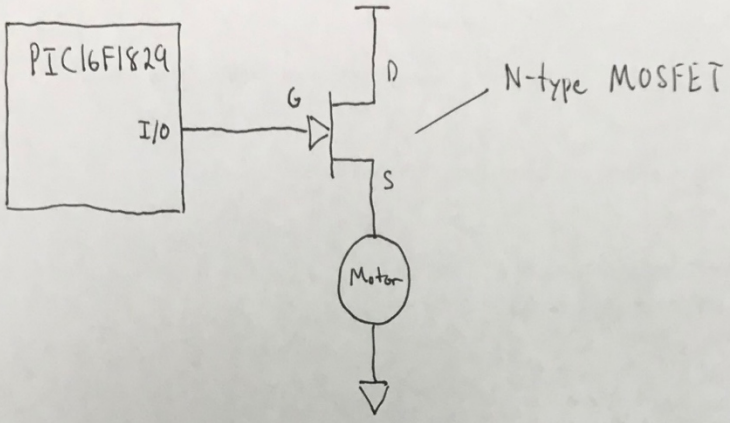
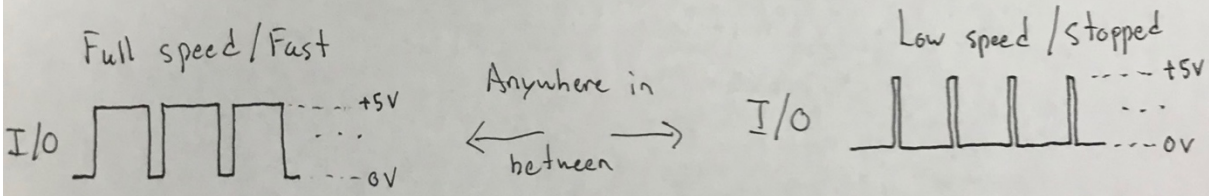
1. The PIC16F1829 microcontroller has 4 pulse width modulation (PWM) modules built in.
2. My diagram for a microcontroller generated PWM output signal is below.





The term duty cycle refers to the portion of time that the signal is on (+5v) with respect to the duration of the ~~period~~ cycle period. Duty cycles can range from 0% to 100%. in most cases and the amount of power delivered to the motor ~~varies~~ corresponds to the duty cycle value.

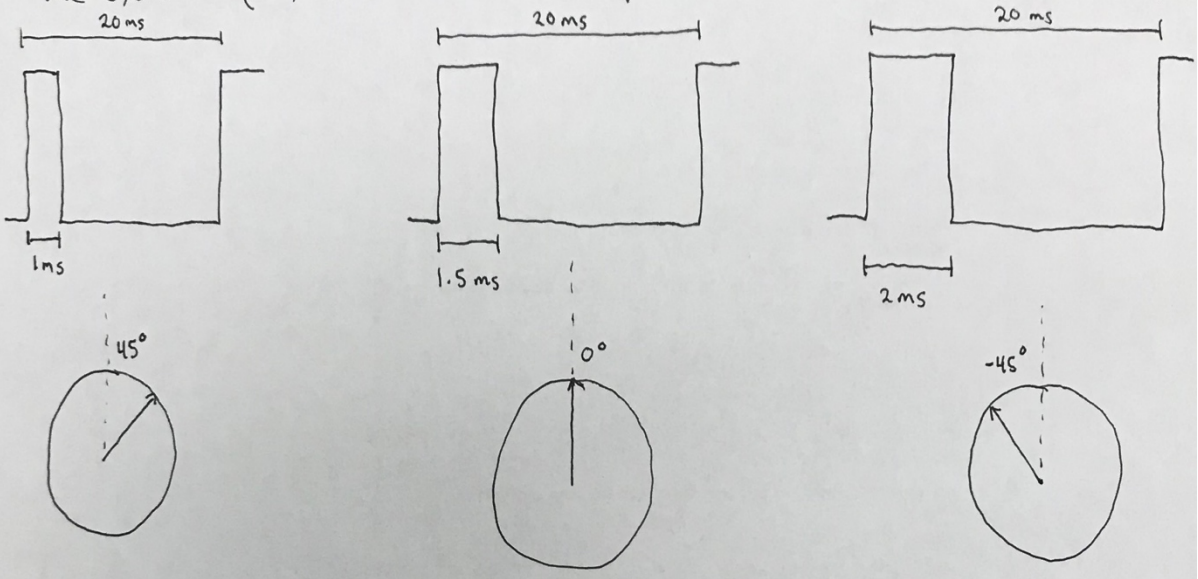
3. Screenshots of my source code using the onboard PWM are pasted below.

```
#include <16F1829.H>
#device ADC=10             // Preprocessor directive to define the size of the ADC return value
// #include <stdlib.h>


//#device adc=8

#use delay (clock=8000000)    //  8 MHz, 0.5 us instruction cycle, this MUST come before the #include <oled_16F1829_i2c.h> to define delays in header file such as "delay_us(1)"
//   #use fast_io (A)         //

#include <oled_16F1829_i2c_Edit.h>

#fuses WDT, INTRC_IO, PUT     // internal oscillator operation with RA6 and RA7 as digital I/O, ref 16F1829.h, enable Power Up Timer
#fuses NOMCLR                 // disable /MCLR pin
#fuses NOPROTECT              // disable code protection
// #fuses NOLVP

// Bob's pin definitions for PIC16F1829 for I2C communication with OLED are in oled_16F1829_i2c.h

#define led pin_a2         // BE SURE TO REDEFINE THE LED PIN TO MATCH YOUR CIRCUIT

unsigned int i = 0;
unsigned int j = 0;

float t = 0;               // floating point variable for "temperature"
float v = 0;               // floating point variable for "voltage"

// Ross' definitions for PIC16F1829 for ADC setup and usage
#define analog4 4          // define channel 4 of the ADC to be analog4
#define max_voltage 4.096  // define the maximum voltage to be Vss (This needs to be chagned if the voltage range changes)
#define num_states 1024    // Number of states for the given bit depth "ADC=8" (This needs to be changed if the bit depth changes)

unsigned int16 ADCout;     // 16 bit integer value for storing ADC output

////////////////////////////////////////
// SUBROUTINES

void init_ports(void) {

   SETUP_OSCILLATOR(OSC_8MHZ|OSC_INTRC);
   SETUP_WDT(WDT_8S);                     //  WDT set to 8 second period

   // Setup the ADC
   setup_adc(ADC_CLOCK_INTERNAL);          // ADC uses the internal clock
   setup_adc_ports(sAN4, VSS_FVR);         // ADC sets all ports to ANALOG and specifies the range to be 0 V to FVR
   set_adc_channel(analog4);               // ADC reads from channel analog4
   delay_us(10);                           // A short delay is required after every time you change ADC channels

   // Setup the reference voltage
   setup_vref(VREF_ON | VREF_ADC_4v096);    // FVR is turned on and set to ADC 4.096 V

   // Setup timer 2 for the CCP module
   setup_timer_2(T2_DIV_BY_1, 127, 1);

   // Setup the CCP3 module
   setup_ccp3(CCP_PWM);
}



////////////////////////////////////////
// PIC12F1829 goes here at RESET

void main() {

   init_ports();              //  Initialize ports
   restart_wdt();
   delay_ms(50);              //  insert a short delay before starting the system up


//  the "bit bang" code below and in the *.h file to emulate I2C
//  to run the OLED Display was developed by Kenny Donnelly

output_low(res);
   delay_ms(10);
   output_high(res);
   delay_ms(10);
```

```
initialise_screen();
   delay_ms(10);
   clear_screen();
   delay_ms(10);
   fill_screen();
   delay_ms(10);
   clear_screen();
   delay_ms(10);

CYCLE:

   i = 0;              //   counter for testing "moving text"
   clear_screen();     //   clear OLED screen

   while(true){

     // Control ADC to only read a new value at about 1 Hz
     ADCout = read_adc();

     // Set the PWM duty cycle to be proportional to the ADC value
     set_pwm3_duty(ADCout/2);  // Scale to increase range

       v = ADCout * (max_voltage / (num_states-1) ); // Calculate the analog voltage
       delay_ms(800);



      restart_wdt();

  oled_write_command(0xb0);
  oled_write_command(0x00);
  oled_write_command(0x10);

//  uncomment the following line to zoom in to text (top four lines of text only)
   oled_zoom();


//  the following command places the next character on line 0 (the 8 text lines are numbered 0 to 7)
//  and column 42 (there are 128 pixels from left to right on this 128 x 64 pixel display)
//  note that text character placement on the OLED display is therefore:
//       128 columns (individual pixels) numbered 0 to 127
//         8 lines, numbered 0 to 7, where each line is one character tall
//           each character is 8 pixels tall
//           8 x 8 = 64 pixels = the full display height
//  By placing the next character at this location the text "BMME580" is centered at the top
//  question: how many characters fit into the 128 characters across the OLED display
  oled_gotoxy(0,0);
 printf(oled_printchar,"ADC Output = %4Lu ", ADCout);  // Displays the digital representation of the voltage


   oled_gotoxy(1,0);
 printf(oled_printchar,"ADC Voltage = %2.3f ", v);   //  see the PIC_C User Manual under "printf()" to format numbers in the display

 printf(oled_printchar,"V");     // notice how the "V" displays directly after the last printed character

    oled_gotoxy(2,0);
 //printf(oled_printchar,"temperature = %1.1f $F", t);
 //  look in the oled_16F1829_i2c.h file to see why the "$" character actually
 //  displays a the temperature degree symbol "∞"
 //  answer: I "remapped" the pixels in the BYTE const TABLE5 for "$".
 //  So I changed the ASCII Character Table for the OLED Display in the *.h file

   oled_gotoxy(3,0);

   for (j=0; j<i; j++) printf(oled_printchar," ");


 printf(oled_printchar,"Ross Rucho");

   t = t + 1.2;      //  increment the "temperature" variable
   i = i + 1;        //  nudge test to the right one character
   if(i>22) {
      i = 0;   // prevent moving text from scrolling down to next row on OLED display
      clear_screen();      // clear OLED screen
   }
 }
      restart_wdt();

goto CYCLE;  //  cycle indefinitely until battery is removed or battery runs out of power
} // main
```

4. My labeled drawing of a model radio servo control signal is below.

Servo motors use pulse width modulation (PWM) to encode position instructions. The cycle frequency is 50 Hz and the pulse width varies from 1 ms to 2 ms.



Typical ~~motors~~ servo motors vary between $\pm 45^\circ$ or $\mp 90^\circ$ with the extreme states corresponding to the pulse widths labeled above. Pulse widths between these extremes vary the position proportionally.