Microprocessors System Design II

EECE.4800

Lab 1: Sensor Design and Analog Digital Conversion

Instructor: Professor Yan Luo

TA: Ioannis Smanis

Group #10

Hans-Edward Hoene

2 October 2017

2 October 2017

Section 2: Contributions /1 points

1. Group Member 1 – Hans-Edward Hoene (Me)
    - Coded light sensor and LED output
    - Worked on counter-based PWM signal generator when PWM initialisation would not work
    - Helped set up and debug hardware issues.
    - Helped design light sensor hardware
    - Cleaned code

2. Group Member 2 - Derek A Teixeira
    - Configured all of the hardware
    - Helped debug and test my code when LED and light sensor initialisations did not work
    - Helped debug PWM initialisation code with Kyle
    - Spent countless hours fixing all hardware-related issues

3. Group Member 3 - Kyle W Marescalchi
    - Built code for PWM
    - Debugged and fixed counter version of PWM code
    - Helped set up PWM hardware
    - Put my code and his together into one main file

Section 3: Purpose /0.5 points

The purpose of this laboratory is to understand the hardware and software development of sensors via analog to digital conversion (ADC), and to understand the generation of pulse-wave modulated (PWM) signals. Specifically, this laboratory incorporated a light sensor, which functioned in conjunction with an ADC to turn a light-emitting diode (LED) on and off accordingly. In addition, this laboratory included a senso-motor, whose arm could be operated by controlling the duty cycle of PWM signals.

Section 4: Introduction /0.5 points

This laboratory required us to use a microcontroller to operate a senso-motor via pulse-wave modulation (PWM) waves, and use a light sensor's analog input to turn a light-emitting diode (LED) on or off.

A PWM wave is a square wave with a constant frequency, which uses the width of the pulse as an input. Usually, the width of the pulse in the square wave is measured as a duty cycle, which is the pulse width as a percent of the square wave's period. PWM waves are perfect in a number of applications for outputting analog values from digital values. For example, if one wanted to dim or brighten an LED based on a light-sensor input, you can send PWM waves to the LED and vary the duty cycle. In this laboratory, PWM waves were used to move a motor with a range of 180 degrees.

A light sensor uses light as an analog input. Then, after converting the input to a 10-bit digital value via an analog-digital-converter (ADC), the program will determine whether or not to turn a digital output on or off. This digital output is connected to an LED, which will be shut on or off corresponding to the output's value.

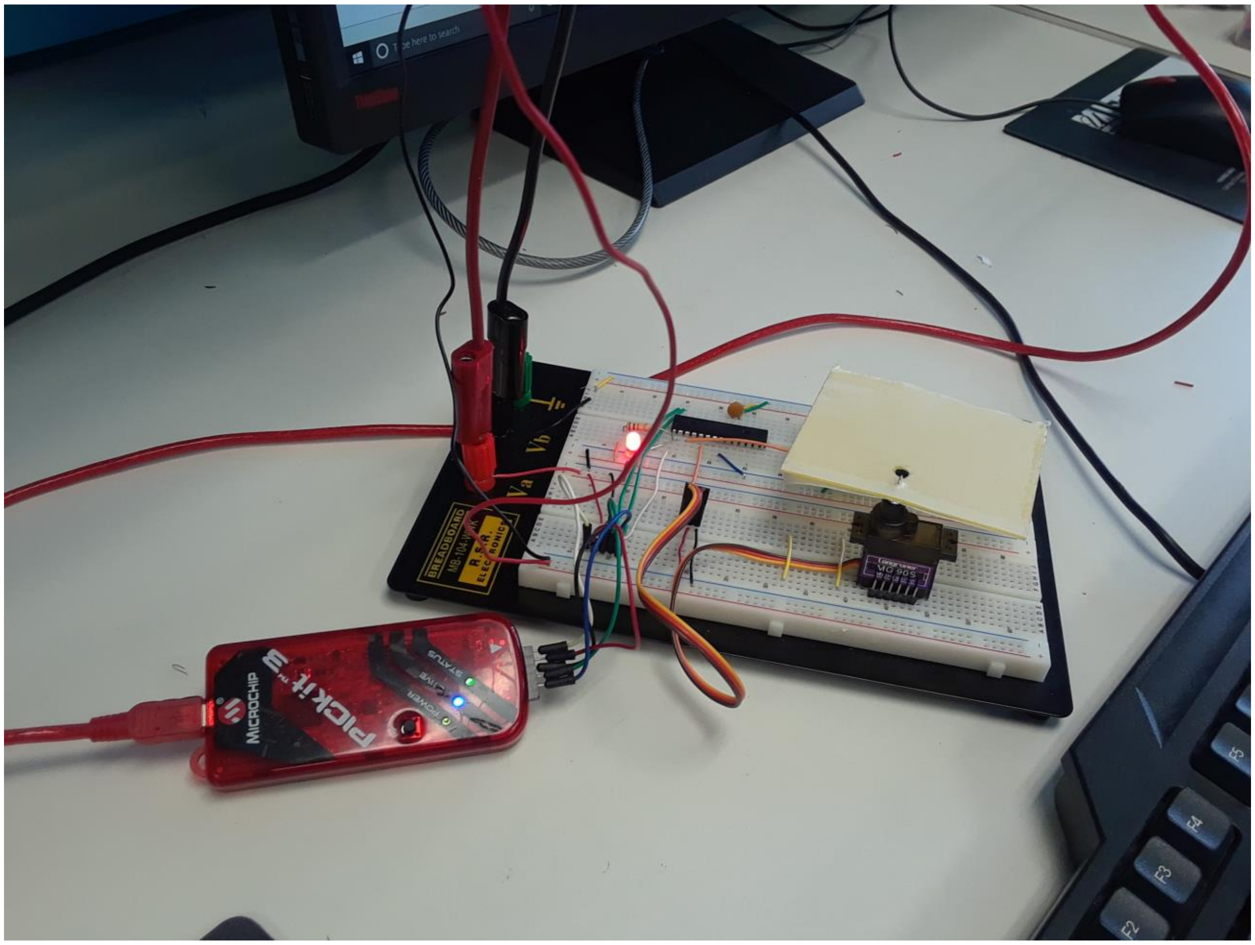
In this laboratory, code was built that was designed to send the correct PWM signals for moving the senso-motor back and forth. The senso-motor's arm had paper taped to it, which would move towards and away from the light sensor, which would correspondingly shut the output LED on and off.

Section 5: Materials, Devices and Instruments /0.5 points

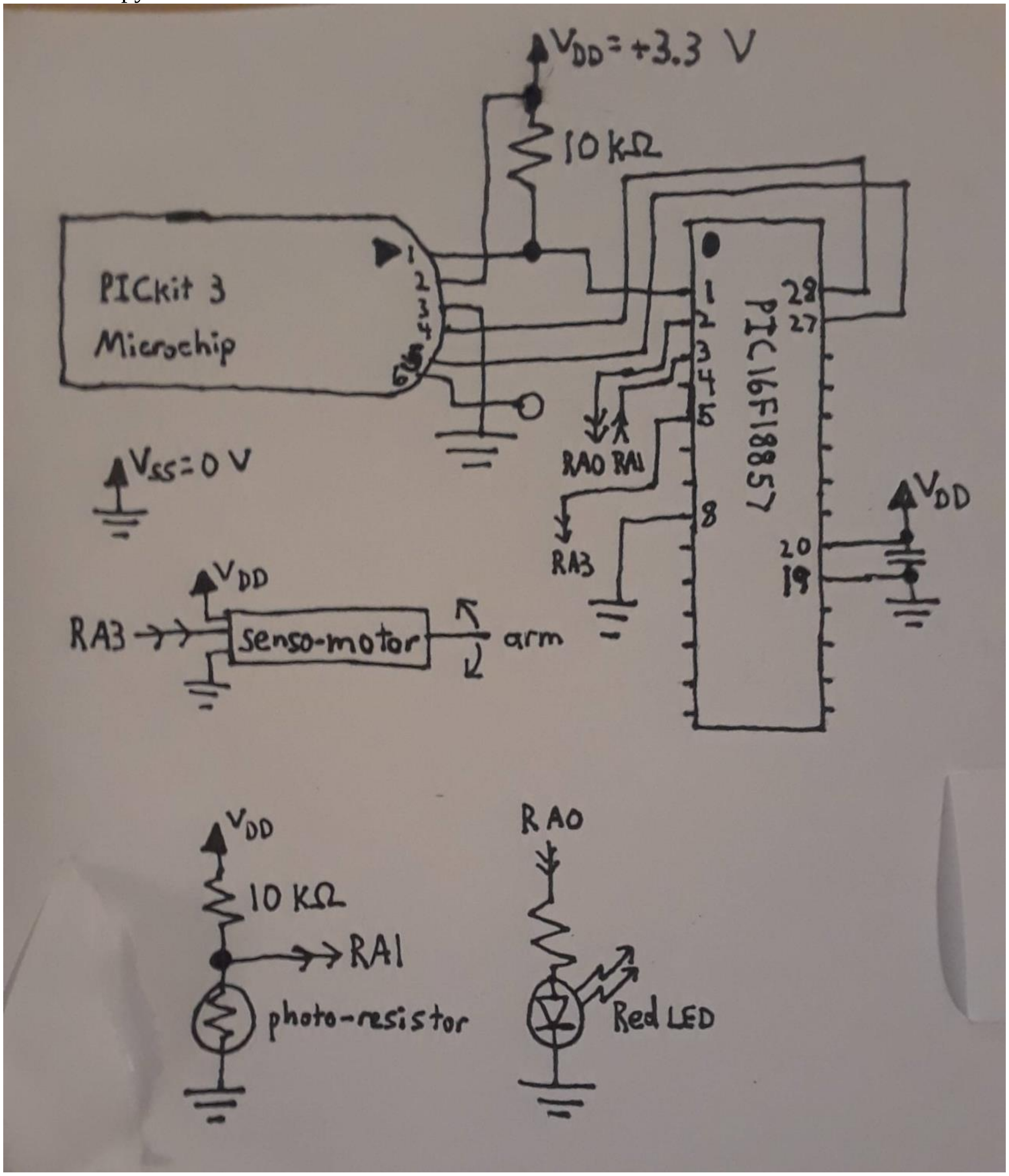
Device Name	Model Number	Description
PICkit	3	Programs microcontroller
PIC Microcontroller	PIC16F18857	Is a microcontroller. Runs code. Operating at 3.3 V in this laboratory.
Servo Motor	SG90	Motor arm controlled by PWM signals
LED	N/A	Light-emitting diode
LDR sensor	N/A	Light sensor

Section 6: Schematics /0.5 points

This is a picture of the final product.



This is a copy of the schematic.



Hardware design:

On our microcontroller, we used only port A for input and output because we just went in order starting with pin 0. Pin 0 was for LED digital output, pin 1 was for analog light sensor input, and pin 3 was for PWM digital output.

From pin 0, we connected a resistor in series with the LED. We assumed that digital outputs had a low output impedance, so it was important to limit the current that came from the output in order to ensure the LED would not be damaged.

To pin 1, we connected the voltage across a photo-resistor as the input. The photo-resistor was in series with another resistor. Here, a voltage divider was used. The resistance of the photo-resistor in relation to the resistor's resistance would determine the input voltage. The only reason that this method worked was because the input pin had a large input impedance. Derek and I measured the photo-resistors resistance under different conditions and we determined with some Calculus that for us to obtain the largest voltage input range across the photo-resistor for the light sensor, we should attach a 13.1 k-ohm resistor in series. We settled for 10 k-ohms, which is good enough for our purposes.

The PWM pin, pin 3, was attached directly to the senso-motor, which had its own internal resistance.

To power the entire setup, an external 3.3 V power supply was used.

Software design:

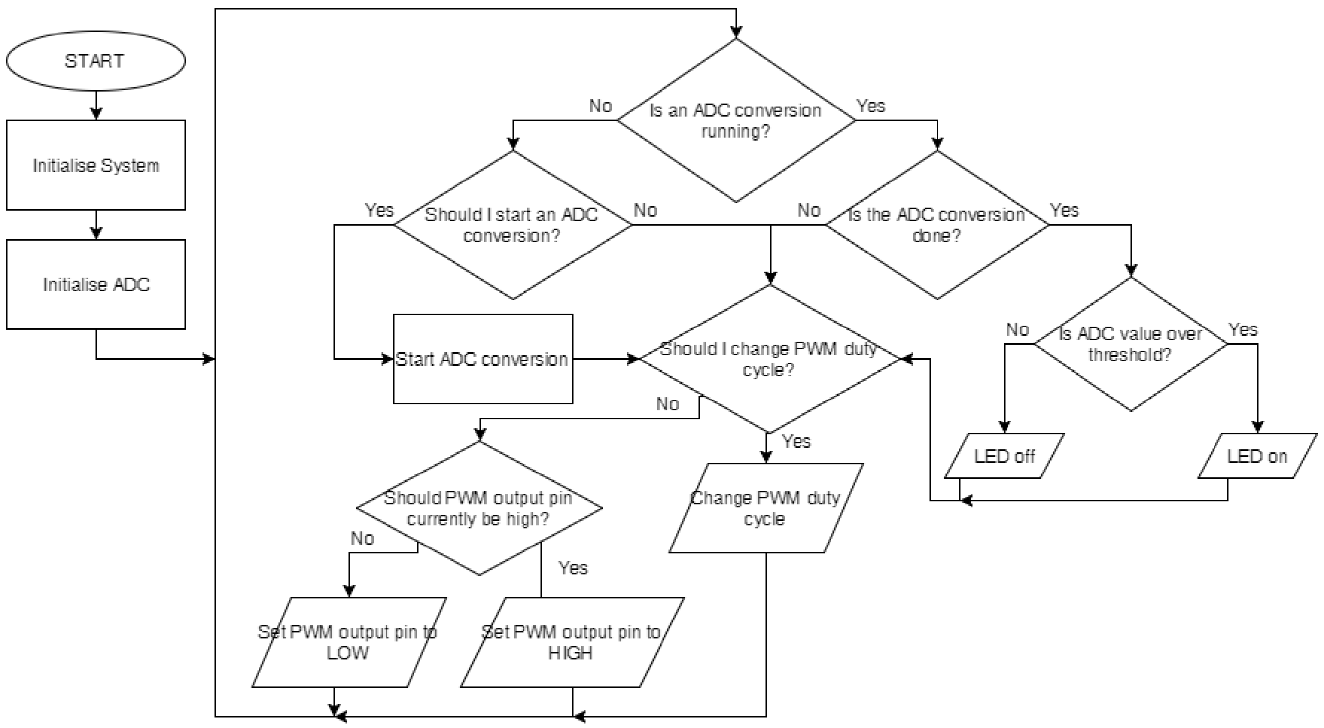
Inside the program, we set up initialisations and inputs/outputs. We needed pin 0 to be a digital output for the LED, pin 1 to be an analog input from the light sensor, and pin 3 to be a digital output for the PWM signals. The PWM signals were generated manually by switching the digital output on and off very quickly, so we only needed to set up initialisations for the ADC. The initialisations for the ADC are in Appendix A1.

After everything was initialized, the heart of the program was put inside an infinite while loop. I look at the while loop as two processes. There are two separate sections of the while loop which do not directly affect each other. The first part deals with the ADC and LED, while the second part deals with the PWM.

In the first part concerning the LED and ADC, the LED counter variable is checked first. The value of the LED counter will indicate whether or not a conversion has been started but not read. If no conversion is pending, then the counter will continue counting until it is time to start the next conversion (when the counter reaches a rollover threshold). When the conversion is started, the program will check to see if it is finished in each loop iteration. When the conversion finishes, the value is read, the value is used to shut the LED on or off, and the counter is reset so that it can continue counting until the next time the program needs to do a conversion. This method saves a lot of clock cycles meaning that this part of the while loop will not likely hinder any other parallel parts of the while loop. On the contrary, the rollover threshold may have to be adjusted depending on how long parallel tasks in the loop take, or else more greedy processes can slow down this code. This code is shown in Appendix A2.

The second part of the loop uses two counters. The first counter will change the duty cycle of the PWM, hence changing the direction of the sesno-motor, every time the counter rolls over. The other counter is used to generate the PWM signal. The counter will keep track of when the digital output needs to be turned on or off. For example, if the intended duty cycle is 10%, the output will be high until the counter reaches ten, then the output will be low until the counter reaches one-hundred and resets. This code is shown in Appendix A3.

Below is a program flowchart to model the course of the program.


START
Initialise System
Initialise ADC
Is an ADC conversion running?
No
Yes
Should I start an ADC conversion?
Yes
No
Is the ADC conversion done?
No
Yes
Start ADC conversion
Should I change PWM duty cycle?
No
Yes
Is ADC value over threshold?
No
Yes
LED off
LED on
Should PWM output pin currently be high?
No
Yes
Change PWM duty cycle
Set PWM output pin to LOW
Set PWM output pin to HIGH

Section 8: Trouble Shooting /1 points

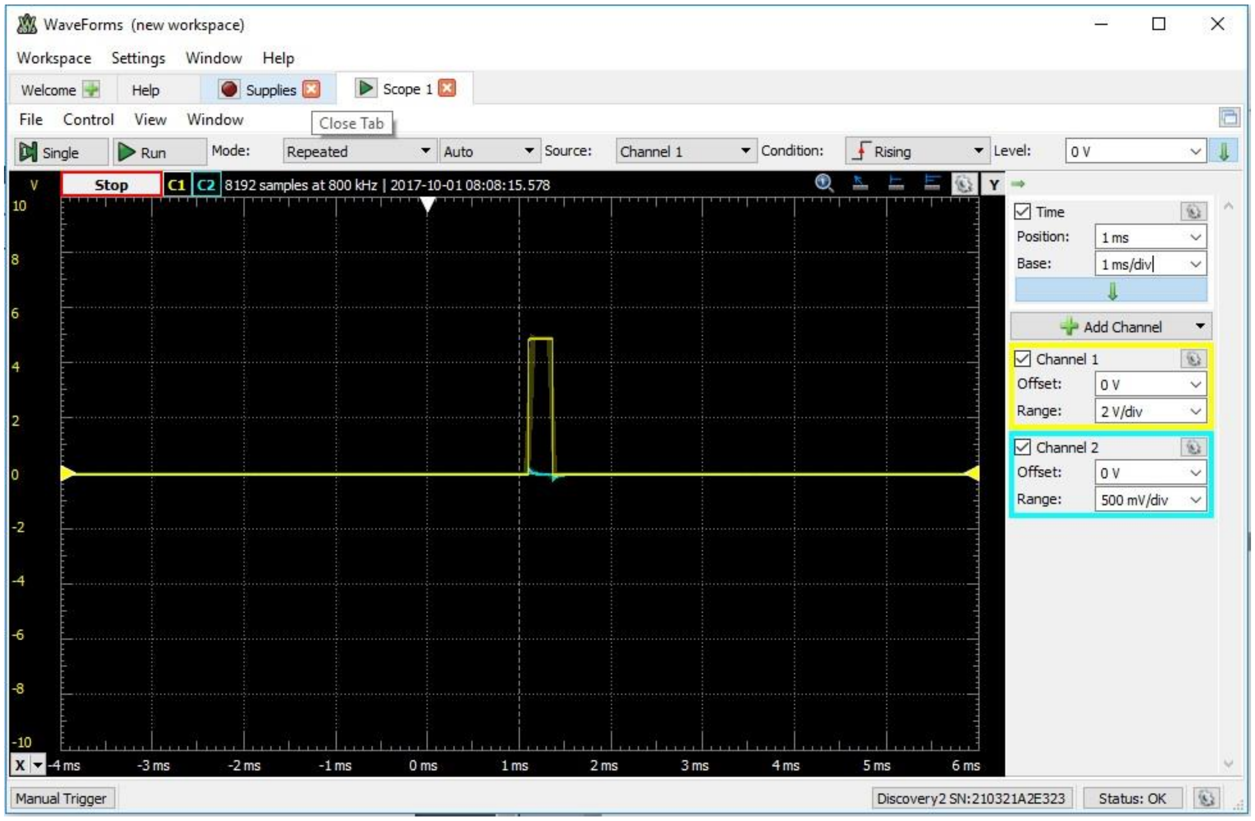
Issue 1:
On 22-Sep-2017, Derek and I were experiencing issues with programming the microcontroller. Every time we tried to program the microcontroller via MPLABX and the PICkit 3, we would receive this error message: <<Target device was not found (could not detect target voltage VDD). You must connect to a target device to use PICkit 3.>>. After five hours of changing settings and trying other computers, we found out that we needed to power the microcontroller via an external source. We originally thought that the microcontroller could be powered through the PICkit.

Issue 2:
On 25-Sep-2017, Derek and I were trying to program the light sensor and LED code onto the microcontroller. Unfortunately, although the code built after a few modifications, the microcontroller did not perform as expected. We believed that there may have been errors with the way we were initializing the ADC. I set up a test program that would just read from the ADC once and blink the LED the same amount of times as the value that was read. For example, if fifty was read from the ADC, the LED would blink fifty times. We soon discovered that the ADC was only reading zero. We spent at least two hours changing initialisation flags and adding new ones from the device specification until we made one important change. In my test code, the program would only read from the ADC once and blink the LED based on what the ADC read. This seems like a decent idea, but the issue is that for some reason, the ADC always reads zero on the first conversion. As soon as we updated the test code so that the ADC would read more than once, the code with the new ADC initialisations worked perfectly.

Issue 3:
On 26-Sep-2017, Derek, Kyle, and I were putting together the LED and PWM code. We spent hours trying to tweak the PWM initialisations so that a PWM signal would be generated. We were unsuccessful and tired, so we went for the non-ideal solution and tried implementing PWM signals via a counter and delay loop. However, we noticed that the microcontroller was no longer reprogramming for an unknown reason. I sent an email to the TA and he responded. Derek tried a few suggestions and got the microcontroller to reprogram again by switching a 1 k-ohm resistor with a 10 k-ohm resistor. We had been using a 1 k-ohm resistor up to that point, so we still do not know why everything suddenly stopped working, but on the bright side, the problem is somehow fixed. The next day, Derek and I tried the code without the delays and only the counter. We got the proper signal but the senso-motor would only spin one way. Kyle came by later to help, and he fixed the problem within minutes by changing the duty cycles. At that point, we now had the whole project working properly.

Section 9: Results /0.5 points

Generating the PWM signal was especially difficult because we could not figure out the ideal method of using the internal clocks and functions to generate PWM signals. Our biggest concern with using a counter though was that it would be difficult to obtain the desired frequency. In the end though, everything worked out fine. We were able to generate PWM signals via counters and we verified that they were legitimate by viewing the waves on an oscilloscope, which also measured frequency and duty cycle. Two screenshots of the varied duty cycles are shown below.


WaveForms (new workspace)
Workspace Settings Window Help
Welcome Help Supplies Scope 1
File Control View Window
Close Tab
Single Run
Stop C1 C2
V
10 8 6 4 2 0 -2 -4 -6 -8 -10
X -4 ms -3 ms -2 ms -1 ms 0 ms 1 ms 2 ms 3 ms 4 ms 5 ms 6 ms
Y
Add Channel
Manual Trigger

Mode: Repeated | Auto | Source: Channel 1 | Condition: Rising | Level: 0 V

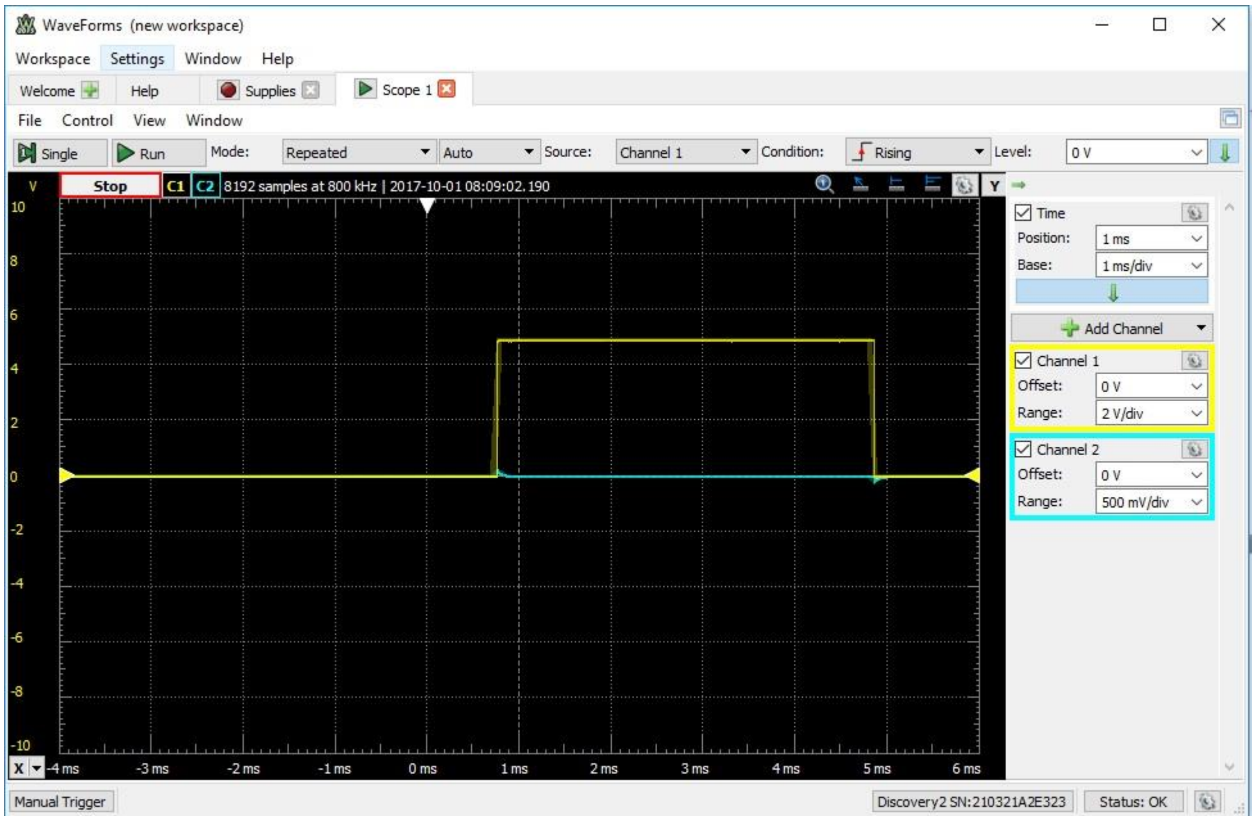
8192 samples at 800 kHz | 2017-10-01 08:08:15.578

Time
Position: 1 ms
Base: 1 ms/div

Channel 1
Offset: 0 V
Range: 2 V/div

Channel 2
Offset: 0 V
Range: 500 mV/div

Discovery2 SN:210321A2E323 Status: OK



Mode: Repeated Auto Source: Channel 1 Condition: Rising Level: 0 V

8192 samples at 800 kHz | 2017-10-01 08:09:02.190

Time
Position: 1 ms
Base: 1 ms/div

Channel 1
Offset: 0 V
Range: 2 V/div

Channel 2
Offset: 0 V
Range: 500 mV/div

Discovery2 SN:210321A2E323 Status: OK

A1.

```
  void ADC_Init(void) {
          //ADCON0 = 0;
    ADCON1 = 1;
    ADCON2 = 0;
    ADCON3 = 0;                    // set adc threshold reg to 0
    ADACT = 0;                     // disable adc auto conversion trigger (is this how????????????)
                           // 5 already disabled, genius
    ADSTATbits.ADAOV = 0;                    // I hope this works?
    ADCAP = 0;                     // 7
    ADPRE = 0;                     // 8

    ADCON0bits.ADFM = 0;                         // left justified alignment?
    // maybe ^ = ADCON0bits.ADFRM0 = 0???
    ADCON0bits.ADON = 1;                // adc enable
    ADCON0bits.ADCS = 0b101;                //  Clock supplied by FOSC, divided according to
ADCLK register? (0) vs. Clock supplied from FRC dedicated oscillator
    ADCON0bits.ADCONT = 0;              // disable continuous operation

    //ADNREF = 0;                //negative voltage reference: Vss
    //ADPREF1 = 0;               //positive voltage reference: Vdd
    //ADPREF0 = 0;               //positive voltage reference

    //ADCON0 = ADCON0 | 0x44;
    //ADCON0 = ADCON0 & 0XE7;

    //ADCON0bits.VCFG = 0;
    ADREFbits.ADNREF = 0;
    ADREFbits.ADPREF0 = 0;
    ADREFbits.ADPREF1 = 0;         //V references

    ADSTATbits.ADSTAT0 = 1;
    ADSTATbits.ADSTAT1 = 1;        //ADC conversion module
    ADSTATbits.ADSTAT2 = 0;

    ADPCHbits.ADPCH0 = 1;
    ADPCHbits.ADPCH1 = 0;          //ADC positive channel select
    ADPCHbits.ADPCH2 = 0;
    ADPCHbits.ADPCH3 = 0;
    ADPCHbits.ADPCH4 = 0;
    ADPCHbits.ADPCH5 = 0;
 }
```

A2.

```
if (led_counter < 0) {
        // adc conversion is running

        // only if conversion is done,
        // update LED based on LED_THRESHOLD (pre-defined)
        // and reset counter
        if (ADCON0bits.GO == 0) {
        // conversion = done

           // update led
           if (ADRESH < LED_THRESHOLD) {
                        // light is high, photoresistor is low, analog input
                //voltage is low, LED goes off
                        LATAbits.LATA0 = 0;
           } else {
                        LATAbits.LATA0 = 1;
           }

           led_counter = 0;       // start counter over again
        }

} else if (led_counter >= LED_ROLLOVER) {
        // it is time to start an adc conversion

        ADCON0bits.GO = 1;   // start conversion
        led_counter = -1;   // indicates that conversion is running

} else {
        // neither so just update counter
        ++led_counter;
}
```

A3.

```
// check if you must change direction
// note: pwm_counter updated at end
if (pwm_counter >= PWM_ROLLOVER) {
        // change direction and reset counter
        switch (direction) {
                case LEFT:
                        direction = RIGHT;
                        break;

                default:
                        direction = LEFT;
        }
        pwm_counter = 0;
} else {
        // keep direction, manage duty cycle

        // check if RA3 is on
        if (LATAbits.LATA3 == 0b1) {

                // turn off only when counter reaches direction, which is percentage of duty
                // cycle
                if (duty_counter >= direction) {
                        LATAbits.LATA3 = 0b0;
                }

                ++duty_counter; // always increment counter

        } else {

                // RA3 is off, turn on only when duty_counter reaches 100 (%) which
                //means that the cycle is complete
                if (duty_counter >= 100) {
                        // turn on and reset counter
                        LATAbits.LATA3 = 0b1;
                        duty_counter = 0;
                } else {
                        // otherwise, increment counter so that it is out of 100 instead of 101
                        // 0-99 before reset = 100
                        duty_counter++;
                }
        }
}
++pwm_counter;      // update pwm counter
```