Yuchen Liu

ME 333 – Week 7 – Assignment 6

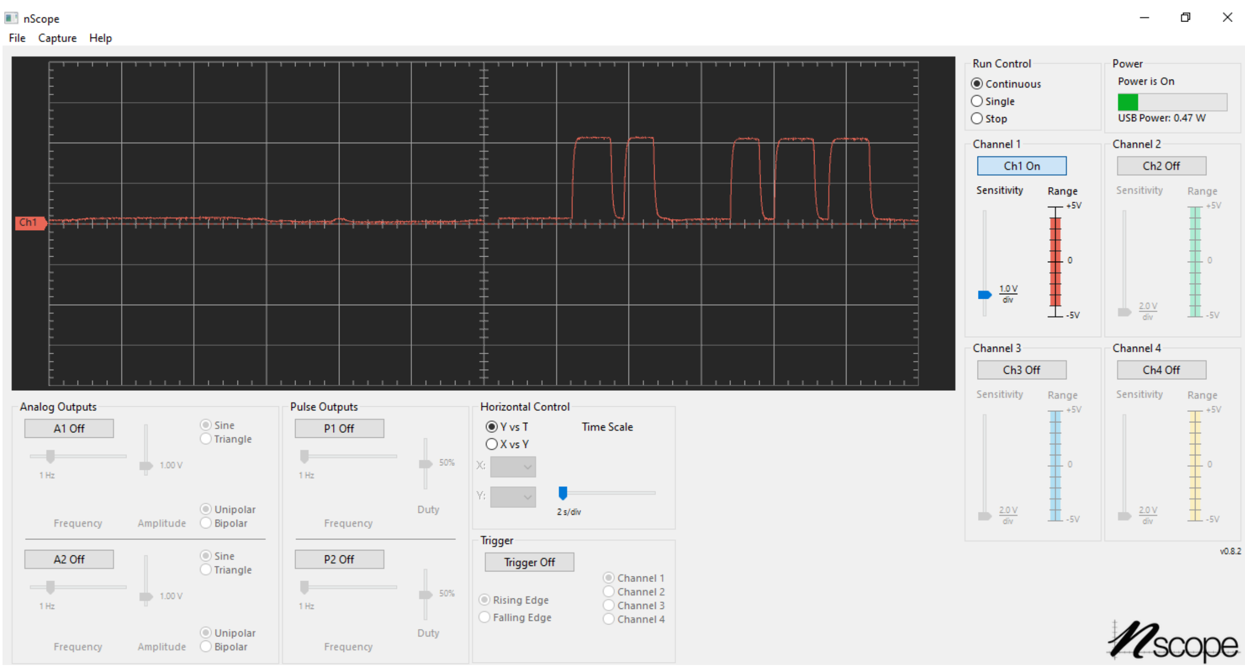
Upload your submission for Chapter 24.1.2, 24.2.1 and 2, 24.3.1 and 2, 24.5, 24.7, and 24.8

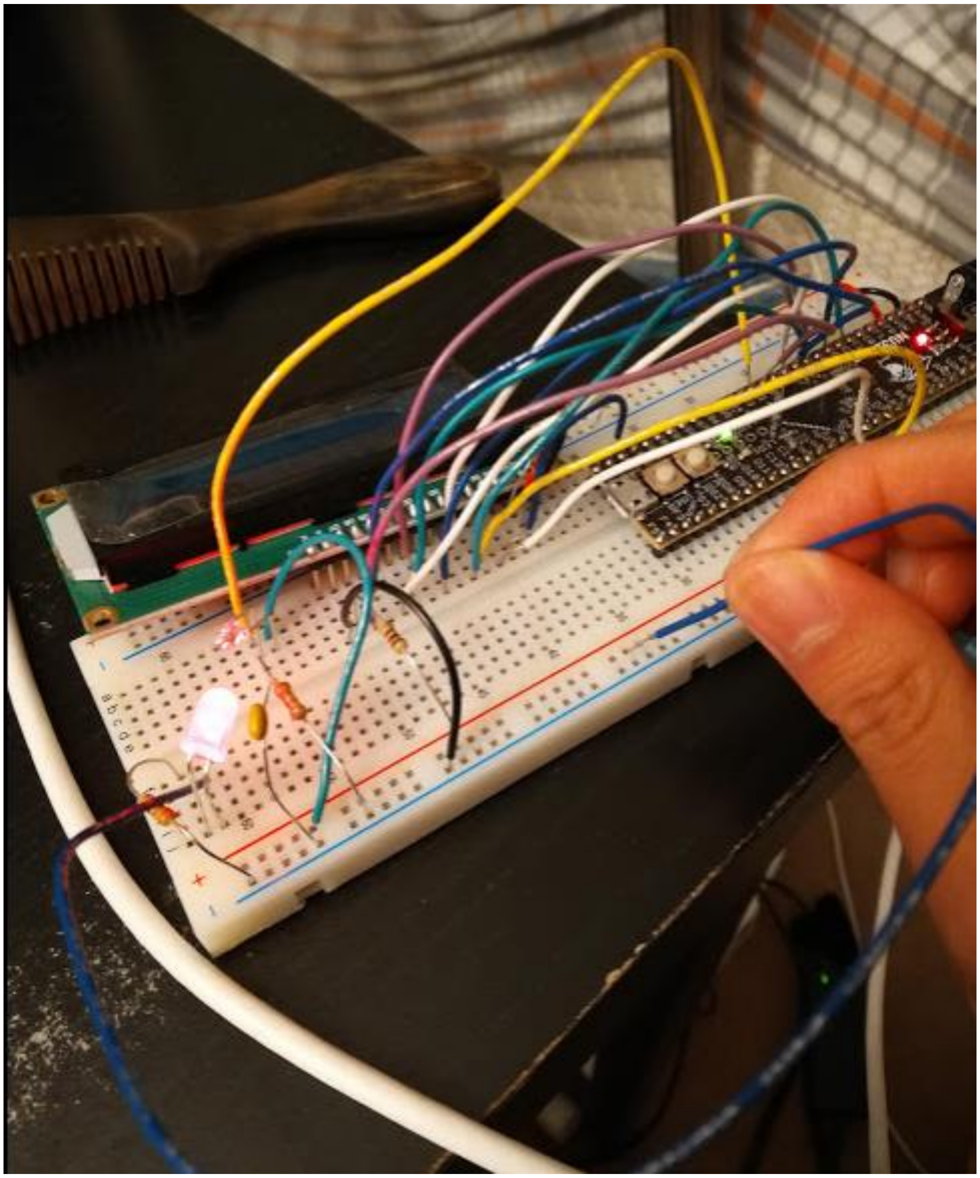
Upload a demo video for 24.8 in the Demonstrations section

Chapter 24.1.2

2. **Choose** R. Wire the circuit as shown in Figure 24.2, except for the connection from the LED to OC1. The LED and phototransistor should be pointing toward each other, with approximately one inch separation, as shown in Figure 24.4. Now choose R to be as small as possible while ensuring that the voltage V_{out} at the phototransistor emitter is close to 3 V when the LED anode is connected to 3.3 V (maximum LED brightness) and close to 0 V when the LED anode is disconnected (the LED is off). (Something in the 10 kΩ range may work, but use a smaller resistance if you can still get the same voltage swing.) Record your value of R. Now connect the anode of the LED to OC1 for the rest of the project.

I found that I had to use R=33000 ohms to get between the 2 and 3V range.





Chapter 24.2.1

1. **PWM calculation.** You will use Timer3 as the timer base for OC1. You want a 20 kHz PWM on OC1. Timer3 takes the PBCLK as input and uses a prescaler of 1. What should PR3 be?

The input of the timer is PBCLK, so it is 80 MHz.

To create a 20 kHz ISR with an 80 MHz PBCLK, the interrupt must be triggered every (80*10^6)/(20*10^3) = 4000 PBCLK cycles.

Given n=1, we have

4000 = (PR3+1)*1, so PR3 = 3999

Chapter 24.2.2

2. **PWM program.** Write a program that uses your previous result to create a 20 kHz PWM output on OC1 (with no fault pin) using Timer3. Set the duty cycle to 75%. Get the following screenshots from your oscilloscope:

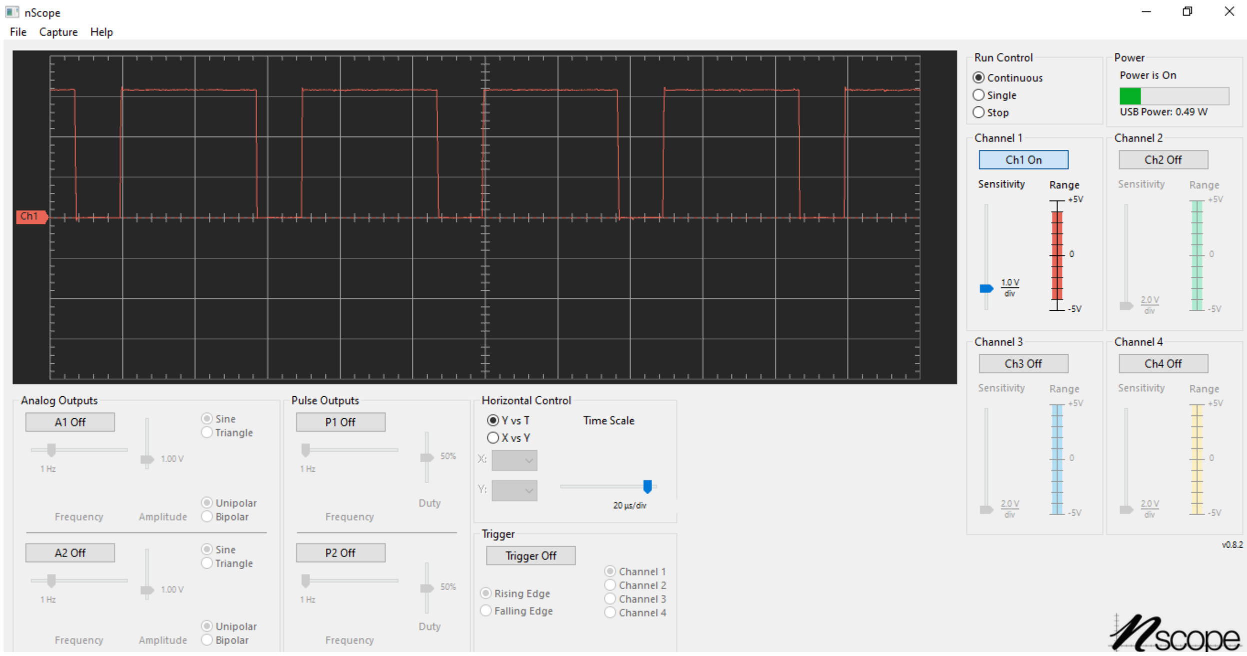
a. The OC1 waveform. Verify that this waveform matches your expectations.
b. The sensor voltage V_{out}.
c. Now remove the 1 µF capacitor and get another screenshot of V_{out}. Explain the difference from the previous waveform.

code, using Code Sample 9.1 as a template:

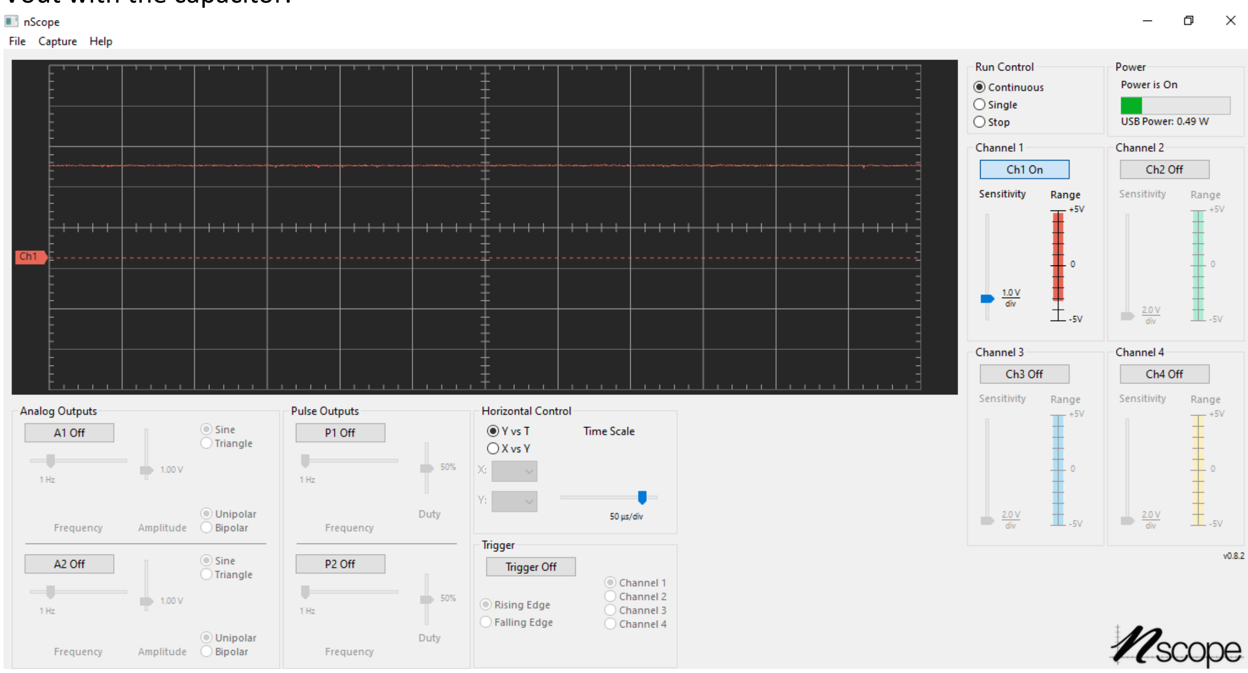
```
#include "NU32.h"          // constants, functions for startup and UART

int main(void) {
  NU32_Startup();          // cache on, interrupts on, LED/button init, UART init

  T3CONbits.TCKPS  = 0b000;    // Timer3 prescaler N=1 (1:1)
  PR3 = 3999;              // calculated in 24.2.1
  TMR3 = 0;                // initial TMR3 count is 0
  OC1CONbits.OCM = 0b110;  // PWM mode without fault pin; other OC1CON bits are d
efaults
  OC1RS = 3000;             // duty cycle = OC1RS/(PR3+1) = 25%
  OC1R = 3000;              // initialize before turning OC1 on; afterward it is
read-only
  T3CONbits.ON = 1;        // turn on Timer3
  OC1CONbits.ON = 1;       // turn on OC1

  _CP0_SET_COUNT(0);       // delay 4 seconds to see the 75% duty cycle on a 'sco
pe
  while(_CP0_GET_COUNT() < 4 * 40000000) {
    ;
  }
  OC1RS = 3000;            // keep duty cycle at 75%
  while(1) {
    ;                      // infinite loop
  }
  return 0;
}
```

[redacted]

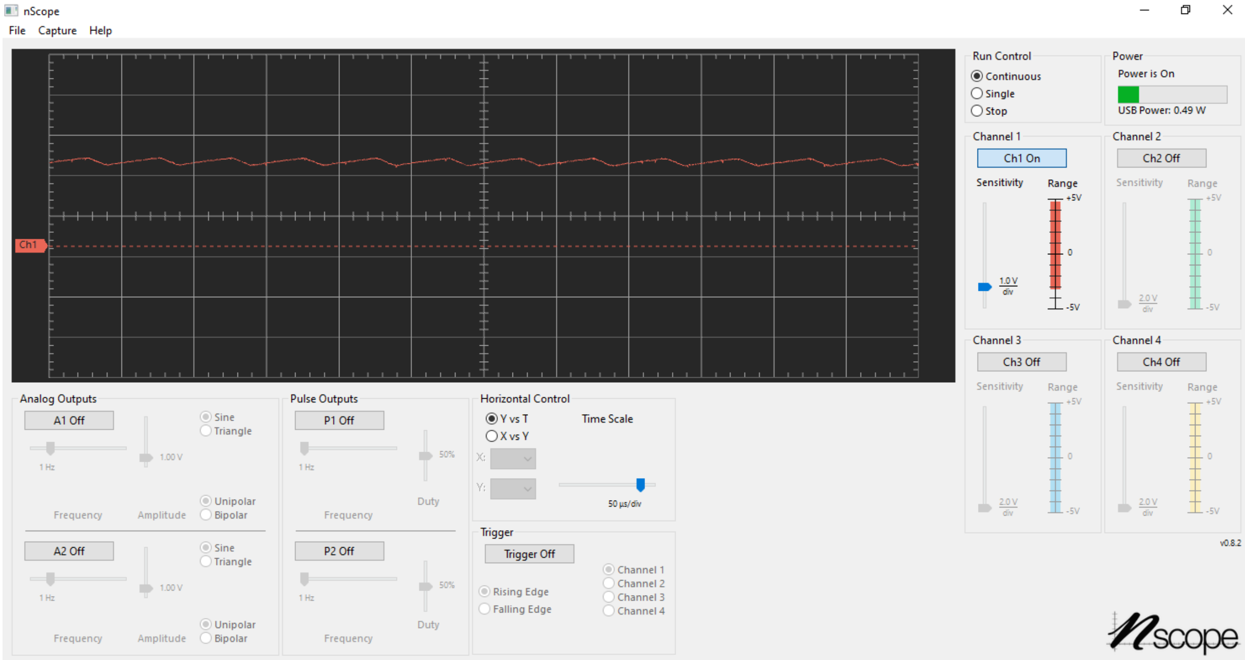
a) OC1 output matches my expectations because the duty cycle is 75% (it's more high than low). Also, we set it to be 20kHz, which is 1/20kHz = 0.00005s, which is on the scale of the 20 mu seconds in our time scale on the nScope.



b) Vout with the capacitor:

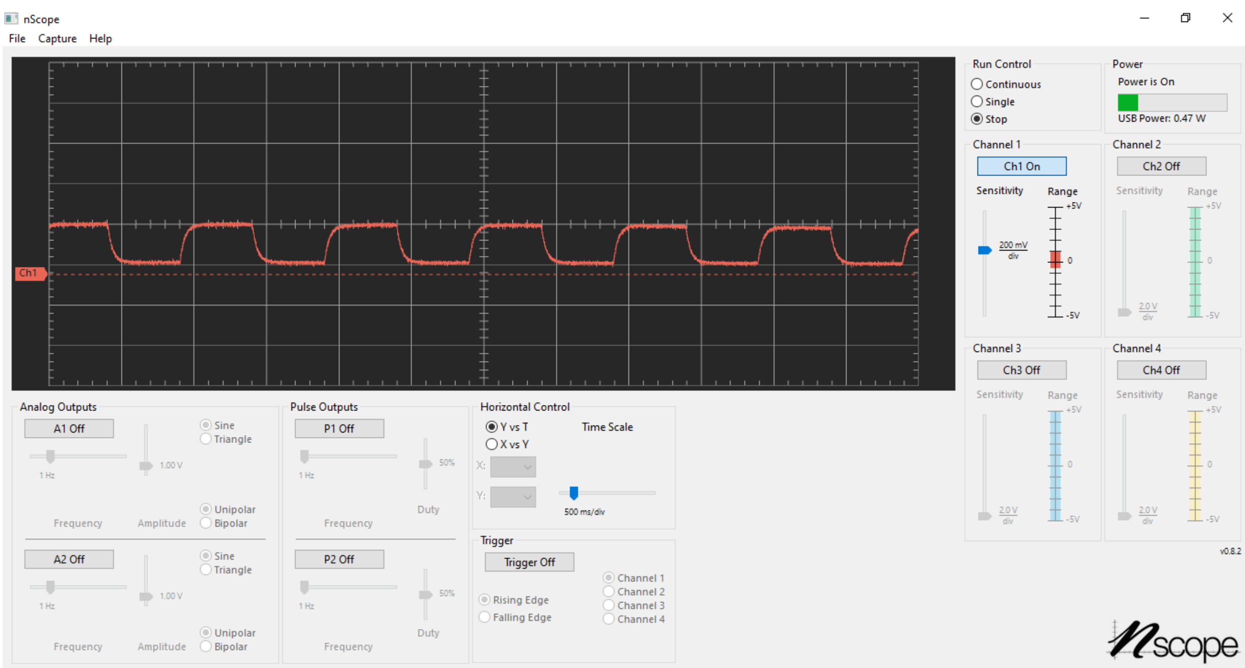


c) Vout without the capacitor. It is wavier because the capacitor isn't there to smooth it out.



Chapter 24.3.1

1. Get a screenshot of your oscilloscope trace of V_{out} showing 2-4 periods of what should be an approximately square-wave sensor reading.



Chapter 24.3.2

2. Turn in your code.

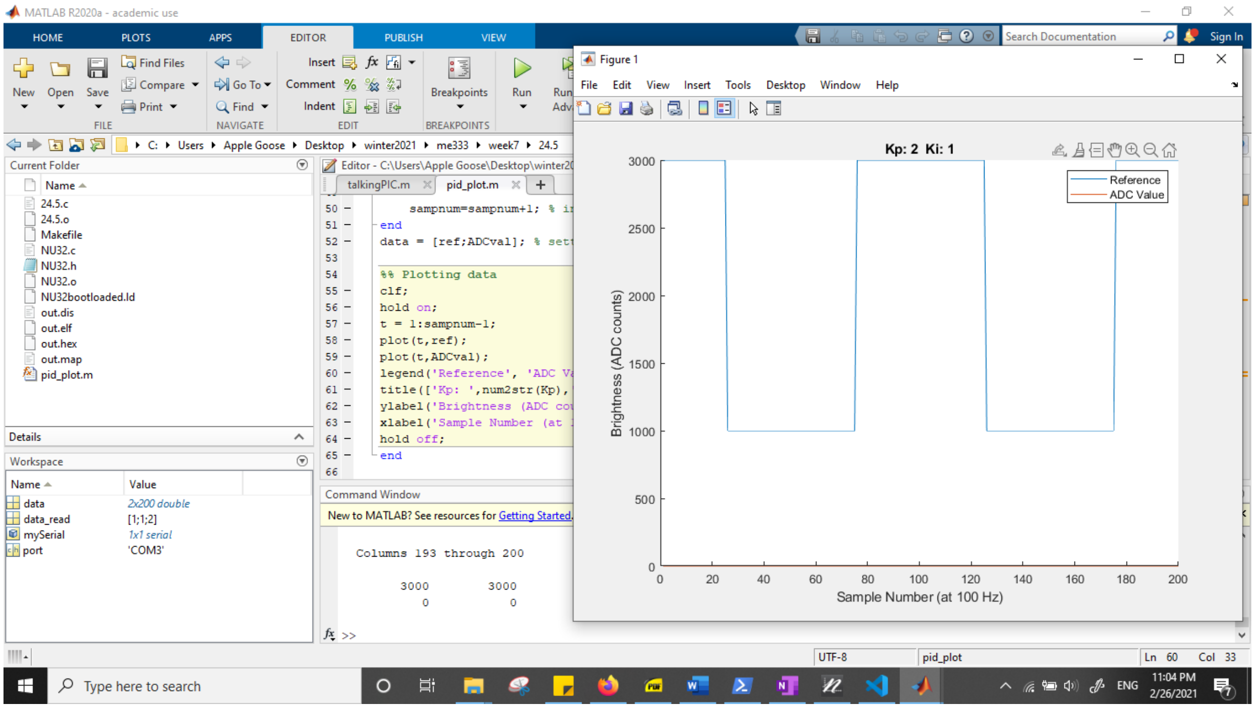
```c
#include "NU32.h" // constants, functions for startup and UART

#define NUMSAMPS 1000                   // number of points in waveform
static volatile int Waveform[NUMSAMPS]; // waveform

void __ISR(_TIMER_2_VECTOR, IPL5SOFT) Controller(void)
{                          // _TIMER_2_VECTOR = 8
  static int counter = 0; // initialize counter once
  OC1RS = Waveform[counter];
  counter++; // add one to counter every time ISR is entered
  if (counter == NUMSAMPS)
  {
    counter = 0; // roll the counter over when needed
  }
  IFS0bits.T2IF = 0;
}

void makeWaveform()
{
  int i = 0, center = 2000, A = 1000; //A is (PR3+1)/4 = (3999+1)/4 = 1000. cente
r is (3999+1)/2
  for (i = 0; i < NUMSAMPS; ++i)
  {
    if (i < NUMSAMPS / 2)
    {
      Waveform[i] = center + A; //this is 3000, which is 75%
    }
    else
    {
      Waveform[i] = center - A; //this is 1000, which is 25%
    }
  }
}

int main(void)
{
  NU32_Startup(); // cache on, interrupts on, LED/button init, UART init
  makeWaveform();
  T3CONbits.TCKPS = 0;     // Timer3 prescaler N=1 (1:1)
  PR3 = 3999;              // calculated in 24.2.1
  TMR3 = 0;                // initial TMR3 count is 0
  OC1CONbits.OCM = 0b110; // PWM mode without fault pin; other OC1CON bits are de
faults
  OC1RS = 3000;            // duty cycle = OC1RS/(PR3+1) = 75%
```

```
  OC1R = 3000;            // initialize before turning OC1 on; afterward it is re
ad-only
  T3CONbits.ON = 1;       // turn on Timer3
  OC1CONbits.ON = 1;      // turn on OC1
  _CP0_SET_COUNT(0);      // delay 4 seconds to see the 75% duty cycle on a 'scop
e
  while (_CP0_GET_COUNT() < 4 * 40000000)
  {
    ;
  }
  OC1RS = 3000; // keep duty cycle at 75%

  __builtin_disable_interrupts();
  T2CONbits.TCKPS = 0b001; // prescalar of 2, since we can't use prescalar of 1,
otherwise it would have P be greater than 2^16-1
  PR2 = 39999;             //want t=1ms, T = (P + 1) × N × 12.5 ns -> (1*10^-
3)/((12.5*10^-9)*2)-1 = P
  TMR2 = 0;
  T2CONbits.ON = 1;
  IPC2bits.T2IP = 5; //priority
  IPC2bits.T2IS = 0;
  IFS0bits.T2IF = 0;
  IEC0bits.T2IE = 1;
  __builtin_enable_interrupts();

  while (1)
  {
    ; // infinite loop
  }
  return 0;
}
```

Chapter 24.5

1. Turn in a MATLAB plot showing `pid_plot.m` is communicating with your PIC32 code.

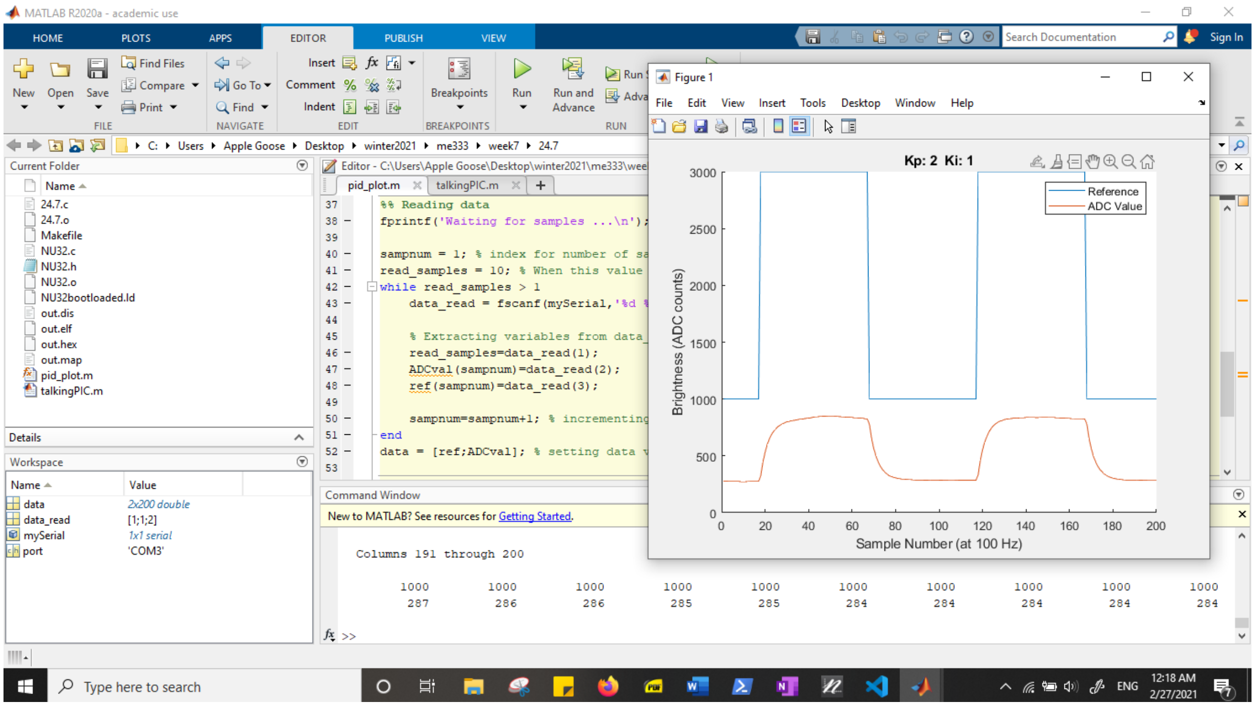
Yay, it worked:



Chapter 24.7

1. Read the ADC value in your ISR, just before the `if (StoringData)` line of code. The value should be called `adcval`, so it will be stored in `ADCarray`. Turn in a MATLAB plot showing the measured `ADCarray` and the `REFArray`. You may wish to use manual sampling and automatic conversion to read the ADC.

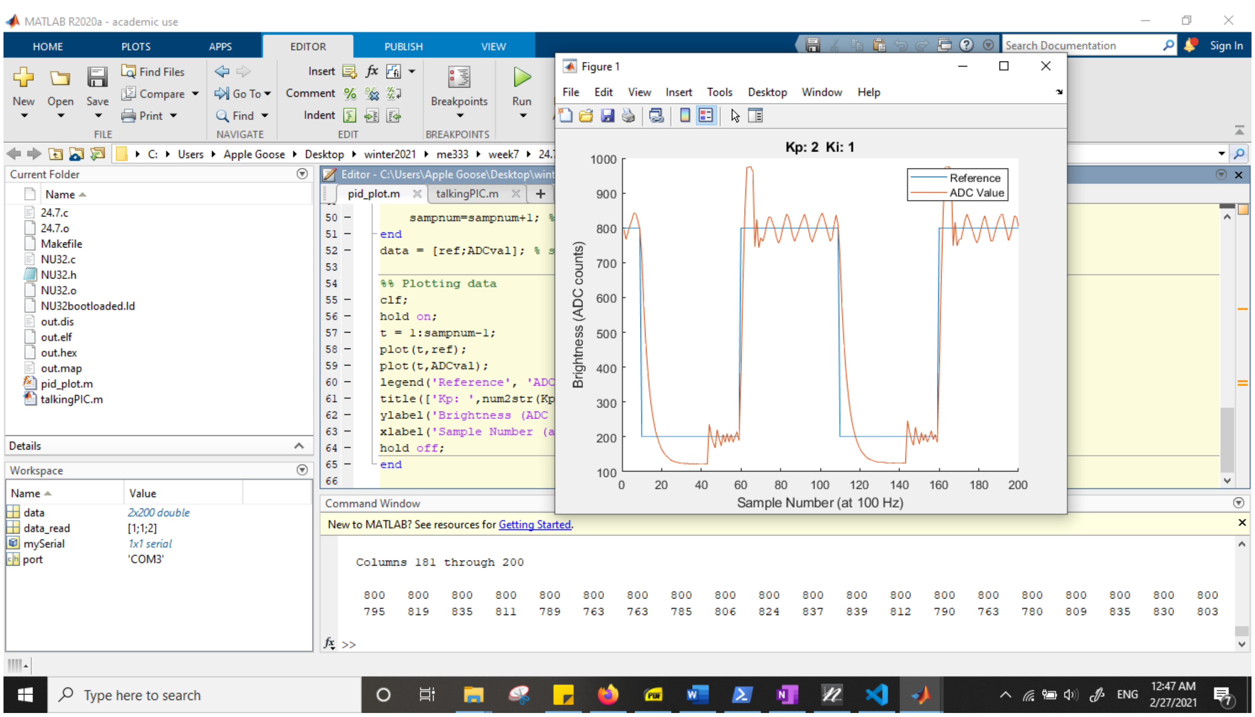
Use the adc_sample_convert function from sample code 10.1. Don't forget to modify main with AD1PCFGbits.



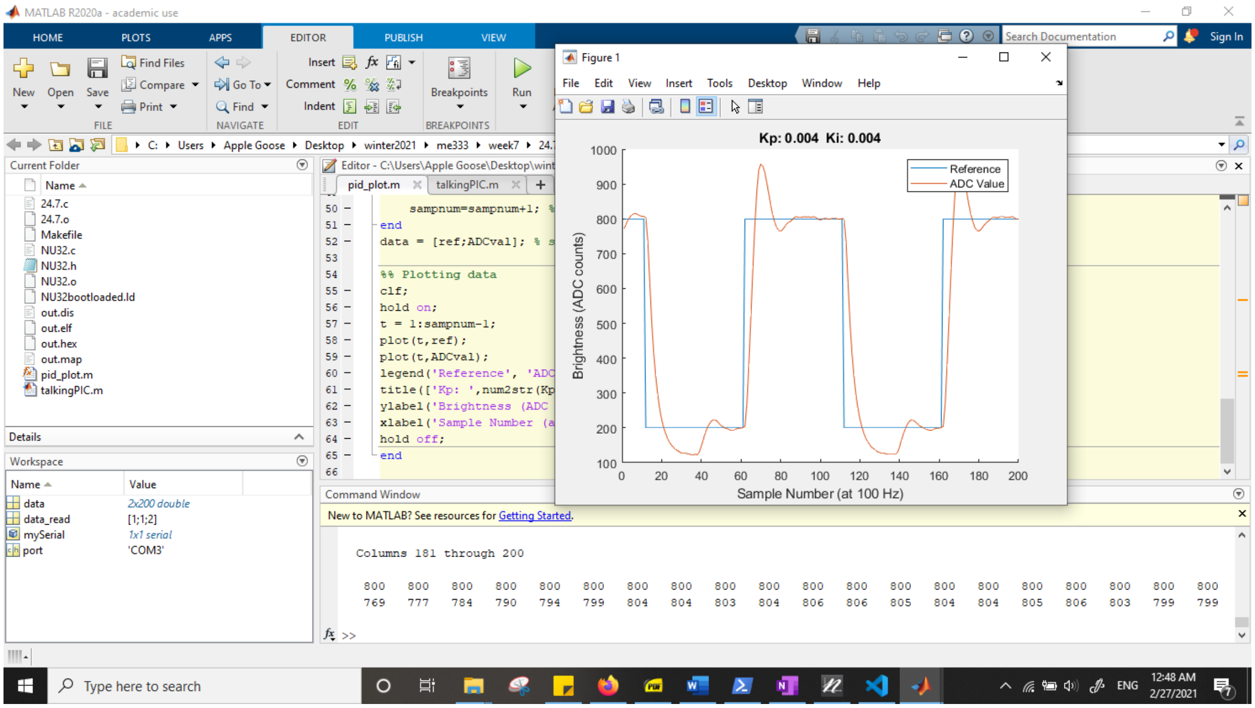
Chapter 24.8

1. Using your MATLAB interface, tune your gains K_p and K_i until you get good tracking of the square wave reference. Turn in a plot of the performance.

The default we've been using is:



The best I can do is at Kp = Ki = 0.004:



MATLAB R2020a - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Figure 1
File Edit View Insert Tools Desktop Window Help
Kp: 0.004 Ki: 0.004
Reference
ADC Value
Brightness (ADC counts)
Sample Number (at 100 Hz)
Current Folder
24.7.c
24.7.o
Makefile
NU32.c
NU32.h
NU32.o
NU32bootloaded.ld
out.dis
out.elf
out.hex
out.map
pid_plot.m
talkingPIC.m
Workspace
data 2x200 double
data_read [1;1;2]
mySerial 1x1 serial
port 'COM3'
sampnum=sampnum+1;
end
data = [ref;ADCval];
%% Plotting data
clf;
hold on;
t = 1:sampnum-1;
plot(t,ref);
plot(t,ADCval);
hold off;
end
Command Window
Columns 181 through 200
800 800 800 800 800 800 800 800 800 800 800 800 800 800 800 800 800 800 800 800
769 777 784 790 794 799 804 804 803 804 806 806 805 804 804 805 806 803 799 799