00_1006_1600</td>
<td>254/375</td>
<td>cs0 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_1</td>
<td>00_1006_1608</td>
<td>254/375</td>
<td>cs1 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_2</td>
<td>00_1006_1610</td>
<td>254/375</td>
<td>cs2 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_3</td>
<td>00_1006_1618</td>
<td>254/375</td>
<td>cs3 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_4</td>
<td>00_1006_1620</td>
<td>254/375</td>
<td>cs4 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_5</td>
<td>00_1006_1628</td>
<td>254/375</td>
<td>cs5 Timing parameter configuration 0.</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_ext_time_cfg0_6</td>
<td>00_1006_1630</td>
<td>254/375</td>
<td>cs6 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_7</td>
<td>00_1006_1638</td>
<td>254/375</td>
<td>cs7 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_0</td>
<td>00_1006_1700</td>
<td>255/376</td>
<td>cs0 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_1</td>
<td>00_1006_1708</td>
<td>255/376</td>
<td>cs1 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_2</td>
<td>00_1006_1710</td>
<td>255/376</td>
<td>cs2 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_3</td>
<td>00_1006_1718</td>
<td>255/376</td>
<td>cs3 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_4</td>
<td>00_1006_1720</td>
<td>255/376</td>
<td>cs4 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_5</td>
<td>00_1006_1728</td>
<td>255/376</td>
<td>cs5 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_6</td>
<td>00_1006_1730</td>
<td>255/376</td>
<td>cs6 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_7</td>
<td>00_1006_1738</td>
<td>255/376</td>
<td>cs7 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_interrupt_status</td>
<td>00_1006_1a00</td>
<td>256/376</td>
<td>Interrupt status for generic bus sources (Read Only, read clears).</td>
</tr>
<tr>
<td>io_interrupt_data0</td>
<td>00_1006_1a10</td>
<td>257/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data1</td>
<td>00_1006_1a18</td>
<td>258/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data2</td>
<td>00_1006_1a20</td>
<td>259/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data3</td>
<td>00_1006_1a28</td>
<td>260/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_addr0</td>
<td>00_1006_1a30</td>
<td>261/378</td>
<td>Address latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_addr1</td>
<td>00_1006_1a40</td>
<td>262/378</td>
<td>Address latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_parity</td>
<td>00_1006_1a50</td>
<td>263/378</td>
<td>Parity latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>pcmcia_cfg</td>
<td>00_1006_1a60</td>
<td>273/393</td>
<td>PCICMA controller configuration.</td>
</tr>
<tr>
<td>pcmcia_status</td>
<td>00_1006_1a70</td>
<td>274/394</td>
<td>PCMCIA controller status (Read Only, read clears interrupt).</td>
</tr>
<tr>
<td><strong>GPIO</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gpio_clr_edge</td>
<td>00_1006_1a80</td>
<td>276/398</td>
<td>Write a 1 to clear the edge detector for that bit (Write Only).</td>
</tr>
<tr>
<td>gpio_int_type</td>
<td>00_1006_1a88</td>
<td>277/398</td>
<td>Select interrupt type for pairs of pins.</td>
</tr>
<tr>
<td>gpio_input_invert</td>
<td>00_1006_1a90</td>
<td>279/398</td>
<td>Set to invert the input.</td>
</tr>
<tr>
<td>gpio_glitch</td>
<td>00_1006_1a98</td>
<td>280/399</td>
<td>Set for 1ms glitch filter, clear for 60 ns.</td>
</tr>
<tr>
<td>gpio_read</td>
<td>00_1006_1aa0</td>
<td>278/398</td>
<td>Shows current value of signal after possible invert and glitch (Read Only).</td>
</tr>
<tr>
<td>gpio_direction</td>
<td>00_1006_1aab</td>
<td>281/399</td>
<td>GPIO pin data direction. 1 for output, 0 for input.</td>
</tr>
<tr>
<td>gpio_pin_clr</td>
<td>00_1006_1ab0</td>
<td>282/399</td>
<td>Write 1 to set that bit to zero (Write Only).</td>
</tr>
<tr>
<td>gpio_pin_set</td>
<td>00_1006_1ab8</td>
<td>283/399</td>
<td>Write 1 to set that bit to one (Write Only).</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output Drive Control</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>io_drive_0</td>
<td>00_1006_1300</td>
<td>264/378</td>
<td>3.3V output drive strength and slew rate control registers. See Section: “Drive Strength Control” on page 373.</td>
</tr>
<tr>
<td>io_drive_1</td>
<td>00_1006_1308</td>
<td>265/379</td>
<td></td>
</tr>
<tr>
<td>io_drive_2</td>
<td>00_1006_1310</td>
<td>266/380</td>
<td></td>
</tr>
<tr>
<td>io_drive_3</td>
<td>00_1006_1318</td>
<td>267/380</td>
<td></td>
</tr>
<tr>
<td><strong>SMBus</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smb_serial_xtra_0</td>
<td>00_1006_0000</td>
<td>294/416</td>
<td>SMB port extra data register.</td>
</tr>
<tr>
<td>smb_serial_xtra_1</td>
<td>00_1006_0008</td>
<td>294/416</td>
<td>SMB port extra data register.</td>
</tr>
<tr>
<td>smb_serial_freq_0</td>
<td>00_1006_0010</td>
<td>289/415</td>
<td>Frequency for SMB port.</td>
</tr>
<tr>
<td>smb_serial_freq_1</td>
<td>00_1006_0018</td>
<td>289/415</td>
<td>Frequency for SMB port.</td>
</tr>
<tr>
<td>smb_serial_status_0</td>
<td>00_1006_0020</td>
<td>292/416</td>
<td>SMB port status (Read clears finish interrupt).</td>
</tr>
<tr>
<td>smb_serial_status_1</td>
<td>00_1006_0028</td>
<td>292/416</td>
<td>SMB port status (Read clears finish interrupt).</td>
</tr>
<tr>
<td>smb_serial_cmd_0</td>
<td>00_1006_0030</td>
<td>290/415</td>
<td>SMB port command.</td>
</tr>
<tr>
<td>smb_serial_cmd_1</td>
<td>00_1006_0038</td>
<td>290/415</td>
<td>SMB port command.</td>
</tr>
<tr>
<td>smb_serial_start_0</td>
<td>00_1006_0040</td>
<td>296/417</td>
<td>SMB port start register.</td>
</tr>
<tr>
<td>smb_serial_start_1</td>
<td>00_1006_0048</td>
<td>296/417</td>
<td>SMB port start register.</td>
</tr>
<tr>
<td>smb_serial_data_0</td>
<td>00_1006_0050</td>
<td>293/416</td>
<td>SMB port data register.</td>
</tr>
<tr>
<td>smb_serial_data_1</td>
<td>00_1006_0058</td>
<td>293/416</td>
<td>SMB port data register.</td>
</tr>
<tr>
<td>smb_serial_control_0</td>
<td>00_1006_0060</td>
<td>291/415</td>
<td>SMB port control register.</td>
</tr>
<tr>
<td>smb_serial_control_1</td>
<td>00_1006_0068</td>
<td>291/415</td>
<td>SMB port control register.</td>
</tr>
<tr>
<td>smb_serial_pec_0</td>
<td>00_1006_0070</td>
<td>295/417</td>
<td>SMB port pec register.</td>
</tr>
<tr>
<td>smb_serial_pec_1</td>
<td>00_1006_0078</td>
<td>295/417</td>
<td>SMB port pec register.</td>
</tr>
<tr>
<td><strong>SCD: Timers</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_0</td>
<td>00_1002_0050</td>
<td>23/59</td>
<td>Watchdog initial count.</td>
</tr>
<tr>
<td>watchdog_timer_cnt_0</td>
<td>00_1002_0058</td>
<td>24/59</td>
<td>Watchdog current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cfg_0</td>
<td>00_1002_0060</td>
<td>25/59</td>
<td>Watchdog configuration (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_init_cnt_0</td>
<td>00_1002_0070</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>general_timer_init_cnt_1</td>
<td>00_1002_0078</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>general_timer_cnt_0</td>
<td>00_1002_0080</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cnt_1</td>
<td>00_1002_0088</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cfg_0</td>
<td>00_1002_0090</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_cfg_1</td>
<td>00_1002_0098</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_1</td>
<td>00_1002_0150</td>
<td>23/59</td>
<td>Watchdog initial count.</td>
</tr>
<tr>
<td>watchdog_timer_cnt_1</td>
<td>00_1002_0158</td>
<td>24/59</td>
<td>Watchdog current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cfg_1</td>
<td>00_1002_0160</td>
<td>25/59</td>
<td>Watchdog configuration (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_init_cnt_2</td>
<td>00_1002_0170</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>general_timer_init_cnt_3</td>
<td>00_1002_0178</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>general_timer_cnt_2</td>
<td>00_1002_0180</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cnt_3</td>
<td>00_1002_0188</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cfg_2</td>
<td>00_1002_0190</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_cfg_3</td>
<td>00_1002_0198</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>zbbus_cycle_cnt</td>
<td>00_1003_0000</td>
<td>29/61</td>
<td>ZBbus cycle count.</td>
</tr>
<tr>
<td>zbbus_cycle_cp0</td>
<td>00_1002_0C00</td>
<td>30/61</td>
<td>ZBbus cycle count compare register zero.</td>
</tr>
<tr>
<td>zbbus_cycle_cp1</td>
<td>00_1002_0C08</td>
<td>30/61</td>
<td>ZBbus cycle count compare register one.</td>
</tr>
</tbody>
</table>

**SCD: System Control**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system_revision</td>
<td>00_1002_0000</td>
<td>14/43</td>
<td>System id and revision (Read Only).</td>
</tr>
<tr>
<td>system_cfg</td>
<td>00_1002_0008</td>
<td>15/43</td>
<td>System configuration register.</td>
</tr>
<tr>
<td>system_scratch</td>
<td>00_1002_0C10</td>
<td>16/45</td>
<td>Scratch register for software use.</td>
</tr>
<tr>
<td>system_manuf</td>
<td>00_1003_8000</td>
<td>14/43</td>
<td>Read Only. Manufacturing Information Register.</td>
</tr>
</tbody>
</table>

**SCD: Address Trap**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addr_trap_index</td>
<td>00_1002_00b0</td>
<td>40/68</td>
<td>Index of interrupting trap (Read Only).</td>
</tr>
<tr>
<td>addr_trap_reg</td>
<td>00_1002_00b8</td>
<td>42/68</td>
<td>Address of interrupting trap (Read Only, read clears interrupt).</td>
</tr>
<tr>
<td>addr_trap_reg_debug</td>
<td>00_1002_04b0</td>
<td>46/68</td>
<td>Address of interrupting trap (Read Only, No side effects).</td>
</tr>
<tr>
<td>addr_trap_up_cfg_0</td>
<td>00_1002_0400</td>
<td>41/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_1</td>
<td>00_1002_0408</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_2</td>
<td>00_1002_0410</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_3</td>
<td>00_1002_0418</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addrTrap_down_cfg_0</td>
<td>00_1002_0420</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addrTrap_down_cfg_1</td>
<td>00_1002_0428</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addrTrap_down_cfg_2</td>
<td>00_1002_0430</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addrTrap_down_cfg_3</td>
<td>00_1002_0438</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addr_trap_cfg_0</td>
<td>00_1002_0440</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_1</td>
<td>00_1002_0448</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_2</td>
<td>00_1002_0450</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_3</td>
<td>00_1002_0458</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
</tbody>
</table>
### SCD: Interrupt Mapper

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_diag_0</td>
<td>00_1002_0010</td>
<td>21/52</td>
<td>Force interrupt diagnostic register.</td>
</tr>
<tr>
<td>interrupt_mask_0</td>
<td>00_1002_0028</td>
<td>21/52</td>
<td>Interrupt mask.</td>
</tr>
<tr>
<td>interrupt_source_status_0</td>
<td>00_1002_0040</td>
<td>21/52</td>
<td>Status of system interrupt sources (Read Only).</td>
</tr>
<tr>
<td>ldt_interrupt_0</td>
<td>00_1002_0018</td>
<td>21/52</td>
<td>Status of HyperTransport interrupt sources (Read Only).</td>
</tr>
<tr>
<td>ldt_interrupt_clr_0</td>
<td>00_1002_0020</td>
<td>21/52</td>
<td>Clear ldt_interrupt_0 by writing 1s to this location.</td>
</tr>
<tr>
<td>interrupt_trace_0</td>
<td>00_1002_0038</td>
<td>21/52</td>
<td>Set bits in this register to cause the corresponding interrupt to be passed to the trace trigger logic.</td>
</tr>
<tr>
<td>ldt_interrupt_set</td>
<td>00_1002_0048</td>
<td>19/49</td>
<td>Write HyperTransport interrupt message to this address to raise a HyperTransport interrupt (Write Only).</td>
</tr>
<tr>
<td>mailbox_cpu_0</td>
<td>00_1002_00c0</td>
<td>17/46</td>
<td>Status of mailbox (Read Only, has alias).</td>
</tr>
<tr>
<td>alias_mailbox_cpu_0</td>
<td>00_1002_1xx0</td>
<td>17/46</td>
<td>Alias of status of mailbox (has incomplete decode, so access from pci won't hang). (Read Only).</td>
</tr>
<tr>
<td>mailbox_set_cpu_0</td>
<td>00_1002_00c8</td>
<td>17/46</td>
<td>Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only).</td>
</tr>
<tr>
<td>alias_mailbox_set_cpu_0</td>
<td>00_1002_1xx8</td>
<td>17/46</td>
<td>Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only, has incomplete decode, so access from pci won't hang).</td>
</tr>
<tr>
<td>mailbox_clr_cpu_0</td>
<td>00_1002_00d0</td>
<td>17/46</td>
<td>Clear bits in mailbox_cpu_0 by writing 1s to this location (Write Only).</td>
</tr>
<tr>
<td>interrupt_status0_0</td>
<td>00_1002_0100</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status1_0</td>
<td>00_1002_0108</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status2_0</td>
<td>00_1002_0110</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status3_0</td>
<td>00_1002_0118</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status4_0</td>
<td>00_1002_0120</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status5_0</td>
<td>00_1002_0128</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status6_0</td>
<td>00_1002_0130</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status7_0</td>
<td>00_1002_0138</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_map0_0</td>
<td>00_1002_0200</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map1_0</td>
<td>00_1002_0208</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map2_0</td>
<td>00_1002_0210</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map3_0</td>
<td>00_1002_0218</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map4_0</td>
<td>00_1002_0220</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map5_0</td>
<td>00_1002_0228</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map6_0</td>
<td>00_1002_0230</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map7_0</td>
<td>00_1002_0238</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map8_0</td>
<td>00_1002_0240</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map9_0</td>
<td>00_1002_0248</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map10_0</td>
<td>00_1002_0250</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_map11_0</td>
<td>00_1002_0258</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map12_0</td>
<td>00_1002_0260</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map13_0</td>
<td>00_1002_0268</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map14_0</td>
<td>00_1002_0270</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map15_0</td>
<td>00_1002_0278</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map16_0</td>
<td>00_1002_0280</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map17_0</td>
<td>00_1002_0288</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map18_0</td>
<td>00_1002_0290</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map19_0</td>
<td>00_1002_0298</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map20_0</td>
<td>00_1002_02a0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map21_0</td>
<td>00_1002_02a8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map22_0</td>
<td>00_1002_02b0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map23_0</td>
<td>00_1002_02b8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map24_0</td>
<td>00_1002_02c0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map25_0</td>
<td>00_1002_02c8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map26_0</td>
<td>00_1002_02d0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map27_0</td>
<td>00_1002_02d8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map28_0</td>
<td>00_1002_02e0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map29_0</td>
<td>00_1002_02e8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map30_0</td>
<td>00_1002_02f0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map31_0</td>
<td>00_1002_02f8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map32_0</td>
<td>00_1002_0300</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map33_0</td>
<td>00_1002_0308</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map34_0</td>
<td>00_1002_0310</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map35_0</td>
<td>00_1002_0318</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map36_0</td>
<td>00_1002_0320</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map37_0</td>
<td>00_1002_0328</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map38_0</td>
<td>00_1002_0330</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map39_0</td>
<td>00_1002_0338</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map40_0</td>
<td>00_1002_0340</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map41_0</td>
<td>00_1002_0348</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map42_0</td>
<td>00_1002_0350</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map43_0</td>
<td>00_1002_0358</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map44_0</td>
<td>00_1002_0360</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map45_0</td>
<td>00_1002_0368</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map46_0</td>
<td>00_1002_0370</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map47_0</td>
<td>00_1002_0378</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_map48_0</td>
<td>00_1002_0380</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map49_0</td>
<td>00_1002_0388</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map50_0</td>
<td>00_1002_0390</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map51_0</td>
<td>00_1002_0398</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map52_0</td>
<td>00_1002_03a0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map53_0</td>
<td>00_1002_03a8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map54_0</td>
<td>00_1002_03b0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map55_0</td>
<td>00_1002_03b8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map56_0</td>
<td>00_1002_03c0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map57_0</td>
<td>00_1002_03c8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map58_0</td>
<td>00_1002_03d0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map59_0</td>
<td>00_1002_03d8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map60_0</td>
<td>00_1002_03e0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map61_0</td>
<td>00_1002_03e8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map62_0</td>
<td>00_1002_03f0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map63_0</td>
<td>00_1002_03f8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>int_mapper_1</td>
<td>00_1002_2000</td>
<td>2000-23f8</td>
<td>2000-23f8 Registers for CPU 1 interrupt mapper (+2000 from CPU 0).</td>
</tr>
</tbody>
</table>

**SCD: Performance Counters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>perf_cnt_cfg</td>
<td>00_1002_04c0</td>
<td>31/61</td>
<td>Performance counter control.</td>
</tr>
<tr>
<td>perf_cnt_0</td>
<td>00_1002_04d0</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_1</td>
<td>00_1002_04d8</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_2</td>
<td>00_1002_04e0</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_3</td>
<td>00_1002_04e8</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
</tbody>
</table>

**SCD: Bus Watcher**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus_err_status</td>
<td>00_1002_0880</td>
<td>35/65</td>
<td>Bus error status (Read Only).</td>
</tr>
<tr>
<td>bus_err_status_debug</td>
<td>00_1002_08d0</td>
<td>36/65</td>
<td>Bus error status (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_0</td>
<td>00_1002_08a0</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_1</td>
<td>00_1002_08a8</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_2</td>
<td>00_1002_08b0</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_3</td>
<td>00_1002_08b8</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_l2_errors</td>
<td>00_1002_08c0</td>
<td>38/66</td>
<td>Count of L2 ECC errors (4 8-bit counters).</td>
</tr>
<tr>
<td>bus_mem_io_errors</td>
<td>00_1002_08c8</td>
<td>39/66</td>
<td>Count of memory ECC errors and I/O errors (3 8-bit counters).</td>
</tr>
</tbody>
</table>
### Table 311: Internal Register Addresses by Function (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SCD: Debug Controller</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jtag_space</td>
<td>00_1000_0000</td>
<td>1000_0000-1001_ffff.</td>
<td>JTAG serviced debug space</td>
</tr>
<tr>
<td><strong>SCD: Trace Buffer</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trace_cfg</td>
<td>00_1002_0a00</td>
<td>48/76</td>
<td>Trace configuration.</td>
</tr>
<tr>
<td>trace_read</td>
<td>00_1002_0a08</td>
<td>50/79</td>
<td>Trace buffer read register (Read Only).</td>
</tr>
<tr>
<td>trace_event_0</td>
<td>00_1002_0a20</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_1</td>
<td>00_1002_0a28</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_2</td>
<td>00_1002_0a30</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_3</td>
<td>00_1002_0a38</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_sequence_0</td>
<td>00_1002_0a40</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_1</td>
<td>00_1002_0a48</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_2</td>
<td>00_1002_0a50</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_3</td>
<td>00_1002_0a58</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_event_4</td>
<td>00_1002_0a60</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_5</td>
<td>00_1002_0a68</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_6</td>
<td>00_1002_0a70</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_7</td>
<td>00_1002_0a78</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_sequence_4</td>
<td>00_1002_0a80</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_5</td>
<td>00_1002_0a88</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_6</td>
<td>00_1002_0a90</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_7</td>
<td>00_1002_0a98</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td><strong>SCD: Data Mover</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dm_dscr_base_0</td>
<td>00_1002_0b00</td>
<td>114/184</td>
<td>Data Mover channel 0 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_0</td>
<td>00_1002_0b08</td>
<td>116/185</td>
<td>Data Mover channel 0 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_0</td>
<td>00_1002_0b10</td>
<td>117/185</td>
<td>Data Mover channel 0 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_0</td>
<td>00_1002_0b18</td>
<td>115/184</td>
<td>Debug alias for channel 0 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_partial_0</td>
<td>00_1002_0b20</td>
<td>114/184</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_dscr_base_1</td>
<td>00_1002_0b20</td>
<td>114/184</td>
<td>Data Mover channel 1 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_1</td>
<td>00_1002_0b28</td>
<td>116/185</td>
<td>Data Mover channel 1 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_1</td>
<td>00_1002_0b30</td>
<td>117/185</td>
<td>Data Mover channel 1 current address. (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_1</td>
<td>00_1002_0b38</td>
<td>115/184</td>
<td>Debug alias for channel 1 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_partial_1</td>
<td>00_1002_0b40</td>
<td>114/184</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_dscr_base_2</td>
<td>00_1002_0b40</td>
<td>114/184</td>
<td>Data Mover channel 2 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_2</td>
<td>00_1002_0b48</td>
<td>116/185</td>
<td>Data Mover channel 2 descriptor count.</td>
</tr>
</tbody>
</table>
**Table 311: Internal Register Addresses by Function (Cont.)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dm_dscr_addr_2</td>
<td>00_1002_0b50</td>
<td>117/185</td>
<td>Data Mover channel 2 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_2</td>
<td>00_1002_0b58</td>
<td>115/184</td>
<td>Debug alias for channel 2 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_partial_2</td>
<td>00_1002_0b60</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_dscr_base_3</td>
<td>00_1002_0b68</td>
<td>114/184</td>
<td>Data Mover channel 3 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_3</td>
<td>00_1002_0b70</td>
<td>116/185</td>
<td>Data Mover channel 3 descriptor count.</td>
</tr>
<tr>
<td>dm_debug_dscr_base_3</td>
<td>00_1002_0b78</td>
<td>115/184</td>
<td>Debug alias for channel 3 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_partial_3</td>
<td>00_1002_0b80</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>crc_def_0</td>
<td>00_1002_0b88</td>
<td>118/185</td>
<td>CRC definition register.</td>
</tr>
<tr>
<td>ctcp_def_0</td>
<td>00_1002_0b90</td>
<td>119/186</td>
<td>CRC and checksum definition register.</td>
</tr>
<tr>
<td>crc_def_1</td>
<td>00_1002_0b90</td>
<td>118/185</td>
<td>CRC definition register.</td>
</tr>
<tr>
<td>ctcp_def_1</td>
<td>00_1002_0b98</td>
<td>119/186</td>
<td>CRC and checksum definition register.</td>
</tr>
</tbody>
</table>
### BCM1250/BCM1125/H Internal Registers Ordered by Address

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jtag_space</td>
<td>00_1000_0000</td>
<td></td>
<td>JTAG serviced debug space 1000_0000-1001_ffff.</td>
</tr>
<tr>
<td>system_revision</td>
<td>00_1002_0000</td>
<td>14/43</td>
<td>System Id and revision (Read Only).</td>
</tr>
<tr>
<td>system_cfg</td>
<td>00_1002_0008</td>
<td>15/43</td>
<td>System configuration register.</td>
</tr>
<tr>
<td>interrupt_diag_0</td>
<td>00_1002_0010</td>
<td>21/52</td>
<td>Force interrupt diagnostic register.</td>
</tr>
<tr>
<td>ldtd_interrupt_0</td>
<td>00_1002_0018</td>
<td>21/52</td>
<td>Status of HyperTransport interrupt sources (Read Only).</td>
</tr>
<tr>
<td>ldtd_interrupt_clr_0</td>
<td>00_1002_0020</td>
<td>21/52</td>
<td>Clear ldtd_interrupt_0 by writing 1s to this location.</td>
</tr>
<tr>
<td>interrupt_mask_0</td>
<td>00_1002_0028</td>
<td>21/52</td>
<td>Interrupt mask.</td>
</tr>
<tr>
<td>interrupt_trace_0</td>
<td>00_1002_0038</td>
<td>21/52</td>
<td>Set bits in this register to cause the corresponding interrupt to be passed to the trace trigger logic.</td>
</tr>
<tr>
<td>interrupt_source_status_0</td>
<td>00_1002_0040</td>
<td>21/52</td>
<td>Status of system interrupt sources (Read Only).</td>
</tr>
<tr>
<td>ldtd_interrupt_set</td>
<td>00_1002_0048</td>
<td>19/49</td>
<td>Write HyperTransport interrupt message to this address to raise a HyperTransport interrupt (Write Only).</td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_0</td>
<td>00_1002_0050</td>
<td>23/59</td>
<td>Watchdog initial count.</td>
</tr>
<tr>
<td>watchdog_timer_cnt_0</td>
<td>00_1002_0058</td>
<td>24/59</td>
<td>Watchdog current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cfg_0</td>
<td>00_1002_0060</td>
<td>25/59</td>
<td>Watchdog configuration (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_init_cnt_0</td>
<td>00_1002_0070</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>general_timer_cnt_1</td>
<td>00_1002_0078</td>
<td>26/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cnt_0</td>
<td>00_1002_0080</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>general_timer_cfg_1</td>
<td>00_1002_0088</td>
<td>27/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>general_timer_cfg_0</td>
<td>00_1002_0090</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>addr_trap_index</td>
<td>00_1002_00b0</td>
<td>40/68</td>
<td>Index of interrupting trap (Read Only).</td>
</tr>
<tr>
<td>addr_trap_reg</td>
<td>00_1002_00b8</td>
<td>42/68</td>
<td>Address of interrupting trap (Read Only, read clears interrupt).</td>
</tr>
<tr>
<td>mailbox_cpu_0</td>
<td>00_1002_00c0</td>
<td>17/46</td>
<td>Status of mailbox (Read Only, has alias).</td>
</tr>
<tr>
<td>mailbox_set_cpu_0</td>
<td>00_1002_00c8</td>
<td>17/46</td>
<td>Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only).</td>
</tr>
<tr>
<td>mailbox_clr_cpu_0</td>
<td>00_1002_00d0</td>
<td>17/46</td>
<td>Clear bits in mailbox_cpu_0 by writing 1s to this location (Write Only).</td>
</tr>
<tr>
<td>interrupt_status0_0</td>
<td>00_1002_0100</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status1_0</td>
<td>00_1002_0108</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status2_0</td>
<td>00_1002_0110</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status3_0</td>
<td>00_1002_0118</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status4_0</td>
<td>00_1002_0120</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_status5_0</td>
<td>00_1002_0128</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status6_0</td>
<td>00_1002_0130</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>interrupt_status7_0</td>
<td>00_1002_0138</td>
<td>21/52</td>
<td>Status of mapped interrupt sources (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_1</td>
<td>00_1002_0150</td>
<td>23/59</td>
<td>Watchdog initial count.</td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_2</td>
<td>00_1002_0170</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>watchdog_timer_init_cnt_3</td>
<td>00_1002_0178</td>
<td>26/60</td>
<td>General timer initial count.</td>
</tr>
<tr>
<td>watchdog_timer_cnt_1</td>
<td>00_1002_0158</td>
<td>24/59</td>
<td>Watchdog current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cnt_2</td>
<td>00_1002_0180</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cnt_3</td>
<td>00_1002_0188</td>
<td>27/60</td>
<td>General timer current count (Read Only).</td>
</tr>
<tr>
<td>watchdog_timer_cfg_1</td>
<td>00_1002_0160</td>
<td>25/59</td>
<td>Watchdog configuration (Write clears interrupt).</td>
</tr>
<tr>
<td>watchdog_timer_cfg_2</td>
<td>00_1002_0190</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td></td>
<td>00_1002_0198</td>
<td>28/60</td>
<td>General timer configuration and enable (Write clears interrupt).</td>
</tr>
<tr>
<td>interrupt_map0_0</td>
<td>00_1002_0200</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map1_0</td>
<td>00_1002_0208</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map2_0</td>
<td>00_1002_0210</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map3_0</td>
<td>00_1002_0218</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map4_0</td>
<td>00_1002_0220</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map5_0</td>
<td>00_1002_0228</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map6_0</td>
<td>00_1002_0230</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map7_0</td>
<td>00_1002_0238</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map8_0</td>
<td>00_1002_0240</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map9_0</td>
<td>00_1002_0248</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map10_0</td>
<td>00_1002_0250</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map11_0</td>
<td>00_1002_0258</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map12_0</td>
<td>00_1002_0260</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map13_0</td>
<td>00_1002_0268</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map14_0</td>
<td>00_1002_0270</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map15_0</td>
<td>00_1002_0278</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map16_0</td>
<td>00_1002_0280</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map17_0</td>
<td>00_1002_0288</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map18_0</td>
<td>00_1002_0290</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map19_0</td>
<td>00_1002_0298</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map20_0</td>
<td>00_1002_02a0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map21_0</td>
<td>00_1002_02a8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map22_0</td>
<td>00_1002_02b0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map23_0</td>
<td>00_1002_02b8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
</tbody>
</table>
Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interrupt_map24_0</td>
<td>00_1002_02e0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map25_0</td>
<td>00_1002_02e8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map26_0</td>
<td>00_1002_02d0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map27_0</td>
<td>00_1002_02d8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map28_0</td>
<td>00_1002_02e0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map29_0</td>
<td>00_1002_02e8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map30_0</td>
<td>00_1002_02f0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map31_0</td>
<td>00_1002_02f8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map32_0</td>
<td>00_1002_0300</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map33_0</td>
<td>00_1002_0308</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map34_0</td>
<td>00_1002_0310</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map35_0</td>
<td>00_1002_0318</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map36_0</td>
<td>00_1002_0320</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map37_0</td>
<td>00_1002_0328</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map38_0</td>
<td>00_1002_0330</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map39_0</td>
<td>00_1002_0338</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map40_0</td>
<td>00_1002_0340</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map41_0</td>
<td>00_1002_0348</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map42_0</td>
<td>00_1002_0350</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map43_0</td>
<td>00_1002_0358</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map44_0</td>
<td>00_1002_0360</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map45_0</td>
<td>00_1002_0368</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map46_0</td>
<td>00_1002_0370</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map47_0</td>
<td>00_1002_0378</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map48_0</td>
<td>00_1002_0380</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map49_0</td>
<td>00_1002_0388</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map50_0</td>
<td>00_1002_0390</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map51_0</td>
<td>00_1002_0398</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map52_0</td>
<td>00_1002_03a0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map53_0</td>
<td>00_1002_03a8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map54_0</td>
<td>00_1002_03b0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map55_0</td>
<td>00_1002_03b8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map56_0</td>
<td>00_1002_03c0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map57_0</td>
<td>00_1002_03c8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map58_0</td>
<td>00_1002_03d0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map59_0</td>
<td>00_1002_03d8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map60_0</td>
<td>00_1002_03e0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------</td>
<td>------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>interrupt_map61_0</td>
<td>00_1002_03e8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map62_0</td>
<td>00_1002_03f0</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>interrupt_map63_0</td>
<td>00_1002_03f8</td>
<td>18/47</td>
<td>Source to IRQ map register.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_0</td>
<td>00_1002_0400</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_1</td>
<td>00_1002_0408</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_2</td>
<td>00_1002_0410</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_up_cfg_3</td>
<td>00_1002_0418</td>
<td>43/68</td>
<td>Top of address trap range.</td>
</tr>
<tr>
<td>addr_trap_down_cfg_0</td>
<td>00_1002_0420</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addr_trap_down_cfg_1</td>
<td>00_1002_0428</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addr_trap_down_cfg_2</td>
<td>00_1002_0430</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addr_trap_down_cfg_3</td>
<td>00_1002_0438</td>
<td>44/69</td>
<td>Bottom of address trap range.</td>
</tr>
<tr>
<td>addr_trap_cfg_0</td>
<td>00_1002_0440</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_1</td>
<td>00_1002_0448</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_2</td>
<td>00_1002_0450</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_cfg_3</td>
<td>00_1002_0458</td>
<td>45/69</td>
<td>Address trap configuration.</td>
</tr>
<tr>
<td>addr_trap_reg_debug</td>
<td>00_1002_0460</td>
<td>41/68</td>
<td>Address of interrupting trap (Read Only, No side effects).</td>
</tr>
<tr>
<td>perf_cnt_cfg</td>
<td>00_1002_04c0</td>
<td>31/61</td>
<td>Performance counter control.</td>
</tr>
<tr>
<td>perf_cnt_0</td>
<td>00_1002_04d0</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_1</td>
<td>00_1002_04d8</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_2</td>
<td>00_1002_04e0</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>perf_cnt_3</td>
<td>00_1002_04e8</td>
<td>32/62</td>
<td>Performance counter.</td>
</tr>
<tr>
<td>bus_err_status</td>
<td>00_1002_0880</td>
<td>35/65</td>
<td>Bus error status (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_0</td>
<td>00_1002_08a0</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_1</td>
<td>00_1002_08a8</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_2</td>
<td>00_1002_08b0</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_err_data_3</td>
<td>00_1002_08b8</td>
<td>37/65</td>
<td>Data or address/control information latched on bus error (Read Only).</td>
</tr>
<tr>
<td>bus_l2_errors</td>
<td>00_1002_08c0</td>
<td>38/66</td>
<td>Count of L2 ECC errors (4 8-bit counters).</td>
</tr>
<tr>
<td>bus_mem_io_errors</td>
<td>00_1002_08c8</td>
<td>39/66</td>
<td>Count of memory ECC errors and I/O errors (3 8-bit counters).</td>
</tr>
<tr>
<td>bus_err_status_debug</td>
<td>00_1002_08d0</td>
<td>36/65</td>
<td>Bus error status (Read Only).</td>
</tr>
<tr>
<td>trace_cfg</td>
<td>00_1002_0a00</td>
<td>48/76</td>
<td>Trace configuration.</td>
</tr>
<tr>
<td>trace_read</td>
<td>00_1002_0a08</td>
<td>50/79</td>
<td>Trace buffer read register (Read Only).</td>
</tr>
<tr>
<td>trace_event_0</td>
<td>00_1002_0a20</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_1</td>
<td>00_1002_0a28</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_2</td>
<td>00_1002_0a30</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trace_event_3</td>
<td>00_1002_0a38</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_sequence_0</td>
<td>00_1002_0a40</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_1</td>
<td>00_1002_0a48</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_2</td>
<td>00_1002_0a50</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_3</td>
<td>00_1002_0a58</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_event_4</td>
<td>00_1002_0a60</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_5</td>
<td>00_1002_0a68</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_event_6</td>
<td>00_1002_0a70</td>
<td>46/72</td>
<td>Trace event selector.</td>
</tr>
<tr>
<td>trace_sequence_4</td>
<td>00_1002_0a80</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_5</td>
<td>00_1002_0a88</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_6</td>
<td>00_1002_0a90</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>trace_sequence_7</td>
<td>00_1002_0a98</td>
<td>47/74</td>
<td>Trace sequence and action.</td>
</tr>
<tr>
<td>dm_dscr_base_0</td>
<td>00_1002_0b00</td>
<td>114/184</td>
<td>Data Mover channel 0 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_0</td>
<td>00_1002_0b08</td>
<td>116/185</td>
<td>Data Mover channel 0 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_0</td>
<td>00_1002_0b10</td>
<td>117/185</td>
<td>Data Mover channel 0 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_0</td>
<td>00_1002_0b18</td>
<td>115/184</td>
<td>Debug alias for channel 0 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_dscr_base_1</td>
<td>00_1002_0b20</td>
<td>114/184</td>
<td>Data Mover channel 1 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_1</td>
<td>00_1002_0b28</td>
<td>116/185</td>
<td>Data Mover channel 1 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_1</td>
<td>00_1002_0b30</td>
<td>117/185</td>
<td>Data Mover channel 1 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_1</td>
<td>00_1002_0b38</td>
<td>115/184</td>
<td>Debug alias for channel 1 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_dscr_base_2</td>
<td>00_1002_0b40</td>
<td>114/184</td>
<td>Data Mover channel 2 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_2</td>
<td>00_1002_0b48</td>
<td>116/185</td>
<td>Data Mover channel 2 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_2</td>
<td>00_1002_0b50</td>
<td>117/185</td>
<td>Data Mover channel 2 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_2</td>
<td>00_1002_0b58</td>
<td>115/184</td>
<td>Debug alias for channel 2 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>dm_dscr_base_3</td>
<td>00_1002_0b60</td>
<td>114/184</td>
<td>Data Mover channel 3 ring base address (Read clears some bits).</td>
</tr>
<tr>
<td>dm_dscr_count_3</td>
<td>00_1002_0b68</td>
<td>116/185</td>
<td>Data Mover channel 3 descriptor count.</td>
</tr>
<tr>
<td>dm_dscr_addr_3</td>
<td>00_1002_0b70</td>
<td>117/185</td>
<td>Data Mover channel 3 current address (Read Only).</td>
</tr>
<tr>
<td>dm_debug_dscr_base_3</td>
<td>00_1002_0b78</td>
<td>115/184</td>
<td>Debug alias for channel 3 ring base address (Read Only, no side effects).</td>
</tr>
<tr>
<td>crc_def_0</td>
<td>00_1002_0b80</td>
<td>118/185</td>
<td>CRC definition register.</td>
</tr>
<tr>
<td>ctcp_def_0</td>
<td>00_1002_0b88</td>
<td>119/186</td>
<td>CRC and checksum definition register.</td>
</tr>
<tr>
<td>crc_def_1</td>
<td>00_1002_0b90</td>
<td>118/185</td>
<td>CRC definition register.</td>
</tr>
</tbody>
</table>
Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctcp_def_1</td>
<td>00_1002_0b98</td>
<td>119/186</td>
<td>CRC and checksum definition register.</td>
</tr>
<tr>
<td>dm_partial_0</td>
<td>00_1002_0ba0</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_partial_1</td>
<td>00_1002_0ba8</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_partial_2</td>
<td>00_1002_0bb0</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>dm_partial_3</td>
<td>00_1002_0bb8</td>
<td>120/186</td>
<td>CRC/Checksum partial result register.</td>
</tr>
<tr>
<td>zbbus_cycle_cp0</td>
<td>00_1002_0c00</td>
<td>30/61</td>
<td>ZBbus cycle count compare register zero.</td>
</tr>
<tr>
<td>zbbus_cycle_cp1</td>
<td>00_1002_0c08</td>
<td>30/61</td>
<td>ZBbus cycle count compare register one.</td>
</tr>
<tr>
<td>system_scratch</td>
<td>00_1002_0c10</td>
<td>16/45</td>
<td>Scratch register for software use.</td>
</tr>
<tr>
<td>alias_mailbox_cpu_0</td>
<td>00_1002_1xx0</td>
<td>17/46</td>
<td>Alias of status of mailbox (has incomplete decode, so access from pci won't hang). (Read Only).</td>
</tr>
<tr>
<td>alias_mailbox_set_cpu_0</td>
<td>00_1002_1xx8</td>
<td>17/46</td>
<td>Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only, has incomplete decode, so access from pci won't hang).</td>
</tr>
<tr>
<td>int_mapper_1</td>
<td>00_1002_2000</td>
<td></td>
<td>2000-2318 Registers for CPU1 interrupt mapper (+2000 from CPU 0).</td>
</tr>
<tr>
<td>zbbus_cycle_cnt</td>
<td>00_1003_0000</td>
<td>29/61</td>
<td>ZBbus cycle count.</td>
</tr>
<tr>
<td>system_manuf</td>
<td>00_1003_8000</td>
<td>14/43</td>
<td>Read Only. Manufacturing Information Register.</td>
</tr>
<tr>
<td>l2_read_address</td>
<td>00_1004_0018</td>
<td>56/100</td>
<td>Read only. Last address/tag in a read (for testing)</td>
</tr>
<tr>
<td>l2_ecc_address</td>
<td>00_1004_0038</td>
<td>56/100</td>
<td>Read only. Last address with ecc error (correctable or not).</td>
</tr>
<tr>
<td>l2_misc_value</td>
<td>00_1004_0058</td>
<td>57/100</td>
<td>Read only. PERIPH_REV3 and later. Value of L2 hidden registers.</td>
</tr>
<tr>
<td>l2_way_disable</td>
<td>00_1004_1x00</td>
<td>/91</td>
<td>Accesses made to this range of addresses will write the value x to l2_wayen[3:0] register. If l2_wayen[i] is clear Way i is removed from the L2 replacement algorithm. See Section: “Using the L2 Cache as Memory” on page 91</td>
</tr>
<tr>
<td>l2_cache_disable</td>
<td>00_1004_2x00</td>
<td>/94</td>
<td>Accesses made to this range of addresses will write the value x to l2_cache_disable[3:0] register. See Section: “Reduced Cache Size” on page 94.</td>
</tr>
<tr>
<td>l2_misc_config</td>
<td>00_1004_3x00</td>
<td>/99</td>
<td>Accesses made to this range of addresses will write the value x to l2_misc_config[3:0] register. See Section: “Cache Configuration Register” on page 99.</td>
</tr>
<tr>
<td>mc_config_0</td>
<td>00_1005_1100</td>
<td>72/135</td>
<td>Channel 0 attributes.</td>
</tr>
<tr>
<td>mc_dramcmd_0</td>
<td>00_1005_1120</td>
<td>75/139</td>
<td>Channel 0 SDRAM command.</td>
</tr>
<tr>
<td>mc_drammode_0</td>
<td>00_1005_1140</td>
<td>76/139</td>
<td>Channel 0 SDRAM mode.</td>
</tr>
<tr>
<td>mc_timing1_0</td>
<td>00_1005_1160</td>
<td>77/140</td>
<td>Channel 0 SDRAM timing 1.</td>
</tr>
<tr>
<td>mc_timing2_0</td>
<td>00_1005_1180</td>
<td>78/141</td>
<td>Channel 0 SDRAM timing 2.</td>
</tr>
<tr>
<td>mc_cs_start_0</td>
<td>00_1005_11a0</td>
<td>79/141</td>
<td>Channel 0 CS[3:0] start address.</td>
</tr>
<tr>
<td>mc_cs_end_0</td>
<td>00_1005_11c0</td>
<td>80/141</td>
<td>Channel 0 CS[3:0] end+1 address.</td>
</tr>
<tr>
<td>mc_interleave_0</td>
<td>00_1005_11e0</td>
<td>81/142</td>
<td>Channel 0 interleaved CS position.</td>
</tr>
<tr>
<td>mc_cs0_row_0</td>
<td>00_1005_1200</td>
<td>82/142</td>
<td>Channel 0 CS0 row address bits.</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>mc_cs0_col_0</td>
<td>00_1005_1220</td>
<td>83/142</td>
<td>Channel 0 CS0 column address bits.</td>
</tr>
<tr>
<td>mc_cs0_ba_0</td>
<td>00_1005_1240</td>
<td>84/143</td>
<td>Channel 0 CS0 bank select.</td>
</tr>
<tr>
<td>mc_cs1_row_0</td>
<td>00_1005_1260</td>
<td>82/142</td>
<td>Channel 0 CS1 row address bits.</td>
</tr>
<tr>
<td>mc_cs1_col_0</td>
<td>00_1005_1280</td>
<td>83/142</td>
<td>Channel 0 CS1 column address bits.</td>
</tr>
<tr>
<td>mc_cs1_ba_0</td>
<td>00_1005_12a0</td>
<td>84/143</td>
<td>Channel 0 CS1 DRAM bank select.</td>
</tr>
<tr>
<td>mc_cs2_row_0</td>
<td>00_1005_12c0</td>
<td>82/142</td>
<td>Channel 0 CS2 row address bits.</td>
</tr>
<tr>
<td>mc_cs2_col_0</td>
<td>00_1005_12e0</td>
<td>83/142</td>
<td>Channel 0 CS2 column address bits.</td>
</tr>
<tr>
<td>mc_cs2_ba_0</td>
<td>00_1005_1300</td>
<td>84/143</td>
<td>Channel 0 CS2 DRAM bank select.</td>
</tr>
<tr>
<td>mc_cs3_row_0</td>
<td>00_1005_1320</td>
<td>82/142</td>
<td>Channel 0 CS3 row address bits.</td>
</tr>
<tr>
<td>mc_cs3_col_0</td>
<td>00_1005_1340</td>
<td>83/142</td>
<td>Channel 0 CS3 column address bits.</td>
</tr>
<tr>
<td>mc_cs3_ba_0</td>
<td>00_1005_1360</td>
<td>83/142</td>
<td>Channel 0 CS3 DRAM bank select.</td>
</tr>
<tr>
<td>mc_cs_attr_0</td>
<td>00_1005_1380</td>
<td>85/143</td>
<td>Channel 0 CS[3:0]attribute.</td>
</tr>
<tr>
<td>mc_test_data_0</td>
<td>00_1005_1400</td>
<td>86/144</td>
<td>Channel 0 ECC error set data bits.</td>
</tr>
<tr>
<td>mc_test_ecc_0</td>
<td>00_1005_1420</td>
<td>87/144</td>
<td>Channel 0 ECC error set ecc bits.</td>
</tr>
<tr>
<td>mc_clock_cfg_0</td>
<td>00_1005_1500</td>
<td>74/138</td>
<td>Channel 0 memory clock ratio.</td>
</tr>
<tr>
<td>mc__1</td>
<td>00_1005_2000</td>
<td>See_0 above</td>
<td>Memory controller channel 1 configuration (at memory controller 0 + 1000).</td>
</tr>
<tr>
<td>smb_serial_xtra_0</td>
<td>00_1006_0000</td>
<td>294/416</td>
<td>SMB port extra data register.</td>
</tr>
<tr>
<td>smb_serial_xtra_1</td>
<td>00_1006_0008</td>
<td>294/416</td>
<td>SMB port extra data register.</td>
</tr>
<tr>
<td>smb_serial_freq_0</td>
<td>00_1006_0010</td>
<td>289/415</td>
<td>Frequency for SMB port.</td>
</tr>
<tr>
<td>smb_serial_freq_1</td>
<td>00_1006_0018</td>
<td>289/415</td>
<td>Frequency for SMB port.</td>
</tr>
<tr>
<td>smb_serial_status_0</td>
<td>00_1006_0020</td>
<td>292/416</td>
<td>SMB port status (Read clears finish interrupt).</td>
</tr>
<tr>
<td>smb_serial_status_1</td>
<td>00_1006_0028</td>
<td>292/416</td>
<td>SMB port status (Read clears finish interrupt).</td>
</tr>
<tr>
<td>smb_serial_cmd_0</td>
<td>00_1006_0030</td>
<td>290/415</td>
<td>SMB port command.</td>
</tr>
<tr>
<td>smb_serial_cmd_1</td>
<td>00_1006_0038</td>
<td>290/415</td>
<td>SMB port command.</td>
</tr>
<tr>
<td>smb_serial_start_0</td>
<td>00_1006_0040</td>
<td>296/417</td>
<td>SMB port start register.</td>
</tr>
<tr>
<td>smb_serial_start_1</td>
<td>00_1006_0048</td>
<td>296/417</td>
<td>SMB port start register.</td>
</tr>
<tr>
<td>smb_serial_data_0</td>
<td>00_1006_0050</td>
<td>293/416</td>
<td>SMB port data register.</td>
</tr>
<tr>
<td>smb_serial_data_1</td>
<td>00_1006_0058</td>
<td>293/416</td>
<td>SMB port data register.</td>
</tr>
<tr>
<td>smb_serial_control_0</td>
<td>00_1006_0060</td>
<td>291/415</td>
<td>SMB port control register.</td>
</tr>
<tr>
<td>smb_serial_control_1</td>
<td>00_1006_0068</td>
<td>291/415</td>
<td>SMB port control register.</td>
</tr>
<tr>
<td>smb_serial_pce_0</td>
<td>00_1006_0070</td>
<td>295/417</td>
<td>SMB port pec register.</td>
</tr>
<tr>
<td>smb_serial_pce_1</td>
<td>00_1006_0078</td>
<td>295/417</td>
<td>SMB port pec register.</td>
</tr>
<tr>
<td>duart_mode_reg_1a</td>
<td>00_1006_0100</td>
<td>197/326</td>
<td>Mode register 1 port A MR1A.</td>
</tr>
<tr>
<td>duart_mode_reg_2a</td>
<td>00_1006_0110</td>
<td>198/326</td>
<td>Mode register 2 port A MR2A.</td>
</tr>
<tr>
<td>duart_status_a</td>
<td>00_1006_0120</td>
<td>200/327</td>
<td>Status register port A (Read Only).</td>
</tr>
<tr>
<td>duart_clk_sel_a</td>
<td>00_1006_0130</td>
<td>201/328</td>
<td>Clock select register port A.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>duart_full_ctl_a</td>
<td>00_1006_0140</td>
<td>202/328</td>
<td>Full control port A.</td>
</tr>
<tr>
<td>duart_cmd_a</td>
<td>00_1006_0150</td>
<td>199/327</td>
<td>Command register port A.</td>
</tr>
<tr>
<td>duart_rx_hold_reg_a</td>
<td>00_1006_0160</td>
<td>203/328</td>
<td>RX holding register port A (Read Only, read pops character from FIFO).</td>
</tr>
<tr>
<td>duart_tx_hold_reg_a</td>
<td>00_1006_0170</td>
<td>204/329</td>
<td>TX holding register port A (Write Only).</td>
</tr>
<tr>
<td>duart_opcr_a</td>
<td>00_1006_0180</td>
<td>211/331</td>
<td>Output port control register alias that only alters port A bits.</td>
</tr>
<tr>
<td>duart_aux_ctrl_a</td>
<td>00_1006_0190</td>
<td>213/331</td>
<td>Aux control register alias that only alters port A bits.</td>
</tr>
<tr>
<td>duart_mode_reg_1b</td>
<td>00_1006_0200</td>
<td>197/326</td>
<td>Mode register 1 port B.</td>
</tr>
<tr>
<td>duart_mode_reg_2b</td>
<td>00_1006_0210</td>
<td>198/326</td>
<td>Mode register 2 port B.</td>
</tr>
<tr>
<td>duart_status_b</td>
<td>00_1006_0220</td>
<td>200/327</td>
<td>Status register port B (Read Only).</td>
</tr>
<tr>
<td>duart_clk_sel_b</td>
<td>00_1006_0230</td>
<td>201/328</td>
<td>Clock select register port B.</td>
</tr>
<tr>
<td>duart_full_ctl_b</td>
<td>00_1006_0240</td>
<td>202/328</td>
<td>Full control port B.</td>
</tr>
<tr>
<td>duart_cmd_b</td>
<td>00_1006_0250</td>
<td>199/327</td>
<td>Command register port B.</td>
</tr>
<tr>
<td>duart_rx_hold_reg_b</td>
<td>00_1006_0260</td>
<td>203/328</td>
<td>RX holding register port B (Read Only, read pops character from FIFO).</td>
</tr>
<tr>
<td>duart_tx_hold_reg_b</td>
<td>00_1006_0270</td>
<td>204/329</td>
<td>TX holding register port B (Write Only).</td>
</tr>
<tr>
<td>duart_opcr_b</td>
<td>00_1006_0280</td>
<td>211/331</td>
<td>Output port control register alias that only alters port B bits.</td>
</tr>
<tr>
<td>duart_aux_ctrl_b</td>
<td>00_1006_0290</td>
<td>213/331</td>
<td>Aux control register alias that only alters port B bits.</td>
</tr>
<tr>
<td>duart_inport_chng</td>
<td>00_1006_0300</td>
<td>206/329</td>
<td>Input port change register (Read Only, read clears).</td>
</tr>
<tr>
<td>duart_aux_cntrl</td>
<td>00_1006_0310</td>
<td>212/331</td>
<td>Aux control register.</td>
</tr>
<tr>
<td>duart_isr_a</td>
<td>00_1006_0320</td>
<td>215/332</td>
<td>Interrupt status register port A (Read Only).</td>
</tr>
<tr>
<td>duart_imr_a</td>
<td>00_1006_0330</td>
<td>218/333</td>
<td>Interrupt mask register port A.</td>
</tr>
<tr>
<td>duart_isr_b</td>
<td>00_1006_0340</td>
<td>216/332</td>
<td>Interrupt status register port B (Read Only).</td>
</tr>
<tr>
<td>duart_imr_b</td>
<td>00_1006_0350</td>
<td>219/333</td>
<td>Interrupt mask register port B.</td>
</tr>
<tr>
<td>duart_out_port</td>
<td>00_1006_0360</td>
<td>222/334</td>
<td>Output port register (Write Only).</td>
</tr>
<tr>
<td>duart_opcr</td>
<td>00_1006_0370</td>
<td>210/330</td>
<td>Output port control.</td>
</tr>
<tr>
<td>duart_in_port</td>
<td>00_1006_0380</td>
<td>205/329</td>
<td>Input port state (Read Only).</td>
</tr>
<tr>
<td>duart_isr</td>
<td>00_1006_0390</td>
<td>214/332</td>
<td>Full interrupt status (Read Only).</td>
</tr>
<tr>
<td>duart_imr</td>
<td>00_1006_03a0</td>
<td>217/333</td>
<td>Full interrupt mask.</td>
</tr>
<tr>
<td>duart_set Opr</td>
<td>00_1006_03b0</td>
<td>220/334</td>
<td>Output port register bit set (Write Only).</td>
</tr>
<tr>
<td>duart_clear Opr</td>
<td>00_1006_03c0</td>
<td>221/334</td>
<td>Output port register bit clear (Write Only).</td>
</tr>
<tr>
<td>duart_inport_chng_a</td>
<td>00_1006_03d0</td>
<td>208/330</td>
<td>Input port change register for channel A (Read Only, read clears channel A change state)</td>
</tr>
<tr>
<td>duart_inport_chng_b</td>
<td>00_1006_03e0</td>
<td>209/330</td>
<td>Input port change register for channel B (Read Only, read clears channel B change state)</td>
</tr>
<tr>
<td>duart_inport_chng_debug</td>
<td>00_1006_03f0</td>
<td>207/330</td>
<td>Alias of input port change register with no read side effects. (Read Only)</td>
</tr>
<tr>
<td>dma_config0_ser_0_rx</td>
<td>00_1006_0400</td>
<td>91/163</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_0_rx</td>
<td>00_1006_0408</td>
<td>92/164</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>dma_dscr_base_ser_0_rx</td>
<td>00_1006_0410</td>
<td>93/166</td>
<td>Receive DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_cnt_ser_0_rx</td>
<td>00_1006_0418</td>
<td>95/166</td>
<td>Receive DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_0_rx</td>
<td>00_1006_0420</td>
<td>96/167</td>
<td>Receive DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_0_rx</td>
<td>00_1006_0428</td>
<td>97/167</td>
<td>Receive DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_0_rx</td>
<td>00_1006_0430</td>
<td>98/167</td>
<td>Receive DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>dma_config0_ser_0_tx</td>
<td>00_1006_0480</td>
<td>91/163</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_0_tx</td>
<td>00_1006_0488</td>
<td>92/164</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_0_tx</td>
<td>00_1006_0490</td>
<td>93/166</td>
<td>Transmit DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_cnt_ser_0_tx</td>
<td>00_1006_0498</td>
<td>95/166</td>
<td>Transmit DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_0_tx</td>
<td>00_1006_04A0</td>
<td>96/167</td>
<td>Transmit DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_0_tx</td>
<td>00_1006_04A8</td>
<td>97/167</td>
<td>Transmit DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_0_tx</td>
<td>00_1006_04B0</td>
<td>98/167</td>
<td>Transmit DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>ser_mode_0</td>
<td>00_1006_0500</td>
<td>231/353</td>
<td>Mode select.</td>
</tr>
<tr>
<td>ser_minfrm_sz_0</td>
<td>00_1006_0508</td>
<td>237/355</td>
<td>Min frame size.</td>
</tr>
<tr>
<td>ser_maxfrm_sz_0</td>
<td>00_1006_0510</td>
<td>238/356</td>
<td>Max frame size.</td>
</tr>
<tr>
<td>ser_addr_mask_0</td>
<td>00_1006_0518</td>
<td>243/358</td>
<td>Address mask.</td>
</tr>
<tr>
<td>ser_usr0_addr_0</td>
<td>00_1006_0520</td>
<td>244/358</td>
<td>Match address 0.</td>
</tr>
<tr>
<td>ser_usr1_addr_0</td>
<td>00_1006_0528</td>
<td>244/358</td>
<td>Match address 1.</td>
</tr>
<tr>
<td>ser_usr2_addr_0</td>
<td>00_1006_0530</td>
<td>244/358</td>
<td>Match address 2.</td>
</tr>
<tr>
<td>ser_usr3_addr_0</td>
<td>00_1006_0538</td>
<td>244/358</td>
<td>Match address 3.</td>
</tr>
<tr>
<td>ser_cmd_0</td>
<td>00_1006_0540</td>
<td>233/354</td>
<td>Command</td>
</tr>
<tr>
<td>ser_tx_rd_thrsh_0</td>
<td>00_1006_0560</td>
<td>235/355</td>
<td>Transmit FIFO read threshold.</td>
</tr>
<tr>
<td>ser_tx_wr_thrsh_0</td>
<td>00_1006_0568</td>
<td>234/355</td>
<td>Transmit FIFO write threshold.</td>
</tr>
<tr>
<td>ser_rx_rd_thrsh_0</td>
<td>00_1006_0570</td>
<td>236/355</td>
<td>Receive FIFO read threshold.</td>
</tr>
<tr>
<td>ser_line_mode_0</td>
<td>00_1006_0578</td>
<td>232/353</td>
<td>Line Interface configuration register.</td>
</tr>
<tr>
<td>ser_dma_enable_0</td>
<td>00_1006_0580</td>
<td>239/356</td>
<td>DMA channel enable register.</td>
</tr>
<tr>
<td>ser_status_0</td>
<td>00_1006_0588</td>
<td>240/356</td>
<td>Serial interface and DMA status (Read Only, read clears).</td>
</tr>
<tr>
<td>ser_int_mask_0</td>
<td>00_1006_0590</td>
<td>242/357</td>
<td>Interrupt mask.</td>
</tr>
<tr>
<td>dma_asic_addr_ser_0</td>
<td>00_1006_0598</td>
<td>94/166</td>
<td>ASIC mode address.</td>
</tr>
<tr>
<td>ser_debug_status_0</td>
<td>00_1006_05A0</td>
<td>241/357</td>
<td>Serial interface and DMA status (Read Only, no side effects).</td>
</tr>
<tr>
<td>ser_tx_byte_lo_0</td>
<td>00_1006_05C0</td>
<td>246/359</td>
<td>Serial interface transmit byte count (low 16 bits).</td>
</tr>
<tr>
<td>ser_tx_byte_hi_0</td>
<td>00_1006_05C8</td>
<td>246/359</td>
<td>Serial interface transmit byte count (high 16 bits).</td>
</tr>
<tr>
<td>ser_rx_byte_lo_0</td>
<td>00_1006_05D0</td>
<td>246/359</td>
<td>Serial interface receive byte count (low 16 bits).</td>
</tr>
<tr>
<td>ser_rx_byte_hi_0</td>
<td>00_1006_05D8</td>
<td>246/359</td>
<td>Serial interface receive byte count (high 16 bits).</td>
</tr>
<tr>
<td>ser_tx_underrun_0</td>
<td>00_1006_05E0</td>
<td>246/359</td>
<td>Serial interface transmit underrun count.</td>
</tr>
<tr>
<td>ser_rx_overflow_0</td>
<td>00_1006_05EE</td>
<td>246/359</td>
<td>Serial interface receive overflow count.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ser_rx_errors_0</td>
<td>00_1006_05F0</td>
<td>246/359</td>
<td>Serial interface receive error packet count.</td>
</tr>
<tr>
<td>ser_rx_badaddr_0</td>
<td>00_1006_05F8</td>
<td>246/359</td>
<td>Serial interface receive address mismatch count.</td>
</tr>
<tr>
<td>ser_rx_table0_0</td>
<td>00_1006_0600</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table1_0</td>
<td>00_1006_0608</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table2_0</td>
<td>00_1006_0610</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table3_0</td>
<td>00_1006_0618</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table4_0</td>
<td>00_1006_0620</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table5_0</td>
<td>00_1006_0628</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table6_0</td>
<td>00_1006_0630</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table7_0</td>
<td>00_1006_0638</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table8_0</td>
<td>00_1006_0640</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table9_0</td>
<td>00_1006_0648</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table10_0</td>
<td>00_1006_0650</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table11_0</td>
<td>00_1006_0658</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table12_0</td>
<td>00_1006_0660</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table13_0</td>
<td>00_1006_0668</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table14_0</td>
<td>00_1006_0670</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table15_0</td>
<td>00_1006_0678</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table0_0</td>
<td>00_1006_0700</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table1_0</td>
<td>00_1006_0708</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table2_0</td>
<td>00_1006_0710</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table3_0</td>
<td>00_1006_0718</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table4_0</td>
<td>00_1006_0720</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table5_0</td>
<td>00_1006_0728</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table6_0</td>
<td>00_1006_0730</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table7_0</td>
<td>00_1006_0738</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table8_0</td>
<td>00_1006_0740</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table9_0</td>
<td>00_1006_0748</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table10_0</td>
<td>00_1006_0750</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table11_0</td>
<td>00_1006_0758</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table12_0</td>
<td>00_1006_0760</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table13_0</td>
<td>00_1006_0768</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table14_0</td>
<td>00_1006_0770</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table15_0</td>
<td>00_1006_0778</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>dma_config0_ser_1_rx</td>
<td>00_1006_0800</td>
<td>91/163</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_1_rx</td>
<td>00_1006_0808</td>
<td>92/164</td>
<td>Receive DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_1_rx</td>
<td>00_1006_0810</td>
<td>93/166</td>
<td>Receive DMA descriptor base address.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dma_dscr_count_ser_1_rx</td>
<td>00_1006_0818</td>
<td>95/166</td>
<td>Receive DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_1_rx</td>
<td>00_1006_0820</td>
<td>96/167</td>
<td>Receive DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_1_rx</td>
<td>00_1006_0828</td>
<td>97/167</td>
<td>Receive DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_1_rx</td>
<td>00_1006_0830</td>
<td>98/167</td>
<td>Receive DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>dma_config0_ser_1_tx</td>
<td>00_1006_0880</td>
<td>91/163</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_config1_ser_1_tx</td>
<td>00_1006_0888</td>
<td>92/164</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>dma_dscr_base_ser_1_tx</td>
<td>00_1006_0890</td>
<td>93/166</td>
<td>Transmit DMA descriptor base address.</td>
</tr>
<tr>
<td>dma_dscr_count_ser_1_tx</td>
<td>00_1006_0898</td>
<td>95/166</td>
<td>Transmit DMA descriptor count.</td>
</tr>
<tr>
<td>dma_cur_dscr_a_ser_1_tx</td>
<td>00_1006_08A0</td>
<td>96/167</td>
<td>Transmit DMA current descriptor A (Read Only).</td>
</tr>
<tr>
<td>dma_cur_dscr_b_ser_1_tx</td>
<td>00_1006_08A8</td>
<td>97/167</td>
<td>Transmit DMA current descriptor B (Read Only).</td>
</tr>
<tr>
<td>dma_cur_daddr_ser_1_tx</td>
<td>00_1006_08B0</td>
<td>98/167</td>
<td>Transmit DMA current descriptor address (Read Only).</td>
</tr>
<tr>
<td>ser_mode_1</td>
<td>00_1006_0900</td>
<td>231/353</td>
<td>Mode select.</td>
</tr>
<tr>
<td>ser_minfrm_sz_1</td>
<td>00_1006_0908</td>
<td>237/355</td>
<td>Min frame size.</td>
</tr>
<tr>
<td>ser_maxfrm_sz_1</td>
<td>00_1006_0910</td>
<td>238/356</td>
<td>Max frame size.</td>
</tr>
<tr>
<td>ser_addr_mask_1</td>
<td>00_1006_0918</td>
<td>243/358</td>
<td>Address mask.</td>
</tr>
<tr>
<td>ser_usr0_addr_1</td>
<td>00_1006_0920</td>
<td>244/358</td>
<td>Match address 0.</td>
</tr>
<tr>
<td>ser_usr1_addr_1</td>
<td>00_1006_0928</td>
<td>244/358</td>
<td>Match address 1.</td>
</tr>
<tr>
<td>ser_usr2_addr_1</td>
<td>00_1006_0930</td>
<td>244/358</td>
<td>Match address 2.</td>
</tr>
<tr>
<td>ser_usr3_addr_1</td>
<td>00_1006_0938</td>
<td>244/358</td>
<td>Match address 3.</td>
</tr>
<tr>
<td>ser_cmd_1</td>
<td>00_1006_0940</td>
<td>233/354</td>
<td>Command</td>
</tr>
<tr>
<td>ser_rx_underrun_1</td>
<td>00_1006_09E0</td>
<td>246/359</td>
<td>Serial interface transmit underrun count.</td>
</tr>
<tr>
<td>ser_rx_overflow_1</td>
<td>00_1006_09E8</td>
<td>246/359</td>
<td>Serial interface receive overflow count.</td>
</tr>
<tr>
<td>ser_rx_errors_1</td>
<td>00_1006_09F0</td>
<td>246/359</td>
<td>Serial interface receive error packet count.</td>
</tr>
</tbody>
</table>

**Note:** The descriptions provided are the official descriptions as stated in the Broadcom Corporation User Manual. The table entries are organized by address and include the name of the register, the address, and a brief description of the register's function or purpose.
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ser_rx_badaddr_1</td>
<td>00_1006_09F8</td>
<td>246/359</td>
<td>Serial interface receive address mismatch count.</td>
</tr>
<tr>
<td>ser_rx_table0_1</td>
<td>00_1006_0A00</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table1_1</td>
<td>00_1006_0A08</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table2_1</td>
<td>00_1006_0A10</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table3_1</td>
<td>00_1006_0A18</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table4_1</td>
<td>00_1006_0A20</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table5_1</td>
<td>00_1006_0A28</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table6_1</td>
<td>00_1006_0A30</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table7_1</td>
<td>00_1006_0A38</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table8_1</td>
<td>00_1006_0A40</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table9_1</td>
<td>00_1006_0A48</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table10_1</td>
<td>00_1006_0A50</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table11_1</td>
<td>00_1006_0A58</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table12_1</td>
<td>00_1006_0A60</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table13_1</td>
<td>00_1006_0A68</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table14_1</td>
<td>00_1006_0A70</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_rx_table15_1</td>
<td>00_1006_0A78</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table0_1</td>
<td>00_1006_0B00</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table1_1</td>
<td>00_1006_0B08</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table2_1</td>
<td>00_1006_0B10</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table3_1</td>
<td>00_1006_0B18</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table4_1</td>
<td>00_1006_0B20</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table5_1</td>
<td>00_1006_0B28</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table6_1</td>
<td>00_1006_0B30</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table7_1</td>
<td>00_1006_0B38</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table8_1</td>
<td>00_1006_0B40</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table9_1</td>
<td>00_1006_0B48</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table10_1</td>
<td>00_1006_0B50</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table11_1</td>
<td>00_1006_0B58</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table12_1</td>
<td>00_1006_0B60</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table13_1</td>
<td>00_1006_0B68</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table14_1</td>
<td>00_1006_0B70</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>ser_tx_table15_1</td>
<td>00_1006_0B78</td>
<td>245/358</td>
<td>Sequence table.</td>
</tr>
<tr>
<td>io_ext_cfg_0</td>
<td>00_1006_1000</td>
<td>251/374</td>
<td>cs0 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_1</td>
<td>00_1006_1008</td>
<td>251/374</td>
<td>cs1 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_2</td>
<td>00_1006_1010</td>
<td>251/374</td>
<td>cs2 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_3</td>
<td>00_1006_1018</td>
<td>251/374</td>
<td>cs3 interface configuration.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_ext_cfg_4</td>
<td>00_1006_1020</td>
<td>251/374</td>
<td>254/374 cs4 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_5</td>
<td>00_1006_1028</td>
<td>251/374</td>
<td>254/374 cs5 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_6</td>
<td>00_1006_1030</td>
<td>251/374</td>
<td>254/374 cs6 interface configuration.</td>
</tr>
<tr>
<td>io_ext_cfg_7</td>
<td>00_1006_1038</td>
<td>251/374</td>
<td>254/374 cs7 interface configuration.</td>
</tr>
<tr>
<td>io_ext_mult_size_0</td>
<td>00_1006_1100</td>
<td>252/374</td>
<td>254/374 cs0 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_1</td>
<td>00_1006_1108</td>
<td>252/374</td>
<td>254/374 cs1 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_2</td>
<td>00_1006_1110</td>
<td>252/374</td>
<td>254/374 cs2 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_3</td>
<td>00_1006_1118</td>
<td>252/374</td>
<td>254/374 cs3 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_4</td>
<td>00_1006_1120</td>
<td>252/374</td>
<td>254/374 cs4 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_5</td>
<td>00_1006_1128</td>
<td>252/374</td>
<td>254/374 cs5 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_6</td>
<td>00_1006_1130</td>
<td>252/374</td>
<td>254/374 cs6 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_mult_size_7</td>
<td>00_1006_1138</td>
<td>252/374</td>
<td>254/374 cs7 region size in 64KB units.</td>
</tr>
<tr>
<td>io_ext_start_addr_0</td>
<td>00_1006_1200</td>
<td>253/375</td>
<td>254/375 cs0 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_1</td>
<td>00_1006_1208</td>
<td>253/375</td>
<td>254/375 cs1 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_2</td>
<td>00_1006_1210</td>
<td>253/375</td>
<td>254/375 cs2 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_3</td>
<td>00_1006_1218</td>
<td>253/375</td>
<td>254/375 cs3 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_4</td>
<td>00_1006_1220</td>
<td>253/375</td>
<td>254/375 cs4 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_5</td>
<td>00_1006_1228</td>
<td>253/375</td>
<td>254/375 cs5 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_6</td>
<td>00_1006_1230</td>
<td>253/375</td>
<td>254/375 cs6 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_ext_start_addr_7</td>
<td>00_1006_1238</td>
<td>253/375</td>
<td>254/375 cs7 region start address (put outside generic bus range to disable).</td>
</tr>
<tr>
<td>io_drive_0</td>
<td>00_1006_1300</td>
<td>264/378</td>
<td>254/375 3.3V output drive strength and slew rate control registers. See Section: &quot;Drive Strength Control&quot; on page 373.</td>
</tr>
<tr>
<td>io_drive_1</td>
<td>00_1006_1308</td>
<td>265/379</td>
<td>254/375 3.3V output drive strength and slew rate control register.</td>
</tr>
<tr>
<td>io_drive_2</td>
<td>00_1006_1310</td>
<td>266/380</td>
<td>254/375 3.3V output drive strength and slew rate control register.</td>
</tr>
<tr>
<td>io_drive_3</td>
<td>00_1006_1318</td>
<td>267/380</td>
<td>254/375 3.3V output drive strength and slew rate control register.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_0</td>
<td>00_1006_1600</td>
<td>254/375</td>
<td>254/375 cs0 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_1</td>
<td>00_1006_1608</td>
<td>254/375</td>
<td>254/375 cs1 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_2</td>
<td>00_1006_1610</td>
<td>254/375</td>
<td>254/375 cs2 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_3</td>
<td>00_1006_1618</td>
<td>254/375</td>
<td>254/375 cs3 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_4</td>
<td>00_1006_1620</td>
<td>254/375</td>
<td>254/375 cs4 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_5</td>
<td>00_1006_1628</td>
<td>254/375</td>
<td>254/375 cs5 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg0_6</td>
<td>00_1006_1630</td>
<td>254/375</td>
<td>254/375 cs6 Timing parameter configuration 0.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_ext_time_cfg0_7</td>
<td>00_1006_1638</td>
<td>254/375</td>
<td>cs7 Timing parameter configuration 0.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_0</td>
<td>00_1006_1700</td>
<td>255/376</td>
<td>cs0 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_1</td>
<td>00_1006_1708</td>
<td>255/376</td>
<td>cs1 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_2</td>
<td>00_1006_1710</td>
<td>255/376</td>
<td>cs2 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_3</td>
<td>00_1006_1718</td>
<td>255/376</td>
<td>cs3 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_4</td>
<td>00_1006_1720</td>
<td>255/376</td>
<td>cs4 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_5</td>
<td>00_1006_1728</td>
<td>255/376</td>
<td>cs5 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_6</td>
<td>00_1006_1730</td>
<td>255/376</td>
<td>cs6 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_ext_time_cfg1_7</td>
<td>00_1006_1738</td>
<td>255/376</td>
<td>cs7 Timing parameter configuration 1.</td>
</tr>
<tr>
<td>io_interrupt_status</td>
<td>00_1006_1a00</td>
<td>256/376</td>
<td>Interrupt status for generic bus sources (Read Only, read clears).</td>
</tr>
<tr>
<td>io_interrupt_data0</td>
<td>00_1006_1a10</td>
<td>257/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data1</td>
<td>00_1006_1a18</td>
<td>258/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data2</td>
<td>00_1006_1a20</td>
<td>259/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_data3</td>
<td>00_1006_1a28</td>
<td>260/377</td>
<td>Data latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_addr0</td>
<td>00_1006_1a30</td>
<td>261/378</td>
<td>Address latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>io_interrupt_addr1</td>
<td>00_1006_1a40</td>
<td>262/378</td>
<td>Address latched on generic interrupt assertion (Read Only).</td>
</tr>
<tr>
<td>gpio_clk_edge</td>
<td>00_1006_1a80</td>
<td>276/398</td>
<td>Write a 1 to clear the edge detector for that bit (Write Only).</td>
</tr>
<tr>
<td>gpio_int_type</td>
<td>00_1006_1a88</td>
<td>277/398</td>
<td>Select interrupt type for pairs of pins.</td>
</tr>
<tr>
<td>gpio_input_invert</td>
<td>00_1006_1a90</td>
<td>279/398</td>
<td>Set to invert the input.</td>
</tr>
<tr>
<td>gpio_glitch</td>
<td>00_1006_1a98</td>
<td>280/399</td>
<td>Set for 1ms glitch filter, clear for 60 ns.</td>
</tr>
<tr>
<td>gpio_read</td>
<td>00_1006_1aa0</td>
<td>278/398</td>
<td>Shows current value of signal after possible invert and glitch (Read Only).</td>
</tr>
<tr>
<td>gpio_direction</td>
<td>00_1006_1aab</td>
<td>281/399</td>
<td>GPIO pin data direction. 1 for output, 0 for input.</td>
</tr>
<tr>
<td>gpio_pin_clr</td>
<td>00_1006_1ab0</td>
<td>282/399</td>
<td>Write 1 to set that bit to zero (Write Only).</td>
</tr>
<tr>
<td>gpio_pin_set</td>
<td>00_1006_1ab8</td>
<td>283/399</td>
<td>Write 1 to set that bit to one (Write Only).</td>
</tr>
<tr>
<td>mac_tx_byte_0</td>
<td>00_1006_4000</td>
<td>167/288</td>
<td>RMON transmit byte counter.</td>
</tr>
<tr>
<td>mac_collisions_0</td>
<td>00_1006_4008</td>
<td>167/288</td>
<td>RMON total collisions.</td>
</tr>
<tr>
<td>mac_late_col_0</td>
<td>00_1006_4010</td>
<td>167/288</td>
<td>RMON late collisions.</td>
</tr>
<tr>
<td>mac_ex_col_0</td>
<td>00_1006_4018</td>
<td>167/288</td>
<td>RMON excessive collisions.</td>
</tr>
<tr>
<td>mac_fcs_error_0</td>
<td>00_1006_4020</td>
<td>167/288</td>
<td>RMON packets Tx with bad FCS.</td>
</tr>
<tr>
<td>mac_tx_abort_0</td>
<td>00_1006_4028</td>
<td>167/288</td>
<td>RMON transmit packets aborted.</td>
</tr>
<tr>
<td>mac_tx_bad_0</td>
<td>00_1006_4038</td>
<td>167/288</td>
<td>RMON total of bad Tx packets.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_tx_good_0</td>
<td>00_1006_4040</td>
<td>167/288</td>
<td>RMON total successfully transmitted packets.</td>
</tr>
<tr>
<td>mac_tx_runt_0</td>
<td>00_1006_4048</td>
<td>167/288</td>
<td>RMON total runt packets transmitted.</td>
</tr>
<tr>
<td>mac_tx_oversize_0</td>
<td>00_1006_4050</td>
<td>167/288</td>
<td>RMON total oversize packets transmitted.</td>
</tr>
<tr>
<td>mac_rx_bytes_0</td>
<td>00_1006_4080</td>
<td>167/288</td>
<td>RMON total received bytes.</td>
</tr>
<tr>
<td>mac_rx_mcast_0</td>
<td>00_1006_4088</td>
<td>167/288</td>
<td>RMON total received multicast packets.</td>
</tr>
<tr>
<td>mac_rx_bcast_0</td>
<td>00_1006_4090</td>
<td>167/288</td>
<td>RMON total received broadcast packets.</td>
</tr>
<tr>
<td>mac_rx_bad_0</td>
<td>00_1006_4098</td>
<td>167/288</td>
<td>RMON total received bad packets.</td>
</tr>
<tr>
<td>mac_rx_good_0</td>
<td>00_1006_40A0</td>
<td>167/288</td>
<td>RMON total received good packets.</td>
</tr>
<tr>
<td>mac_rx_runt_0</td>
<td>00_1006_40A8</td>
<td>167/288</td>
<td>RMON total runt packets received.</td>
</tr>
<tr>
<td>mac_rx_oversize_0</td>
<td>00_1006_40B0</td>
<td>167/288</td>
<td>RMON total oversize packets received.</td>
</tr>
<tr>
<td>mac_rx_fcs_error_0</td>
<td>00_1006_40B8</td>
<td>167/288</td>
<td>RMON total packets received with bad FCS.</td>
</tr>
<tr>
<td>mac_rx_length_error_0</td>
<td>00_1006_40C0</td>
<td>167/288</td>
<td>RMON total packets received with length error.</td>
</tr>
<tr>
<td>mac_rx_code_error_0</td>
<td>00_1006_40C8</td>
<td>167/288</td>
<td>RMON total packets received with code error.</td>
</tr>
<tr>
<td>mac_rx_align_error_0</td>
<td>00_1006_40D0</td>
<td>167/288</td>
<td>RMON total packets received with align error.</td>
</tr>
<tr>
<td>mac_cfg_0</td>
<td>00_1006_4100</td>
<td>176/301</td>
<td>Ethernet interface configuration register.</td>
</tr>
<tr>
<td>mac_thrsh_cfg_0</td>
<td>00_1006_4108</td>
<td>179/306</td>
<td>Ethernet interface FIFO threshold configuration register.</td>
</tr>
<tr>
<td>mac_vlantag_0</td>
<td>00_1006_4110</td>
<td>181/309</td>
<td>VLAN tag for insertion into packets on transmit.</td>
</tr>
<tr>
<td>mac_frame_cfg_0</td>
<td>00_1006_4118</td>
<td>180/307</td>
<td>Ethernet interface MAC frame configuration.</td>
</tr>
<tr>
<td>mac_rx_fifo_ptrs_0</td>
<td>00_1006_4120</td>
<td>186/313</td>
<td>MAC FIFO pointers (Debug, Read Only).</td>
</tr>
<tr>
<td>mac_tx_fifo_ptrs_0</td>
<td>00_1006_4128</td>
<td>186/313</td>
<td>MAC FIFO pointers. (Debug, Read Only.)</td>
</tr>
<tr>
<td>mac_adfilter_cfg_0</td>
<td>00_1006_4200</td>
<td>192/315</td>
<td>MAC receive address filter configuration register.</td>
</tr>
<tr>
<td>mac_ethernet_addr_0</td>
<td>00_1006_4208</td>
<td>190/314</td>
<td>MAC source ethernet address for insertion during transmission.</td>
</tr>
<tr>
<td>mac_type_cfg_0</td>
<td>00_1006_4210</td>
<td>191/315</td>
<td>MAC packet type table for receive packet comparisons.</td>
</tr>
<tr>
<td>mac_admask0_0</td>
<td>00_1006_4218</td>
<td>188/314</td>
<td>Address filter exact match mask register (PERIPH_REV3).</td>
</tr>
<tr>
<td>mac_admask1_0</td>
<td>00_1006_4220</td>
<td>188/314</td>
<td>Address filter exact match mask register (PERIPH_REV3).</td>
</tr>
<tr>
<td>mac_hash0_0</td>
<td>00_1006_4240</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash1_0</td>
<td>00_1006_4248</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash2_0</td>
<td>00_1006_4250</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash3_0</td>
<td>00_1006_4258</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash4_0</td>
<td>00_1006_4260</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash5_0</td>
<td>00_1006_4268</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash6_0</td>
<td>00_1006_4270</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_hash7_0</td>
<td>00_1006_4278</td>
<td>189/314</td>
<td>Address filter hash map.</td>
</tr>
<tr>
<td>mac_addr0_0</td>
<td>00_1006_4280</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr1_0</td>
<td>00_1006_4288</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
</tbody>
</table>
### Table 312: Internal Registers Ordered by Address (Cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Table/Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mac_addr2_0</td>
<td>00_1006_4290</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr3_0</td>
<td>00_1006_4298</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr4_0</td>
<td>00_1006_42A0</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr5_0</td>
<td>00_1006_42A8</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr6_0</td>
<td>00_1006_42B0</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_addr7_0</td>
<td>00_1006_42B8</td>
<td>187/313</td>
<td>Address filter exact match.</td>
</tr>
<tr>
<td>mac_chup0_0</td>
<td>00_1006_4300</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup1_0</td>
<td>00_1006_4308</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup2_0</td>
<td>00_1006_4310</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chup3_0</td>
<td>00_1006_4318</td>
<td>193/317</td>
<td>Receive DMA channel select MSB.</td>
</tr>
<tr>
<td>mac_chlo0_0</td>
<td>00_1006_4320</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo1_0</td>
<td>00_1006_4328</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo2_0</td>
<td>00_1006_4330</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_chlo3_0</td>
<td>00_1006_4338</td>
<td>193/317</td>
<td>Receive DMA channel select LSB.</td>
</tr>
<tr>
<td>mac_enable_0</td>
<td>00_1006_4400</td>
<td>177/305</td>
<td>MAC enable register.</td>
</tr>
<tr>
<td>mac_status_0</td>
<td>00_1006_4408</td>
<td>182/309</td>
<td>MAC status/error register (Read Only, read clears).</td>
</tr>
<tr>
<td>mac_int_mask_0</td>
<td>00_1006_4410</td>
<td>185/313</td>
<td>MAC interrupt mask register.</td>
</tr>
<tr>
<td>dma_asic_addr_mac_0</td>
<td>00_1006_4418</td>
<td>94/166</td>
<td>ASIC mode base address.</td>
</tr>
<tr>
<td>mac_txd_ctl_0</td>
<td>00_1006_4420</td>
<td>178/305</td>
<td>Transmit DMA control register.</td>
</tr>
<tr>
<td>mac_mdio_0</td>
<td>00_1006_4428</td>
<td>194/317</td>
<td>MDIO pin control register.</td>
</tr>
<tr>
<td>mac_status1_0</td>
<td>00_1006_4430</td>
<td>183/312</td>
<td>MAC status/error register (Read Only, read clears ch 1).</td>
</tr>
<tr>
<td>mac_debug_status_0</td>
<td>00_1006_4448</td>
<td>184/312</td>
<td>MAC status/error register (Debug, Read Only, no side effects).</td>
</tr>
<tr>
<td>dma_config0_mac_0_rx_ch_0</td>
<td>00_1006_4800</td>
<td>91/163</td>
<td>DMA config 0 register.</td>
</tr>
<tr>
<td>dma_config1_mac_0_rx_ch_0</td>
<td>00_1006_4808</td>
<td>92/164</td>
<td>DMA config 1 register.</td>
</tr>
<tr>
<td>dma_dscr_base_mac_0_rx_ch_0</td>
<td>00_1006_4810</td>
<td>93/166</td>
<td>DMA descriptor base register.</td>
</tr>
<tr>
<td>dma_dscr_cnt_0_rx_ch_0</td>
<td>00_1006_4818</td>
<td>95/166</td>
<td>DMA descriptor count.</td>
</tr>
<tr>
<td>dma_dscr_a_mac_0_rx_ch_0</td>
<td>00_1006_4820</td>
<td>96/167</td>
<td>DMA current descriptor A.</td>
</tr>
<tr>
<td>dma_dscr_b_mac_0_rx_ch_0</td>
<td>00_1006_4828</td>
<td>97/167</td>
<td>DMA current descriptor B.</td>
</tr>
<tr>
<td>dma_cur_dscr_addr_mac_0_rx_ch_0</td>
<td>00_1006_4830</td>
<td>98/167</td>
<td>DMA current descriptor address.</td>
</tr>
<tr>
<td>dma_oddpktlost_mac_0_rx_ch_0</td>
<td>00_1006_4838</td>
<td>99/168</td>
<td>DMA packet lost counter. (PERIPH_REV3)</td>
</tr>
<tr>
<td>dma_mac_0_rx_ch_1</td>
<td>00_1006_4900</td>
<td>100.0</td>
<td>Rx channel 1 at Rx channel 0 + 100.</td>
</tr>
<tr>
<td>dma_mac_0_tx_ch_0</td>
<td>00_1006_4C00</td>
<td>400.0</td>
<td>Tx channel 0 at Rx channel 0 + 400.</td>
</tr>
<tr>
<td>dma_mac_0_tx_ch_1</td>
<td>00_1006_4D00</td>
<td>500.0</td>
<td>Tx channel 1 at Rx channel 0 + 500.</td>
</tr>
<tr>
<td>mac_1</td>
<td>00_1006_5000</td>
<td></td>
<td>MAC 1 registers (+1000 from mac_0 registers).</td>
</tr>
<tr>
<td>mac_2</td>
<td>00_1006_6000</td>
<td></td>
<td>MAC 2 registers (+2000 from mac_0 registers).</td>
</tr>
<tr>
<td>Name</td>
<td>Address</td>
<td>Table/Page</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>Type 00 PCI header</td>
<td>00_FE00_0000 127/236</td>
<td>Configuration space bus=0, dev=0.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00_FE00_00FF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type 01 PCI header</td>
<td>00_FE00_0800 140/244</td>
<td>Configuration space bus=0, dev=1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>00_FE00_08FF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
INDEX

Numerics
16-Bit GMII Style Packet FIFO 299

A
A Comment on the term Bank 103
A_CMD 22, 78
A_L1CA 20, 23, 69, 78, 439
A_L2CA 23, 26, 77, 90, 93, 164, 206, 212, 224, 230, 242, 439
Accesses from the BCM1250 to the PCI or HyperTransport 214
Accesses from the HyperTransport to the BCM1250 216
Accesses from the PCI to the BCM1250 217
Accessing the BCM1250 from an BCM1250 on a Double Hosted Chain 212
Accessing the BCM1250 from HyperTransport Devices 210
Accessing the BCM1250 from PCI Devices 205
Acknowledgement Read Access 369
Acknowledgement Write Access 370
addr_trap_cfg 67, 69
addr_trap_down 67, 69
addr_trap_index 67, 68
addr_trap_reg 53, 67, 68
addr_trap_reg_debug 68
addr_trap_up 67, 68
Address Trapping 67
alias_mailbox_cpu 46
alias_mbox_set_cpu 46
ASIC mode 159
Asynchronous Mode 322
Audience 3

B
Bank 103
Bank Address 117
baud rate 322
Baud Rate Generators 322
BCM1125 1, 3, 4, 120, 136, 190, 194, 196, 264, 292, 425, 430
BCM1250 Block Diagram 2, 3
Big Endian System
- Match Bit Lanes 203
- Match Byte Lanes 202
Block 5
Broadcom Use Only 6
Bus Error 53, 184, 211, 249
Bus error 77, 136, 137
Bus Error Exceptions 18
Bus Width 365
bus_err_data 65
bus_err_status 17, 18, 53, 65
bus_err_status_debug 65
bus_io_error 364
bus_i2_errors 17, 66
bus_mem_io_errors 17, 18, 66
BusErr-DPA 18
Cache Block 5
cache error 17
Cache Error Exceptions 17
Cache Line 5
Cache Management Access 94
cacheability attribute 68, 69, 89
CacheErr-D 17
CacheErr-DPA 17
CacheErr-i 17
CAS time check policy 122
Cause 47
CF+ cards 390
Channel Select 109
checksum 178
Checksum Generation 178
Chip Select 110
Choosing Interleave Parameters 120
Clock Ratios and Clocking Scheme 107
Clock, Reset and Test 8
Closed Page policy 122
COLDRES_L 26, 41, 260
Column Address 117
Comments on Using the L2 as Memory 93
CompactFlash 390
Configuration Header Descriptions 236
Configuration Resistors 27, 41
Configuration Space 194
Connecting A PCMCIA Slot 384
CPU Speculative Execution 16
CPU to CPU Communication 17, 19
CRC 178, 180, 269
CRC Generation 179
crc_def 179, 185
CRC32 (Ethernet) 180
CRC32C (iSCSI) 180
CRC-CCITT (HDLC) 180
ctcp_def 186
CTS 322, 323, 324
cache error 17
Cache Error Exceptions 17
Cache Line 5
Cache Management Access 94
cacheability attribute 68, 69, 89
CacheErr-D 17
CacheErr-DPA 17
CacheErr-i 17
CAS time check policy 122
Cause 47
CF+ cards 390
Channel Select 109
checksum 178
Checksum Generation 178
Chip Select 110
Choosing Interleave Parameters 120
Clock Ratios and Clocking Scheme 107
Clock, Reset and Test 8
Closed Page policy 122
COLDRES_L 26, 41, 260
Column Address 117
Comments on Using the L2 as Memory 93
CompactFlash 390
Configuration Header Descriptions 236
Configuration Resistors 27, 41
Configuration Space 194
Connecting A PCMCIA Slot 384
CPU Speculative Execution 16
CPU to CPU Communication 17, 19
CRC 178, 180, 269
CRC Generation 179
crc_def 179, 185
CRC32 (Ethernet) 180
CRC32C (iSCSI) 180
CRC-CCITT (HDLC) 180
ctcp_def 186
CTS 322, 323, 324

D
D_CODE 24, 65, 77
D_ID 77
D_MOD 25, 77
D_RSP 77
data buffers 147
Data Mover Operation 176
Data Phase 24
DDR FCRAM 123
DDR SDRAMS 123
DEBUG_L 71, 72, 74, 76
descriptors 147
Destination Address Filtering 278
Device Mode 190
DINT 47, 48, 50
Direct Connection of a Memory Only Card 385
DLL 128, 138
dm_cur_dscr_addr 185
dm_debug_dscr_base 184
dm_dscr_base 177, 184
dm_dscr_base_0 54
dm_dscr_base_1  54
dm_dscr_base_2  54
dm_dscr_base_3  54
dm_dscr_count  185
dm_partial  186
DMA
  ASIC mode  159
  buffer  148
  buffers  147
  bytes before offset overwritten  155
  Coherence  155
  completion interrupts  158
  descriptor  149
  descriptor chain  153
  descriptor interrupts  158
  descriptor ring  151
  descriptors  147,  151,  154,  156
  disable  152
  Ethernet priority  156
  L2 allocate  155
  packet  149
  pause  152
  scatter/gather  155
  watermarks  157
DMA Coherence and Cache Options  155
DMA Configuration  351
DMA Configurations  156
DMA Controllers  147
dma_asic_addr  159,  160,  166
dma_conf0  148,  156,  157,  163
dma_conf1  152,  156,  159,  164,  285
dma_cur_daddr  167
dma_cur_dscr_a  167
dma_cur_dscr_addr  153
dma_cur_dscr_b  167
dma_dscr_base  166
dma_dscr_cnt  166
dma_oodpktlost  158,  168,  283
DRAM  103
dscr_a  149,  149,  154,  160,  169,  170
dscr_b  149,  149,  154,  169,  170
DUART  322
duart_aux_ctrl  332
duart_clear_opr  324,  335
duart_clk_sel  323,  329
duart_cmd  323,  328
duart_full_ctl  326,  329
duart_imr  325,  334
duart_imr_a  325
duart_imr_b  325
duart_in_port  324,  330
duart_inport_chg  324,  330
duart_inport_chg_a  331
duart_inport_chg_b  331
duart_inport_chg_debug  331
duart_isr  325,  333,  334
duart_isr_a  325
duart_isr_b  325
duart_mode_reg_1  323,  324,  326,  327
duart_mode_reg_2  323,  324,  327
duart_opcr  322,  324,  331,  340,  342
duart_out_port  324,  335
duart_rx_hold  326,  329
duart_set_opr  324,  335
duart_status  323,  328
duart_tx_hold  323,  330
ECC  125
ECC Diagnostic Management Accesses (ECC diag bits nonzero)  98
ECC error  64,  66,  89,  91,  94,  96,  97,  104,  125
Endian policy
  as an optimization  204
  Little endian system  201
  match bit lanes  203
  match byte lanes  202
EPC  17
ErrCtl  17,  18
Ethernet  180
  alignment error  276,  291
  Code Error  276
  Code error  291
  Collision  273
  collisions  289,  302
  CRC  269
  CRC Error  276
  CRC error  274,  291
  Dribble Error  276
  dribble error  291
  Excessive Collision  273
  excessive collisions  289,  302
  filter  276
  flow control  284
  frame format  268
  inter-frame gap  308
  interrupts  286
  Late Collision  273
  late collisions  289,  302
  Length Error  276,  291
  length error  303
  MDC  287
  MDIO  287
  Overflow  273,  277
  oversize  290
  Oversize Packet  276
  oversize packets  291,  303
  packet type  282
  Pause Frame  284
  PHY  271
  prepended header  270
  receive FIFO  266,  277
  receive fifo  275
  RMON statistics  289
  runt  303
  Runt Packet  276
  runt packets  290,  291
  transmit FIFO  266,  272
  Underflow  273,  277
  underflow  302
  underrun  290
Ethernet and Serial DMA Engines  157
Ethernet MAC  264
Ethernet MACs  264
Example Startup Code to clear the L2 Cache 99
Explicit Descriptor Interrupts 158

Introduction

Feature Control 219
FIFO Configuration 351
Fixed Cycle Write Access 368
Flow Control 284
Flow control 294
flow control 152, 271
watermark 157

General Timers 58
general_timer_cfg 60
general_timer_cnt 60
general_timer_init_cnt 60

Generic Bus
address range 363
address space 362
burst mode 371
chip selects 362
data width 362, 365
error conditions 374
parity 364
Timing 365

Generic Bus Errors 374

HyperTransport Interface Overview 265
Interface to PHY 271
interrupt_diag 50, 52
interrupt_ldt 48, 49, 50, 52
interrupt_ldt_clr 52
interrupt_ldt_set 48, 52
interrupt_map 52
interrupt_mask 50, 52
interrupt_source_status 50, 52
interrupt_status 52
interrupt_trace 51, 52
interrupts
Arbitrated 49
Fixed 49
Non-vectored 49

I/O Bridge 1 157
I/O Bridge Clocks 32
IFG 269

HyperTransport Configuration Header 245
HyperTransport Expansion Space 194
HyperTransport Fabric to PCI Bus 221
HyperTransport Resets 260

I/O Bridge 1 157
I/O Bridge Clocks 32
IFG 269

Interface Overview 265

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>io_interrupt_addr0</td>
<td>364</td>
</tr>
<tr>
<td>io_interrupt_addr1</td>
<td>364</td>
</tr>
<tr>
<td>io_interrupt_data0</td>
<td>378</td>
</tr>
<tr>
<td>io_interrupt_data1</td>
<td>378</td>
</tr>
<tr>
<td>io_interrupt_data2</td>
<td>378</td>
</tr>
<tr>
<td>io_interrupt_data3</td>
<td>378</td>
</tr>
<tr>
<td>io_interrupt_parity</td>
<td>379</td>
</tr>
<tr>
<td>io_interrupt_status</td>
<td>18, 364, 374, 377</td>
</tr>
<tr>
<td>io_multi_size</td>
<td>390</td>
</tr>
<tr>
<td>io_start_addr</td>
<td>390</td>
</tr>
<tr>
<td>IPv4 Header Checksum</td>
<td>283</td>
</tr>
<tr>
<td>iSCSI</td>
<td>180</td>
</tr>
<tr>
<td>IsocBAR</td>
<td>212</td>
</tr>
<tr>
<td>IsocIgnMask</td>
<td>212</td>
</tr>
<tr>
<td>JTAG</td>
<td>41, 70</td>
</tr>
<tr>
<td>JTAG and Debug</td>
<td>422</td>
</tr>
<tr>
<td>L1CA</td>
<td>23</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>89</td>
</tr>
<tr>
<td>L2 cache</td>
<td>89</td>
</tr>
<tr>
<td>L2_cache_disable</td>
<td>94</td>
</tr>
<tr>
<td>L2_ecc_tag</td>
<td>89, 97, 100</td>
</tr>
<tr>
<td>L2_misc_config</td>
<td>99</td>
</tr>
<tr>
<td>L2_misc_value</td>
<td>91, 94, 99, 100</td>
</tr>
<tr>
<td>L2_read_tag</td>
<td>97, 100</td>
</tr>
<tr>
<td>L2_way_enable</td>
<td>91, 94</td>
</tr>
<tr>
<td>LDT</td>
<td>190</td>
</tr>
<tr>
<td>ldt_interrupt</td>
<td>55</td>
</tr>
<tr>
<td>ldt_interrupt_clear</td>
<td>50</td>
</tr>
<tr>
<td>LDT_PWROK</td>
<td>26, 260</td>
</tr>
<tr>
<td>LDT_RESET_L</td>
<td>26, 57, 260</td>
</tr>
<tr>
<td>Line 5</td>
<td>5</td>
</tr>
<tr>
<td>Line Interface Configuration</td>
<td>352</td>
</tr>
<tr>
<td>Little Endian System</td>
<td></td>
</tr>
<tr>
<td>No Swaps</td>
<td>201</td>
</tr>
<tr>
<td>MAC Registers</td>
<td>302</td>
</tr>
<tr>
<td>mac_addr</td>
<td>314</td>
</tr>
<tr>
<td>mac_adfilter_cfg</td>
<td>171, 172, 279, 282, 283, 285, 293, 316</td>
</tr>
<tr>
<td>mac_admask</td>
<td>315</td>
</tr>
<tr>
<td>mac_cfg</td>
<td>271, 272, 273, 275, 277, 281, 282, 284, 293, 294, 299, 302</td>
</tr>
<tr>
<td>mac_chlo</td>
<td>281, 294, 299, 318</td>
</tr>
<tr>
<td>mac_chup</td>
<td>281, 294, 299, 318</td>
</tr>
<tr>
<td>mac_debug_status</td>
<td>313</td>
</tr>
<tr>
<td>mac_enable</td>
<td>301, 306</td>
</tr>
<tr>
<td>mac_ethernet_addr</td>
<td>274, 285, 315</td>
</tr>
<tr>
<td>mac_frame_cfg</td>
<td>271, 276, 284, 308</td>
</tr>
<tr>
<td>mac_hash</td>
<td>315</td>
</tr>
<tr>
<td>mac_int_mask</td>
<td>271, 286, 314</td>
</tr>
<tr>
<td>mac_mdio</td>
<td>287, 294, 318</td>
</tr>
<tr>
<td>mac_rx_fifo_ptrs</td>
<td>314</td>
</tr>
<tr>
<td>mac_status</td>
<td>157, 271, 273, 277, 285, 286, 310</td>
</tr>
<tr>
<td>mac_status_0</td>
<td>54</td>
</tr>
<tr>
<td>mac_status_1</td>
<td>54</td>
</tr>
<tr>
<td>mac_status_2</td>
<td>54</td>
</tr>
<tr>
<td>mac_status_debug</td>
<td>286</td>
</tr>
<tr>
<td>mac_status1</td>
<td>286, 313</td>
</tr>
<tr>
<td>mac_status1_0</td>
<td>56</td>
</tr>
<tr>
<td>mac_status1_1</td>
<td>56</td>
</tr>
<tr>
<td>mac_status1_2</td>
<td>56</td>
</tr>
<tr>
<td>mac_thrh_cfg</td>
<td>272, 275, 294, 307</td>
</tr>
<tr>
<td>mac_txd_ctl</td>
<td>274, 306</td>
</tr>
<tr>
<td>mac_type_cfg</td>
<td>316</td>
</tr>
<tr>
<td>mac_viantag</td>
<td>152, 164, 274, 285, 310</td>
</tr>
<tr>
<td>MACs</td>
<td>401</td>
</tr>
<tr>
<td>mailbox</td>
<td>48, 49</td>
</tr>
<tr>
<td>mailbox_0</td>
<td>46</td>
</tr>
<tr>
<td>mailbox_1</td>
<td>46</td>
</tr>
<tr>
<td>mailbox_cpu</td>
<td>46</td>
</tr>
<tr>
<td>Management Interface to PHY</td>
<td>287</td>
</tr>
<tr>
<td>Mapping</td>
<td>109</td>
</tr>
<tr>
<td>mbox_clear</td>
<td>46</td>
</tr>
<tr>
<td>mbox_clr_cpu</td>
<td>46</td>
</tr>
<tr>
<td>mbox_set</td>
<td>46</td>
</tr>
<tr>
<td>mbox_set_cpu</td>
<td>46</td>
</tr>
<tr>
<td>mc_clock_cfg</td>
<td>127, 138</td>
</tr>
<tr>
<td>mc_config</td>
<td>105, 106, 112, 119, 121, 126, 128, 135</td>
</tr>
<tr>
<td>mc_cs_end</td>
<td>111</td>
</tr>
<tr>
<td>mc_cs_interleave</td>
<td>111, 119, 121</td>
</tr>
<tr>
<td>mc_cs_start</td>
<td>111</td>
</tr>
<tr>
<td>mc_csN_ba</td>
<td>121</td>
</tr>
<tr>
<td>mc_csN_col</td>
<td>121</td>
</tr>
<tr>
<td>mc_csN_row</td>
<td>121</td>
</tr>
<tr>
<td>mc_dramcmd</td>
<td>126, 139, 139</td>
</tr>
<tr>
<td>mc_drammode</td>
<td>123, 124, 126, 139</td>
</tr>
<tr>
<td>mc_test_data</td>
<td>125</td>
</tr>
<tr>
<td>mc_test_ecc</td>
<td>107, 125</td>
</tr>
<tr>
<td>mc_timing1</td>
<td>125, 131, 135, 140</td>
</tr>
<tr>
<td>mc_timing2</td>
<td>125</td>
</tr>
<tr>
<td>memory barrier</td>
<td>107</td>
</tr>
<tr>
<td>Memory clock</td>
<td>33</td>
</tr>
<tr>
<td>Memory Configurations</td>
<td>109</td>
</tr>
<tr>
<td>Memory Controller Architecture</td>
<td>104</td>
</tr>
<tr>
<td>Memory Locked in the L2 Cache</td>
<td>92</td>
</tr>
<tr>
<td>Memory Mapped Devices</td>
<td>194</td>
</tr>
<tr>
<td>MESI</td>
<td>21</td>
</tr>
<tr>
<td>MIL</td>
<td>264, 267</td>
</tr>
<tr>
<td>NMI</td>
<td>47, 50</td>
</tr>
<tr>
<td>Normal Operation</td>
<td>89</td>
</tr>
<tr>
<td>Not Implemented</td>
<td>5</td>
</tr>
<tr>
<td>Open Page policy</td>
<td>122</td>
</tr>
<tr>
<td>Other Documentation</td>
<td>4</td>
</tr>
<tr>
<td>Overview of the ZBus Protocol</td>
<td>20</td>
</tr>
<tr>
<td>P_RESET_L</td>
<td>57</td>
</tr>
<tr>
<td>P_RST_L</td>
<td>36</td>
</tr>
<tr>
<td>Packet FIFO</td>
<td>292</td>
</tr>
<tr>
<td>16-bit encoded mode</td>
<td>300</td>
</tr>
<tr>
<td>16-bit GMII style</td>
<td>299</td>
</tr>
<tr>
<td>8-bit Encoded mode</td>
<td>296</td>
</tr>
<tr>
<td>8-bit EOP flagged</td>
<td>298</td>
</tr>
<tr>
<td>8-bit GMII style</td>
<td>295</td>
</tr>
<tr>
<td>8-bit SOP flagged</td>
<td>297</td>
</tr>
<tr>
<td>CRC</td>
<td>294, 299</td>
</tr>
<tr>
<td>flow control</td>
<td>294</td>
</tr>
<tr>
<td>Packet FIFO interface</td>
<td>264</td>
</tr>
<tr>
<td>Packets Dropped by the DMA Channel</td>
<td>283</td>
</tr>
<tr>
<td>Page policy</td>
<td></td>
</tr>
</tbody>
</table>
CAS time check  122
Closed  122
Hint Based  122
Open  122
Pause Frame  279, 315
Pause frame  305
pause frame  152, 164, 168, 267, 271, 284, 284, 303, 313, 317
PCI  190, 401

- match byte lanes  202
- access to mailbox registers  207
- Adaptive Extend Register  243, 244
- adaptive retry  217
- Additional Status and Control Register  242
- arbiter  190, 222

BAR  205
BAR0 Map Table Entry  242
bus number  234
bus zero  191
Bypass Control Register  244
Cache Line Size  241
ClassRevSet Register  233
Command Register  240
Compatibility space  197
Configuration  194
collection  234
collection address  234
Device Mode  199
device number  234
DEVSEL  206
endian policy (BAR0)  206
errors  235
Feature Control  242
collection  234
I/O space  195
IDSEL generation  235
INTA Control Register  233, 243
interrupt acknowledgment  199
Latency Timer  241
Little endian system  201
match bit lanes  203
memory mapped  194
ordering rules  231
Read  217
Read Host Register  243
ReadLine  217
ReadMultiple  217
register  234
Status Register  240
SubSysSet Register  233
subtractive decode  195, 196, 199
Timeout Register  241
VendorIdSet  233
PCI and HyperTransport Address Range  192
PCI BAR2  46
PCI BAR3  46
PCI Bus and HyperTransport Fabric  190
PCI Bus To HyperTransport Fabric  219
PCI Configuration Header  236
PCI Full Access Space  199
PCI I/O Space  195

PCMCIA  384
- attribute memory  386
- status signals  386
PCMCIA Power Control Pins  401
pomcia_cfg  386, 387, 390, 393, 394
pomcia_status  395
PEC  406
Peer-to-Peer Accesses  219
perf_cnt  62
perf_cnt_cfg  61
Performance Monitoring Features  134
Performance of the PCI and HyperTransport Interfaces  213
PHY  271
Physical Address  34
Physical Addresses  5
PLL  31, 41, 43
Preamble  269
Prlld  29, 42
Protocol Engine and GMII/MII  267
Protocol Engine Configuration  271, 351

R
R_EXC  23, 23, 77
R_L2HIT  23, 77
R_SHD  23, 77
Read Only  5
Receive Path  277
Receiver Operation  275
Reference  446
Reserved  5
Reset  26
RESET_L  26, 27, 30, 41, 232, 260, 422, 430, 432, 443
RESETOUT_L  26, 27, 41, 44, 57
Response Phase  23
Rings and Chains  151
RMON  353
RMON Counters  289, 289, 353
RMON statistical counters  265
Row Address  117
Row, Column and Bank Configuration  112, 117
RTS  322, 323, 324

S
sb_softres  41
SCD  41, 89
SDRAM Refresh  126
SDRAM Timing  125
ser_addr_mask  348, 359
ser_clk  340
ser_cmd  355
ser_dma_enable  357
ser_err_mask  346
ser_int_mask  358
ser_line_mode  354
ser_maxfrm_sz  345, 348, 357
ser_min_frm_sz  346
ser_minfrmsz  345, 356
ser_mode  341, 343, 344, 345, 346, 348, 354
ser_rx_rd_thres  356
ser_rx_table 359
ser_status 157, 341, 343, 346, 349, 357
ser_status_debug 358
ser_tx_rd_thres 346, 356
ser_tx_table 359
ser_tx_wr_thres 346, 356
ser_usr0_addr 359
Serial Configuration Interface 404
Serial Interfaces 321
Serial Ports 401
SFD 269
Signal Description 7
smb_cmd 410, 412, 416
smb_control 413, 416
smb_data 410, 412, 417
smb_freq 412, 416
smb_pec 406, 408, 410, 412, 418
smb_start 408, 410, 412, 418, 419
smb_status 406, 410, 412, 413, 417
smb_xtra 412, 417
SMBus 404
SMBus Overview 404
SMBus Registers 416
SouthBridge 195
Southbridge 195
southbridge 199
southOnLDT 196
Special HyperTransport Space 199
SRI Command Register 194, 256
Standard Management Mode Accesses (both ECC diag address
bits zero) 97
Standard RAM 92
Subtractive Decode 195
Supported DRAMs and DIMMs 123
Synchronous Interface Configuration 351
Synchronous Serial Loopback 352
Synchronous Serial Protocol Engine 344
system 41
System Control and Debug Unit 41
System Overview 9
System Reset Initialization of the HyperTransport Interface 256
system_cfg 41, 42, 43, 321, 373
system_manuf 43
system_reset 41
system_revision 41, 42
system_scratch 42, 45
TCP 178, 180
TCP checksum 283
Terminology 5
The GPIO Pins 397
Timer Registers 59
Timer Special Cases 58
Timers 57
Timing Parameter Guidelines 131
trace_buffer 70, 76, 79
trace_cfg 75, 76, 79, 84, 85, 86
trace_event 72
trace_read 76
trace_sequence 74
Transaction ID 21
transaction ID 67
Transmitter Operation 272
Transport Protocol 404
trigger event 70
trigger sequence 73
UDP 178
UDP checksum 283
UNDEFINED 5
UNPREDICTABLE 5
Using the L2 Cache as Memory 91
Using the PCI in Device Mode 232
VGA 195
VGA display controller 196
Viewing Endian Policy as an Optimization 204
Virtual Address 5
VLAN TAG 269
VLAN tag 270, 274
Watchdog Timers 57
watchdog_timer_cfg 59
watchdog_timer_cnt 59
watchdog_timer_init_cnt 59
Write Only 5
ZBBus Monitoring 134
zbbus_cycle_count 58, 61
zbbus_cycle_cp0 54, 61
zbbus_cycle_cp1 54, 61