Section 1: General Lab Info   /0.5   points

Microprocessors II and Embedded Systems

EECE.4800

Lab 2: Interfacing with a sensor device on an embedded computer system

Yan Luo

Group 1

Grayson Colwell

Handed in: October 30, 2017

Lab Due: October 30, 2017

Section 2: Contributions /1 points

1. Group Member 1 – Grayson Colwell

Responsible for creating the PIC module. Created the servoRotate0 function which would rotate the servo to the 0 position. Created the servoRotate30 function which would send the Ack message back to the Galileo and rotate the servo 30 degrees. Created the servoRotate90 function which would send the Ack message back to the Galileo and rotate the servo 90 degrees. Created the servoRotate120 function which would send the Ack message back to the Galileo and turn the servo 120 degrees. Created the set_receive function which enabled the PIC to receive digital signals from the Galileo. Created the receive_msg function which would decode the message sent from the Galileo and based off that message, return a value to the main loop. Created the sensorReset function which sends the ACK message back to the Galileo and sets the servo to position 0 by calling the servoRotate0 function. Added the ADC_Init function from lab 1 to initialize the ADC of the PIC. Added the ADC_conversion_results function from lab 1 to send the value read in by the ADC to the main function to determine whether or not the LED turns on. Created the sensorPing function which sends the ACK message to the Galileo and refreshes the ADC value. Helped create the sendADCResults function which sends the most recent ADC value in hexadecimal to the Galileo and the ACK message. Created the main function which initializes the system and ADC, and then loops infinitely and calls each function based off the message received from the Galileo

2. Group Member 2 – Andrew Macgregor

Responsible for configuring the GPIO ports of the Galileo. Created the setGPIOMode function, setGPIODirection function, writeGPIO function, openGPIO function, readGPIO function, writeNibble function, readNibble function, and all the functions pertaining to the message received back from the PIC.

3. Group Member 3 – Jose Velis

Responsible for creating the main galileo code. This code stated on the computer connected to the Galileo the options for the message to be sent to the PIC. This function also sets the output data pins of the Galileo to send the message to the PIC.

Section 3: Purpose /0.5 points

The purpose of this lab was to gain an understanding of how to connect the PIC16F18857 to an embedded computer. The embedded computer being connected to in this lab is the Intel Galileo. In order to do this, a customized bus protocol needs to be created to allow sensor data read in by the internal ADC of the PIC to be sent to the Galileo. The PIC should also be able to read in commands sent by the Galileo to rotate the servo.

Section 4: Introduction /0.5 points

To be successful with this lab, there were a few key concepts that needed to be understood. This lab was centered on using the Intel Galileo to communicate with the PIC16F18857. In order to have the Galileo function correctly, a file system needed to be created in order to ensure that each of the data buses was able to send and receive messages during the correct timing cycle as defined in the specification. Once this was configured correctly, the Galileo needed to be able to send specific digital messages to the PIC which specified which command the Galileo wished the PIC to do. The PIC then needed to be configured in able to read these digital messages sent from the Galileo and based off that message, do a specified task. The most important thing to make sure when coding the PIC to respond to these messages was the timing. In order for the Galileo to receive the messages correctly, the PIC needed to follow the bus protocol to ensure the Acknowledge messages were sent at the correct time. If the Acknowledge message was not sent at the correct time, the Galileo would never receive a message back from the PIC and would crash.

Section 5: Materials, Devices and Instruments /0.5 points

- Servo Motor SG90: Motor which rotates between 0 and 180 degrees depending on the duty cycle.
- Photoresistor (LDR): Resistor with a variable resistance depending on the amount of light that is exposed to it.
- PIC16F18857: Embedded microcontroller which responds to messages sent by the Galileo to control the servo or light the LED based on the ADC value.
- Light Emitting Diode: Used as a visual indicator for when the LDR detects darkness.
- Analog Discovery, Serial #210244630597: Used the Discovery to act as a voltage source and also utilized the logic module to view the timing of the Galileo to ensure our timing on the PIC lined up with the Galileo.
- 1kΩ and 330Ω Resistors: the 1k Resistor was used to form a voltage divider with the LDR and the 330 Resistor was in series with the LED to ensure that it didn't burn out from being given too much voltage.
- Intel Galileo Gen2 embedded computer: The embedded computer used to communicate with the PIC

Section 6: Schematics /0.5 *points*

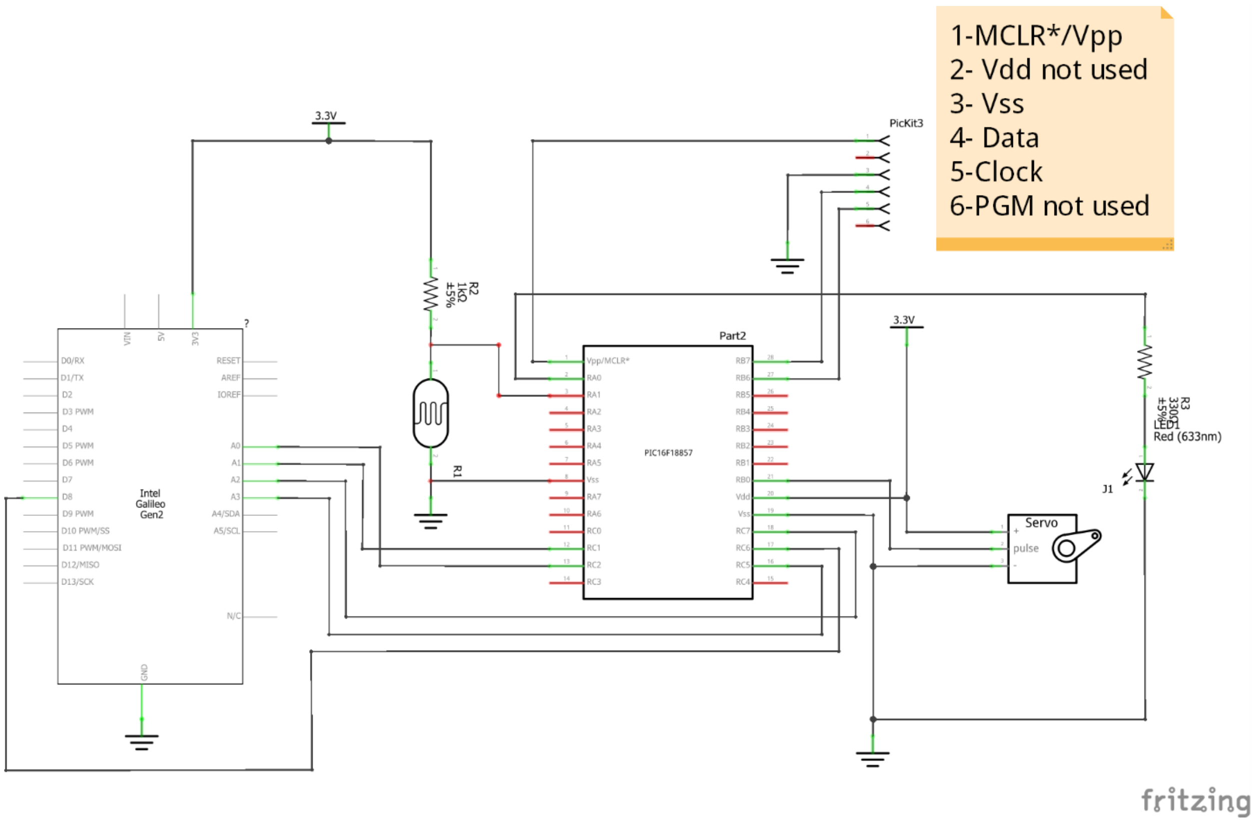


Figure 1: Pinout and layout of the circuit used for this lab drawn using fritzing.

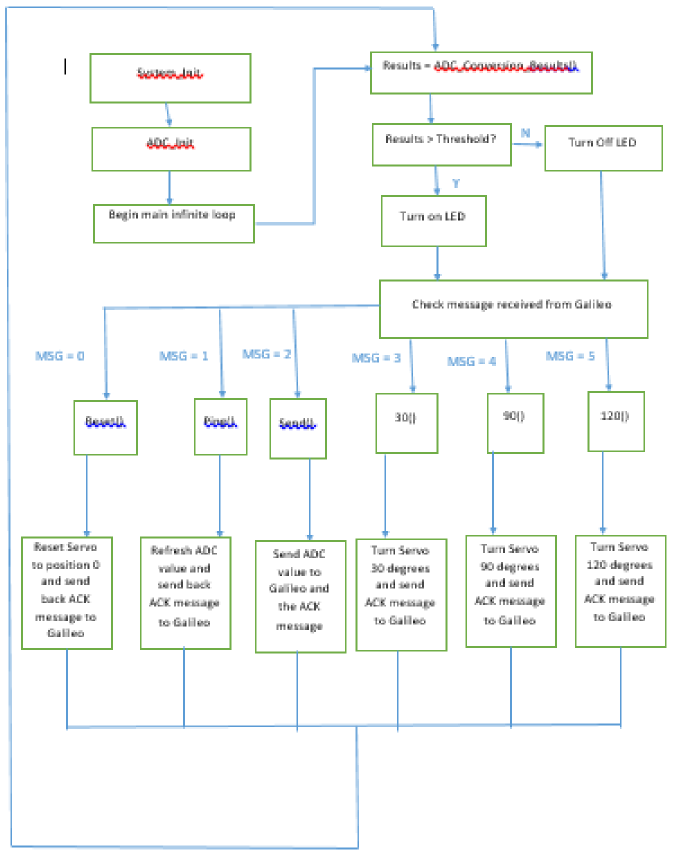
In Figure 1, the schematic shown is how the circuit was implemented by our group. One modification that we needed to make which makes the diagram a little unclear is how we oriented the data inputs from the Galileo to the PIC. Because the RC4 pin on our PIC was broken, we had to move the signal coming from D2 to RC7.

Section 7: Lab Methods and Procedure /2 *points*

Hardware Design:

When looking at the created circuit, there were essentially no changes from the circuit that was used in the previous lab compared to the circuit used in this lab. Once again, Vcc was set to 3.3V to ensure that the PIC did not receive overvoltage. Five new digital connections were added to the PIC, the strobe signal connected to RC6 and then four data input/output pins were connected to pins RC2, RC1, RC7, and RC5. These data input pins would connect to D0, D1, D2, and D3, respectively, on the Galileo. The analog circuit remained the same from the previous lab because the PIC needed to respond to commands from the Galileo and because we were able to have last lab successfully work, we felt like we didn't need to change any of the circuit from the previous lab.

Software Design:



System_Init
ADC_Init
Begin main infinite loop
Results = ADC_Conversion_Results()
Results > Threshold?
N
Turn Off LED
Y
Turn on LED
Check message received from Galileo
MSG = 0
MSG = 1
MSG = 2
MSG = 3
MSG = 4
MSG = 5
Reset()
Ping()
Send()
30()
90()
120()
Reset Servo to position 0 and send back ACK message to Galileo
Refresh ADC value and send back ACK message to Galileo
Send ADC value to Galileo and the ACK message
Turn Servo 30 degrees and send ACK message to Galileo
Turn Servo 90 degrees and send ACK message to Galileo
Turn Servo 120 degrees and send ACK message to Galileo

Figure 2: Software Flowchart for the PIC code.

In Figure 2, the software design flow chart for the PIC code is illustrated. Starting from the main function, the first thing done is to initialize the system using the system_initialize function. From there, the ADC was configured using the ADC_Init function, which was the same function used in the previous lab. After doing this, the main function would enter an infinite loop where it would wait for a command from the Galileo. The first thing that was done in this loop would be to check the ADC value obtained from the ADC_Conversion_Results function, the same function used in the previous lab, and compare this value to a threshold value. If the value obtained was greater than the threshold value, the LED connected to RA0 would turn on and if the value was less than the threshold value, the LED would turn off. This checking would occur after every time the Galileo sent a message to the PIC so the ADC value would refresh constantly. After checking the ADC value, the PIC would then check for a message from the Galileo. There were 6 message options that the Galileo could send to the PIC, reset the sensor, ping the sensor, send the ADC results to the Galileo, Rotate the servo motor 30, 90, or 120 degrees, and no message. If the PIC received the reset command, the PIC would wait one timing cycle before sending the Acknowledge message back to the Galileo and then reset the servo to its initial position of 0 by calling the servoRotate0 function. All of the servo functions worked by using the __delay_ms(x) function to manually input the duty cycle. To have the servo rotate to the zero position, a delay of 1.4ms was used, to have the servo rotate 30 degrees, a delay of 1.75ms was used, to have the servo rotate 90 degrees, a delay of 2.1ms was used, and to have the servo rotate 120 degrees, a delay of 1.1ms was used. If the PIC received the ping command, the PIC would wait one timing cycle before sending back the Acknowledge message and then wait three timing cycles before ending. These three extra timing cycles for this function were necessary as the Galileo would not receive a proper response without them. If the PIC received the send ADC results command, the PIC would obtain the most recent ADC value using the ADC_conversion_results function and store the value into three nibbles. The first nibble contained the highest 2 bits, the second nibble contained the next highest 4 bits, and the last nibble contained the lowest 4 bits. After waiting two timing cycles, the PIC would send the first nibble to the Galileo. After waiting four timing cycles, the PIC would then send the second nibbles bits. After waiting four more timing cycles, the PIC would send the last nibbles bits to the Galileo. Lastly, after waiting to go through four more timing cycles, the PIC would send the Acknowledge message to the Galileo so that it knows to stop requesting the ADC value. The last three commands that the PIC could receive were all very similar due to the fact that they all turned the servo motor. The first command would turn the servo 30 degrees, the second command would turn the servo 90 degrees, and the last command would turn the servo 120 degrees.

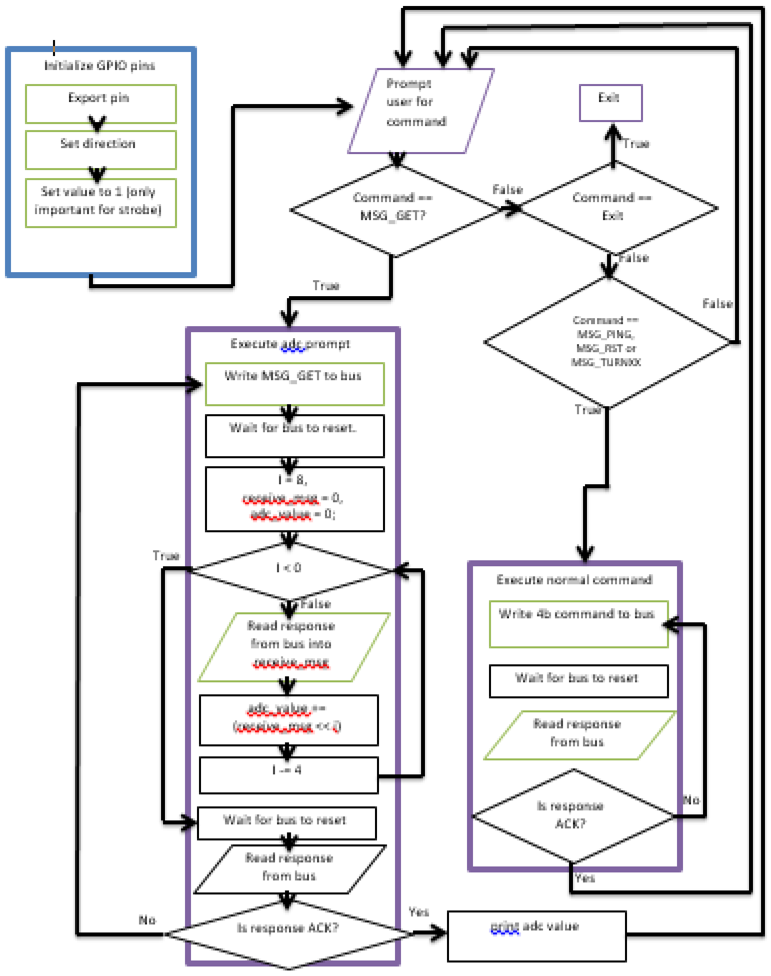


Figure 3: Software Flowchart for the Galileo Code.

In Figure 3, the software flowchart that the Galileo Code followed is illustrated. The Main function of code essentially works by initially setting the five GPIO pins. One pin is set to always be outputting, the strobe pin, which is used as the timer for receiving and sending messages. The other four pins were used as the input/output pins for sending and receiving messages to and from the PIC. The initial state of all the pins were set to be sending a message out. After initializing the pins, the Galileo would then prompt the user to choose a command to send to the PIC. These commands were: Reset, Ping, Get ADC Value, Turn servo 30 degrees, Turn servo 90 Degrees, and Turn servo 120 degrees. After sending any of these commands, the Galileo would wait for an acknowledge message from the PIC before exiting out of its command and prompting the user to see if they want to do another command. The one command that was different from the others was the Get ADC value function because this function required 10 bits of data to be read in by the Galileo. To do this, the Galileo utilized a four loop that ran 3 times and received the highest two bits first, followed by the next four bits, and lastly the last four bits. After receiving all 10 bits, the Galileo would then wait to receive an Acknowledge message from the PIC to ensure the PIC sent the right data.

Section 8: Trouble Shooting /1 points

Issue 1: Broken Pin on the PIC:

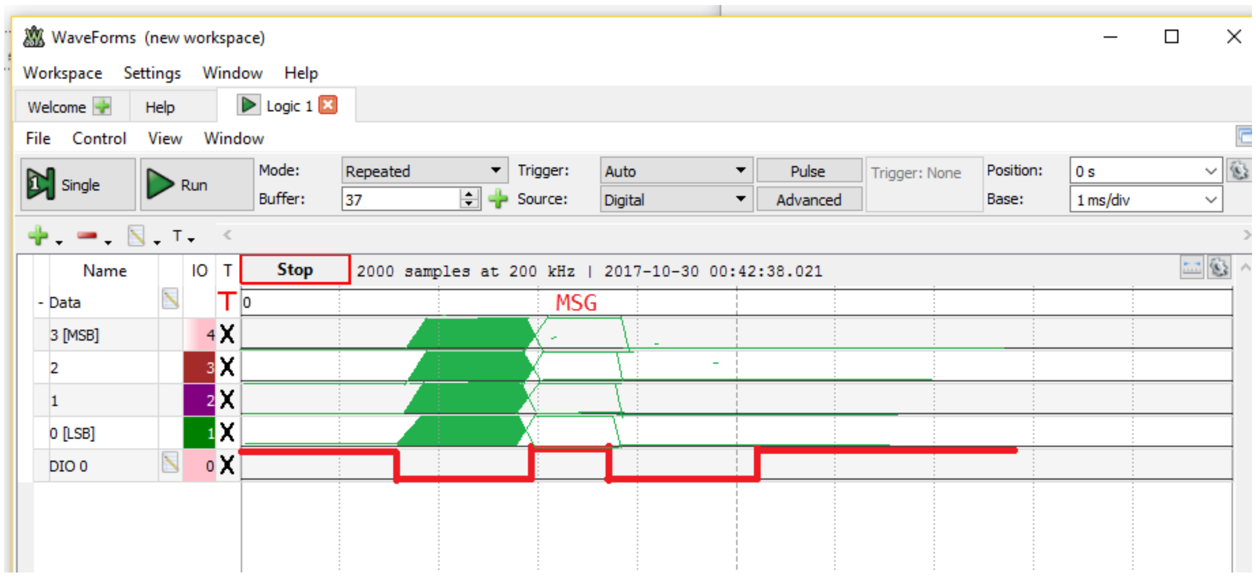
After doing each of our code sections and setting up the data input/output pins on the Galileo, we attempted to communicate with the PIC by connecting the data in/out of the Galileo to the PIC on PORTC pins RC1, RC2, RC4, and RC5. After testing to see whether or not the PIC was receiving the right values from the Galileo, we discovered that RC4 was never receiving data other than 0. After pulling out the PIC, we discovered that the pin RC4 was actually broken off from the PIC and would not be able to be used. To remedy this situation, all we had to do was go through our code and replace RC4 with RC7 and move the wire connected to RC4 to RC7. After doing this, we no longer had an error with reading in values from the Galileo to the PIC.

Issue 2: Timing errors in our return value functions:

As we began testing the sending and receiving of messages between the PIC and the Galileo, we quickly discovered that the Galileo would never receive a proper acknowledge message back from the PIC and cause each of the Galileo functions to loop infinitely trying to get an acknowledge message from the PIC. This error was happening on almost every command send from the Galileo and we were stumped for a while because it seemed like we were following the proper bus protocol. When we began debugging the PIC code, we noticed that one command seemed to always be working. Our reset command was always working and looked nearly identical to all of our other commands and so we spent a lot of unnecessary time trying to figure out what was wrong with our code. In the end, to solve our problem, we had to add a printf() statement to the beginning of each of the PIC functions in order to correctly get the timing. This makes very little sense, but, after adding these printf() statements to the beginning of each function, the timing lined up and the acknowledge message was sent back to the Galileo and was recognized. A tool that proved to be very useful in debugging our timing problems was the Analog Discovery's Logic module and from this module, we were able to see in real time the timing of the Galileo as seen from the strobe. This saved us countless hours of debugging as we could tell when our timing was off and could remedy the timing by adding delay loops.

Section 9: Results /0.5 points

When considering the results from this lab, there were not many tangible results that could be identified. The bulk majority of this lab was building a communication protocol between the PIC and Galileo and all of the results from this lab were nearly identical to the previous lab. The same ADC module was used in this lab and the only new thing added was how we read the ADC value. This value was sent to the Galileo by sending the upper two bits first, followed the next highest four bits, and lastly sending the lowest four bits.



Mode: Repeated
Buffer: 37
Trigger: Auto
Source: Digital
Trigger: None
Position: 0 s
Base: 1 ms/div

```
2000 samples at 200 kHz | 2017-10-30 00:42:38.021
```

Figure 4: Timing Diagram of a successful receive and acknowledge message.

In Figure 4, the timing diagram shown for the Galileo sending a message to the PIC and the PIC sending back the acknowledge message is displayed. The Pic receives the message and based off the message, waits a timing cycle to send back the acknowledge message to let the Galileo know it responded to the message and was ready to receive another message.

Section 10: Appendix

A1: PIC Code;

```
#include <pic16f18857.h>
#include "mcc_generated_files/mcc.h" //default library

#define MSG_RESET   0x0
#define MSG_PING    0x1
#define MSG_GET     0x2
#define MSG_TURN30  0x3
#define MSG_TURN90  0x4
#define MSG_TURN120 0x5
#define MSG_ACK     0xE
#define MSG_NOTHING 0xF

/* Circuit Connections
   Signal STROBE   RC6
   Signal D0       RC2
   Signal D1       RC1
   Signal D2       RC7
   Signal D3       RC5
 *
 * Analog - RA1
 */

void servoRotate0() //0 Degree -> reset servo position
{
  unsigned int i;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(1.4);
    PORTB = 0;
    __delay_ms(18.6);
  }
}

void servoRotate30() //30 Degree
{
  printf("Delay me just a little bit\n");
  while(PORTCbits.RC6 == 1)
  {

  }
  unsigned int i;
  TRISC = 0b01000000;
    //send the ACK message to the galileo
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(1.75);
    PORTB = 0;
    __delay_ms(18.33);
  }
}

void servoRotate90() //90 Degree
{
  printf("REEEEEEEEEEEEEEEEEEEEEEEE\n");
  while(PORTCbits.RC6 == 1)
  {

  }
  unsigned int i;
  TRISC = 0b01000000;
    //send the ACK message to the galileo
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(2.1);
    PORTB = 0;
    __delay_ms(17.9);
  }
}

void servoRotate120() //120 Degree
{
  printf("Delay me just a little bit\n");
  while(PORTCbits.RC6 == 1)
  {

  }
  unsigned int i;
  TRISC = 0b01000000;
    //send the ACK message to the galileo
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;
  for(i=0;i<50;i++)
  {
    PORTB = 1;
    __delay_ms(1.1);
    PORTB = 0;
    __delay_ms(18.9);
  }

}

void set_receive()
{
  /*
   1.set RC6 as digital input
   2.set RC2, RC3, RC4 and RC5 as digital inputs
  */
    ANSELC = 0; //set portc to digital
    PORTC = 0; //clear portc
    TRISC = 0b11101100; //setting RC2-6 as digital inputs
}

unsigned char receive_msg()
{
    set_receive();
    unsigned char results;
    while(PORTCbits.RC6 == 1)
    {

    }

    while(PORTCbits.RC6 == 0)
    {
        if(PORTCbits.RC5 == 0 &&
           PORTCbits.RC7 == 0 &&
           PORTCbits.RC1 == 0 &&
           PORTCbits.RC2 == 0)
        {
           results = 0x0;
        }
        else if(PORTCbits.RC5 == 0 &&
               PORTCbits.RC7 == 0 &&
               PORTCbits.RC1 == 0 &&
               PORTCbits.RC2 == 1)
        {
           results = 0x1;
        }
        else if(PORTCbits.RC5 == 0 &&
               PORTCbits.RC7 == 0 &&
               PORTCbits.RC1 == 1 &&
               PORTCbits.RC2 == 0)
        {
           results = 0x2;
        }
        else if(PORTCbits.RC5 == 0 &&
               PORTCbits.RC7 == 0 &&
               PORTCbits.RC1 == 1 &&
               PORTCbits.RC2 == 1)
        {
           results = 0x3;
        }
        else if(PORTCbits.RC5 == 0 &&
               PORTCbits.RC7 == 1 &&
               PORTCbits.RC1 == 0 &&
               PORTCbits.RC2 == 0)
        {
           results = 0x4;
        }
        else if(PORTCbits.RC5 == 0 &&
               PORTCbits.RC7 == 1 &&
               PORTCbits.RC1 == 0 &&
               PORTCbits.RC2 == 1)
        {
           results = 0x5;
        }
        else
           results = 0xF;
    }
    while( PORTCbits.RC6 == 1)
    {

    }
```

```
    while(PORTCbits.RC6 == 0)
    {

    }
    return results;
 /* 1.wait strobe high
    2.wait strobe low
    3.read the data
    4.wait strobe high
    5.return the data
    */

}

void sensorReset()
{
    printf("REEEEEEEEEEE");
    while(PORTCbits.RC6 == 1)
    {

    }
    TRISC = 0b01000000;
    //send the ACK message to the galileo
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;
    //reset the servo position
    servoRotate0();
}
void ADC_Init(void)  {
 //  Configure ADC module
 //---- Set the Registers below::
 // 1. Set ADC CONTROL REGISTER 1 to 0
 // 2. Set ADC CONTROL REGISTER 2 to 0
 // 3. Set ADC THRESHOLD REGISTER to 0
 // 4. Disable ADC auto conversion trigger control register
 // 5. Disable ADACT
 // 6. Clear ADAOV ACC or ADERR not Overflowed  related register
 // 7. Disable ADC Capacitors
 // 8. Set ADC Precharge time control to 0
 // 9. Set ADC Clock
 // 10 Set ADC positive and negative references
 // 11. ADC channel - Analog Input
 // 12. Set ADC result alignment, Enable ADC module, Clock Selection Bit, Disable ADC
Continuous Operation, Keep ADC inactive
```

```
    TRISA = 0b11111110;   //set PORTA to input except for pin0
    TRISAbits.TRISA1 = 1;   //set pin A1 to input
    ANSELAbits.ANSA1 = 1;   //set as analog input
    ADCON1 = 0;
    ADCON2 = 0;
    ADCON3 = 0;
    ADACT = 0;
    ADSTAT = 0;
    ADCAP = 0;
    ADPRE = 0;
    ADCON0 = 0b10000100; // bit7 = enabled; bit 2 = 1: right justified
    ADREF = 0;
    ADPCH = 0b00000001; //set input to A1
    //UART initialization
    TX1STA = 0b00100000;
    RC1STA = 0b10000000;
    printf("Initialized ADC\n");
}

unsigned int ADC_conversion_results() {
    ADPCH = 1;

    ADCON0 |= 1;          //Initializes A/D conversion

    while(ADCON0 & 1);             //Waiting for conversion to complete
    unsigned result = (unsigned)((ADRESH << 8) + ADRESL); //0bXXXXXXHHLLLLLLLL
    return result;
}

void sensorPing()
{
    printf("PING\n");
    //PORTA ^= 1;
    while(PORTCbits.RC6 == 1)
    {

    }

    TRISC = 0b01000000;
    //send the ACK message to the galileo
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;
    //PORTA = 0;
    while(PORTCbits.RC6 == 0)
    {

    }
    while(PORTCbits.RC6 == 1)
    {

    }
    while(PORTCbits.RC6 == 0)
    {

    }
}

void sendADCResults()
{
    printf(";)\n");
    unsigned results;
    unsigned char nib1, nib2, nib3;
    //get the ADC value and break it into 3 nibbles
    results = ADC_conversion_results();
    nib1 = (results & 0x300) >> 8;
    nib2 = (results & 0x0F0) >> 4;
    nib3 = (results & 0x00F);
    //only start when the bus is high
    while(PORTCbits.RC6 == 0)
    {

    }
    //waits for the bus to go low
    while(PORTCbits.RC6 == 1)
    {

    }

    TRISC = 0b01000000;
    //send the first nibble
    PORTCbits.RC2 = (nib1 & 0x1);
    PORTCbits.RC1 = (nib1 & 0x2) >>1;
    PORTCbits.RC7 = 0;
    PORTCbits.RC5 = 0;

    //wait for the bus to go high again
    while(PORTCbits.RC6 == 0)
    {
    }

    //wait for strobe to go low again
    while(PORTCbits.RC6 == 1)
    {

    }
    //wait for the bus to go high again - end the write
    while(PORTCbits.RC6 == 0)
    {

    }

    //start the second write when bus goes low
    while(PORTCbits.RC6 == 1)
    {

    }
    //send the second nibble.
    PORTCbits.RC2 = (nib2 & 0x1);
    PORTCbits.RC1 = (nib2 & 0x2) >>1;
    PORTCbits.RC7 = (nib2 & 0x4) >> 2;
    PORTCbits.RC5 = (nib2 & 0x8) >> 3;

    //wait for the bus to go high again
    while(PORTCbits.RC6 == 0)
    {

    }

    //wait for strobe to go low again
    while(PORTCbits.RC6 == 1)
    {

    }
    //bus goes high, end the write
    while(PORTCbits.RC6 == 0)
    {

    }

    //wait for strobe to go low again to start third write
    while(PORTCbits.RC6 == 1)
    {
    }
    //send the third nibble.
    PORTCbits.RC2 = (nib3 & 0x1);
    PORTCbits.RC1 = (nib3 & 0x2) >>1;
    PORTCbits.RC7 = (nib3 & 0x4) >> 2;
    PORTCbits.RC5 = (nib3 & 0x8) >> 3;

    //wait for the bus to go high again
    while(PORTCbits.RC6 == 0)
    {

    }

    //wait for strobe to go low again
    while(PORTCbits.RC6 == 1)
    {

    }

    //wait for strobe to go high again END
    while(PORTCbits.RC6 == 0)
    {

    }

    //wait for strobe to go low again - WRITE ACK
    while(PORTCbits.RC6 == 1)
    {

    }
    PORTCbits.RC2 = 0;
    PORTCbits.RC1 = 1;
    PORTCbits.RC7 = 1;
    PORTCbits.RC5 = 1;

    //wait to go high - G reads the ACK
    while(PORTCbits.RC6 == 0)
    {

    }
    //wait to go low - G done reading
    while(PORTCbits.RC6 == 1)
    {

    }
```

```
        //wait to go high - G is DONE
    while(PORTCbits.RC6 == 0)
    {

    }
}

// Main program
#define ADC_THRESHOLD 0x0380
void main (void)
{
    SYSTEM_Initialize();
    unsigned results;

    unsigned char msg;
    TRISB = 0;
    ADC_Init();
    printf("Starting main\n");
    TRISAbits.TRISA0 = 0; //make sure portA0 is ouput for the LED
    while(1)
    {
    results = ADC_conversion_results();
    if(results > ADC_THRESHOLD)
            PORTA |= 0x01; //turn on LEd
    else
            PORTA &= !0x01; //turn off LED
    msg=receive_msg();
    if(msg == MSG_RESET)
                sensorReset();
    else if (msg == MSG_PING)
        sensorPing();
    else if (msg == MSG_GET)
    {
        sendADCResults();
    }
    else if (msg == MSG_TURN30)
        servoRotate30();
    else if (msg == MSG_TURN90)
        servoRotate90();
    else if (msg == MSG_TURN120)
        servoRotate120();
    else
        (void) 0;
    }
}
```

```
/**
 End of File
*/
```

A2: Galileo Code;

```
/*
 * File:   PIC and Galileo communication
 *
 * simple Galileo program example for main function
 * for UMass Lowell 16.480/552
 *
 * Author: Jose Velis, Andy MacGregor,Grayson Colwell
 * Lab 2 main function Rev_1
 *
 * Created on 10/17/2017
 */

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>


//Linux GPIO number // Shield Pin Name
#define Strobe                (40) // 8
#define GP_4                  (48) // A0
#define GP_5                  (50) // A1
#define GP_6                  (52) // A2
#define GP_7                  (54) // A3

#define GPIO_DIRECTION_IN       (1)//Go HIGH (acoording to handout timing diagram))
#define GPIO_DIRECTION_OUT      (0)//Go LOW (acoording to handout timing diagram))
#define HIGH                  (1)
#define LOW                   (0)

#define GPIO_MODE_PULLUP        (1)
#define GPIO_MODE_PULLDOWN      (2)
#define GPIO_MODE_HIZ           (3)
#define GPIO_MODE_STRONG        (4)

#define SUCCESS                (0)
#define ERROR                 (-1)
```

```
/*User Commands*/
#define MSG_RESET   0x0
#define MSG_PING    0x1
#define MSG_GET     0x2
#define MSG_TURN30  0x3
#define MSG_TURN90  0x4
#define MSG_TURN120 0x5

#define MSG_ACK     0xE
#define MSG_NOTHING 0xF

//Constants
#define BUFFER_SIZE            (256)
#define EXPORT_FILE            "/sys/class/gpio/export"
#define STROBE_DELAY           (1000*20) //10 ms in us
/*************************/

//returns the mode set on success
//returns ERROR(negative) on failure
int setGPIOMode(char* gpioDirectory, int mode)
{
   //find Drive file
   char gpioDrive[BUFFER_SIZE];
   strcpy(gpioDrive, gpioDirectory);
   strcat(gpioDrive, "drive");
   FILE* driveFh = fopen(gpioDrive, "w");

   int n = -1;
   if(mode == GPIO_MODE_HIZ)
   {
      n = fputs("hiz", driveFh);
   }
   else if(mode == GPIO_MODE_STRONG)
   {
      n = fputs("strong", driveFh);
   }
   else if(mode == GPIO_MODE_PULLUP)
   {
      n = fputs("pullup", driveFh);
   }
   else if(mode == GPIO_MODE_PULLDOWN)
   {
      n = fputs("pulldown", driveFh);
   }
```

```
    fclose(driveFh);
    if(n < 0 )
    {
        printf("Error writing to gpio drive file in %s", gpioDirectory);
        return ERROR;
    }

    return mode;
}
//Sets the GPIO pin specified to a new direction
// ALSO sets the mode of the pin.
//returns the direction set on success
//returns ERROR(negative) on failure
int setGPIODirection(char* gpioDirectory, int direction)
{
    //find direciton file
    char gpioDirection[BUFFER_SIZE];
    strcpy(gpioDirection, gpioDirectory);
    strcat(gpioDirection, "direction");
    FILE* directionFh = fopen(gpioDirection, "w");

    //find Drive file
    char gpioDrive[BUFFER_SIZE];
    strcpy(gpioDrive, gpioDirectory);
    strcat(gpioDrive, "drive");
    FILE* driveFh = fopen(gpioDrive, "w");

    int n = -1;
    int m = -1;
    if(direction == GPIO_DIRECTION_IN)
    {
        n = fputs("in", directionFh);
        m = fputs("hiz", driveFh);
    }
    else if(direction == GPIO_DIRECTION_OUT)
    {
        n = fputs("out", directionFh);
        m = fputs("strong", driveFh);
    }

    fclose(driveFh);
    fclose(directionFh);
    if(n < 0 || m < 0)
    {
        printf("Error writing to gpio value file in %s", gpioDirectory);
        return ERROR;
    }

    return direction;
}

//write value (HIGH or LOW) to port specified
//returns value written on success
//returns ERROR (negative) on failure
int writeGPIO(char* gpioDirectory, int value)
{
    char gpioValue[BUFFER_SIZE];
    strcpy(gpioValue, gpioDirectory);
    strcat(gpioValue, "value");
    FILE* valueFh = fopen(gpioValue, "w");

    char numBuffer[5];
    snprintf(numBuffer, 5, "%d", value);

    int n = fputs(numBuffer, valueFh);
    fclose(valueFh);
    if(n < 0)
    {
        printf("Error writin to gpio value file in %s", gpioDirectory);

        return ERROR;
    }

    return value;
}

//open GPIO and set the direction
//returns pointer to string containing the gpio pin directory if successful
//  this needs to get freed after you're done with it
//returns null ptr on error
char* openGPIO(int gpio_handle, int direction )
{
    int n;
    //   1. export GPIO
    FILE* exportFh = fopen(EXPORT_FILE, "w");
    if(exportFh== NULL)
    {
        printf("Couldn't open export file\n");
        fclose(exportFh);
        return NULL;
    }
    char numBuffer[5];
    snprintf(numBuffer, 5, "%d", gpio_handle);
    n = fputs(numBuffer, exportFh);
    fclose(exportFh);
    if(n < 0)
    {
        printf("error writing to export file\n");
        return NULL;
    }

    //form the file name of the newly created gpio directory
    char *gpioDirectory = malloc(BUFFER_SIZE);

    n = snprintf(gpioDirectory, BUFFER_SIZE, "/sys/class/gpio/gpio%d/", gpio_handle);
    if(n >= BUFFER_SIZE)
    {
        printf("Buffer overflow when creating directory name\n");
        free(gpioDirectory);
        return NULL;
    }

    //    2.set the direction
    setGPIODirection(gpioDirectory, direction);

    //    3.set the voltage
    writeGPIO(gpioDirectory, HIGH);

    //return the new gpio directory
    return gpioDirectory;
}


//read value (HIGH or LOW) from port specified
//port direction must be set to input.
//returns port value on success
//returns ERROR (negative) on failure
int readGPIO(char* gpioDirectory)
{
    char gpioValue[BUFFER_SIZE];
    strcpy(gpioValue, gpioDirectory);
    strcat(gpioValue, "value");
    FILE* valueFh = fopen(gpioValue, "r");

    char numBuffer[5];
    char* test = fgets(numBuffer, 5, valueFh);
    fclose(valueFh);
    if(test == NULL)
    {
        printf("Error reading from gpio value %s", gpioDirectory);
        return ERROR;
    }

    return atoi(numBuffer);
}

//Sends a nibble(4 bytes) along the bus following the Bus Protocol.
//does not wait for an ACK.
void writeNibble(unsigned char data,
    char* d0,
    char* d1,
    char* d2,
    char* d3,
    char* strobe)
{
    //set all the ports to output
    setGPIODirection(d0, GPIO_DIRECTION_OUT);
    setGPIODirection(d1, GPIO_DIRECTION_OUT);
    setGPIODirection(d2, GPIO_DIRECTION_OUT);
    setGPIODirection(d3, GPIO_DIRECTION_OUT);
    setGPIODirection(strobe, GPIO_DIRECTION_OUT);

    //start the bus protocol
    //1: pull strobe low
    writeGPIO(strobe, LOW);

    //2: output the nibble to the bus
    writeGPIO(d0, data & 1);
    writeGPIO(d1, (data & 2) >> 1);
    writeGPIO(d2, (data & 4) >> 2);
    writeGPIO(d3, (data & 8) >> 3);

    //3: raise strobe and wait at least 10ms
    writeGPIO(strobe, HIGH);
    usleep(STROBE_DELAY);

    //4: Pull strobe low again
    writeGPIO(strobe, LOW);
    usleep(STROBE_DELAY); //and delay a little bit

    //5: clear the bus
    writeGPIO(d0, LOW);
    writeGPIO(d1, LOW);
    writeGPIO(d2, LOW);
    writeGPIO(d3, LOW);

    //....let the bus float high again
    writeGPIO(strobe, HIGH);
    usleep(STROBE_DELAY); //and delay a little bit
}

//Reads a 4 bit nibble from the bus following the protocol
//returns the nibble in the lower 4 bits of the return value
//returns a negative on error
int readNibble(char* d0,
    char* d1,
    char* d2,
    char* d3,
    char* strobe)
{
    unsigned char data = 0x00;
    int test = 1;
    //set all the data ports to input, but the strobe to output
    setGPIODirection(d0, GPIO_DIRECTION_IN);
    setGPIODirection(d1, GPIO_DIRECTION_IN);
    setGPIODirection(d2, GPIO_DIRECTION_IN);
    setGPIODirection(d3, GPIO_DIRECTION_IN);

    //start the bus protocol
    //1: pull strobe low to signal the start of the read
    writeGPIO(strobe, LOW);

    //2: the PIC should output to the bus now.

    //3: We give it 10ms
    usleep(STROBE_DELAY);

    //4: raise strobe and start reading the value from the data bus
    writeGPIO(strobe, HIGH);
    test = readGPIO(d0);
    if(test == ERROR)
        return ERROR;
    data += test;

    test = readGPIO(d1);
    if(test == ERROR)
        return ERROR;
    data += test << 1;

    test = readGPIO(d2);
    if(test == ERROR)
        return ERROR;
    data += test << 2;

    test = readGPIO(d3);
    if(test == ERROR)
        return ERROR;

    data += test << 3;

    if(data > 0xF)
    {
        printf("Uncaught error reading nibble from the bus");
        return ERROR;
    }
    //leave strobe high for a bit
    usleep(STROBE_DELAY);

    //4: Pull strobe low again to signal that data has been read
    writeGPIO(strobe, LOW);
    usleep(STROBE_DELAY);
    //5: the PIC will clear the bus

    //....let the bus float high again
    writeGPIO(strobe, HIGH);
    usleep(STROBE_DELAY); //and delay a little bit
    return (int)data;
}

// tests the GPIO write and exits
// connect a scope or something to the strobe port and watch for output
void testGPIOWrite(char * fh)
{
    int i;
    //test read and write.
    for(i =0; i <1000; i++)
    {
        int w = writeGPIO(fh, HIGH);
        assert(w == HIGH && "Write high");
        usleep(10);
        w = writeGPIO(fh, LOW);
        assert(w == LOW && "Write Low");
        usleep(10);
    }
    exit(0);
}
```

```
// tests the GPIO writenibble and exits
//repeatedly sends nibbles from 0x0 to 0xF
//How to test: Connect to logic analyzer, watch values
void testGPIOWriteNibble(char* strobe_fh,
                         char* d4, //48
                         char* d5, //50
                         char* d6, //52
                         char* d7) //54
{
   unsigned i;
   //test writeNibble
   for( i =0; i <1000; i++)
   {
      printf("%X\n",i % 0xF);
      writeNibble( i %0xF, d4, d5, d6, d7, strobe_fh);
      usleep(20);
   }
   exit(0);
}

// tests the GPIO readnibble
void testGPIOReadNibble(char* strobe_fh,
                         char* d4, //48
                         char* d5, //50
                         char* d6, //52
                         char* d7) //54
{
   unsigned i;
   unsigned data;
   //test writeNibble
   for( i =0; i <1000; i++)
   {
      data = readNibble(d4, d5, d6, d7, strobe_fh);
      printf("read: %X\n",data);

      usleep(STROBE_DELAY);
   }
   exit(0);
}

//Functions definitions - for commands
void reset();
void ping();
void adc_value();
void servo_30();
void servo_90();
void servo_120();


//File handles for the pins
char* fileHandleGPIO_4;
char* fileHandleGPIO_5;
char* fileHandleGPIO_6;
char* fileHandleGPIO_7;
char* fileHandleGPIO_S;  //Should these 5 variables be used globally? (jk you're right they
should be global)-

int main(void)
{

   fileHandleGPIO_4 = openGPIO(GP_4, GPIO_DIRECTION_OUT);
   fileHandleGPIO_5 = openGPIO(GP_5, GPIO_DIRECTION_OUT);
   fileHandleGPIO_6 = openGPIO(GP_6, GPIO_DIRECTION_OUT);
   fileHandleGPIO_7 = openGPIO(GP_7, GPIO_DIRECTION_OUT);
   fileHandleGPIO_S = openGPIO(Strobe, GPIO_DIRECTION_OUT);

   int input;
   int scanf_test;

   do{

      printf("Select a number for desired action: \n\n");
      printf("1. Reset\n");
      printf("2. Ping\n");
      printf("3. Get ADC value\n");
      printf("4. Turn Servo 30 degrees\n");
      printf("5. Turn Servo 90 degrees\n");
      printf("6. Turn Servo 120 degrees\n");
      printf("7. Exit\n");

      //check for input.
      input = 0;
      scanf_test = scanf("%d", &input);
      //If ipmroperly formatted,
      //  set input to 0 to prompt user to input again
      if(scanf_test == 0)
         input = 0;
      switch (input)
      {
         case 1 :
            reset();
```

```
                break;
            case 2  :
                ping();
                break;
            case 3  :
                adc_value();
                break;
            case 4  :
                servo_30();
                break;
            case 5  :
                servo_90();
                break;
            case 6  :
                servo_120();
                break;
            default :
                printf("Please enter a valid number (1 - 6)\n");
                break;
        }
    }while(input != 7);
}


//stubs for the command functions
void reset()
{
    int receive_msg = 0;
    while(receive_msg != MSG_ACK)
    {
        printf("Starting to send reset\n");
        writeNibble(MSG_RESET,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        printf("Wrote Nibble to line\n");
        usleep(STROBE_DELAY);
        receive_msg = readNibble(fileHandleGPIO_4,
                                fileHandleGPIO_5,
                                fileHandleGPIO_6,
                                fileHandleGPIO_7,
                                fileHandleGPIO_S);
        printf("Received message from PIC: %x \n", receive_msg);
        usleep(STROBE_DELAY);
    }
    printf("Reset message sent\n");
}

void ping()
{
    int receive_msg = 0;
    while(receive_msg != MSG_ACK)
    {
        printf("Starting to send ping\n");
        writeNibble(MSG_PING,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        printf("Wrote ping to bus\n");
        usleep(STROBE_DELAY);
        receive_msg = readNibble(fileHandleGPIO_4,
                                 fileHandleGPIO_5,
                                 fileHandleGPIO_6,
                                 fileHandleGPIO_7,
                                 fileHandleGPIO_S);
        printf("Received message from PIC: %x \n", receive_msg);
        usleep(STROBE_DELAY);
    }
    printf("Ping message sent\n");
}

//requests the PIC to send its current adc value MSN (most significant nibble) first
void adc_value()
{
    int i;
    int receive_msg = 0;
    int adc_value = 0;
    while(receive_msg != MSG_ACK)
    {
        adc_value = 0;
        receive_msg = 0;
        printf("Starting to send ADC_request\n");
        writeNibble(MSG_GET,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        //expect to receive 3 messages containing ADC values, msn first
        usleep(STROBE_DELAY);
        for(i = 8; i >= 0; i -= 4)
        {
           receive_msg = readNibble(fileHandleGPIO_4,
                                    fileHandleGPIO_5,
                                    fileHandleGPIO_6,
                                    fileHandleGPIO_7,
                                    fileHandleGPIO_S);
           adc_value += ((unsigned) receive_msg) << i;
           printf("Received ADC Nibble: 0x%x\n", receive_msg);
           usleep(STROBE_DELAY);
        }
        //expect one last ACK message
        receive_msg = readNibble(fileHandleGPIO_4,
                                 fileHandleGPIO_5,
                                 fileHandleGPIO_6,
                                 fileHandleGPIO_7,
                                 fileHandleGPIO_S);
        printf("Received ADC ACK(?): 0x%x\n", receive_msg);
        usleep(STROBE_DELAY);
     }
     printf("adc message received successfully: 0x%x\n", adc_value);
  }

  void servo_30()
  {
     int receive_msg = 0;
     while(receive_msg != MSG_ACK)
     {
        printf("Starting to send Servo30\n");
        writeNibble(MSG_TURN30,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        printf("Wrote Nibble to Line Servo30\n");
        usleep(STROBE_DELAY);
        receive_msg = readNibble(fileHandleGPIO_4,
                                 fileHandleGPIO_5,
                                 fileHandleGPIO_6,
                                 fileHandleGPIO_7,
                                 fileHandleGPIO_S);
        printf("Received message from PIC: %x \n", receive_msg);
        usleep(STROBE_DELAY);
    }
    printf("servo_30 message sent\n");
}

void servo_90()
{
    int receive_msg = 0;
    while(receive_msg != MSG_ACK)
    {
        printf("Starting to send Servo90\n");
        writeNibble(MSG_TURN90,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        printf("Wrote Nibble to bus\n");
        usleep(STROBE_DELAY);
        receive_msg = readNibble(fileHandleGPIO_4,
                                 fileHandleGPIO_5,
                                 fileHandleGPIO_6,
                                 fileHandleGPIO_7,
                                 fileHandleGPIO_S);
        printf("Received message from PIC: %x \n", receive_msg);
        usleep(STROBE_DELAY);
    }
    printf("Servo_90 message sent\n");
}

void servo_120()
{
    int receive_msg = 0;
    while(receive_msg != MSG_ACK)
    {
        writeNibble(MSG_TURN120,
                    fileHandleGPIO_4,
                    fileHandleGPIO_5,
                    fileHandleGPIO_6,
                    fileHandleGPIO_7,
                    fileHandleGPIO_S);
        printf("Wrote Nibble to bus\n");
        usleep(STROBE_DELAY);
        receive_msg = readNibble(fileHandleGPIO_4,
                                 fileHandleGPIO_5,
                                 fileHandleGPIO_6,
                        fileHandleGPIO_7,
                        fileHandleGPIO_S);
      printf("Received message from PIC: %x \n", receive_msg);
      usleep(STROBE_DELAY);
   }
   printf("Servo_120 message sent\n");
}
```