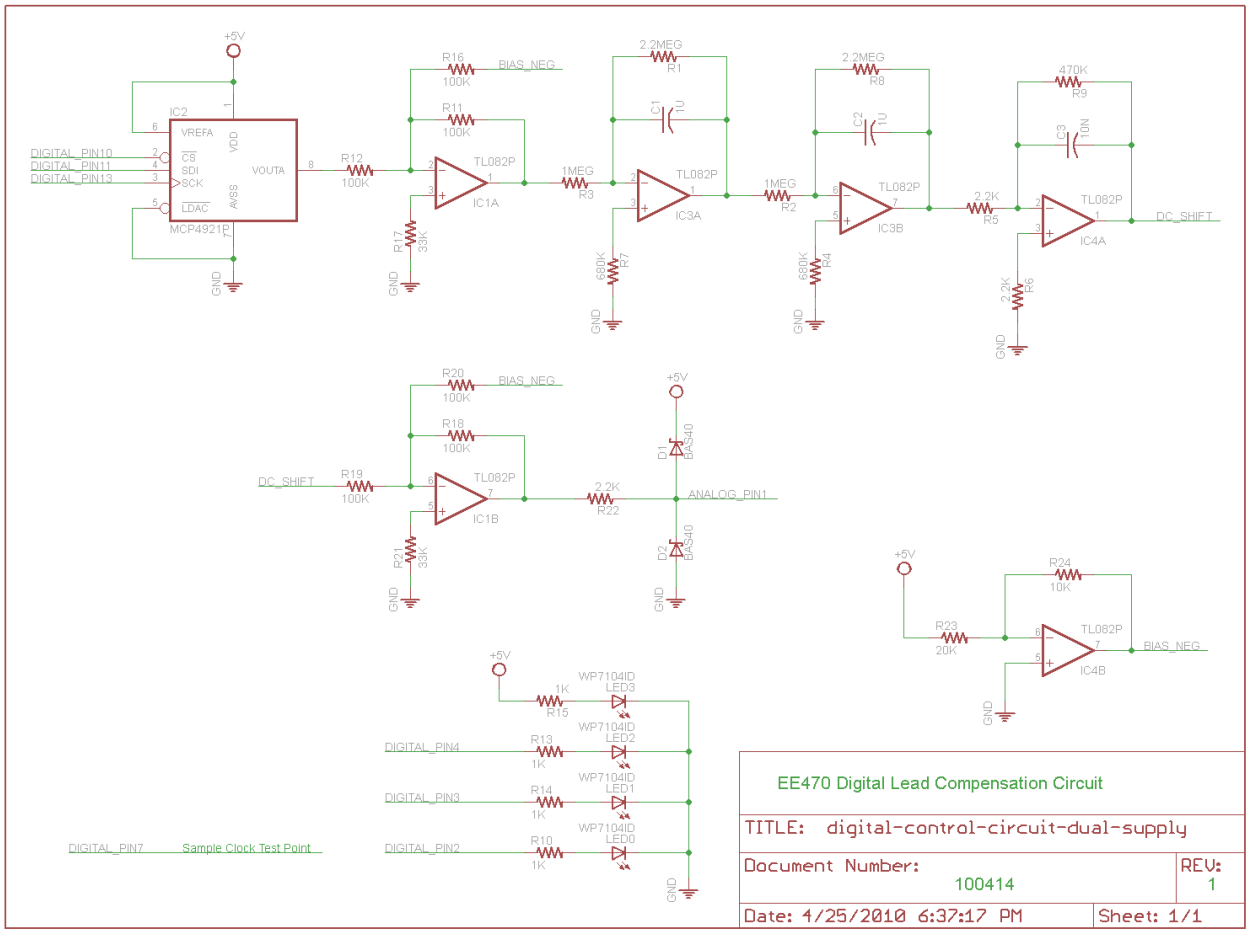
Digital Lead Compensation Circuit

Outline

1.0 Hardware Schematic 1
2.0 Software 2
3.0 Photo Montage 7
Stage 1: 12-bit DAC to Down Level Shift 7
Stage 2: First Integration Stage 8
Stage 3: Second Integration Stage 8
Stage 4: Analog "Gain" Stage..... 9
Stage 5: DC Shift and ADC Protection 10
4.0 Debugging..... 11
4.1 Closing the Loop 16
5.0 Noise Reduction 17
5.1 Ground Loops 17
6.0 Lessons Learned 17

1.0 Hardware Schematic



2.0 Software

To compile the provided code you will need to install the Spi and MsTimer2 folders from the Arduino site to your arduino-00nn/hardware/libraries folder.

Before attempting to control your circuit with the provided code take a few moments and make sure you understand how it works. To help you work through the operation of the software, I have written the following questions. See if you can find the answers.

1. What analog pin on the Arduino is input to the 10-bit ADC of the ATmega328P microcontroller?
2. How do you change the half-cycle period?
3. What is the sample frequency of the digital lead-compensation circuit?
4. How much time is provided for the system to stabilize?
5. What is the name of the C++ function containing the lead-compensation equations?
6. How does this function get called?
7. Why doesn't the loop function include any code?

```
//*************************************************************//
//  Name        : Lead Compensation Digital Control            //
//              : Analog In, 12-bit SPI DAC Microchip MCP4921  //
//  Author      : Gary Hill                                    //
//  Date        : 9 April, 2010                                //
//  Version     : 1.0                                          //
//  Reference(s): http://www.arduino.cc/playground/Code/Spi    //
//  Notes       : Download Spi and MsTimer2 folders and place  //
//              : in arduino-00nn/hardware/libraries folder    //
//*************************************************************

// Digital Compensator Output
// SPI Interface  SS_PIN(PB2), SCK_PIN(PB5), MOSI_PIN(PB3), MISO_PIN
// Arduino Pin    10           13            11             12
// MCP4921 DAC    /CS          SCK           SDI            n/a
// MCP4921 Pin     2            3             4

#include <Spi.h>

#include <MsTimer2.h>

// ATmega328P ADC
const int analogPin = 1;   // analog input channel

// diagnostics
const int triggerPin = 2;  // LED 2 - trigger start of test
const int ledPin2    = 3;  // LED 1
const int ledPin3    = 4;  // LED 0
const int samplePin  = 7;  // 1 kHz sample clock
```

```
volatile unsigned int sample_clk  = LOW;
volatile unsigned int trigger_clk = LOW;

// Op-Amp constants
// LM324
// output voltage swing 0V to V+ - 1.5V
// note: const instructs compiler to save in Flash program memory
const unsigned int OpAmp_max  = 0x0FFF;    // 5v/5v * 2^12 = 0xFFF
const unsigned int OpAmp_min  = 0x0000;

// Experiment
// note: volatile insures variable is saved in SRAM data memory and not a register.
const unsigned int one_half_cycle = 2000;  // number of samples in a half-cycle (1 sec / 1 msec)
volatile unsigned int count = 0;           // current number of samples during this half-cycle

// External circuit wiring
// note: do not need volatile qualifier because variable is not persistent across interrupts
unsigned int dn_in = 0;        // analog input wired to plant output Y
unsigned int dn_out = 0;       // analog output wired to plant input U

// Difference circuit constants
// all constants and variables are 32-bit real numbers
// Arduino implements data type double as float
const double v_dn = 5.0 / 4096.0;    // dn to voltage conversion factor
const double dn_v = 4096.0 / 5.0;    // voltage to dn conversion factor
const double zero_offset = 2.5;      // 2.5 v
const double plus_one  = + 1.000;    // +/- 1000 mV swing around ground
const double minus_one = - 1.000;

// Un = ((2-bT)Un-1 + K(2+aT)En + K(aT - 2En-1))/(2+bT), where a = 1 and b = 10
const double T = 1.0/1000.0;  // 1 millisecond (f = 1 kHz)
const double k = 1.0;         // gain
const double A = (2.0-10.0*T)/(2.0+10.0*T);  // first constant  (A = 0.99)
const double B = k*(2.0+T)/(2.0+10.0*T);     // second constant (B = 0.9955)
const double C = k*(T-2.0)/(2.0+10.0*T);     // third constant  (C = 0.9945)

// Difference circuit variables
// volatile probably not required due to double binary format
double R = minus_one;     // system input to digital control network
double zero_adjust = 0.0; // experimentally determined during setup
double Y  = 0.0;          // plant output (input to summing junction)
double E1 = 0.0;          // input to difference circuit at time t
double E0 = 0.0;          // input to difference circuit at time t-1
double U1 = 0.0;          // output of difference circuit at time t
double U0 = 0.0;          // output of difference circuit at time t-1

// Timer 2 ISR
// We have 1 msec to execute this ISR.
void sample() {

  // generate sample clk
```

```
  sample_clk = !sample_clk;
  digitalWrite(samplePin, sample_clk);

  // loop manager
  count++;
  if (count > one_half_cycle){
    count = 0;

    trigger_clk = !trigger_clk;
    digitalWrite(triggerPin, trigger_clk);

    if (R == plus_one){
      R = minus_one;
    }
    else{
      R = plus_one;
    }
  }
  digitalWrite(samplePin, sample_clk);

  // digital control circuit

  // input (0.125 msec)
  dn_in = readADC();      // unipolar ADC 12-bit digital number
    //Serial.println(dn_in,HEX);
  Y = ((double)dn_in)*v_dn - zero_offset;   // convert input to +/- voltage

  // summing junction with zero adjust
  E1 = -(R - Y);
  // E1 -= zero_adjust;

  // lead-compensation difference equation (approx. 0.174 msec)
  U1 = A*U0 + B*E1 + C*E0;

  // update variables for next iteration
  U0 = U1;            // save current output as previous output
  E0 = E1;            // save current error as previous error

  // shift the dc level
  //U1 = R;              // Enable for testing (Warning: may cause overheating and damage to the
Plant Op-Amps)
  double U1_unipolar = U1 + zero_offset;      // convert output to + voltage (unipolar)
  dn_out = (unsigned int)(U1_unipolar*dn_v);  // convert to digital number

  // output to DAC with limit check
  spiDAC(dn_out, SS_PIN);
}

void setup() {
  // open serial port, set data rate to 9600 bps
  Serial.begin(9600);
```

```
  MsTimer2::set(1, sample); // 1 ms period

  //set pin(s) to input and output
  pinMode(ledPin3, OUTPUT);        // red indicator LED - digital pin 4
  pinMode(ledPin2, OUTPUT);        // red indicator LED - digital pin 5
  pinMode(triggerPin, OUTPUT);     // red indicator LED - digital pin 2
  pinMode(samplePin, OUTPUT);      // sample period (oscilliscope test point) - digital pin 7


  pinMode(analogPin, INPUT);       // plant output (digital control input) - analog pin 1

  Serial.println("waiting for system to stabilize");

  // Wait 3 seconds to let system stabilize
  // Christmas tree sequence
  digitalWrite(ledPin3, HIGH);
  delay(500);
  digitalWrite(ledPin2, HIGH);
  delay(500);
  digitalWrite(triggerPin, HIGH);
  delay(1000);
  digitalWrite(triggerPin, LOW);
  delay(500);
  digitalWrite(ledPin2, LOW);
  delay(500);
  digitalWrite(ledPin3, LOW);

  // measure and adjust zero
  // zeroOffset();

  MsTimer2::start();
}

void loop() {
}

void zeroOffset(){
  // this feature is currently not implemented
  zero_adjust = 0.0;
}

unsigned int readADC()
{
  unsigned int analogValue;
  // read the value from the sensor:
  analogValue = analogRead(analogPin);   // comment out this line to test DAC
  // Y = 0x0200;               // 0x03FF = Vref, 0x0200 = 1/2 Vref, 0x0000 = 0
  analogValue = analogValue << 2 ;       // 10 bit ADC to 12-bit DAC word

  // check limits
```

```
  if (analogValue > OpAmp_max) {
    analogValue = OpAmp_max;
    digitalWrite(ledPin3, HIGH);
  }
  else if (analogValue < OpAmp_min) {
    analogValue = OpAmp_min;
    digitalWrite(ledPin2, HIGH);
  }
  else {
    digitalWrite(ledPin3, LOW);
    digitalWrite(ledPin2, LOW);
  }

  return analogValue;
}

void spiDAC(unsigned int _word, int _ss_pin)
{
  // Byte of data to output to DAC
  byte data;

  // check limits
  if (_word > OpAmp_max) {
    _word = OpAmp_max;
    digitalWrite(ledPin3, HIGH);
  }
  else if (_word < OpAmp_min) {
    _word = OpAmp_min;
    // Serial.print("DAC output less than OpAmp minimum = ");
    // Serial.println(_word, HEX);
  }
  else {
    digitalWrite(ledPin3, LOW);
  }

  // set SS pin low, beginning 16-bit (2 byte) data transfer to DAC
  digitalWrite(_ss_pin, LOW);
  // send high byte
  data = highByte(_word);
  data = 0b00001111 & data;     // clear 4-bit command field (optional)
  data = 0b00110000 | data;     // set command: 0 = DACA, 0 = buffered, 1 = 1x, 1 = output buffer
enabled
  Spi.transfer(data);           // alternate: shiftOut(MOSI, SCK, MSBFIYST, data);
  // send low byte
  data = lowByte(_word);
  Spi.transfer(data);           // alternate: shiftOut(MOSI, SCK, MSBFIYST, data);
  // set SS pin high, completing 16-bit transfer to DAC
  digitalWrite(_ss_pin, HIGH);
}
```

3.0 Photo Montage

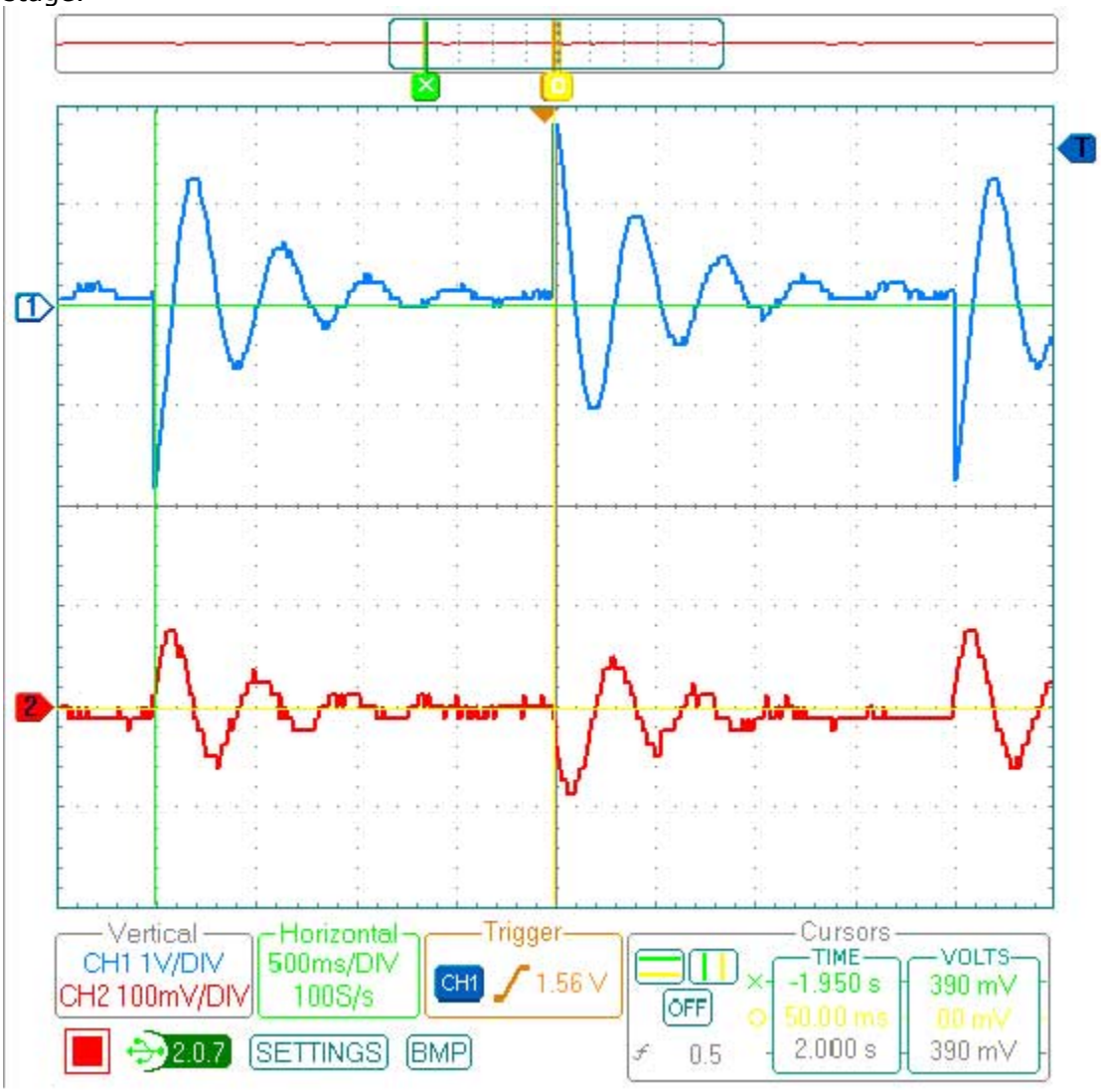
Here is a photo montage showing the input and output of each stage of my digital lead compensation circuit. In all photos the input to the stage is in **blue** at the top of the digital oscilloscope display and output in **red** at the bottom.

Stage 1: 12-bit DAC to Down Level Shift

In the first stage the input (blue waveform) to our analog system is the output of the unipolar 12-bit Digital-to-Analog converter. The stage comprises the Down Level Shift Op-Amp circuit. The yellow line is 0 volts for both the input and output (also indicated by the overlapping red and blue chevrons to the left of the display) . The green line is 5 volts. As you can see the input zero point is at 2.5 v and the waveform is always greater than 0v and less than 5v. The output (red waveform) shows that the unipolar input signal has now been down shifted by 2.5 v, with the zero point now at 0 v.

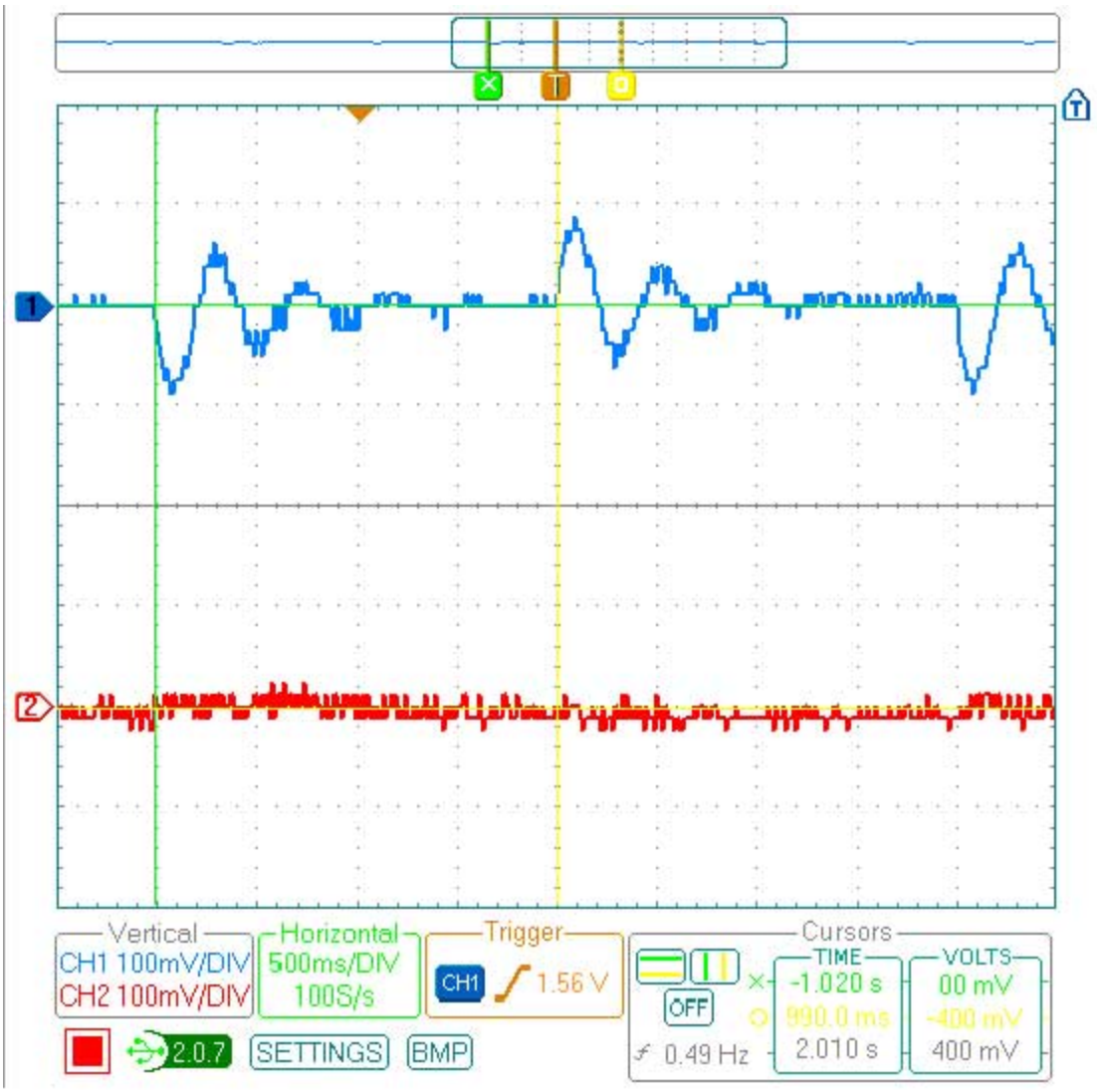
Stage 2: First Integration Stage

The bipolar control signal U(s) now enters the first integration stage of the plant. The green line is the 0v line for the input (blue chevron at left) and the yellow line is the 0v line for the output (red chevron at left). As in the first stage, our vertical resolution is set to 1 v/div for the input (blue waveform). In contrast, there are now 100 mV/div for the output (red waveform) of the first integration stage, reflecting the attenuation from our integration stage.



Stage 3: Second Integration Stage

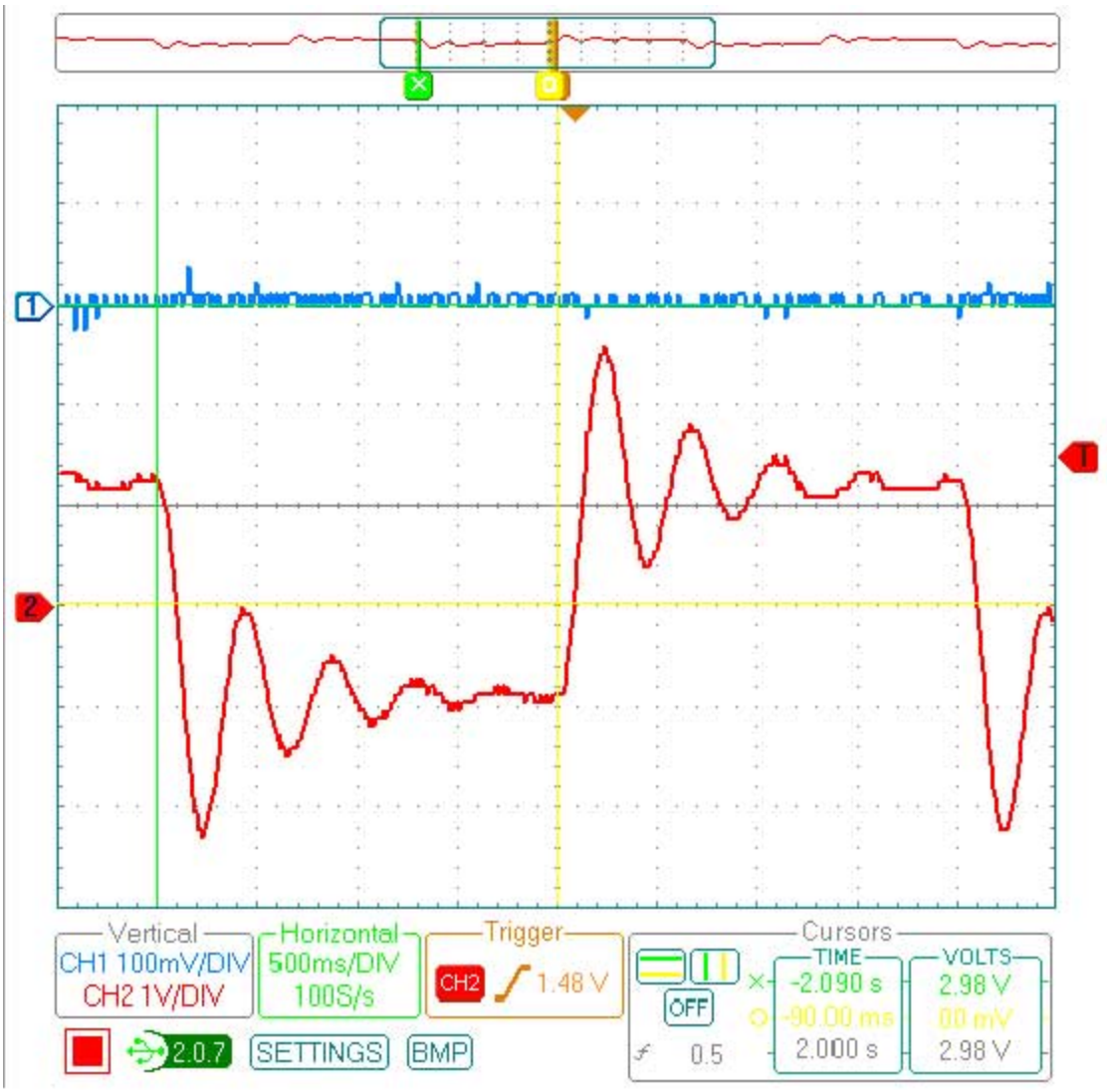
Our input is now the output of the first integration stage. Now shown in blue, the vertical resolution is still set to 100 mV/div. The output is from the second integration stage and is the output of the plant. This signal is also viewed at 100 mV/div which is the lowest setting available on my Parallax USB Oscilloscope. As you can see for my scope, any output signal is lost in the noise. Clearly, we need to separate the signal from the noise before we can send the output of the plant to the 10-bit ADC of the ATmega328P.



Stage 4: Analog "Gain" Stage

With a gain factor of 261, our Op-amp circuit has successfully separated the output of the plant from the noise. While the vertical input resolution is still set at 100 mV/div, the output is now back to 1 v/div, with the signal not exceeding 5 v p-p.

Important: The design of the gain stage is critically important to the success of your circuit. If your circuit is to work, close consideration must be made with respect to the circuit gain, compensation resistor, bi-pass capacitors, and filter capacitor(s).



Stage 5: DC Shift and ADC Protection

In this final stage a 2.5v DC bias is summed with the output of our gain stage (shown in blue). The vertical resolution of both signals is 2 v/div. The 0v line for the output is the at the bottom of the display (red chevron). The yellow line is 5v. As you can see the output of the plant is now amplified and dc biased so it nicely oscillates within the 0 to 5v range of the ATmega328P's 10-bit unipolar ADC - and this gives our digital compensation network something to work with.

4.0 Debugging

Help! Pictures of the working circuit are nice, but my circuit is not working. To help you debug your circuit lets start with the complete transfer function (input-to-output) of our working circuit. Both channel's vertical resolutions are set to 1 v/div, and both the green and yellow lines are set at the 2.5 v point of their respective channels. As you can see the input to the circuit (lead-compensation output U(s)) has minimum and maximum values of approximately 0.5v to 4.5v (2.5 v -/+ 2 v). As expected, the steady state values of the output circuit Y(s) are 1.5v and 3.5v (2.5v -/+ 1 v) respectively.

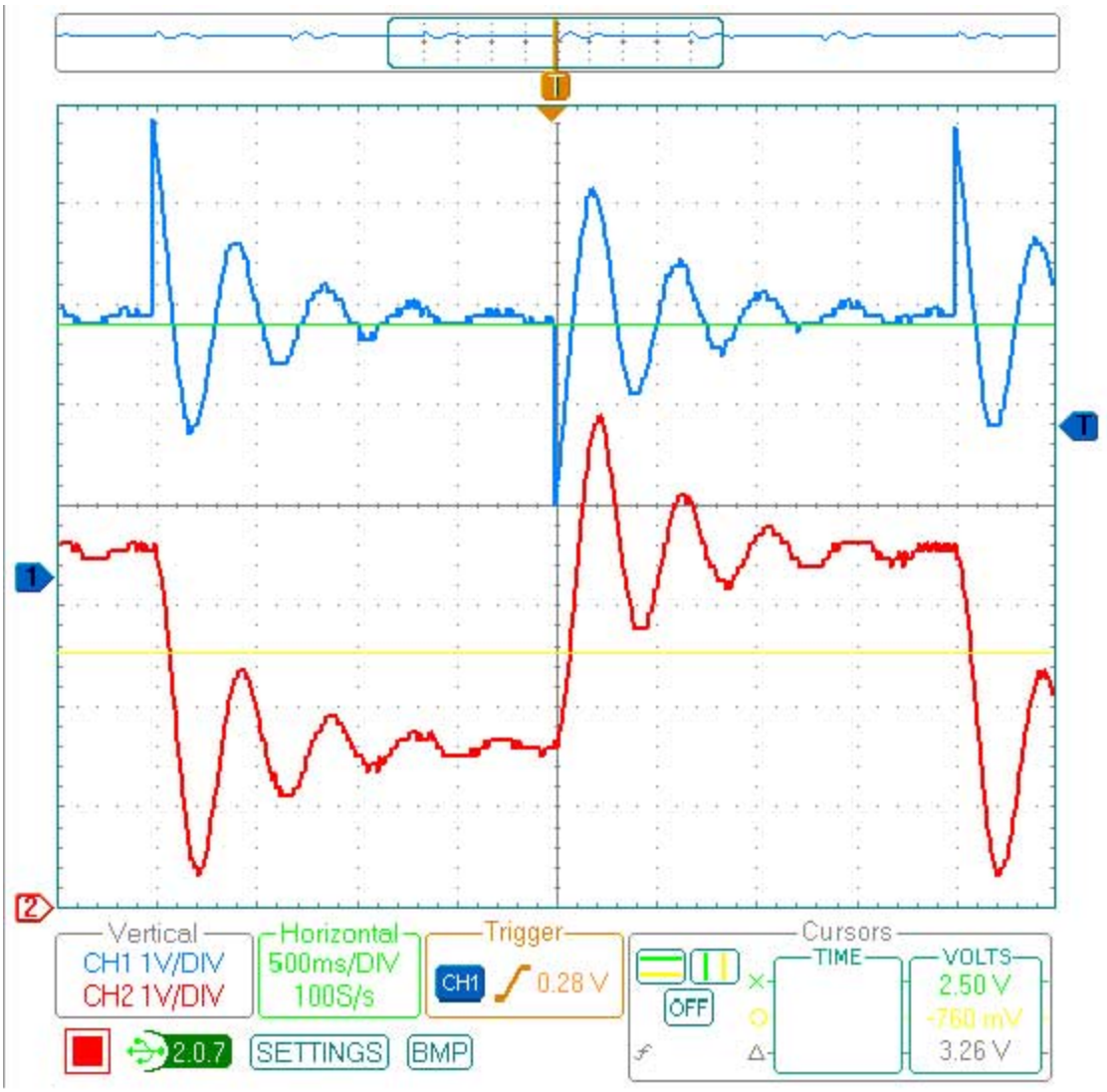


Figure 4.0 Input and Output Waveforms for a Working Digital Lead Compensation Network

In the next figure I have grounded the input to the 10-bit ADC of the ATmega328P. Specifically, Arduino analog pin 1. As you know this is the input Y(s) to the digital lead compensation network. I have not changed any of the scope settings from the working version. Both channel's vertical resolutions are set to 1 v/div, and both the green and yellow lines are still set at the 2.5 v point of their respective channels. As I debugged my circuit, I became very familiar with these waveforms. The blue input waveform, with its characteristic exponential function and the saturated red output signal.

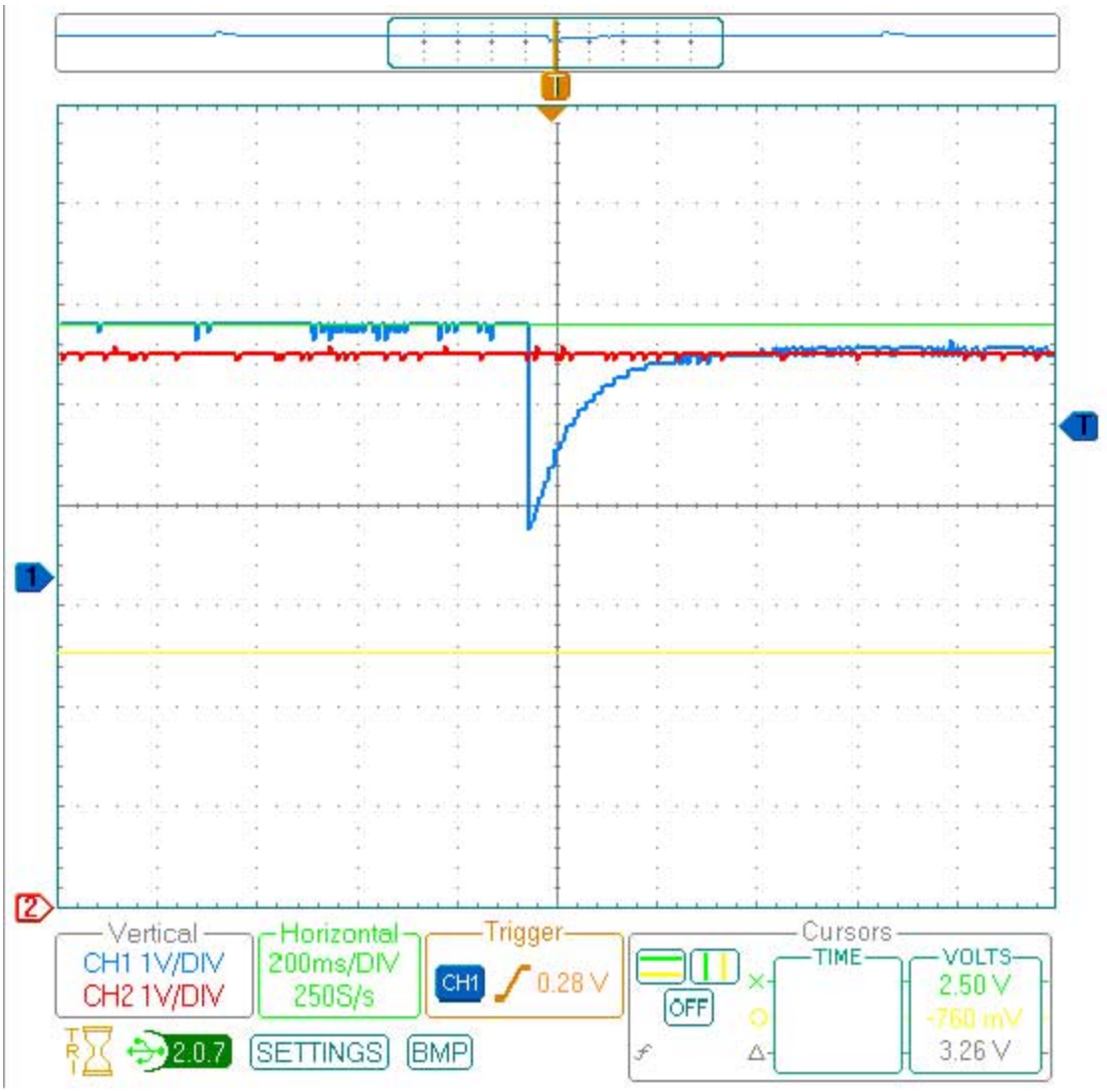


Figure 4.1 Input and Output Waveforms with input to ADC grounded

So how do we isolate the problem? Debugging a control circuit is difficult because everything is interconnected. Software bugs effect the performance of the hardware and vice versa. In our case, I am providing the software which helps; but, you really need to understand the software in order to correctly interface it with your hardware - so take a few moments and make sure you understand the code.

To isolate your hardware problem we begin by implementing an open-loop solution. This breaks the interdependency between the hardware and software. Once completed, the following steps will result in the Arduino outputting a 1 Hz square wave going from 1.5 to 3.5v p-p.

1. Remove the wire connecting the output of the circuit to Analog Pin 1 on the Arduino. Now add a wire from ground to Analog Pin 1. Make sure you have not just grounded the output of your circuit - we only want the input of the Arduino to think the output is 0v.
2. Decrease the Analog sample frequency from 2000 ms to 500 ms. It should now look like this.

```
const unsigned int one_half_cycle = 500;
```

3. Remove the comments from the line assigning R to the output variable of the difference equation U1. It should now look like this.

```
U1 = R
```

4. Set the gain of your analog "Gain" stage between 20 and 40. For my circuit this meant changing the 2.2 K resistor feeding the gain stage to 12 K. My feedback resistor was 470 K.

Now we can check the operation of each stage. First verify that the input square wave is dc biased to 0 v by the level shifter. Specifically, the output of this stage should be a 1 Hz square wave going from -1.0v to 1.0v.

Now lets check the operation of the first integrator (Figure 4.2). In blue is the output of the DAC. If you followed the steps above it should be a 1 Hz square wave going from 1.5 to 3.5v p-p. In red is the output of the first integrator. Notice that the vertical resolution of the input is 1 V/DIV, while the vertical resolution of the output is only 100 mV/DIV reflecting significant attenuation of the input signal by the integrator.

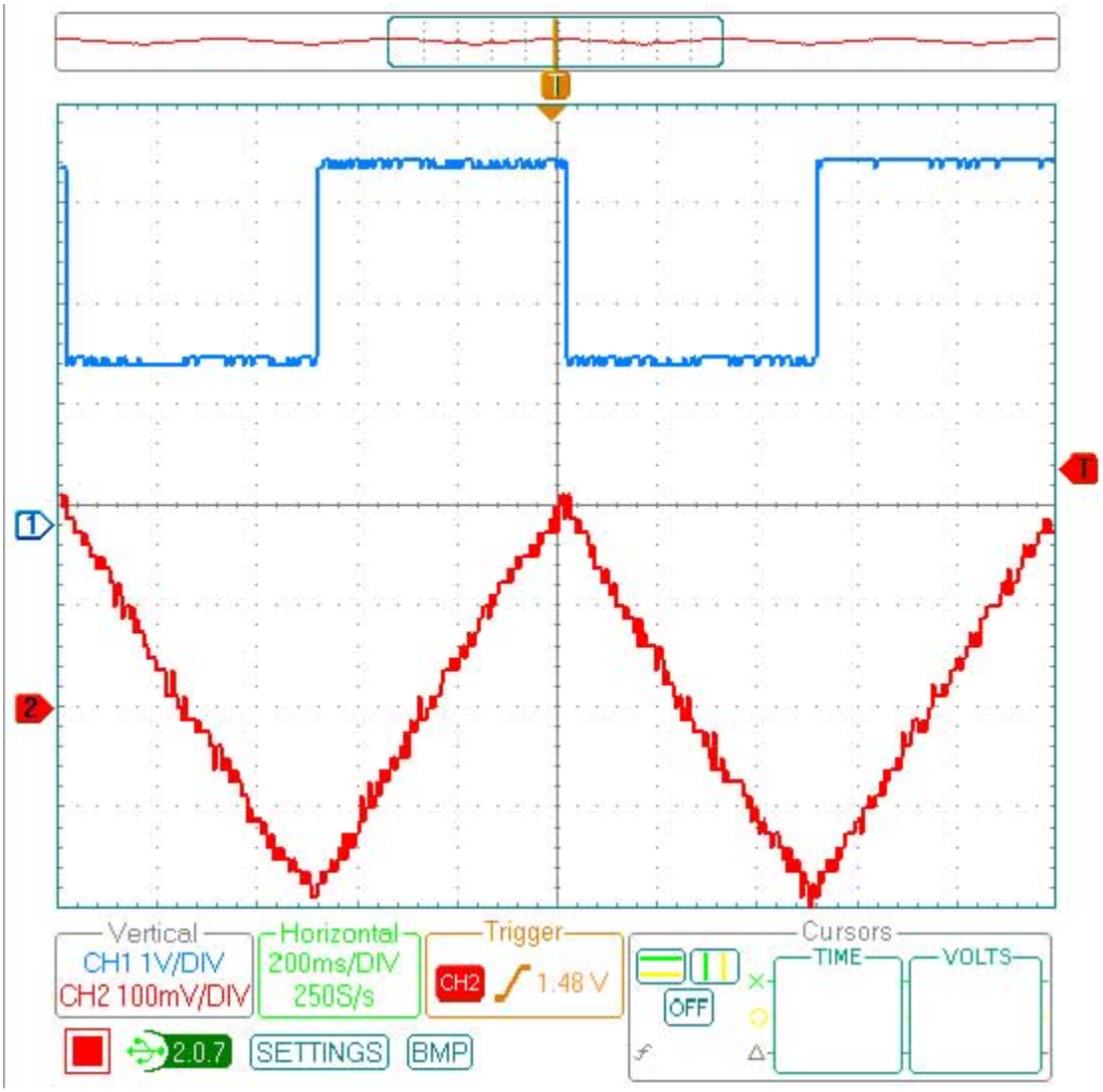


Figure 4.2 Testing the output of the first integration stage

If you are interested, my first attempt at building the circuit failed this test. *The cause was a wiring error between the second and first integration stage resulting in my op-amp for a short period of time becoming a toaster, followed shortly thereafter as a piece of non-functional art.*

Once you have verified correct operation of the first integration stage, lets move the second. In blue is our 1 Hz square wave going from 1.5 to 3.5v p-p. In red is the output of the second integrator. The output is 100 mV/DIV, the minimum vertical resolution available on my inexpensive digital scope. Even with the noise you can see a nice x^2 function, which looks a lot like a sine wave.

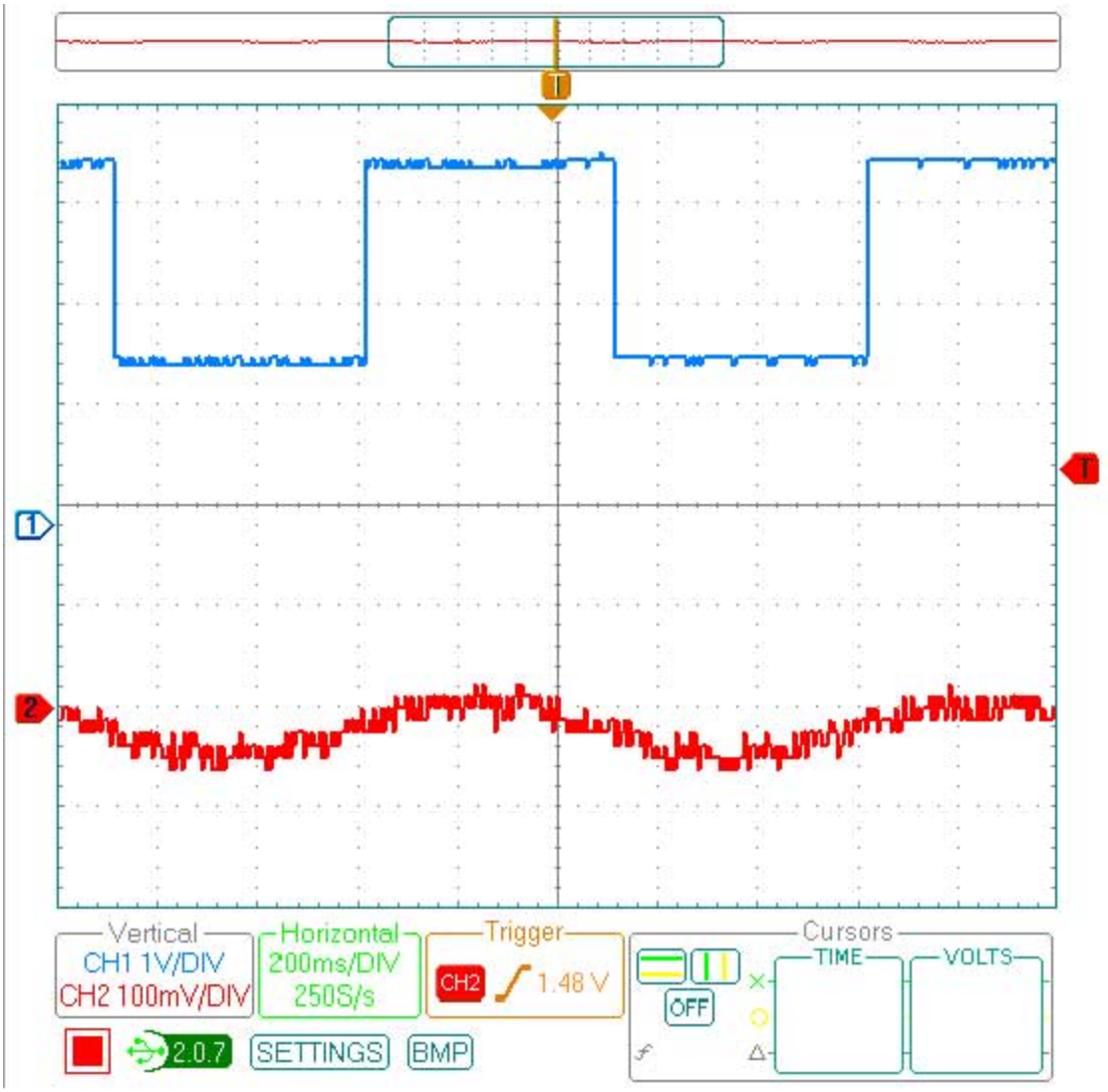
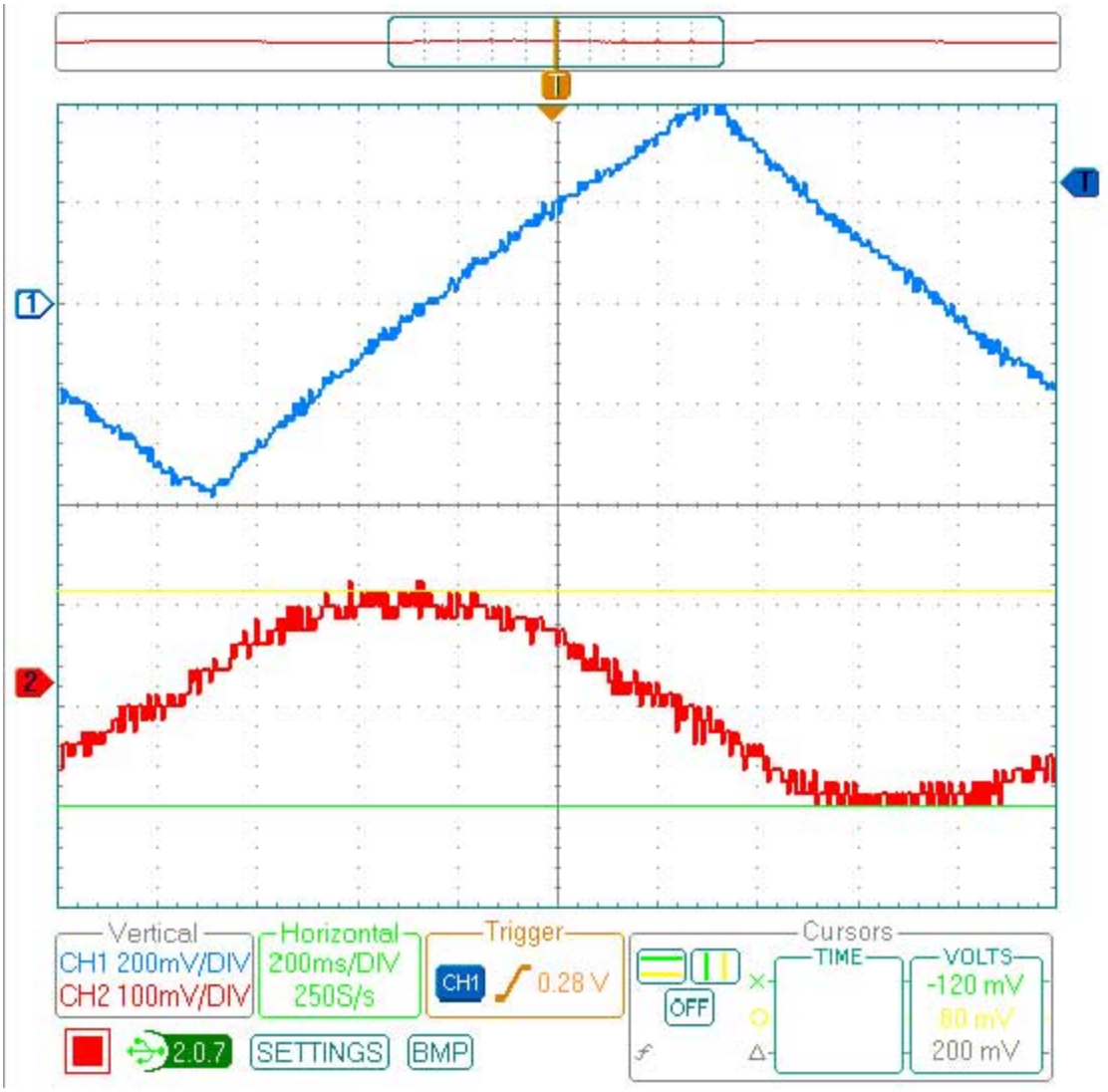


Figure 4.2 Testing the output of the second integration stage

If you like, you can increase the magnitude of these integration stages by simply decreasing the input frequency of the square wave. For example from 1 Hz down to 0.5 Hz as shown in Figure 4.3. Notice the vertical resolution of first integration stage is now set to 200 mV/DIV. Warning: if it wasn't before the output of your gain stage is now definitely hitting the rails now. The yellow line is set to 80mV while the green line is set to a value of -120mv. Thus

my second integrator has an amplitude of 200 mV p-p with a unwanted DC bias of -20 mV. If your circuit has a significant dc bias you may want to consider adding a summing junction. Because my circuit was now operational, I did not correct this dc bias (see earlier discussion).



4.1 Closing the Loop

Once you have the test circuit working it is time to put it all back the way it was (i.e., close the loop).

1. Set the gain of your analog "Gain" stage back between 100 and 200. For my circuit this meant changing the 12 K resistor feeding the gain stage back to the 2.2 K. My feedback resistor was 470 K.
2. Add the comments from the line assigning R to the output variable of the difference equation U1. It should now look like this.

```
// U1 = R
```
3. Increase the Analog sample frequency from 500 ms back to 2000 ms. It should now look like this.
```
const unsigned int one_half_cycle = 2000;
```
4. Remove the wire connecting the input of Analog Pin 1 to ground. Now add a wire from the output of your circuit to Analog Pin 1. Make sure you have not just grounded the output of your circuit.
5. Place one of your scope probes to the output to verify that the input to the ADC is between 0 and 5 volts.
6. Another point is that it may take your closed loop circuit a long time to settle (sometimes measured in 10s of seconds) so be a little patient.

5.0 Noise Reduction

Noise reduction is an art form unto itself. For my circuit a significant amount of noise was removed by adding the following capacitors. Unless otherwise indicated all capacitors are ceramic.

- If you have not done so already begin by adding 0.1 uF bi-pass capacitors between the V+ and V- supply inputs and ground for each of your op-amps.
- Next add 10 uF *tantalum* capacitors between the V+ and V- power supply rails and ground on the breadboard.
- A .01 uF capacitor across the feedback resistor (470 K) of my gain stage reduced a significant amount of noise.

5.1 Ground Loops

In the last noise example (Section 5.0) a ground loop is expected, but never proven. Unless you have isolated power sources and measurement equipment (which I do not), *always use a common ground*. For my circuit that means my Arduino and analog plant are powered by my laptop batteries. Plus, my Parallax USB oscilloscope is also wired to my laptop.

Finally, a special shout out to Scott for all his help debugging and *fine-tuning* my digital lead compensation circuit. Any noise shown in the images above are the sole responsibility of the author.

6.0 Lessons Learned

Here is a compilation of all the problems encountered, that I know of, in the development of this lab. Use this as a check list if you are having problems.

- OpAmps are important. One of the advantages of a closed loop system is it can improve system performance in spite of inexpensive parts. Having said that, starting with inexpensive op-amps can introduce problems that could have been avoided. The 741 and its variants is an example of a good general purpose part that is not optimized for our application (low level dc signals). The LM358 is better but still not recommended. For many students this part introduced a significant dc bias, especially when the stage included a gain factor. Here are some solutions.

    - Select an op-amp with a low input bias current, input offset voltage and thermal stability. When comparing specifications between op-amps you may want to use the Analog Devices OP27 and especially the AD797 as the gold standards. I would not not recommend buying either due to their cost. Begin your search by looking for op-amps with an input bias current specification of 100 nA or less. One quick way to identify this type of op-amp is to look for ones with a JFET input stage. For example, the TL082 has an input bias specification of 50 pA. Another JFET op-amp is the LF356.
    - Next look for a small input offset voltage, typically 15 mV or less.
    - Find an op-amp with good thermal stability. One symptom of this problem is the "it was working fine and now everything is broken." If this sounds like you, then poor thermal stability of your op-amp may be the answer. This is yet another problem with using the 741 op-amp.
    - Avoid quad op-amp packages. Dual op-amp ICs represent a good compromise between cost, packaging density, and performance.
- Have a large number of discrete resistors and bypass/filter capacitors on hand. Radio Shack sells a great resistor variety pack at a very reasonable price. Unfortunately, their capacitor variety package is a complete waste of money. For capacitors, I would recommend purchasing an assortment of 0.01 and 0.1 uF ceramic, 1.0 and 10.0 uF tantalum capacitors.
- Have a few variable resistors on hand to allow you to experimentally tune your circuit and observe closed-loop performance with different gains.
- In the debug section of this document I provided step-by-step instructions for debugging an open-loop version of your circuit. While this is important, a closed loop system, if working properly, may actually remove some of these errors. For example, while the use of a Miller integrator (large resistance placed in the feedback loop) is required to get an open-loop solution to operate, it is not required for the closed loop circuit. If you try and debug an open loop version of your circuit without these resistors you will see the output slowly drift to a rail. In a similar way a small dc bias can simply go away once the circuit is closed. I still recommend getting an open-loop version working first, because debugging a closed loop circuit is extremely difficult.
- Many students simply used the software provided without understanding how it worked. I believe the idea is that if it works don't change it. Many students lost days if not weeks making this assumption. Here are two common mistakes.
    - You need to specify which analog pin the output of your plant is wired to (after the gain and level shifter).
    - The program assumes a phase for the input signal. A simple change in the sign of the digital implementation of the summing junction `(E1 = -(R - Y);)` can be all it takes to make the circuit work.
- Do not introduce gain in your first integration stage. Any dc bias introduced at this early stage will also be amplified and then integrated in the next integrator.
- Adding a little gain factor, on the order of x2 or x4 to your second stage can actually help provide a cleaner signal to your gain stage. It can again add a magnified dc bias, so you may want to introduce a way of manually removing this dc bias (at least it is not integrated) in your gain stage.
- To minimize dc bias in all stages, I would recommend adding compensation resistors to your op-amp circuits. These resistors are critical to the design of your plant; while they may be skipped in some of your other stage (gain, dc bias source, etc.). Using the correct compensation resistor in the first integrator, can sometimes be all it takes to get your circuit working. For the first integrator I really do mean the correct value, not one that looks like it is close enough. Some students found a variable resistor placed here in series with one that was close enough would allow

them to remove small dc bias problems. To be safe, I always included compensation resistors in all my op-amp circuit designs.

- Do not add digital gain. While, I kept my digital gain at x1, Scott actually introduced a gain of less than 1 (attenuation). With a gain of x4 or more the DAC will saturate (input of greater than 0x0FFF). One symptom of this problem is a significant asymmetry component to the output of your plant.
- I would recommend removing all potentials for ground loops and accidents. One good way to avoid these two issues is to use 9v batteries for your power sources or the TDK DC-to-DC converter used by Scott and myself.
- Never input 5 volts directly to the 5v header pin on the Arduino. Instead go through the excellent power conditioning circuits on the Arduino. I know some folks, say it is fine, but why ask for a disaster to occur when there are some many that can happen when you do everything correctly.
- DC bias problems, as discussed earlier can arise from a number of sources, one good general purpose solution is to introduce a variable summing junction into your gain stage. To tune out any dc bias, ground the input to the plant and then using your variable resistor null out any dc bias in the output.
- The software is designed to work with LED indicators as shown in my schematic. Add them! They are there to tell your when things are not going well. For example if your output is saturating. Many hours of hardware debugging could have been avoided if students would have taken the few minutes required to add these LEDs.
- Bypass and filter capacitors as discussed in the main article really do help.