Section 1 : General Lab Info

EECE.4800 Microprocessors II and Embedded System Design

11024

Lab 2: Interfacing with a Sensor Device on an Embedded Computer System

Professor Yan Luo

Group number: 1

Jose Velis

October 29, 2017

October 29, 2017

Section 2 : Contributions

1. Group Member 1 – Jose Velis

   Responsible for ADC configuration and ADC results, LDR implementation, and some circuit layout on the breadboard. Also made the main function logic running on the Galileo board. Circuit schematic.

2. Group Member 2 – Grayson Colwell

   Responsible for servo/PWM implementation as well as for the ADC, main logic implementation on the PIC16, and serial communication. Also did communication code for the PIC16 and wiring. Code debugging and PIC16 flow chart.

3. Group Member 3 – Andy MacGregor

   Responsible for main code logic as well as timer configuration and serial communications. Also, ADC results and most of the code running on the Galileo. This code was the pin output/input setup, communications handling code, debugging, writing/reading bus contents, Galileo flow chart.

Section 3: Purpose

To further advanced our microprocessor knowledge, a menu system was used in the project. This menu option was made to run on an Embedded computer system (the Galileo Gen. 2 board). The user selectable operations were chosen on the Galileo board and a communications protocol was implemented to the send the selection to the PIC16. The PIC16 was used to “drive” the hardware outputs. The purpose of this lab was also to further understand the operation of an analog-to-digital converter by interfacing an analog sensor to an embedded microcontroller, also to understand the design of sensor circuitry, understand the operation of Pulse Width Modulation (PWM) signals, and to learn how to control a mechanical actuator (a servo motor for this lab).

Section 4 : Introduction

In this lab, a communications protocol was implemented to send command signals to the PIC16 microcontroller. The microcontroller was used to control a servo, acquire ADC values from a photoresistor network, and to respond to other, more software based, requests. The Intel Galileo Gen 2 board was used as the master and the PIC16 as the slave. As mentioned, the PIC16 was simply receiving, acknowledging, and transmitting a response back to the Galileo through a simple bus protocol. The protocol was implemented through the Galileo GPIO pins and the PIC16 GPIO pins. The bus protocol was designed as required by the lab manual.

Section 5 : Materials, Devices and Instruments

- Microchip PicKit, Version 3, used to program the PIC device, operates at 5V
- Microchip PIC16F18857 microcontroller, 28 dip device, operates at 3.3 volts
- MPLAB X IDE with XC8 compiler
- Servo motor, SG90, operates between 4.8v -5V
- FTDI cable, operates at 5V
- Photoresistor, measures light intensity, operates between 0V-5V
- Resistors, 1kΩ and 330Ω
- LED
- Breadboard & wiring
- Lab oscilloscope and power supply
- Intel Galileo Gen 2 embedded computer system

Section 6: Schematics

The schematic for the circuit and programmer can be seen in figure 1.

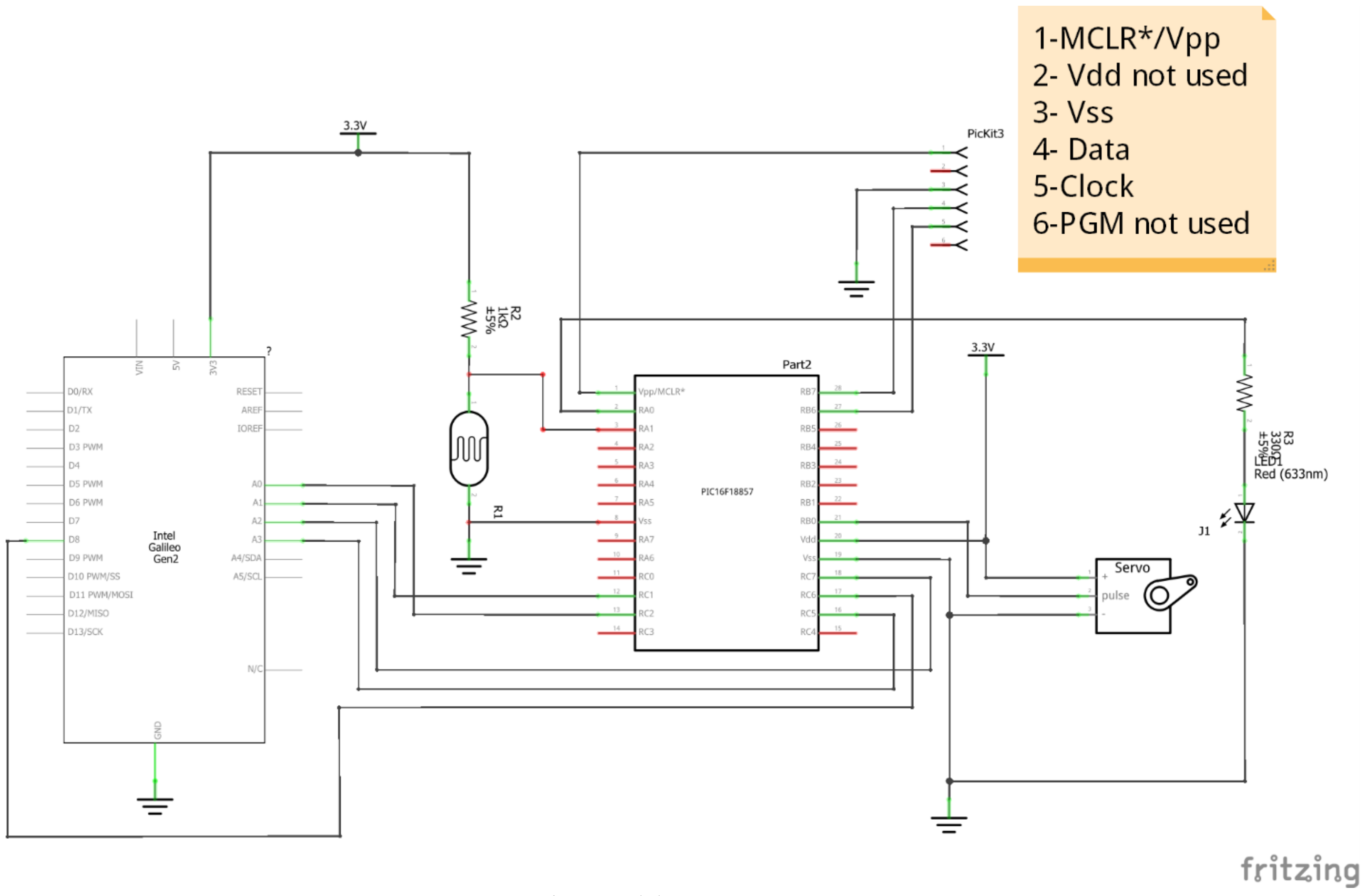


Figure (1)

Section 7 : Lab Methods and Procedure

Hardware design:

For this lab, we first started by figuring out the pins, programming pins and power requirements of the PIC16. Then we moved on to how the servo motor works and decided to control it with delay functions instead of using the PWN modules. We became acquainted with the servo pinout and power demands. The servo has a 4.8V-5V power, ground, and signal pin. The signal pin was connected to pin 21 (RB0) of the PIC16 microcontroller due to the pin being a general-purpose pin. The next step was connecting the photoresistor, resistors and LED. A simple voltage divider circuit between the 1kΩ resistor and photoresistor was used to obtain a change in voltage value for when the LDR reacted to different light intensity. The voltage divider network was connected to the 3.3V supply and the changing voltage signal was connected to pin 3 (RA1) of the PIC16. The LED anode was connected to a 330Ω resistor and then the resistor to pin 2 of the PIC16. LED cathode was grounded. Much of the PIC16 software remained the same as in lab 1, except for the code that had to do with the bus communications protocol.

For the Galileo, the main problem to overcome was getting to know the software and how the code should be implemented. On the Galileo, a version of Yocto Linux OS was installed. With the OS installed it was possible to configure the device for WIFI communications. The WIFI allowed us to communicate and program the device wirelessly. A text editor was used as the "IDE" for writing the code. The pins selected for implementing the bus protocol were chosen because they are GPIO and can be configured to input and output at any time.

Software design:

The hardware controlling software for the PIC16 remained relatively the same. The program flow chart for the PIC16 can be seen in figure 2. The new code was added to determining the selected menu option and to configure the bus protocol required for replies.

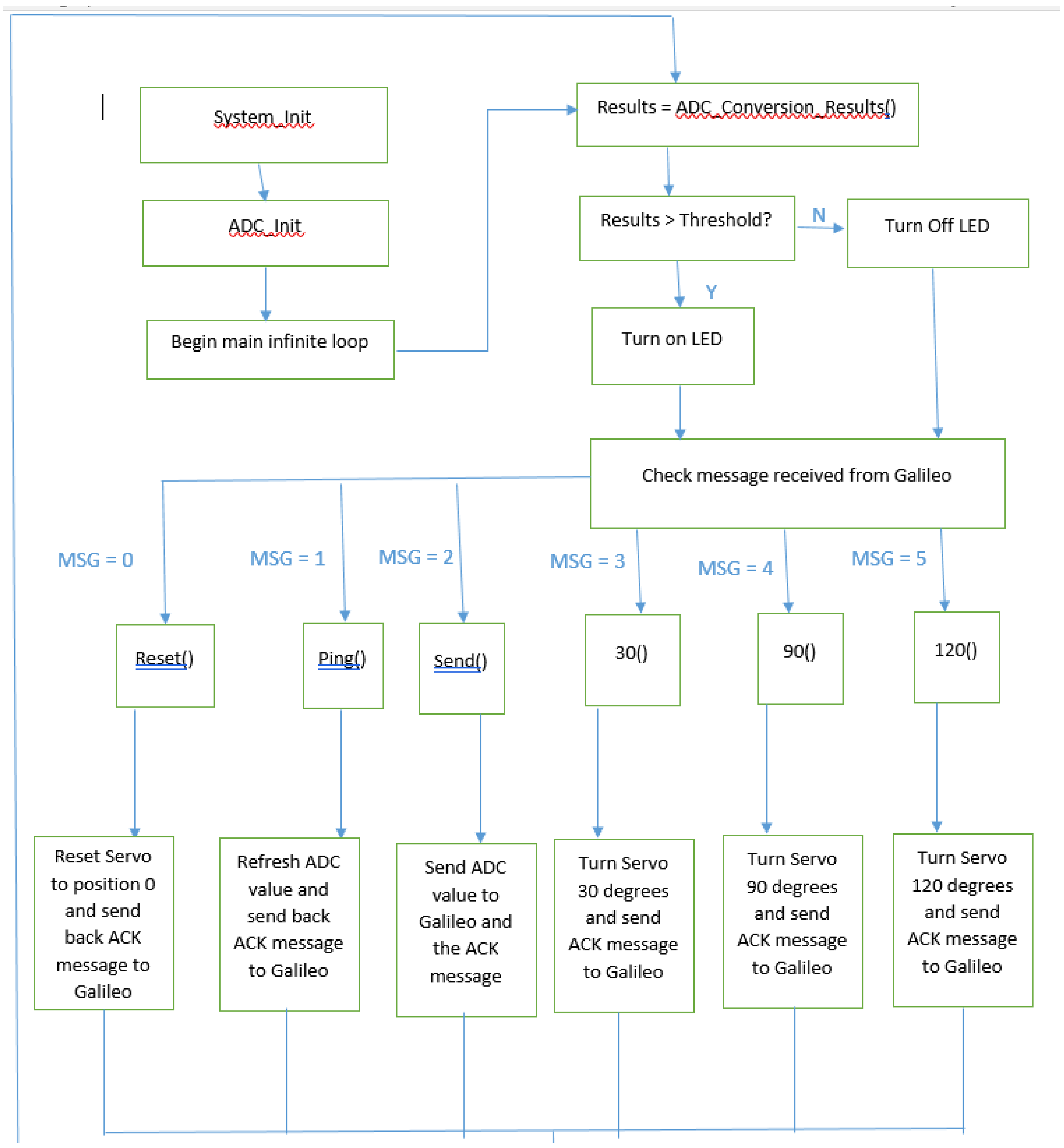

System_Init
ADC_Init
Begin main infinite loop
Results = ADC_Conversion_Results()
Results > Threshold?
N
Turn Off LED
Y
Turn on LED
Check message received from Galileo
MSG = 0
MSG = 1
MSG = 2
MSG = 3
MSG = 4
MSG = 5
Reset()
Ping()
Send()
30()
90()
120()
Reset Servo to position 0 and send back ACK message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

The basic working of the PIC16 program are as follows:

- Initialize the chip along with the modules that will be used. In this case it was just the ADC module.
- The PIC16 sits waiting for a message from the Galileo board, the steps after this are executed based on the message received.

- Data from the ADC was recovered, and made available to the main logic loop for the send message function.
- In the main logic loop, the ADC conversion results function was called and the value was compared to an LED threshold value. If the ADC value was greater than the threshold, the LED is turned on via the PORTA registers. If not, the LED is kept off.
- The location of the servo was also tracked and used in the main logic to control the direction of rotation based on the what the received message sent from the Galileo was.
- The PIC just keeps waiting for instructions from the Galileo.

Code snips main logic loop and communications handling for the PIC16 are shown below.

```
#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;
                    }
                }




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
    */

}
```

The basic setup and workings of the Galileo code are briefly given below:

- System initializations
- Define the pins that will be used for communications and set their GPIO mode
- Define the values for menu selection
- Set the GPIO selected direction (input or output) through a specific function call
- Create functions that will enable the GPIO to write and read the contents on the bus
- Create the user selectable functions with their individual bus protocol identifiers
- Create a menu interface so that the user can select which operation to send to the PIC16
- When an option is chosen, the Galileo calls the function, the function itself writes a message to the bus. This message is read by the PIC and then a reply is sent back to the Galileo as an acknowledgement and the PIC performs the desired action.
- The read, write, gpio direction functions also have steps to clear the bus.
- The process keeps running as long as the user doesn't exit the program

A flow chart for the Galileo can be seen below in figure 3:

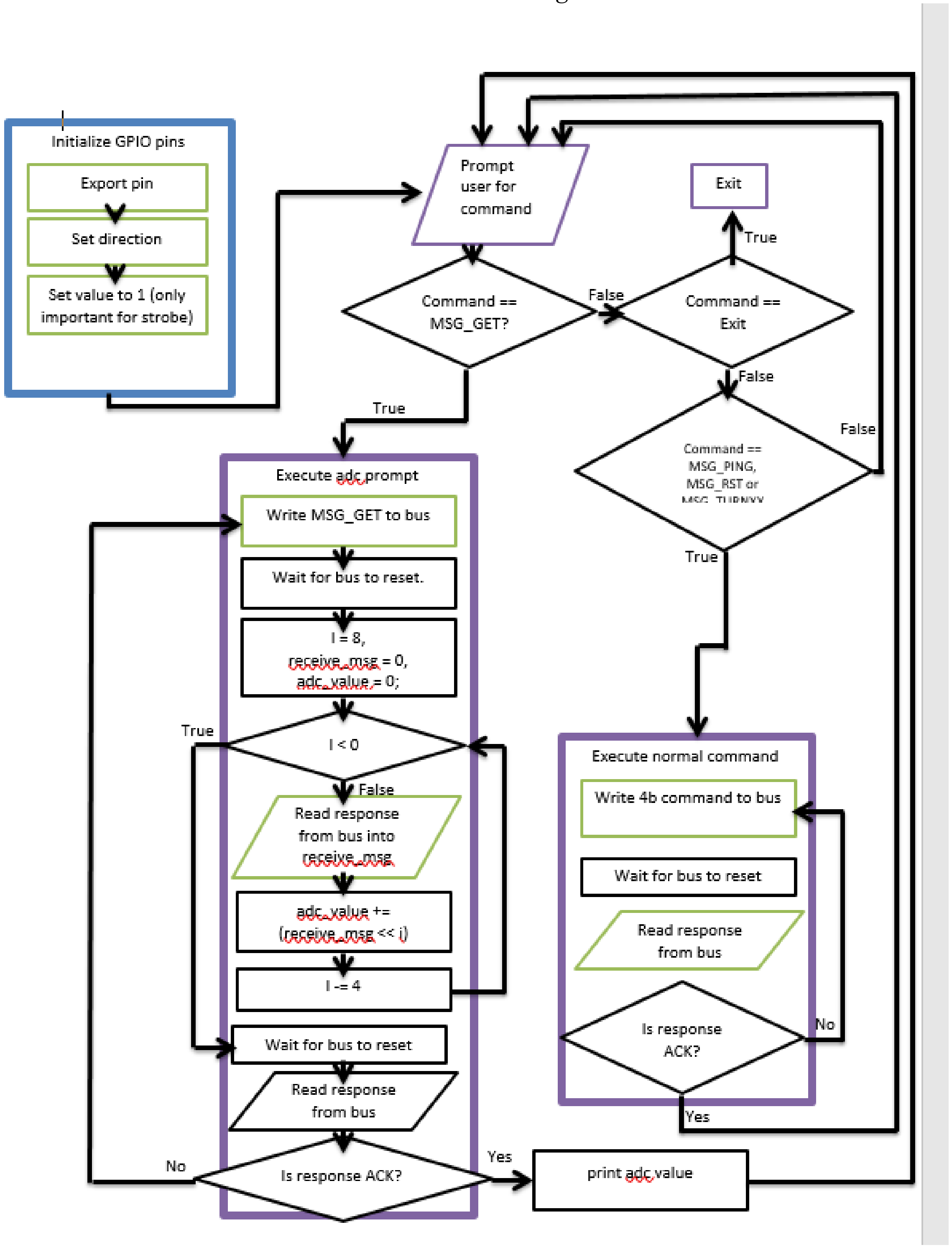


Figure (3)

In the following snips of code, the Galileo pin setup, menu code and some of the bus protocol code can be seen. It's not all included because it is quite long.

```
//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;
            }

//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);
          }


//read
value
```

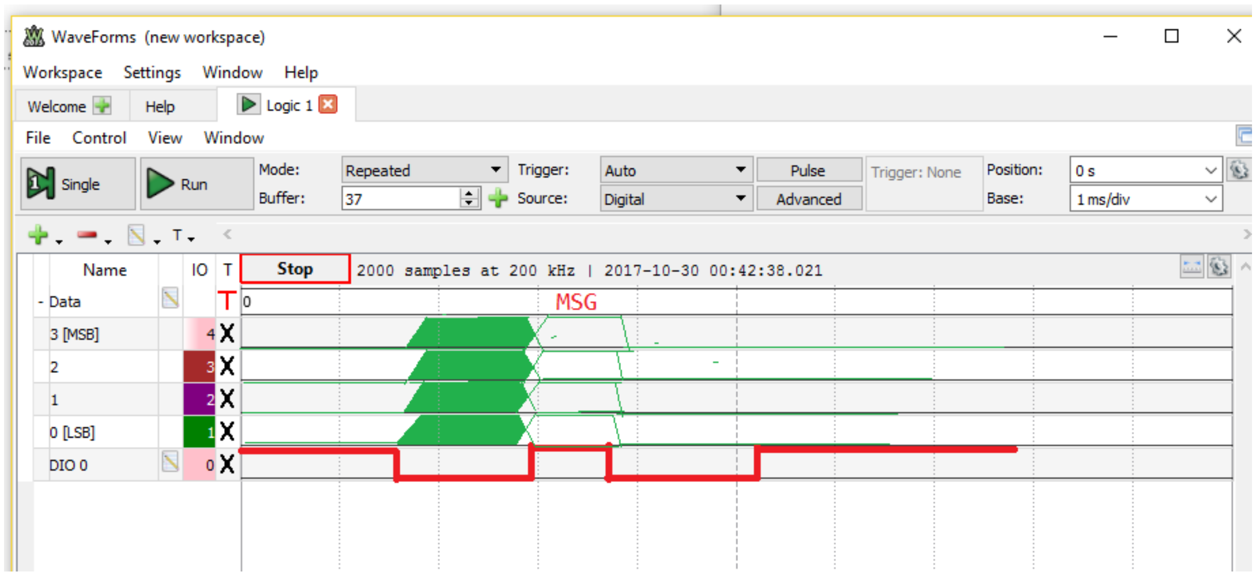
```
(HIGH or
LOW) from
port
specified
            //port direction must be set to input.
            //returns port value on success
            //returns ERROR (negative) on failure
            int readGPIO(char* gpioDirectory)
            {
                char gpioValue[BUFFER_SIZE];
                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;
    }
```

Section 8 : Trouble Shooting

Issue 1: We had some trouble with just general functioning of the code. This was troubleshooted by simply rebuilding code, writing debug statements in functions, looking over code and following its logic to see if it was doing what we intended for it to do. Many hours of rebuilding the communications protocol were had. The figure below shows the logic analyzer that we used to help debug the timing signals.



Mode: Repeated
Buffer: 37
Trigger: Auto
Source: Digital
Position: 0 s
Base: 1 ms/div

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

Name	IO	T
- Data		T
3 [MSB]	4	X
2	3	X
1	2	X
0 [LSB]	1	X
DIO 0	0	X

Issue 2: There weren't any hardware issues apart from running into the occasional PicKit3 "not enough power" errors. These were remedied once we decided to just simply use a bench top power supply for the 3.3V that we needed.

Section 9: Results

Charts/Measurement Tables: The ADC values are really the only measurement that we "obtained". These values can be seen by running the code.

Terminal Screenshots: None were taken. We will remember to do so for the next lab.

Section 10 : Appendix

A1. The code won't be included in the print out of this report, but it will be in the digital version. There far too many pages to print if all the code is included. The following code is for the Galileo:

```
/*
        * 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
```

```c
//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
```

```c
//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);
}
```

```c
// 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
```

```c
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");
```

```
}
```

A2. The following code is for the PIC16:

```
#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("REEEEEEEEEEEEEEEEEEEEEEE\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
*/
```