ME 333
Assignment 7 and 8 - PI Control of LED/Phototransistor Pair

Overview

For this assignment, you will be controlling the light emitted from and received by an LED/phototransistor pair. There are many steps in the process, so we will be breaking it up into 4 main parts:

1. Implement a PWM output circuit that will drive the LED at a given frequency.
2. Implement a timer that will cycle through an array and update a current reference value of desired light level.
3. Set up a phototransistor circuit to read analog voltage levels and then convert these levels to digital values.
4. Implement a proportional-integral controller that will take feedback from the phototransistor circuit, calculate an error from the current reference value, and adjust the LED PWM accordingly.

Step 1 - In class 2/11

- Wire the LED in series with digital output D0 and a 330 ohm resistor.
    - The long leg of the LED will go into D0, the short leg will go to the 330 ohm resistor, and the 330 ohm resistor will go to ground.
- Setup Timer2 on the NU32 to interrupt every 0.1 seconds. To set the timer to interrupt you must:
    - Set the correct timer prescaler (TCKPS bits in T2CON)
    - Set the input to PBCLK (TCS bit in T2CON)
    - Turn the timer on (ON bit in T2CON)
    - Set the period register so the timer rolls over every 0.1 seconds (PR2)
    - Reset the Timer2 count (TMR2) to 0
    - Set the Timer2 interrupt priority and subpriority bits
    - Clear the Timer2 interrupt flag
    - Enable Timer2 interrupt

- Within the interrupt, invert the current value of D0 (This will toggle the output LED).
    - This should create a very simple 5 Hz PWM signal that has a 50% duty cycle. That means that it spends half of its time high, the other half low, and that it repeats itself once per cycle.

- Hook up your nScope on channel A to verify your timing is correct.

Exercises:

1. Using this timer based PWM, generate a 50% duty cycle PWM signal running at 100Hz, 1kHz, and 10kHz. Verify these times on the nScope and take a snapshot of each.

2. What are the values of the timer prescaler, N, and period register, PR2, you used in question 1?

3. What is the slowest 50% duty cycle PWM signal you can generate using this method, and what prescaler would you use to create it?

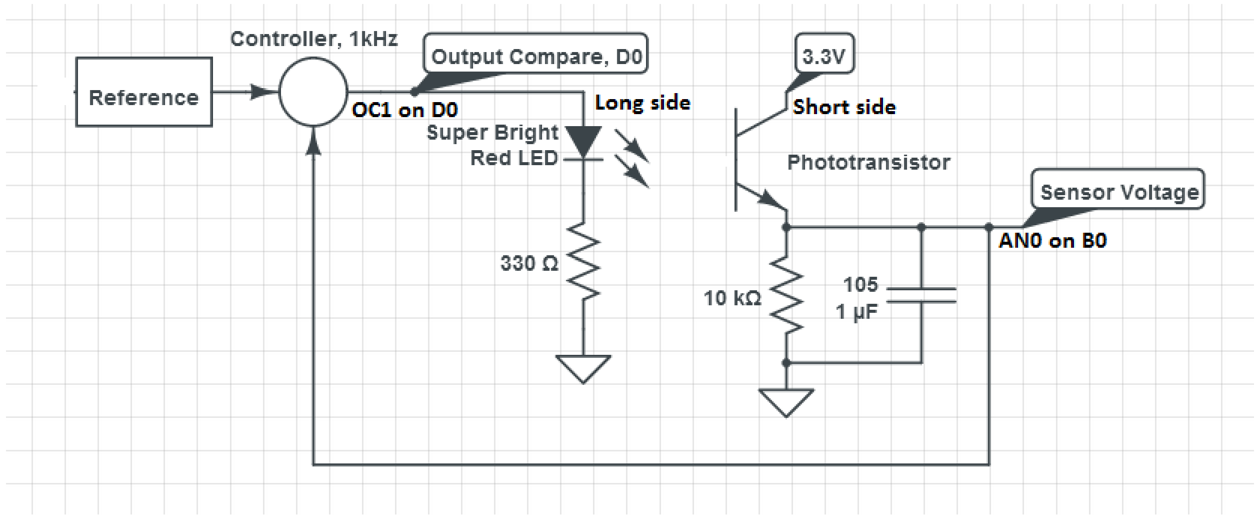
Overview

The goal of this project is to control the light emitted by an LED. We provide a signal that specifies the brightness of the LED over time, and program the PIC to make the LED brightness follow the signal. The steps to make this work are outlined here: they are extremely similar to what must be done to control motors.

The project consists of 4 parts.
Parts 1 & 2 are due on Tuesday 2/18 before class. You will demo the response to part 2 question 7 in class. Include answers to all the questions (including nScope snapshots) and the source code for the answer to part 2 question 7.

Parts 3 & 4 will be posted soon and will be due on Tuesday 2/25 before class

Here is a diagram, including the circuits and control loop. The reference signal is stored as an array on the PIC.

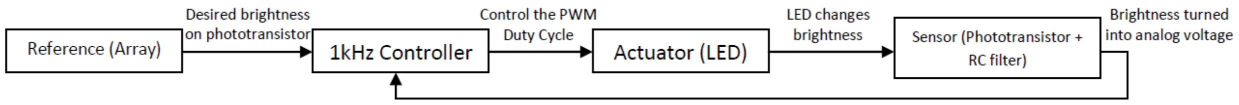


The project consists of several components:

1. The LED circuit. We control the brightness of the LED by turning it on and off really fast using a PWM signal. The switching is too fast for the human eye to see, so it appears as different brightness levels, depending on the duty cycle of the signal.
    - Part 1 of the assignment had you control the light level of this LED
    - In Part 2, you will use the Output Compare feature to control the light level, rather than manually changing it in a timer.
2. The reference generator generates the signal (aka the reference) that we want the LED to track. The signal is stored in an array on the PIC, and is played back by a timer interrupt.
    - In Part 2, you will create this reference signal and use a timer to change the PWM duty cycle. This will cause the LED to approximately follow your signal.
3. The sensing circuit. This is so we can measure how much light is emitted, and adjust the brightness of the LED accordingly. The voltage will be measured on the PIC using the analog to digital converter (ADC), which converts voltage into a digital integer value.

- The sensing circuit includes a low-pass RC filter. In Part 2, you will see how this RC filter converts the on/off signal from the PWM into a voltage that corresponds to the brightness of the led
- In Part 3, you will hook the output of this signal up to an analog input pin on the PIC and read the values of this voltage.

4. The control loop. In the timer where you change the reference signal, you will also read the sensor value. Using the reference and sensed value, you will implement a control law that uses the error between what you want the LED brightness to be and what your sensor reads to change the PWM signal, thus making the LED follow the reference signal.
    - In part 1, the timer you created can be thought of as a primitive control loop. Each time the ISR triggered, you turned the LED either on or off, this controlled the LED
    - In part 2, you will use the control loop to change the PWM signal to the LED, according to a reference signal. This is known as “open loop” control because you are sending a control signal to the LED, but are not using any sensor data.
    - In part 3, you will read sensor data in addition to outputting the reference signal. You will also be able to output the reference and sensed signal on the computer and compare them. You will see that the sensed value does not match the reference signal.
    - In part 4, you will implement the controller. Your data will show that the controller makes the sensor reading match the reference signal. Mission Accomplished!

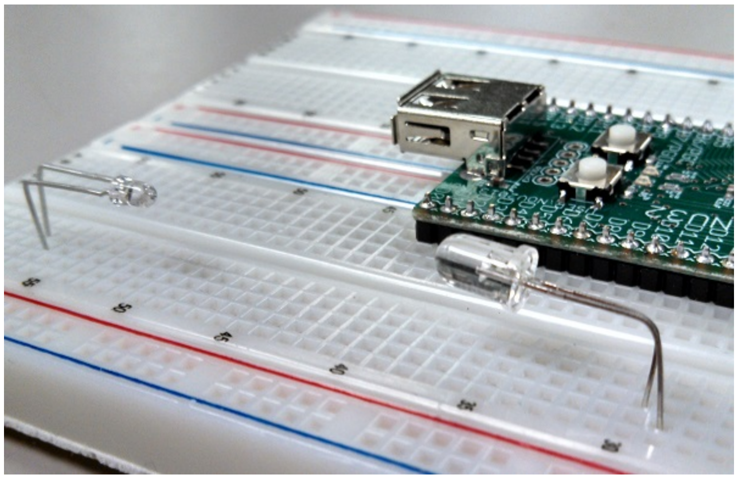

Reference (Array)
Desired brightness on phototransistor
1kHz Controller
Control the PWM Duty Cycle
Actuator (LED)
LED changes brightness
Sensor (Phototransistor + RC filter)
Brightness turned into analog voltage

Overview of System, including the control loop

Step 2 - In class 2/13

1. The sample code we have provided generates a 1kHz PWM signal on OC1 (which is output on pin D0, the pin your LED should be connected to). The OC1RS tick count is entered by the user over serial. View the output of this PWM signal on the nScope (channel A). Play around with different numbers and observe how the wave changes. Approximately how many ticks yield a 25% duty cycle? A 75% duty cycle?

2. Wire up the phototransistor circuit, EXCLUDING THE CAPACITOR. Use a 10kOhm resistor to ground. Place the photo transistor so it is pointed directly at the LED. View the output of the phototransistor on the nScope (Channel B). What do you observe?


Approximate orientation of the LEDs

3. Add a 1uF capacitor in parallel with the 10k resistor. How does the signal change? Try a few different duty cycles.

4. We have provided an array named waveform. We will use this array to set the values of the OC1RS register. This array is designed to be played back at 10 Hz. It is initialized by the makeWaveform function, called from main(). Setup Timer 2 to operate at 10 Hz. In the Timer 2 ISR, cycle through the waveform array and set the OC1RS register according to the value at the current position in the waveform. Observe the output at the LED and at the phototransistor. Are they the same?

Exercises:

4. Generate a 20KHz PWM signal on pin OC1. Use two different combinations of prescaler and PR3 register. Which combination gives you more precise control over the duty cycle?

5. What are the advantages of using the built in PWM function rather than manually performing PWM in a timer, as in Part 1?

6. Setup Timer 2 to run at 1 kHz. Write a function that fills the duty cycle array with the appropriate values to create a 1 Hz square wave, sampled at 1 kHz. Include the ability to specify the minimum and maximum value for the square wave.

7. Generate a square wave with a minimum of roughly 1V and a maximum of roughly 2V. Observe on the nScope. Do the minimum and maximum match with what you specified? What about the frequency?

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 = setDuty(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_WriteUART1Async** (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 comm.txt (also posted on the wiki).

Note about conversions: Remember that the reference signal and the signal coming from the ADC should have the same ranges (same units) before you use them in your control algorithm. That is, you should keep them in terms of **ADC counts, which range from 0-1023**. This range corresponds to **0-3.3V if you are talking about voltage**.

Exercises:

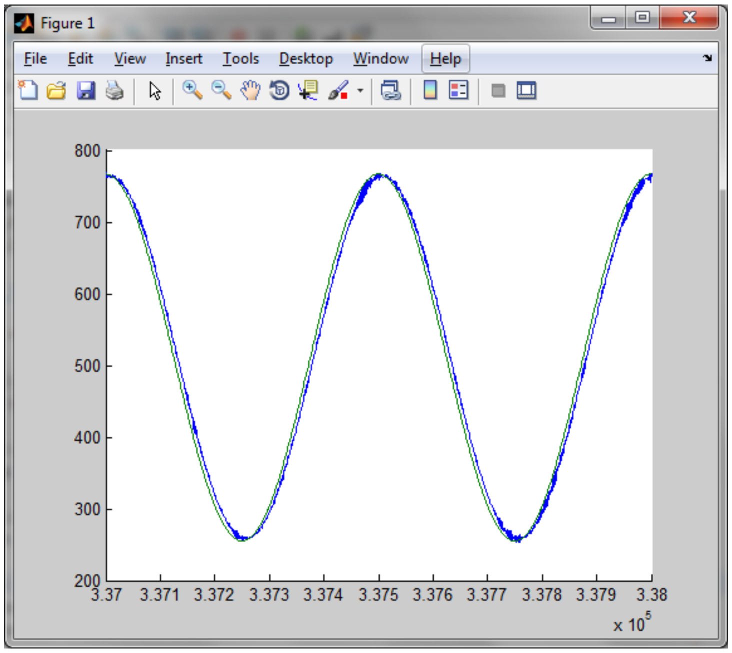
11. What happens to the sensor reading if Kp is much too large? What if it is much too small?

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 (make sure to modify the waveform so it is in ADC counts, not duty cycle).

13. Include a screenshot of the matlab plot. How close are you to tracking the desired signal (a 1V to 2V square wave @ 1Hz)? 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