上 海 交 通 大 学

密西根学院

UM–SJTU Joint Institute

University of Michigan

Shanghai Jiao Tong University

X

UM-SJTU Joint Institute

Ve270 Introduction to Logic Design

by

Xia Fangzhou 5113709225

Xue Jiong 5113709218

2013.7.23

Design of a Digital Device

Objectives

In this lab, we are asked to design a digital device which integrates the function of a character roller and a simple calculator. The user should be able to switch between these two functions by toggling a switch on the FPGA board.

Problem Definition

In this lab, we are asked to integrate all the components we have used to design a digital device which has the function of a roller and a simple calculator which can add two 4-bit 2's complement numbers.

The character roller designed in this lab should be able to roll the student ID over four SSDs automatically which is suitable for human eyes. The rolling should start automatically once the controlling switch is turned up.

The 4-bit 2's complement number simple calculator should be able to add the next four bits input to the four bit number stored in the FPGA when an equal button is pressed. Over flow should be indicated using a LED if detected. The initial value of the calculator should be set to zero when the controlling switch is turned down.

As a whole, the digital device designed in this lab should have five switches, one equal button for input and four SSDs, one LED for output.

System Partitioning

Due to the complexity of the device, we use Verilog HDL to describe the behavior of the circuit. Things become relatively easy in this way since we just need to specify the system input and output and the timing issues of the circuit. As for system partitioning, we divide the system into the following parts:

Clock Dividers: Two clock divider for character rolling and SSD displaying.

SSD Displayer: To display the number on four SSD according to the input.

SSD Driver: Generate the signal to control the character position displayed on the SSD driver.

Character Roller: Predefined character shifter.

Adder: Simple calculator which adds the value of the 2's complement numbers.

Debouncer: Optional part to stabilize the button.

Design Entry

Based on the system partition, we describe the behavior of the circuit directly using Verilog HDL. The Verilog HDL code goes as the following.

ForReportOnly.v Tue Jul 23 20:57:31 2013

```
`timescale 1ns / 1ps

module Lab8(reset, control,equal_button, clock,operand,ssd_driver, ssd_shine,
overFlowIndicator);
input reset;
input equal_button;
input control;
input clock;
input [3:0] operand;
output [6:0] ssd_driver ;
output [3:0] ssd_shine ;
output overFlowIndicator;
wire second0;
wire display;
wire reset ;
wire equal;
wire control ;
wire clock;
wire [3:0] ssd_shine;
wire [6:0] ssd_driver ;
wire [3:0] operand;
wire ssd_clock;
wire [15:0] ssd_feed_signal;
wire overFlowTester;
reg [3:0] displayInfo [3:0];
reg [3:0] numericalResult;
reg [3:0] nextNumericalResult;
reg [3:0] signIndicator;
reg overFlowIndicator;
reg [25:0] div0;
reg [10:0] divd;
reg [51:0] studentID;
reg [3:0] temp;
reg [3:0] absoluteValue;
reg [24:0] equal_count;


assign second0 = (div0==26'd2000_0000);
assign equal = (equal_count==25'd50000);
assign ssd_clock = display;
assign ssd_feed_signal = {displayInfo[3],displayInfo[2],displayInfo[1],displayInf
assign overFlowTester = ((nextNumericalResult[3]^numericalResult[3]) && (numerica
[3]==operand[3])) ? 1:0;
ssdRollerModule ssdDisplay(reset,ssd_clock,ssd_feed_signal,ssd_shine,ssd_driver);

//equal debounce
always @ (posedge clock,  posedge reset, posedge equal)
 begin: equalDebounce
  if(reset==1'b1) equal_count<=25'd0;
  else begin
      if(equal==1'b1)
      begin
         if (equal_button==1'b0) equal_count<=25'd0;
         else;
      end
      else begin
         if(equal_button==1'b1) equal_count<= equal_count+25'd1;
         else  equal_count<=25'd0;
      end
```

Page 1

```
        end
       end


      //clock divider0
      always @ (posedge clock, posedge reset,posedge second0)
       begin: clock0
        if(reset==1'b1) div0<=26'd0;
        else begin
        if(second0==1'b1) div0<=26'd0;
        else
        div0 <= div0+26'd1;
        end
       end

      //display clock divider
      always @ (posedge clock, posedge reset, posedge display)
       begin: SSDdisplay
        if(reset==1'b1) divd<=11'd0;
        else begin
        if (display) divd<=11'd0;
        else
        divd <= divd+11'd0001;
        end
       end

      //roller
      always @ (posedge second0, posedge reset)
       begin : roller
        if (reset == 1'b1) begin
          studentID<={4'b1110,4'b1110,4'b1110,4'b1110,4'd5,4'b1,4'd1,4'd3,4'd7,4'd0,4'd
      4'd2,4'd5};
          temp<=4'b1110;
          end
          else begin
            if (control ==1'b1)
            begin
            studentID[51:4]<=studentID[47:0];
            studentID[3:0]<=temp[3:0];
            temp[3:0]<=studentID[51:48];
            end
            else;
          end
       end

      //adder
       always @ (posedge equal, posedge reset)
       begin : adder
       if (reset == 1'b1) begin
          overFlowIndicator<=1'b0;
          numericalResult<=4'b0000;
          signIndicator<=4'b1111;
          end
          else begin
            if (control ==1'b0)
            begin
            overFlowIndicator<=overFlowTester;
            numericalResult<=nextNumericalResult;
            if (nextNumericalResult[3]==1'b1)
```

```
                begin
                signIndicator<=4'b1110;
                absoluteValue<=(~nextNumericalResult)+1;
                end
                else if (nextNumericalResult[3]==1'b0)
                begin
                signIndicator<=4'b1111;
                absoluteValue<=nextNumericalResult;
                end
            end
            else signIndicator<=4'b1111;
          end
        end

        // Display Control
        always @ (posedge clock)
        begin: displayControl
          nextNumericalResult<=numericalResult+operand;
          if(control==1'b1) begin
             displayInfo[3]<=studentID[51:48];
             displayInfo[2]<=studentID[47:44];
             displayInfo[1]<=studentID[43:40];
             displayInfo[0]<=studentID[39:36];
          end
          else if(control==1'b0) begin
             displayInfo[3]<=4'b1111;
             displayInfo[2]<=4'b1111;
             displayInfo[1]<=signIndicator;
             if(absoluteValue==4'b1000) displayInfo[0]<=absoluteValue;
             else  displayInfo[0]<={1'b0,absoluteValue[2:0]};
          end
          else;
        end
       endmodule

       //ssd Controller
       module ssdRollerModule(reset_signal,clock_signal,input_desplay_signal,ssd_selecti
       output_7_bits );
       parameter n=4;
       input clock_signal;
       input reset_signal;
       input [n*4-1:0] input_desplay_signal;
       wire [3:0] ssd_signal [n-1:0];
       output [6:0] output_7_bits;
       output [n-1:0] ssd_selection;
       reg [3:0] input_4_bits;
       reg [6:0] output_7_bits;
       reg [n-1:0] ssd_selection;

       assign {ssd_signal[3],ssd_signal[2],ssd_signal[1],ssd_signal[0]}=input_desplay_si

       always @ (posedge clock_signal,posedge reset_signal)
        if (reset_signal) ssd_selection<=4'b0111;
        else begin
        case(ssd_selection)
          4'b0111:
          begin
          ssd_selection<=4'b1110;
          input_4_bits<=ssd_signal[0];
```

```
          end
          4'b1110:
          begin
          ssd_selection<=4'b1101;
          input_4_bits<=ssd_signal[1];
          end
          4'b1101:
          begin
          ssd_selection<=4'b1011;
          input_4_bits<=ssd_signal[2];
          end
          4'b1011:
          begin
          ssd_selection<=4'b0111;
          input_4_bits<=ssd_signal[3];
          end
          default ssd_selection<=4'b0111;
          endcase
        end

       always @(input_4_bits,ssd_selection)
       begin: SSDCASE
       case (input_4_bits)
         4'b0000: output_7_bits <= 7'b0000001; //0
         4'b0001: output_7_bits <= 7'b1001111; //1
         4'b0010: output_7_bits <= 7'b0010010; //2
         4'b0011: output_7_bits <= 7'b0000110; //3
         4'b0100: output_7_bits <= 7'b1001100; //4
         4'b0101: output_7_bits <= 7'b0100100; //5
         4'b0110: output_7_bits <= 7'b0100000; //6
         4'b0111: output_7_bits <= 7'b0001111; //7
         4'b1000: output_7_bits <= 7'b0000000; //8
         4'b1001: output_7_bits <= 7'b0000100; //9
         4'b1010: output_7_bits <= 7'b0001000; //A
         4'b1011: output_7_bits <= 7'b1100000; //b
         4'b1100: output_7_bits <= 7'b0110001; //C
         4'b1101: output_7_bits <= 7'b1000010; //d
         4'b1110: output_7_bits <= 7'b1111110; //-
         4'b1111: output_7_bits <= 7'b1111111; //nothing
         default: output_7_bits <= 7'b1111111;
       endcase
       end
       endmodule
```

Test Plan

In order to test the design, generate the RTL and check if all the necessary components have been implemented based on the Verilog code.

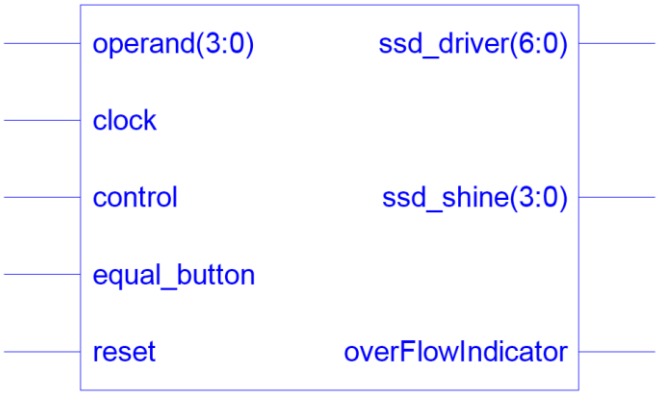

Figure 1 Block Diagram of the Designed Digital Device

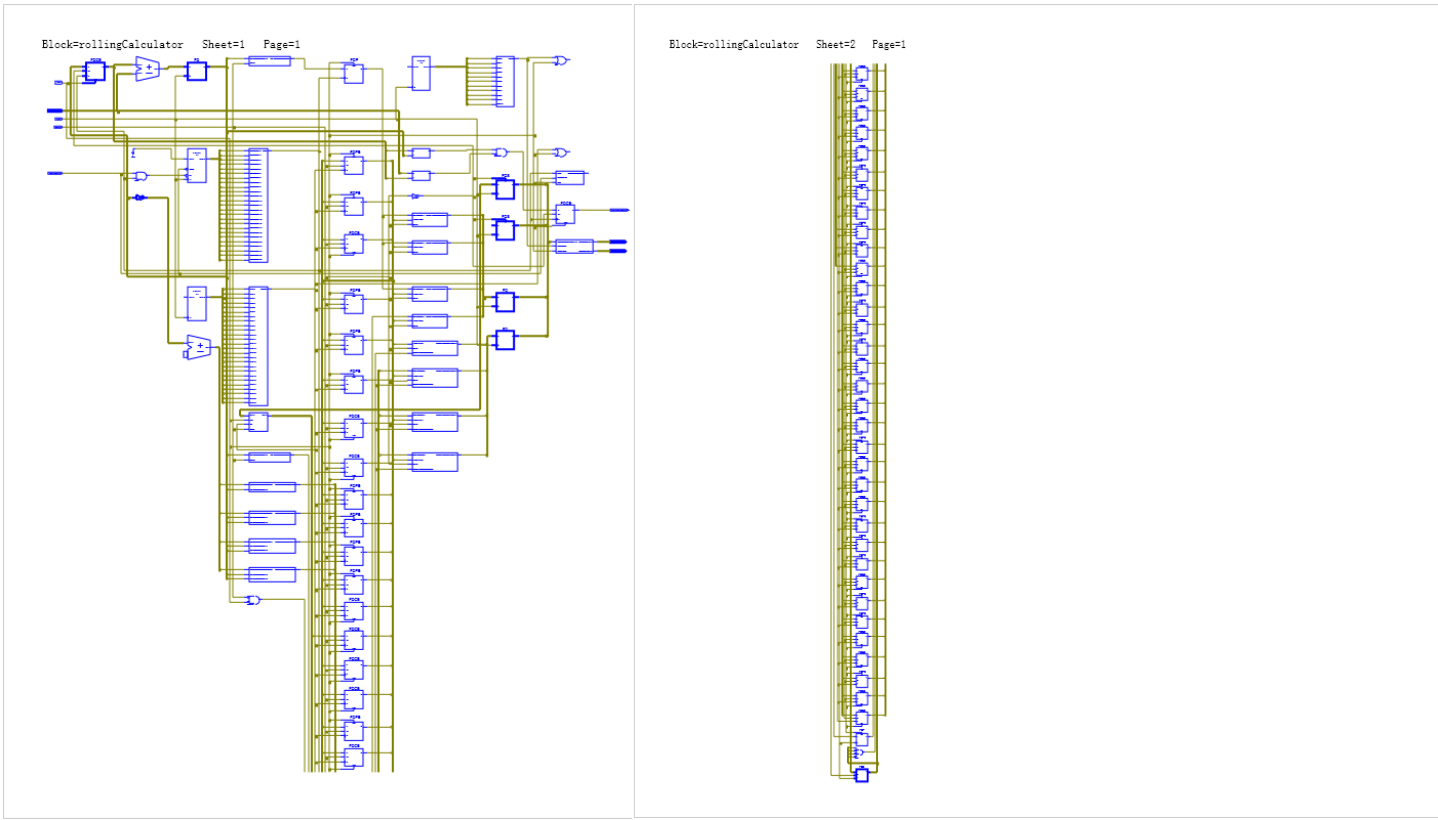

Figure 2 Generated RTL Schematic

Simulation Results

Due to the complexity of the design, it is not reasonable to simulate the entire circuit using test bench wave form. Since we integrated all the previous working components except for the adder to design this digital device, simulations for the individual components are not necessary.

Hardware Implementation and Testing

To test our design, we used the XC3S1200E FPGA board and assign the corresponding IO pins on the board. We demonstrated the functionality of our driver design to the TAs during lab. For the additional debouncing functionality, we demonstrated by comparing the effect with no debuncing version of the design and actually noticed the difference.

Design Object List - I/O Pins

I/O Name	I/O Direction	Loc	Bank
clock	Input	B8	BANK0
control	Input	R17	BANK1
equal_button	Input	B18	BANK1
operand<0>	Input	G18	BANK1
operand<1>	Input	H18	BANK1
operand<2>	Input	K18	BANK1
operand<3>	Input	K17	BANK1
overFlowIndicator	Output	J14	BANK1
reset	Input	H13	BANK1
ssd_driver<0>	Output	H14	BANK1
ssd_driver<1>	Output	J17	BANK1
ssd_driver<2>	Output	G14	BANK1
ssd_driver<3>	Output	D16	BANK1
ssd_driver<4>	Output	D17	BANK1
ssd_driver<5>	Output	F18	BANK1
ssd_driver<6>	Output	L18	BANK1
ssd_shine<0>	Output	F17	BANK1
ssd_shine<1>	Output	H17	BANK1
ssd_shine<2>	Output	C18	BANK1
ssd_shine<3>	Output	F15	BANK1

Figure 3 Assignment for IO Pins

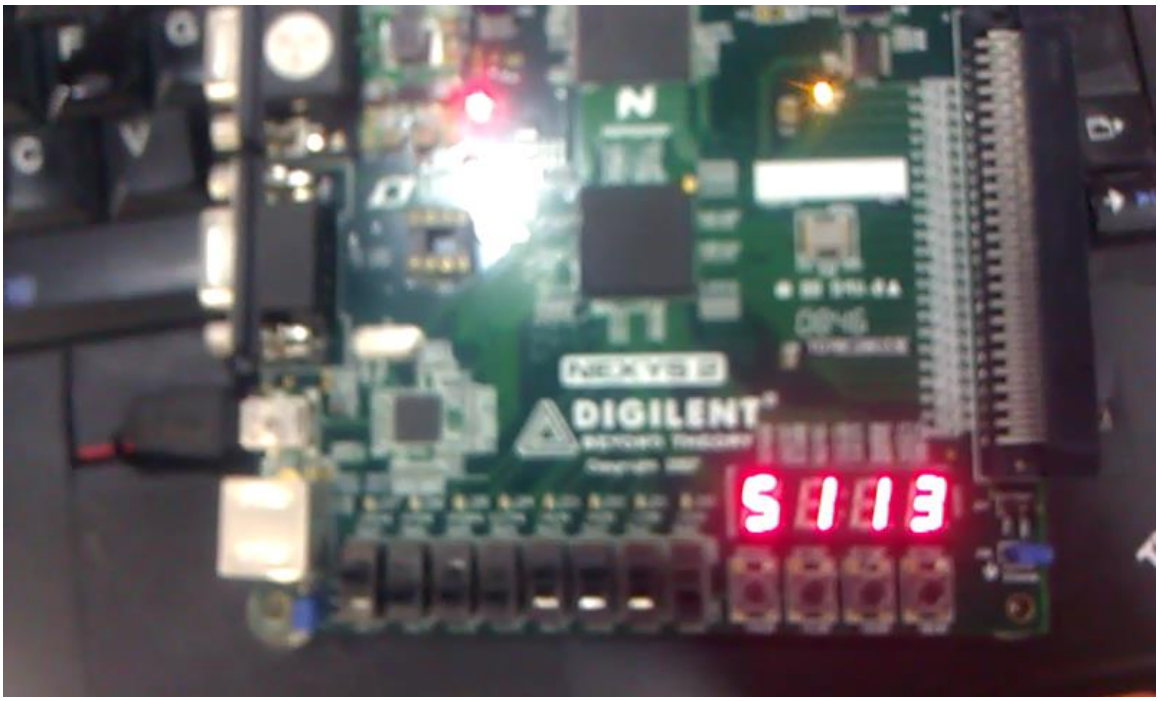

Figure 4 Character Rolling

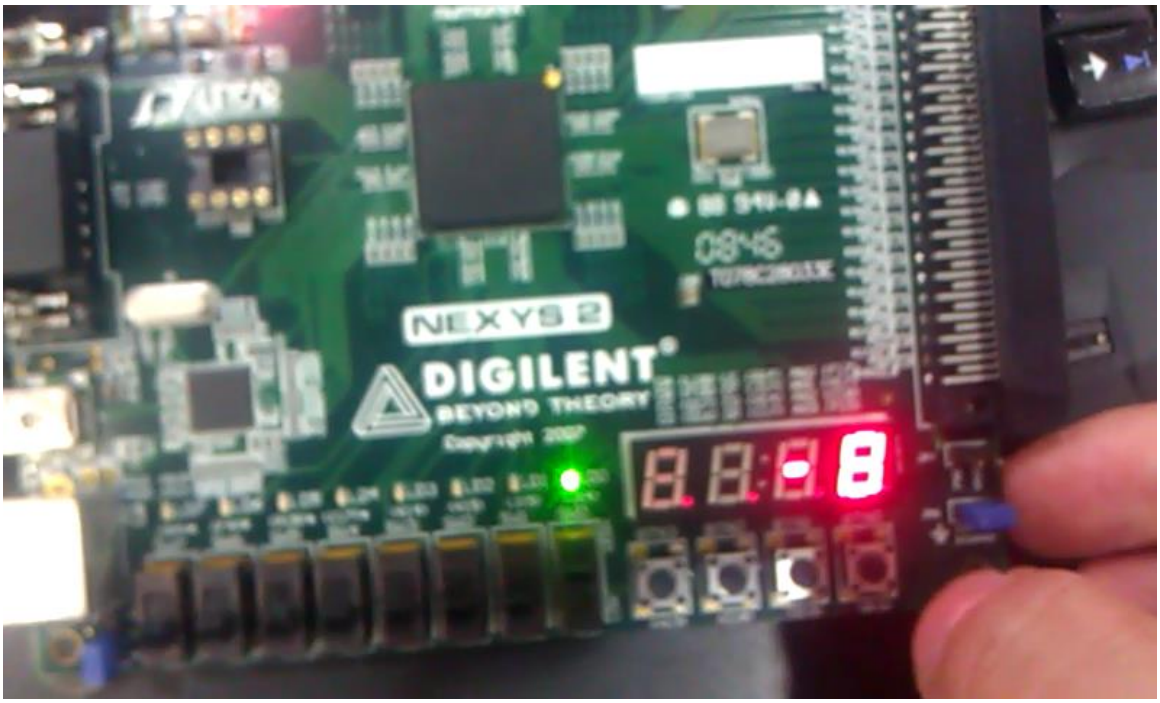

Figure 5 Number Adding

Conclusions

In this lab, we managed to design a relatively complicated digital system. This experience is very useful since it is the first time for us to combine different module designed before to accomplish a complicated design. This lab greatly cultivated our ability to describe the behavior of the circuit using Verilog, which is indeed a powerful tool.