Programmable State-Variable Filter Design For a Feedback Systems Web-Based Laboratory

by
Rayal Johnson

February 17, 2004

Advanced Undergraduate Project
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
Supervisor: Dr. Kent Lundberg

Abstract

This document discusses the first installation of a weblab for a feedback systems class. A programmable state-variable filter accessible through the web is designed and analyzed. The filter's parameters such as the natural frequency and damping ratio are controlled through a web client and the effect of each parameter is analyzed and documented.

Contents

1 Introduction

The programmable state-variable filter is the first part of a internet laboratory, also known as a weblab, for 6.302 (Feedback Systems). The weblab allows students access to real systems through the web much like the one used for 6.012 (Microelectronic Devices). The weblab allows students to perform experiments from their computer and eliminates the need to go to lab.

The types of experiments that are performed are frequency responses. To fully understand the filter responses, these measurements must be made while changing filter characteristics such as natural frequencies and damping ratio/quality factor. The natural frequencies determine the bandwith of the filter and the damping ratio/quality factor determine how much peaking occurs at the corner frequency. In order to change these filter characteristics from a remote terminal, the circuit representation of the filter needs to be adjusted by the server to which it is connected.

In order for the remote terminal to interface with the server, it must communicate with the server through a scripting language. The server chosen for this project is the Apache Webserver, which is a free open source server that is easily configurable and extendable. The scripting language used is the PHP scripting language, another open-source product. It is used because Apache supports it and it has a powerful set a functions to create almost any type of application. One of those applications is the full control of the server's RS232 serial port which will be used in this project to both send data to the circuit and to receive data from the circuit.

The overall weblab system consists of the PC/Server controlled by the remote client through the PHP scripting language, a level shifter, a microcontroller to interpret the server signals and set filter characteristics, and the filter itself. A system level block diagram of the weblab system is shown in Figure 1. The DACs are latch free, so as long as the DAC select bits are set at the same time as the data bits, they will work correctly. This is great for testing the DACs since there are no timing issues associated with writing to the DACs.

The level shifter changes RS232 logic (+12 and −12 volts) from the server to TTL logic (+5 and 0 volts) which can be interpreted by the PIC microcontroller. The microcontroller uses the signals from the server to determine the values to be sent to each digital to analog converter (DAC) over 8 bits of data. The PIC processor has two input/output ports known as PORTA and PORTB, both 8 bits wide. PORTA is used to DAC selection and PORTB is used to transfer the data.

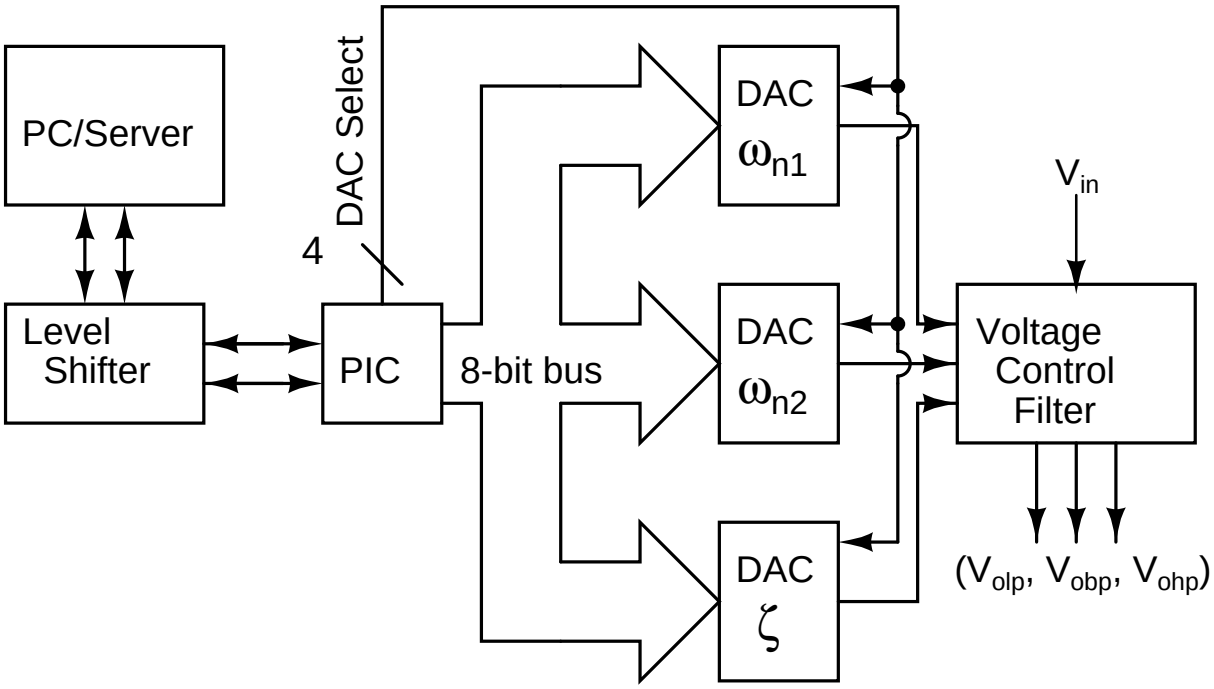

Figure 1: System Level Block Diagram of the Programmable State-Variable Filter

2 State Variable Filter: Theory

2.1 Overview

A very useful filter for teaching any signals class is the state-variable filter. The state-variable filter's topology is such that depending on where the output of the filter is read, a low-pass, band-pass, or high-pass characteristic can be realized. This unique characteristic comes from the filter's implementation using only integrators and gain blocks. A block diagram of the state-variable filter is shown in Figure 2.

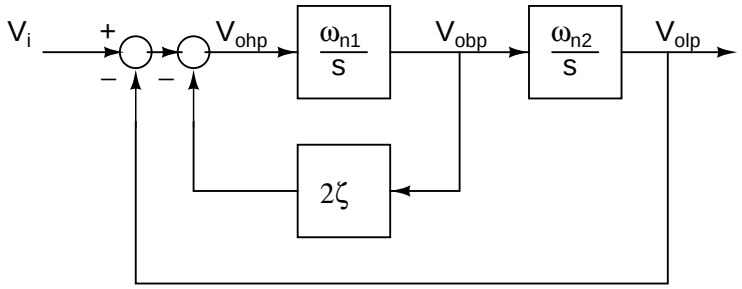

Figure 2: State Variable Filter Block Diagram

The three outputs of the filter are shown in Figure 2 as V_{olp}, V_{obp}, and V_{ohp} corresponding to the low-pass, band-pass and high-pass filters respectively. Utilizing Black's formula, the system function for each filter is as follows:

$$\frac{V_{olp}}{V_i}(s) = \frac{\omega_{n1}\omega_{n2}}{s^2 + 2\zeta\omega_{n1}s + \omega_{n1}\omega_{n2}} \tag{1}$$

$$\frac{V_{obp}}{V_i}(s) = \frac{\omega_{n1}s}{s^2 + 2\zeta\omega_{n1}s + \omega_{n1}\omega_{n2}} \tag{2}$$

$$\frac{V_{ohp}}{V_i}(s) = \frac{s^2}{s^2 + 2\zeta\omega_{n1}s + \omega_{n1}\omega_{n2}} \tag{3}$$

where in all cases:

$$s = j\omega \tag{4}$$

It is the placement of the integrators in the signal path that determine the filters' responses. The low-pass filter has both integrators in the forward path. The band-pass filter has one integrator in the forward path and one in the feedback path. And the high-pass filter has both integrators in the feedback path.

2.2 Integrator Blocks

The most important part of the state-variable filter is the integrator. And in order to build a programmable filter, the gain of the integrator needs to be adjustable. An adjustable integrator can be implemented as shown in Figure 3 with the use of a transconductance amplifier, a capacitor and an operational amplifier.

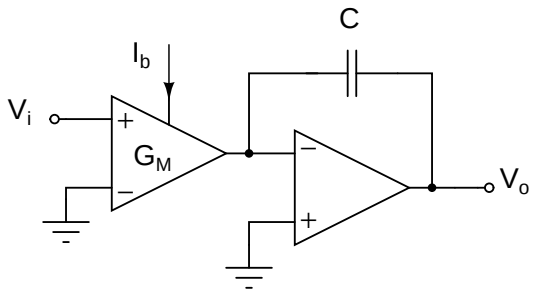

Figure 3: Integrator with adjustable gain

The gain of this block is:

$$\frac{V_o}{V_i}(s) = -\frac{G_M}{Cs} \tag{5}$$

which is exactly what is needed. When this block is used as part of the overall filter, the natural frequency (ω_n) is:

$$\omega_n = \frac{G_M}{C} \tag{6}$$

This natural frequency can be adjusted by changing the bias current I_b. The bias current is related to the transconductance G_M by some constant factor k. The value of this constant was measured using the circuit in Figure 4 using two sets of R_B and R_L to simulate current levels used to set ω_n and ζ. The experiment was performed as follows:

Table 1: Experiment to determine value of k

1. The voltage across R_B was set to set a bias current I_o.
2. V_{id} is set to a 500 Hz sinusoid.
3. The gain from V_{id} to V_o is measured.
4. The G_M is determined as the ratio of the gain to the load resistor R_L.
5. The constant k is determined as the ratio of G_M to the bias current I_b.

The natural frequency's bias current level is in the millamp range, from 0 to $0.5mA$. The damping ratio's bias current level is in the microamp range, from 0 to $2.3\mu A$. The results of the experiment are presented in two graphs, Figure 5 for the natural frequencies and Figure 6 for the damping ratio. The equations that define the relationships between the the gain A_v, the transconductance G_M and bias current I_b are:

$$G_M = I_b k \tag{7}$$

$$A_v = G_M R_L \tag{8}$$

Using the values of the constants k_{ω_n} and k_ζ determined from experiment, the state-variable filter was designed to meet the ranges of ω_n and ζ required, which are discussed in the implementation section for the state-variable filter.

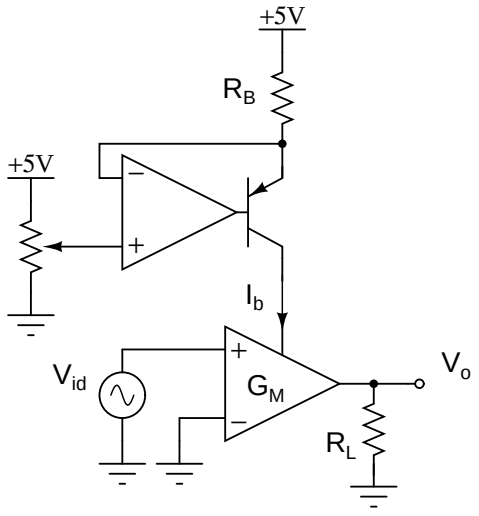

Figure 4: Voltage Controlled Amplifier

The transconductance amplifier is slower than the operational amplifier, therefore contributing non-negligible phase shifts correctable through compensation. [1]

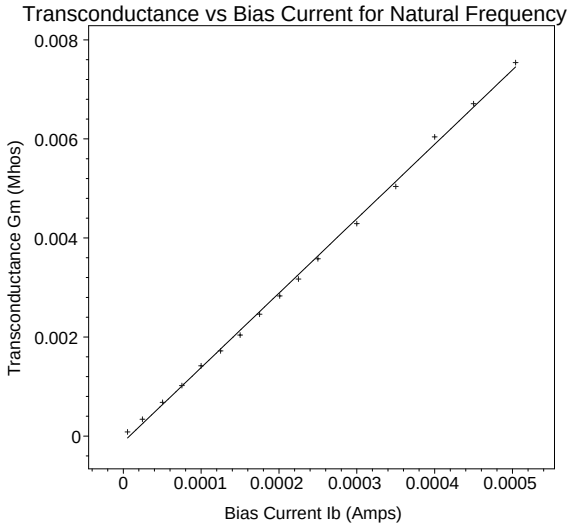

Figure 5: Results of ω_n bias currents. $k_{\omega_n} = 15.029$

[1]The use of a lead compensator can correct negative phase shifts by adding positive phase margin to the system.

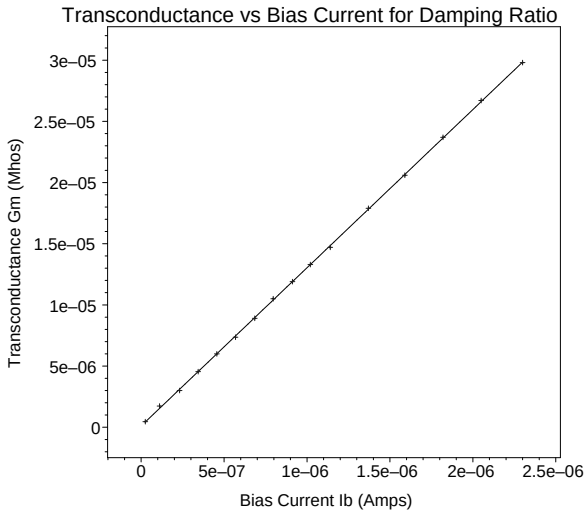

Figure 6: Results of ζ bias currents. $k_\zeta = 12.928$

2.3 Low-Pass Function

The state-variable low-pass filter has the system function $H_{lp}(s)$.

$$H_{lp}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{9}$$

The low-pass filter contains two poles, and therefore it has a roll-off of 40dB/decade. Depending on the relative locations of the two poles, peaking can occur. If the gain is high enough or in this case, if the natural frequencies are high enough, complex poles result and peaking occurs. A bode plot of the filter's frequency response with changing damping ratio/quality factor is shown in Figure 7.

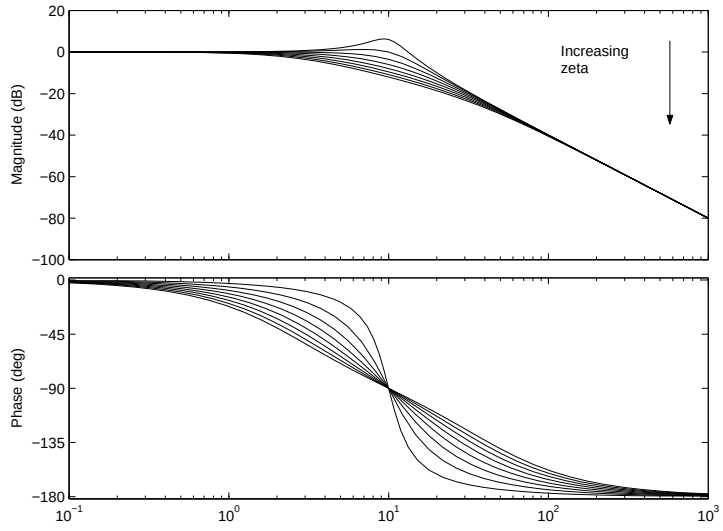

Figure 7: Bode Plot of $H_{lp}(s) = \frac{\omega_n}{s^2+2\zeta\omega_n s+\omega_n^2}$ with varying ζ

The low-pass filter does just that, it lets low frequencies pass and attenuates frequencies higher than its corner frequency. The corner frequency (ω_n), where $\omega_n = \omega_{n1} = \omega_{n2}$ in this bode plot is 10 radians per second (krad/s), therefore, frequencies higher than 10 rad/s are attenuated.

2.4 Band-Pass Function

The state-variable band-pass filter has the system function $H_{bp}(s)$.

$$H_{bp}(s) = \frac{\omega_n s}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{10}$$

The band-pass filter exhibits the peaking seen in the frequency response of the low-pass filter. This characteristic comes from the fact that the filters have the same closed loop poles.

The band-pass filter's frequency response explains what it does. The band-pass filter lets a band of frequencies around its natural frequency, $\omega_n = \omega_{n1} = \omega_{n2}$ is 10 rad/s in this case to pass, and attenuates frequencies much higher or much lower than the pass band. The band-pass filter's frequency response is shown in Figure 8.

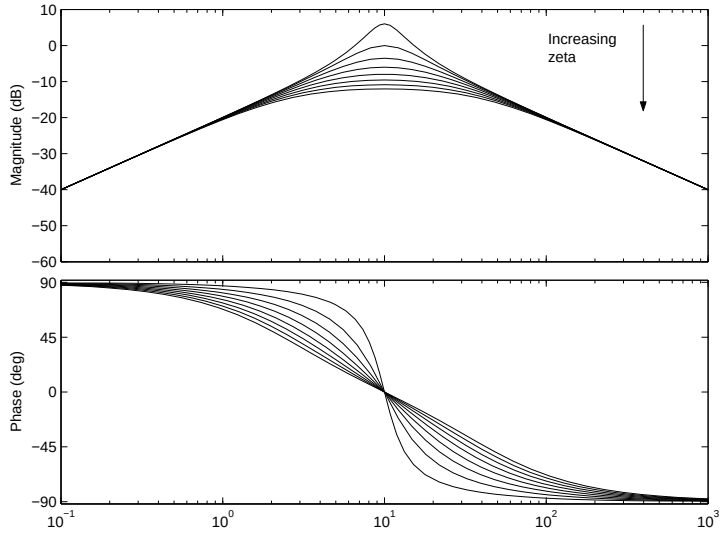

Figure 8: Bode Plot of $H_{bp}(s) = \frac{\omega_n s}{s^2+2\zeta\omega_n s+\omega_n^2}$ with varying ζ

2.5 High-Pass Function

The state-variable high-pass filter has the system function $H_{hp}(s)$.

$$H_{hp}(s) = \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{11}$$

The high-pass filter exhibits the peaking seen in the frequency response of the low-pass and the band-pass filters due to the fact that it too has the same closed loop poles. The band-pass filter's frequency response is shown in Figure 9. This filter lets high frequencies, those higher than its

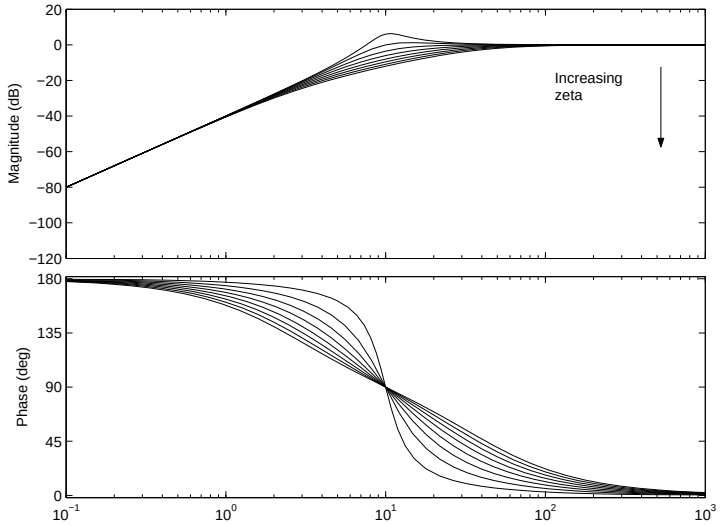

Figure 9: Bode Plot of $H_{hp}(s) = \frac{s^2}{s^2+2\zeta\omega_n s+\omega_n^2}$ with varying ζ

natural frequency, $\omega_n = \omega_{n1} = \omega_{n2}$ in this case, also 10 rad/s to pass and attenuates all other frequencies.

3 State Variable Filter: Implementation

3.1 Hardware

3.1.1 Filter

Moving from the state-variable filter theory, the circuit implementation of the filter can be synthesized using common analog building blocks such as: integrators, gain blocks, adders and subtractors. The circuit implementation is appended as Figure 26.

From some hand analysis, a block diagram can be developed showing exactly how the circuit implementation is related to the block diagram shown in Figure 2. Kirchoff's current rule at the inverting input of U12 of Figure 26 gives:

$$\frac{V_i}{R_5} + \frac{V_{ohp}}{R_8} + \frac{V_{olp}}{R_{17}} + V_{obp}G_{MU23} = 0 \tag{12}$$

Assuming $R_5 = R_{17} = R_8$ and multiplying both sides of the equation by R_8, and solving for V_{ohp}, we get:

$$V_{ohp} = -(V_{olp} + V_i + V_{obp}G_{MU23}R_8) \tag{13}$$

which represents the first adders in Figure 2. Continuing along the signal path from V_{ohp} to V_{obp}, we get the equation:

$$V_{obp} = V_{ohp}\frac{G_{MU14}}{C_7 s} \tag{14}$$

which is the first integrator of the block diagram [2]. Immediately apparent from this block is the value of ω_{n1}, which is:

$$\omega_{n1} = \frac{G_{MU14}}{C_7}. \tag{15}$$

The next block that is easily recognized is the feedback block from V_{obp} to the adder, which represents the value of 2ζ. From this circuit, we see that this value translates into:

$$2\zeta = G_{MU23}R_8. \tag{16}$$

Continuing in the signal path from V_{obp} to V_{olp} we encounter another integrator of the form:

$$V_{olp} = V_{obp}\frac{G_{MU6}}{C_1 s} \tag{17}$$

which gives the value of ω_{n2}, which is:

$$\omega_{n2} = \frac{G_{MU6}}{C_1}. \tag{18}$$

The last part of the block diagram synthesis is the feedback path from V_{olp} to the adder, which we already got from equation 13 which turns out to be unity because of the equal resistor assumption.

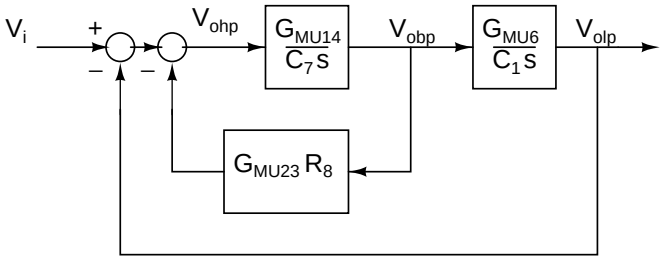

Figure 10: State-Variable Filter Implementation Block Diagram

The fact that the integrators have an noninverting topology is important to get negative feedback as in the block diagram of Figure 2. The inversion of signals V_{olp}, and V_{obp} of Equation 13 is what gives the negative feedback, but since V_i is also inverted there is a phase shift of $\frac{\pi}{2}$ from the input to the outputs. The resulting block diagram from this analysis is shown in Figure 10.

Now that it is confirmed that this circuit matches the block diagram of Figure 2, the parameters that determine system dynamics can be analyzed. The system's dynamics is adjustable through the transconductance amplifiers, and the specific relationships of ω_{n1}, ω_{n2} and 2ζ to their control voltages are shown below:

$$\omega_{n1} = \frac{k_{\omega_n}(V_{dd} - V_{ref1})}{R_6 C_7} \tag{19}$$

$$\omega_{n2} = \frac{k_{\omega_n}(V_{dd} - V_{ref2})}{R_7 C_1} \tag{20}$$

$$2\zeta = \frac{k_{\zeta}(V_{dd} - V_{ref3})R_8}{R_{18}} \tag{21}$$

The the last 3 equations were used to determine values for resistors and capacitors based on desired ranges of ω_{n1}, ω_{n2}, and ζ. The desired range of both ω_{n1} and ω_{n2} is 0 to 62.832 krad/s which corresponds to 10kHz. The desired range for ζ is from 0 to 2. Using the standard capacitor value of 68nF, $V_{ref-max}$ of 5 volts, and V_{BE} of 0.6 volts, the desired value for R_6 and R_7 is 15 $k\Omega$, and the desired value for R_{18} is 3.2 $M\Omega$.

To understand where these equations originate, it is important to take a look at the circuit in Figure 11. This circuit is used to set the bias current for each transconductance amplifier pertaining to each system variable, i.e. ω_{n1}, ω_{n2} and ζ. To see how the output bias current I_o is related to the reference voltage V_{ref}, a feedback block diagram of this circuit is analyzed. The block diagram is shown in Figure 12.

The circuit in Figure 11 works through the use of two feedback loops to create a bias current dependent only on the resistor R_e and the reference voltage V_{ref}. There is a major and minor feedback loop in this circuit. The minor loop is provided by the emitter follower circuit and the

[2]Note, this integrator has a noninverting topology since it is the noninverting pin of the transconductance amplifier that is grounded as opposed to the inverting pin.

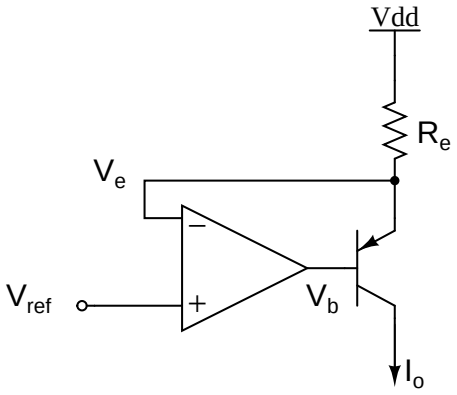

Figure 11: Circuit for Setting Bias Currents

major loop is provided by the operational amplifier. Using Black's formula, the reference voltage V_{ref} is related to the output bias current I_o by Equation 22.

$$\frac{I_o}{V_{ref}} = \frac{\frac{A(s)gm}{1+gmR_e}}{1+\frac{A(s)gmR_e}{1+gmR_e}} \tag{22}$$

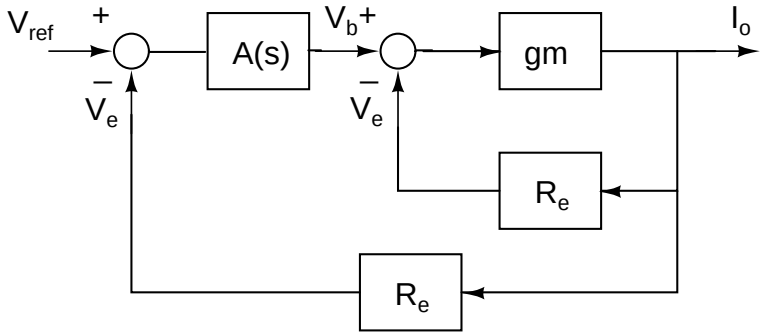

Figure 12: Block Diagram for Bias Current Setting Circuit

The variables $A(s)$, and gm represent the frequency-dependent gain of the operational amplifier, and the transconductance of the bipolar junction transistor respectively. Assuming V_{ref} is changing slowly, the gain of the operational amplifier is at DC is 10^5 according to the TL082 datasheet and it is assumed that:

$$A(s)gmR_e \gg 1 + gmR_e \tag{23}$$

and the relationship between V_{ref} and I_o simplifies to:

$$\frac{I_o}{V_{ref}} \approx \frac{1}{R_e} \tag{24}$$

and therefore:

$$I_o \approx \frac{V_{ref}}{R_e} \tag{25}$$

One minor detail omitted when creating the block diagram of Figure 12 is that V_{dd} has an effect on the bias current. V_{dd} is assumed to be a small signal ground, and therefore omitted in the block diagram but the large signal bias current is dependent on this value and therefore the equation that relates V_{ref} and I_o is more correctly:

$$I_o = \frac{V_{dd} - V_{ref}}{R_e} \tag{26}$$

3.1.2 DAC-Filter Interface

The bias currents that set the dynamics of the filter are set through the use of digital-to-analog converters. Each DAC receives eight bits of data pertaining to the value of the filter's characteristics, which are set by the user. The implementation of the DAC-Filter interface is shown in Figure 25. The data that is received by the DACs determines the reference voltages V_{ref1-3} derived from Equations 19-21 which are reinterpreted here as:

$$V_{ref1} = V_{dd} - \frac{\omega_{n1} R_6 C_7}{k_{\omega_n}} \tag{27}$$

$$V_{ref2} = V_{dd} - \frac{\omega_{n1} R_7 C_1}{k_{\omega_n}} \tag{28}$$

$$V_{ref3} = V_{dd} - \frac{2\zeta R_{18}}{k_\zeta R_8} \tag{29}$$

Future improvements on the programmable filter should use these equations in software to convert the user's inputs to eight bit values to set the reference voltages.

3.1.3 Computer-DAC Interface

In order to use the signals from the RS232 interface, they need to be converted from RS232 logic to TTL logic. This means that the voltages need to be changed from +12 and −12 volts to 0 and +5 volts. This can done with the use of the Maxim 233 level shifter chip. The Maxim 232 takes RS232 logic and converts it to TTL for transmitting and vice versa for receiving. The only pins needed in this project are the 'transmitted data', 'received data' and 'ground.' With these three pins, a serial interface between the server and the PIC microcontoller is possible. The PIC has an onboard USART, with a receive and a transmit pin, therefore, the only configuration needed can be done in software.

A diagram of the interface is shown in Figure 13. The protocol for checking if the right variable is sent is as follows. The PIC is initally set up to wait for a serially transmitted byte of data, meaning one bit after the other, least significant bit first. There are 8 bits of data, 1 stop bit and no parity. The processor then stores the variable and sends it back to the server. The server is then expected to send back to the PIC a '1' if the data matches and a '0' if it does not. The PIC then checks a counter to determine which variable was sent. The counter only updates when confirmed

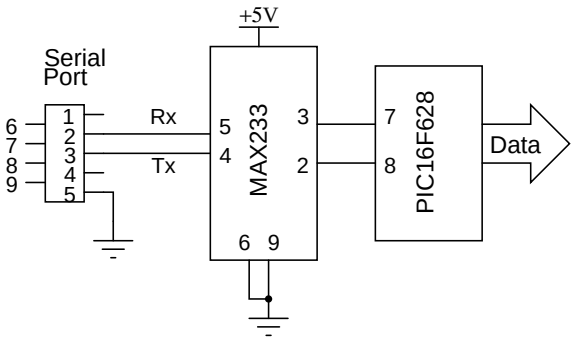

Figure 13: Computer-PIC Interface

values are received. The expected order is ω_{n1}, ω_{n2}, then ζ. When the data is confirmed and determined, it is sent to its respective digital to analog converter through PORTB. A simulation of a conversation between the PIC and the server is shown in words in Table 2. This simulated conversation shows the same process as the flowchart in Figure 14 in pseudo code. Processes done in the background implies that the PIC does not tell the server that it is sending the variable to its corresponding DAC, the server however "knows" because the PIC "asks" for the next variable in the set of variables. The conversation also shows what happens if an error was to occur and how it would be handled.

3.2 Software

The filter's software comes in two interdependent parts. The first part is the PIC microcontroller's program which creates an interface between the signals coming from the server and the state-variable filter itself. The second part is the PHP program which creates an interface between the remote user and the Apache web server. The values operated on by the two parts of the software are interdependent and therefore a "handshaking" mechanism is used. The mechanism is also useful for error checking, making sure that values received by the PIC are values sent by the server. A complete flow chart of the "handshaking" mechanism is shown in Figure 14.

3.2.1 Microcontroller Interface: PIC Program

In order to interpret the signals sent through the serial port by the server, a microcontroller is used. The Microchip PIC16F628 microcontroller was used here because it is cheap, requires few external components, has a Reduced Instruction Set (RISC) processor and has an onboard Universal Synchronous/Asynchronous Receiver/Transmitter module (USART) for serial interfaces.

The only external components required for immediate programming of the PIC microcontroller are: two capacitors and a crystal oscillator. These three components generate the processor's clock. A 20 MHz crystal oscillator was used in this case making the instruction cycle 200 ns (the PIC has an internal divide by four, which makes the clock frequency actually 5 MHz).

The RISC architect means that there are few instructions needed to fully control the processor. In this case, there are only 35 instructions.

Table 2: Simulated PIC-Server Conversation

Server:	Here is ω_{n1}
PIC:	Here is the ω_{n1} you sent,
	Does it match?
Server:	Yes
PIC:	Ready for ω_{n2} now (sent ω_{n1} to ω_{n1}'s DAC in background)
Server:	Here is ω_{n2}
PIC:	Here is the ω_{n2} you sent,
	Does it match?
Server:	Yes
PIC:	Ready for ζ now (sent ω_{n2} to ω_{n2}'s DAC in background)
Server:	Here is ζ
PIC:	Here is the ζ you sent,
	Does it match?
Server:	No (Assuming some error occurred)
	Resynchronizing
PIC:	Waiting for ζ
Server:	Here is ζ
PIC:	Here is the ζ you sent,
	Does it match?
Server:	Yes
PIC:	Waiting for next 3 variables (sent ζ to
	ζ's DAC in background)

The USART allows the processor to communicate to any other device serially. This module was the most important part of this project because it provided a way for the system under test to receive the instructions presented by the webserver.

The flow chart of the server/system interface is presented in Figure 14, along with the actual implementation.

3.2.2 PHP Web Interface

As stated before, in order to set the variables for the filter, control of the server's serial port is required. The PHP scripting language has such capabilities. PHP along with its COM functions allows the control the server's serial port running Windows. COM is a technology which allows the reuse of code written in any language (by any language) using a standard calling convention and hiding behind APIs the implementation details such as what machine the Component is stored on and the executable which houses it.

The COM module that was used for this project is provided by ActivExperts from www.activexperts.com on a trial basis and can be licensed for a fee of $150.00. Alternatives to the COM module, is to use Apache on Linux, where PHP offers functions to directly control the serial

port, or write a Dynamic Linked-Library (.dll) file so that a windows based server can control the serial port. The PHP code that makes use of this module is shown in Appendix B. The code takes the user values entered in a form and prepares them for transmission to the webserver and to its serial port, which is later on interpreted by the PIC processor for setting filter values. There is a "hand-shaking" mechanism implemented such that PHP and the microcontroller have the same values presented by the user.

4 Measurements

To determine if the programmable filter works as expected, step response experiments were performed. The natural frequency, along with the damping ratio were set for each experiment through the use of the PHP web interface and the peak overshoot (P_o) and the time to peak (t_p) were measured. The natural frequency and the damping ratio were then determined from the following equations and compared to the theoretical values for accuracy. For all experiments $\omega_{n1} = \omega_{n2} = \omega_n$. In the measurement tables ω_{nt} and ζ_t are the theoretical values, i.e. set by server, while ω_{nc} and ζ_c are values calculated from measured values of P_o and t_p. The errors are represented by variables ζ_{err} and $\omega_{n_{err}}$. The two trends that should be visible are: the independence of P_o on ω_n and the inverse proportionality between t_p and ω_n. If these trends are observed, the rest is just calibration.

$$P_o = 1 + e^{\frac{-\pi\zeta}{\sqrt{1-\zeta^2}}} \tag{30}$$

$$t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \tag{31}$$

Table 3: Low-Pass Filter Step Response

$f_{nt}(kHz)$	$\omega_{nt}(krad/s)$	ζt	P_o	$t_p(\mu s)$	ζ_c	$\omega_{nc}(krad/s)$	ζ_{err} (%)	$\omega_{n_{err}}$(%)
2	12.566	0.2	1.53	282	0.198	11.365	-1.0	-9.5
		0.4	1.51	284	0.210	11.314	-47.5	-10
		0.6	1.19	259	0.467	13.717	-22.2	9.2
		0.707	1.11	289	0.575	13.286	-18.7	5.7
		0.8	1.17	237	0.491	15.216	-38.6	21.1
		1.0	1.07	274	0.650	15.087	35.0	20.1
4	25.133	0.2	1.54	155	0.192	20.652	-1.0	-17.8
		0.4	1.51	142	0.210	22.627	-47.5	-9.97
		0.6	1.19	123	0.467	28.884	-22.2	15
		0.707	1.11	119	0.575	32.267	-18.7	28.4
		0.8	1.17	122	0.491	29.558	-38.6	17.6
		1.0	1.07	136	0.650	30.396	35.0	20.9
6	37.699	0.2	1.54	85.6	0.192	37.396	-1.0	0.8
		0.4	1.51	89	0.210	36.103	-47.5	-4.23
		0.6	1.19	75.5	0.467	47.056	-22.2	24.8
		0.707	1.11	77.6	0.575	49.481	-18.7	31.25
		0.8	1.17	74.5	0.491	48.404	-38.6	28.4
		1.0	1.07	88	0.650	46.976	35.0	24.6
8	50.265	0.2	1.54	62.2	0.192	51.464	-1.0	2.4
		0.4	1.51	64.8	0.210	49.586	-47.5	-1.35
		0.6	1.19	57.8	0.467	61.465	-22.2	22.3
		0.707	1.11	55	0.575	69.226	-18.7	37.7
		0.8	1.17	53.2	0.491	67.784	-38.6	34.9
		1.0	1.07	57.8	0.650	71.571	35.0	42.4

From the measurements table, it is clear that the two required trends are present, but aside from that, the measurements seem to contain a large amount of error within the center values of ζ. Reasons for this could be a nonlinear G_M of the transconductance amplifiers when they are used in the overall filter, measurement errors, calibration techniques, i.e. rounding the final value to be sent to the DAC to rid the data of leading decimals. This rounding function under PHP will round both 4.5 and 4.99 up to 5. Another problem discovered by the author was the value of k_{ω_n} and k_ζ. The values determined from the test circuit and the values observed in the circuit were not consistent. The values were adjusted on the server side until an expected response was observed, but then, this was not the case at all points of measurement as apparent from the low error in both variables at $\zeta = 0.2$. There were no other problems other than calibration errors, because the signals overall appeared to match the step reponses in Figures 15-17

Some of the step response signals for each output are shown in Figures 18-23. These Figures were taken at a time when ζ was calibrated well by ω_n was not.

5 Future Work

Future work to be completed for the weblab project includes the construction of more systems/filters and the implementation of a graphical user interface (GUI) to control the experiments and to gather data from those experiments. More robust calibration techiques need also to be implemented for this specific filter, most likely could be done in the software on the server side.

Other work includes the construction of more filters/systems for experimentation. Although a state-variable filter is a great example of a low-pass, band-pass and high-pass filter, a more extensive set of filters are required for feedback systems. Filters that are more specific to 6.302 are the lag, lead, and the dominant pole compensators/filters. These filters demonstrate the benefits of feedback to achieve more desirable system dynamics than those of the open-loop or uncompensated system.

6 Conclusion

The programmable state-variable filter is a good start to the installation of a feedback systems weblab, but the first version is not perfect and can be improved upon. For a version 2, a PIC with the USART on a separate port from the data port should be chosen. The PIC16F628 was chosen because it had 2 byte sized ports, an onboard USART, and is inexpensive. The replacement should have an onboard USART separate from the data ports, in addition to the requirements above.

The shared port of the PIC16F628 causes a 2-bit error because the receive pin is not available to send data to the DACs since it is configured as an input for the USART. The receive pin therefore either sits at either 5 volts or ground depending on the output from the level shifter, and cannot be connected to any other potential. A diagram of the connections between the PIC and the level shifter is shown in Figure 25. Since the receive pin cannot be used as an output, data bit DB1 is connected to 5 volts in this case.

The 2 bit error might be acceptable for a first version of the filter but future filters/systems may require higher precision for acceptable use. In its present state, the DAC is referenced to 5 volts therefore a 2 bit error represents an error in voltage of just:

$$V_{error} = \frac{2}{256}5 = 0.039mV \tag{32}$$

which represents an error of 981 rad/s in ω_n and 0.03 in ζ.

Although the two bits of error is important, this was not the greatest of problems. The greatest problem is the calibration of the filter which was discussed before. For although the two bit error caused a maximum of 5.7% error, it does not explain the large 20 to 48% errors observed in the measurements.

A PIC16F628 Assembly Code

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Author: Rayal Johnson                                   ;
; Title: PIC16F628 Programmable State-Variable Filter     ;
;                                                         ;
; Description: Programmable state-variable filter that    ;
; interacts with a PHP web interface to receive, check    ;
; and set values for the filter's natural frequencies     ;
; and the damping ratio/Quality factor.                   ;
;                                                         ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        list p=16f628,f=inhx32
        #include <p16f628.inc> ;Available at microchip.com
        __config 0x3FE1

PC      equ     0x02
CNT     equ     0x20
TEMP    equ     0x21

        org     0x0000
;Initialize porta and portb
start   movlw   0x00
        bcf     STATUS,RP0      ;change to
        bcf     STATUS,RP1      ;bank 0

        movwf   PORTA           ;clear PORTA
        movlw   0x07            ;turn off comparators and
        movwf   CMCON           ;enable pins for I/O
        bsf STATUS,RP0          ;bank 1
        movlw   0x20
        movwf   TRISA           ;set RA<4:0> as outputs
                                ;TRISA<5> always '1'

;Ports are now initialized

;Set up USART
        movlw   b'00000010'     ;set Receive pin as input
        movwf   TRISB           ;all outputs except RX pin
        bcf     OPTION_REG,NOT_RBPU ;enable pull-up resistors
        movlw   0x81            ;set baud rate to 9600
        movwf   SPBRG           ;assuming BRGH =1 (address 99h)
        movlw   b'00100100'     ;bit 6 clear - 8-bit transmission
                                ;bit 5 set - enable transmission
```

```
                                ;bit 4 clear - asynchronous mode
                                ;bit 2 set - high speed
                                ;bits 7,3,1,0 - don't care
        movwf   TXSTA           ;(address 98h)
        bcf     STATUS,RP0      ;bank 0
        movlw   b'10010000'     ;bit 7 set - enable serial port
                                ;bit 6 clear - 8-bit reception
                                ;bit 4 set - enables continuous mode
                                ;bits 5, 3:0 - don't care
        movwf   RCSTA           ;(address 18h)
;USART is now set up

;Programmable Filter

getvars
        movlw   0x00            ;set up counter
        movwf   CNT             ;for variables

rcvgo
        btfss   PIR1,RCIF       ;check receive data
        goto    rcvgo           ;
        movf    RCREG,W         ;move data temporarily
        movwf   TEMP            ;to TEMP, to check with
        movwf   TXREG           ;server, if data matches
check
        btfss   PIR1,RCIF       ;check server for confirmation
        goto    check
        bcf     STATUS,Z        ;
        movlw   0x01            ;check data
        andwf   RCREG           ;for 1 -> for Yes ->data matches
        btfsc   STATUS,Z        ;
        goto    rcvgo           ;data doesn't match, get rid of it
        movf    CNT,W           ;if data matches,
                                ;send to correct DAC
        addwf   PC
        goto    sndDAC1         ;match frequency and damping
        goto    sndDAC2         ;with the correct
        goto    sndDAC3         ;DAC

sndDAC1
        movlw   b'00001001'     ;select DAC1 with PORTA
        movwf   PORTA           ;DAC1 (omega1) selected
        movf    TEMP,W          ;send omega1 to
        movwf   PORTB           ;omega1-DAC
        movlw   0x01            ;set up CNT for
```

```
        movwf   CNT             ;the next variable
        goto    rcvgo

sndDAC2
        movlw   b'00000110'     ;select DAC2 with PORTA
        movwf   PORTA           ;DAC2 (omega2) selected
        movf    TEMP,W          ;send omega2 to
        movwf   PORTB           ;omega2-DAC
        movlw   0x02            ;set up CNT for
        movwf   CNT             ;the next variable
        goto    rcvgo

sndDAC3
        movlw   b'00001000'     ;select DAC3 with PORTA
        movwf   PORTA           ;DAC3 (damping) selected
        movf    TEMP,W          ;send damp to
        movwf   PORTB           ;damp-DAC
        movlw   b'00001010'     ;hold values in DACs by
        movwf   PORTA           ;setting write pins high
        goto    getvars         ;get ready for next user

        end
```

B PHP Code

```
<html>
<head>
<title>Serial Interface Test</title>
</head>
<body>
<?php import_request_variables("p", "svf_");

$vdd_n = 5;
$vdd = 5;
$R_6 = 15000;
$C_7 = 6.8e-8;
$k_1 = 16.92;

$R_7 = 15000;
$C_1 = 6.8e-8;
$k_2 = 16.92;

$R_8 = 200000;
$R_18 = 2.0e6;
$k_3 = 18.46;

$true = 1;
$false = 0;
$status = 1;

$s1 = round((($vdd_n - (($svf_omega1*$R_6*$C_7)/($k_1)))/($vdd))*256);
$s2 = round((($vdd_n - (($svf_omega2*$R_7*$C_1)/($k_2)))/($vdd))*256);
$sd = round((($vdd_n - (($svf_damping*2*$R_18)/($k_3*$R_8)))/($vdd))*256);

$v1 = $s1*(5/256);          //voltages in circuit
$v2 = $s2*(5/256);          //used for testing
$v3 = $sd*(5/256);          //and troubleshooting

$svfvals = array($s1, $s2, $sd);         //store converted data for DACs
$svfret = array();
$obj = new COM("ActivXperts.Comport");
$vers= com_get($obj, 'Version');
$exp = com_get($obj, 'ExpirationDate');

//process form
//set comport ID, baudrate, parity,
//number of databits, stopbits, then open port
```

```
//send info, then close port

// Object Oriented Version

    if (isset($svf_submit) && $svf_action == 'submitted'){

com_set($obj, PortID, 1);          //set port to COM1
com_set($obj, BaudRate, 9600);     //set baud rate to 9600kbps
com_set($obj, DataBits, 8);        //set data bits to 8 bits
com_set($obj, Stopbits, 1);        //set stopbits to 1

$obj->Open();                                  //open serial port

for ($i = 0; $i <= sizeof($svfvals)-1; $i++) {
  $obj->WriteByte($svfvals[$i]);               //Write data to serial port
        if ($obj->ReadByte() == $svfvals[$i]) { //Check if PIC has same value
          $obj->WriteByte($true);              //if so, tell it true
           array_push($svfret, $svfvals[$i]);  //store correct value
        } else { $obj->WriteByte($false);      //if not right, tell it false
           $i=$i-1; }                  //and resynchronize
}
$obj->Close();                                 //close serial port

        if (($svfret[0] == $svfvals[0]) &&     //check values to tell user
            ($svfret[1] == $svfvals[1]) &&
            ($svfret[2] == $svfvals[2])) {
    $errors = ''No errors <br>'';
 } else { $errors = ''There is an error <br>''; }
echo $errors;
echo ''The inputs to the filter are <br>
&#969_1 = $svf_omega1  <br> V(&#969_1) = $v1 <br>
&#969_2 = $svf_omega2  <br> V(&#969_2) = $v2 <br>
&#950 = $svf_damping  <br> V(&#950) = $v3 <br> <br>'';

echo '<a href="index.html">Back</a>';

} else {  //show form
?>
<hr>
<h1>6.302 Web-Based Lab v1.0 Demo</h1>
Expiration date: <?php echo $exp; ?><br>
</hr>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
Inputs <br>
omega1 (&#969_1): <input type="text" name="omega1"><br>
```

```
omega2 (&#969_2): <input type="text" name="omega2"><br>
damping (&#950): <input type="text" name="damping"><br>
<input type="hidden" name="action" value="submitted">
<input type="submit" name="submit" value="Enter information">
</form>
<?php
 }
?>
</body>
</html>
```

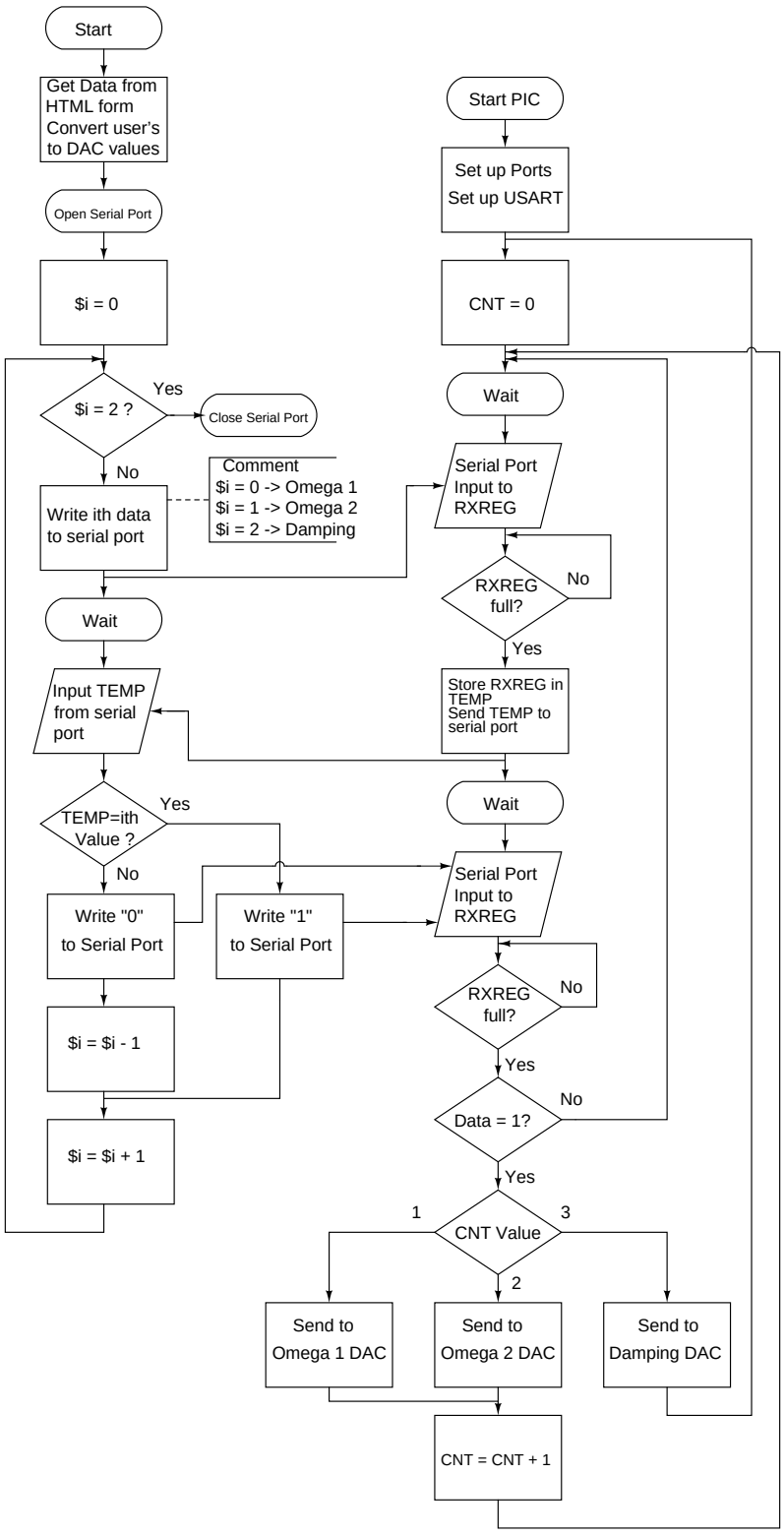

Figure 14: Handshaking Between PIC and Webserver

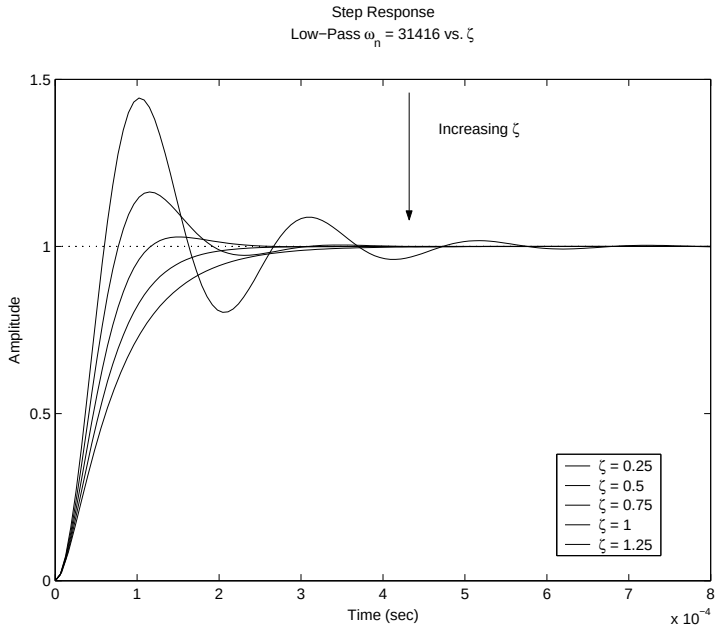

Figure 15: Low-Pass Step Response vs. ζ

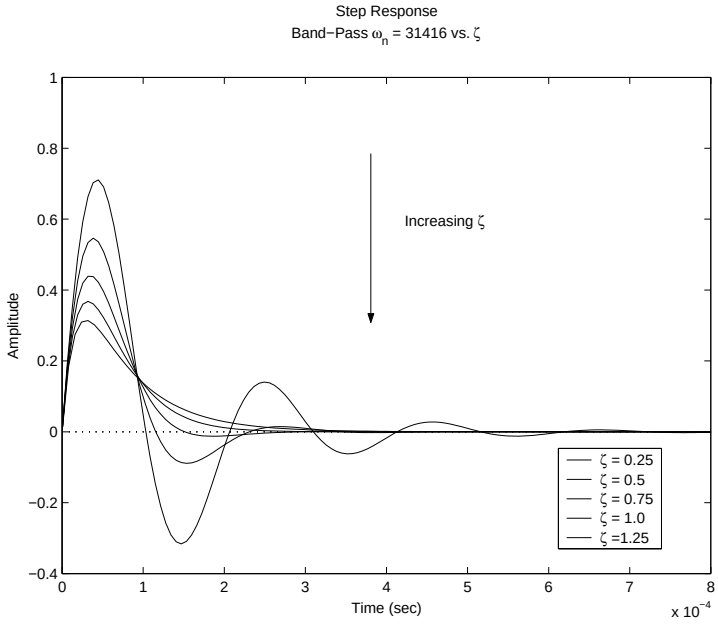

Figure 16: Band-Pass Step Response vs. ζ

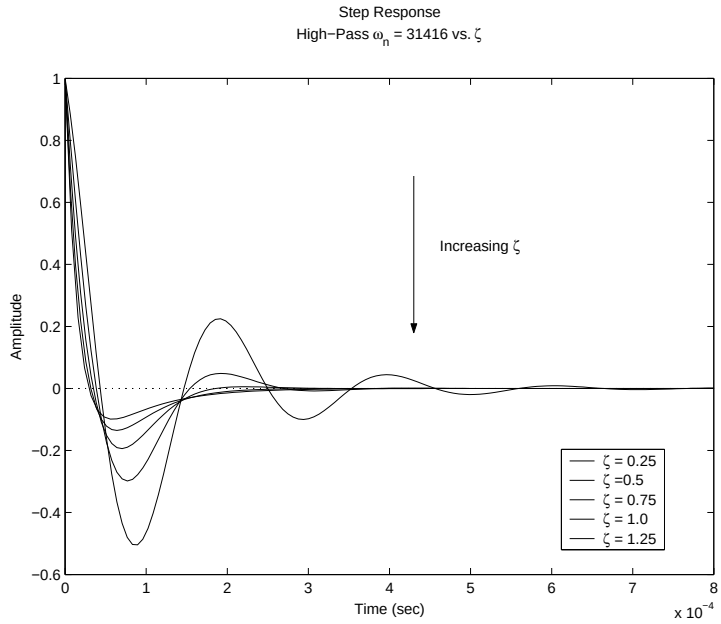

Figure 17: High-Pass Step Response vs. ζ

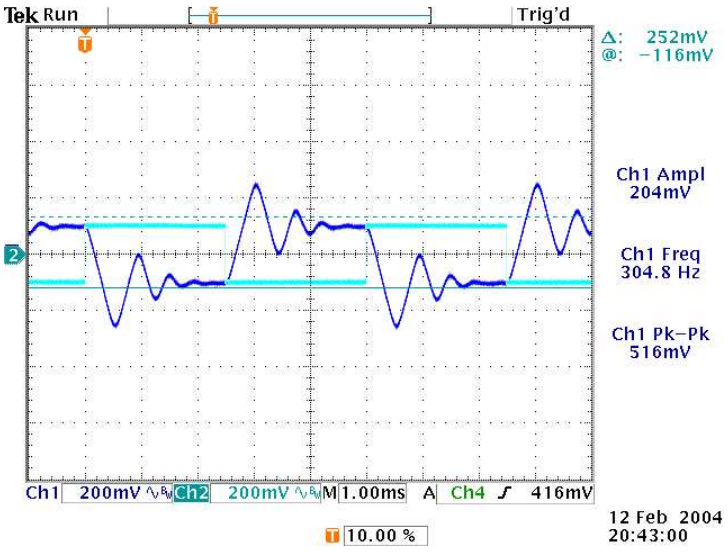

Figure 18: Low-Pass Step Response ($\zeta = 0.2$, $\omega_n = 12566$ rad/s

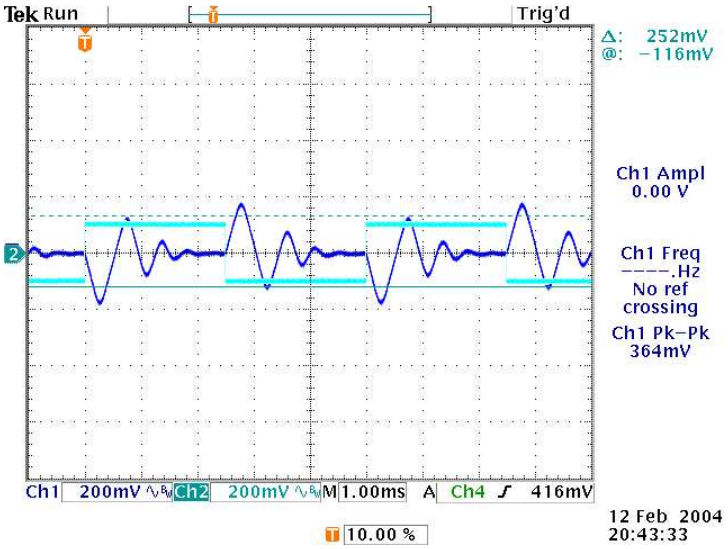

Figure 19: Band-Pass Step Response ($\zeta = 0.2$, $\omega_n = 12566$ rad/s

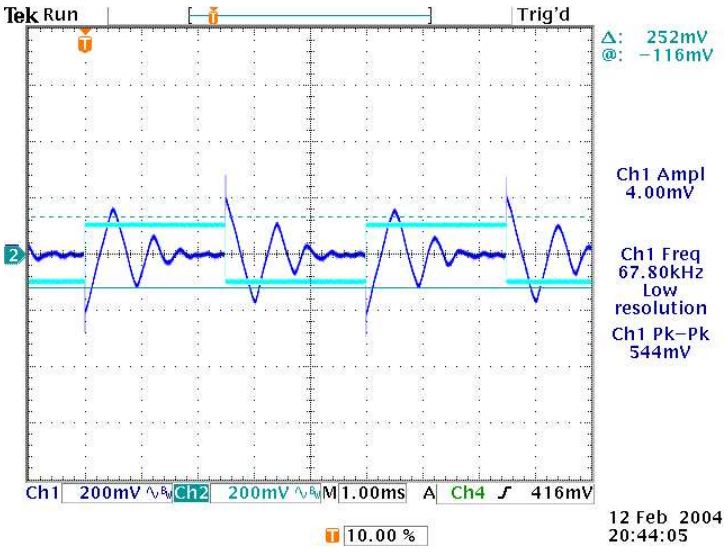

Figure 20: High-Pass Step Response ($\zeta = 0.2$, $\omega_n = 12566$ rad/s

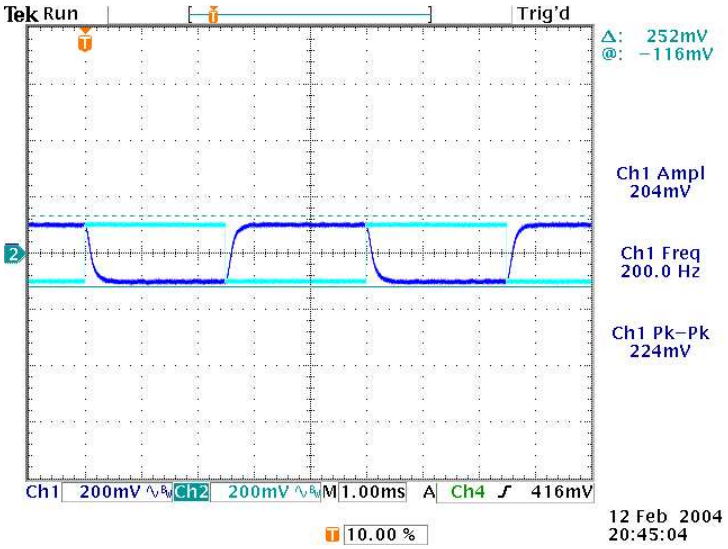

Figure 21: Low-Pass Step Response ($\zeta = 1.0$, $\omega_n = 12566$ rad/s

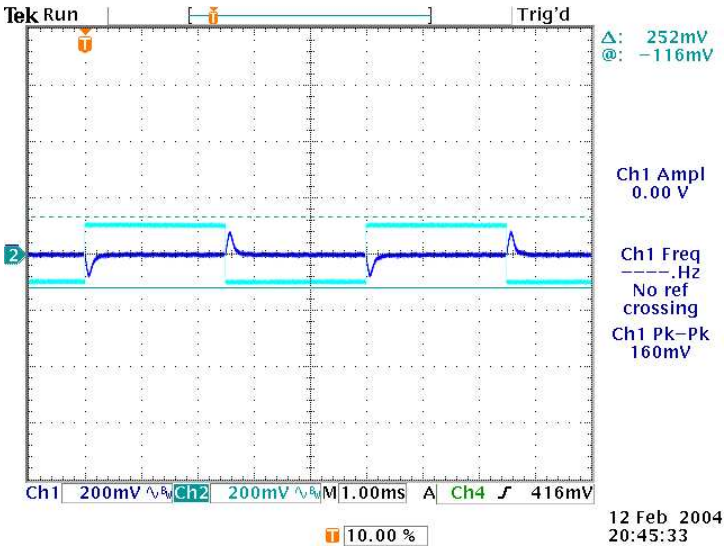

Figure 22: Band-Pass Step Response $\zeta = 1.0$, $\omega_n = 12566$ rad/s

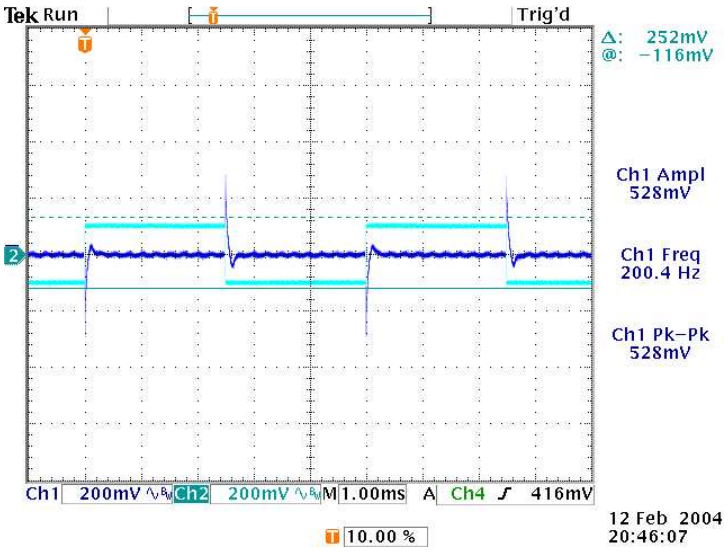

Figure 23: High-Pass Step Response $\zeta = 1.0$, $\omega_n = 12566$ rad/s

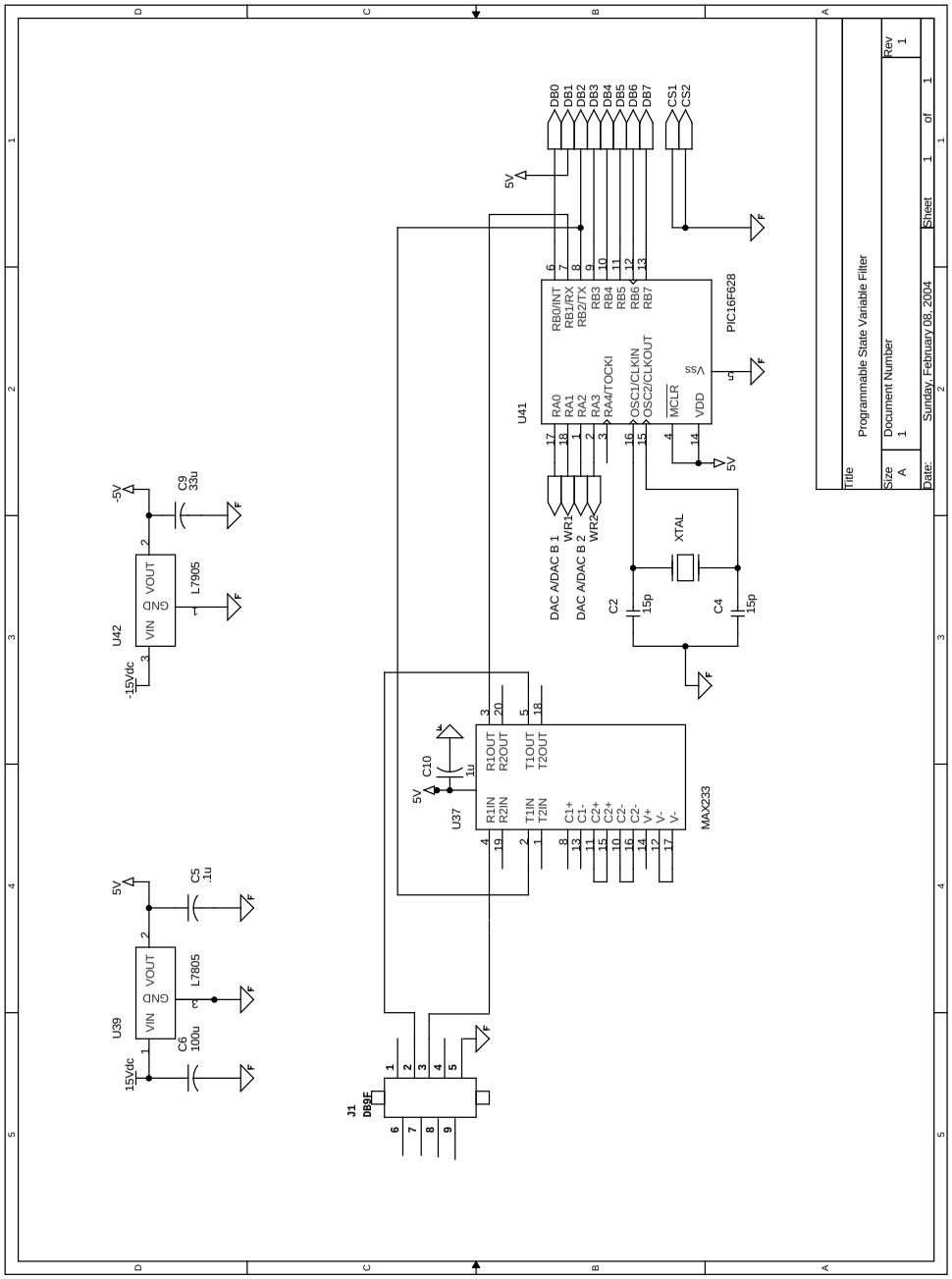

Figure 24: Serial Port/ PIC interface

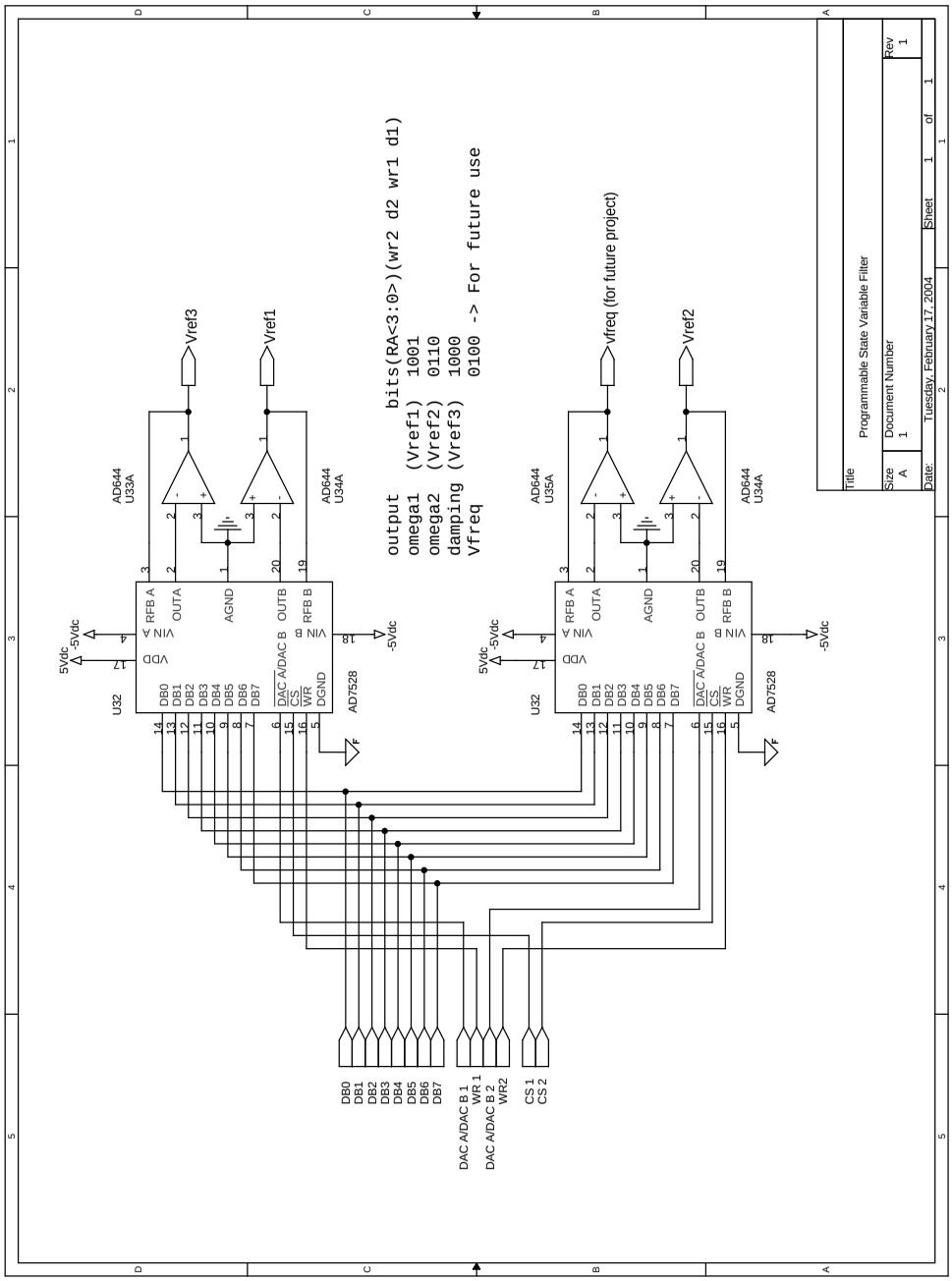

Figure 25: DACs with Data and Write Lines

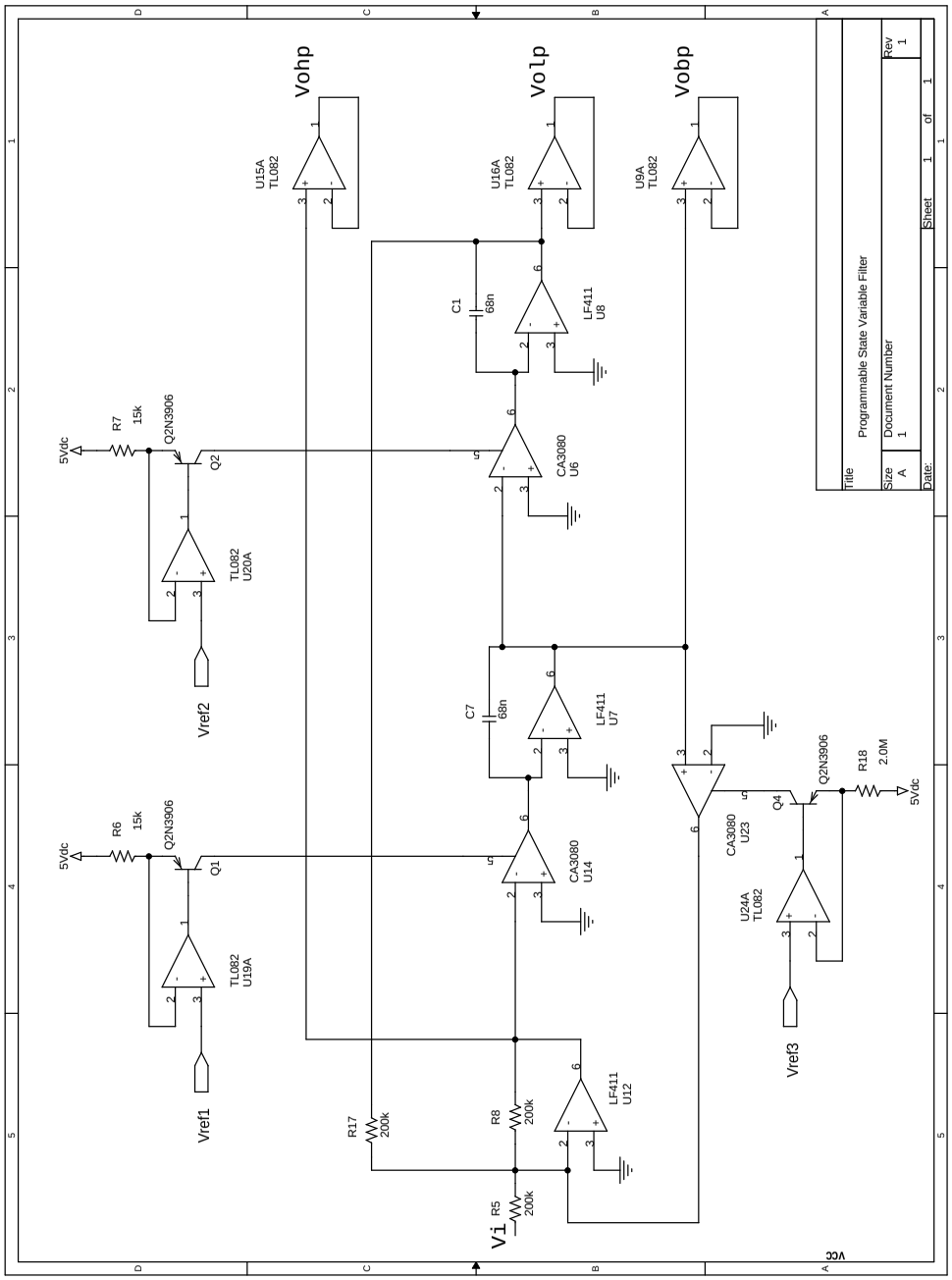

Figure 26: Filter