Overview

In parts 1 and 2 of this project we looked at **open loop** control of the phototransistor sensor voltage (the output for this system). In parts 3 and 4 we will be **'closing the loop'** with a feedback control PI algorithm. Don't worry, it's a lot easier than it sounds!

Parts 3 & 4 will be due on Tuesday 2/25 before class. Answer all of the exercise questions, and include plots or code when asked. You will be asked to demo Exercise 13 (the complete project tracking a 1V to 2V square wave) in class on 2/25.

Part 3 -
You will have one main objective. That is to read in the sensor voltage from the phototransistor and convert it to a digital value from 0-1023. You will do this using the PIC's built in analog to digital converter (ADC). Don't worry about the specifics for now, we have given you sample code functions setupADC() and readADC() to both setup and read from the ADC peripheral.

setupADC() can be called in main when you setup your timers and output compare.
readADC() can be called within your 1kHz control loop. It will return an integer value 0-1023 which corresponds to 0-3.3V analog voltages. It is predefined to use AN0 which is also pin B0.

Both of these functions are posted up on the wiki.

In class 2/18

- Connect the ADC (B0) to the output voltage of the phototransistor. Read the ADC in the 1kHz control loop and output it over serial along with the reference waveform. Your output should be formatted as two numbers with a space separating them followed by \r\n. Do not include additional text.
- Plot in matlab using uart_plot.m. Call it by running uart_plot(2,1000,'comport');
  (the 2 means that you are plotting two numbers, 1000 means that it is plotting 1000 data points on the x axis in between updates). 'Comport' is the name of the com port.
  Press q to stop plotting.
- If you are having trouble plotting with matlab, you can also view the data coming in from the serial port by opening up your terminal emulator program. You may want to slow down the rate at which the PIC sends data to the terminal in this case. (Maybe to 10 Hz or so).

Exercises:

8. Do the ADC readings match what you specified in the waveform array? Why or why not?

9. Manually convert the voltage displayed on the scope into ADC ticks (show your calculations). Do the ADC readings and the values you calculated match (your calculation is an estimate so it may not be exact)?

10. In part 4 you will have to implement a function called setDuty(u) which takes an integer control effort (u) and outputs a value for controlling the duty cycle of the PWM via the OC1RS register. In this function, the control effort input can be a positive or negative value, and will typically range from around -FULL_DUTY/2 to +FULL_DUTY/2 (it may try to go over or under these values if your Kp and Ki are high). The OC1RS register, however, can only ever have a value of 0 to FULL_DUTY! For this exercise the, write a function int setDuty(int u) that takes in a computed control effort, adds an offset of FULL_DUTY/2 to make sure the output won't have a negative value, and limits the variable it returns to be less than FULL_DUTY and greater than zero. Turn in the function you wrote.

Part 4 -

This is where everything comes together. In this part, you will design the feedback controller. This feedback controller will live in the 1kHz control loop you have previously created. The basic algorithm will work like this:

1. Read in the current ADC value from the phototransistor sensor.
2. Calculate the error (e) as the difference between the current reference value (r) and value of the ADC you just read.
3. Calculate the proportional error term using Kp constant.
4. Calculate the integral error (eint) as the sum of the current error (e) and all of the previous error. Multiply the integral error by the integral error constant Ki to get the integral error term.
5. Add the proportional and integral error terms together to form the control effort variable (u).
6. Set the duty cycle of the PWM proportional to the control effort (u).

Pseudocode of a PI controller might look like this:

```
adcValue = readADC();
e = referenceValue - adcValue;
eint = e + eint;
u = Kp*e + Ki*eint;
OC1RS = setControl(u);
```

In class 2/20

- Download the .hex file provided for class (on the wiki) and load it on the NU32. This .hex file contains the completed project, and you can use it to play around with Kp and Ki gains to see what the end goal of this project should look like. Use the matlab function pid_plot.m (on the wiki) to interact with the NU32. Try different Kp and Ki values and see how the controller performs.
    - The pid_plot function takes 3 arguments, the COM port you are using, the Kp value for the controller, and the Ki value for the controller.

- Using the reference waveform you generated in exercise 7, the function you wrote in exercise 10, the sample code given in part 3, and the pseudocode given above, implement a proportional controller with a Kp of 1. That is, set the constant Ki to 0 to begin with.

- Output the sensor reading, the reference signal, and the control effort over serial, plot in matlab. You can use the c function **NU32_WriteUART1Asnc** (provided on the wiki) to send data to matlab from the NU32. This will replace NU32_WriteUART1 for this case only (when writing to matlab). For further information about writing to matlab from the NU32, see com.txt (also posted on the wiki).

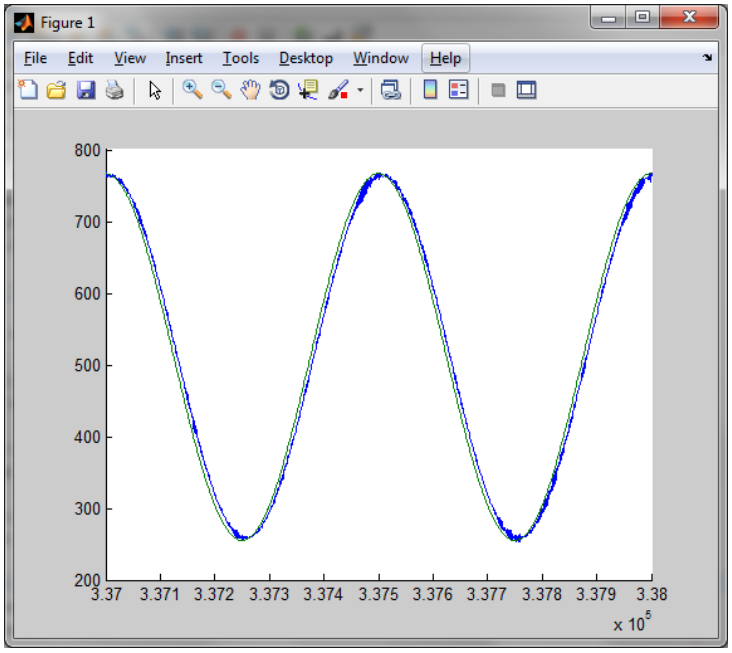
Exercises:

11. What happens to the sensor reading if Kp is much too large? What if it is much too small?

12. Now add the integral control term. It is best to start with a relatively small value and work your way up. Adjust the control constants Kp and Ki so that the ADC sensor reading tracks the reference waveform from exercise 7.

13. Include a screenshot of the matlab plot. How close are you to tracking the desired signal? Include a screenshot of the matlab plot. Also include the final values you have chosen for Kp and Ki, and attach your final code.

14. **For extra credit,** try your controller with a sine wave reference. You can generate a sin wave using the included makeSinWave() function. This function is designed to create a 2 Hz sin wave and put it in the array ‘waveform’. Waveform should have a length of 1000, and should be played back at 1 kHz.



An example output tracking a sin wave reference