Sean Webster

Microprocessors II

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

Group 8 – Sean Webster, Eric Craaybeek, Munib Elkhatib

Handed in 11/6/2016 - Due 11/6/2016

**Personal contributions to the lab assignment:**

* Wrote Galileo code
* Troubleshooted Galileo code
* Reviewed pic code

**Purpose:**

The purpose of this lab was to learn how to interface sensors with embedded microcontrollers, understand bus protocols, and understand the operation of GPIO ports

**Introduction:**

Sensors are the way to create physical input into microcontrollers from outside sources. Designing sensors and their associated circuity, and obtaining the data output by them are important tasks in designing an embedded system. Interfacing embedded systems with each other is the next step in designing embedded systems, and allows for their complexity and abilities to grow exponentially.

**Materials, Devices, and Instruments:**

* PIC16F18857 Microcontroller
* Galileo board
* Serial cable
* Galileo power cable
* Photoresistor
* LED
* 10k ohm resistors
* PIC Kit 3 + USB cable
* Breadboard + jumpers

**Schematic:**

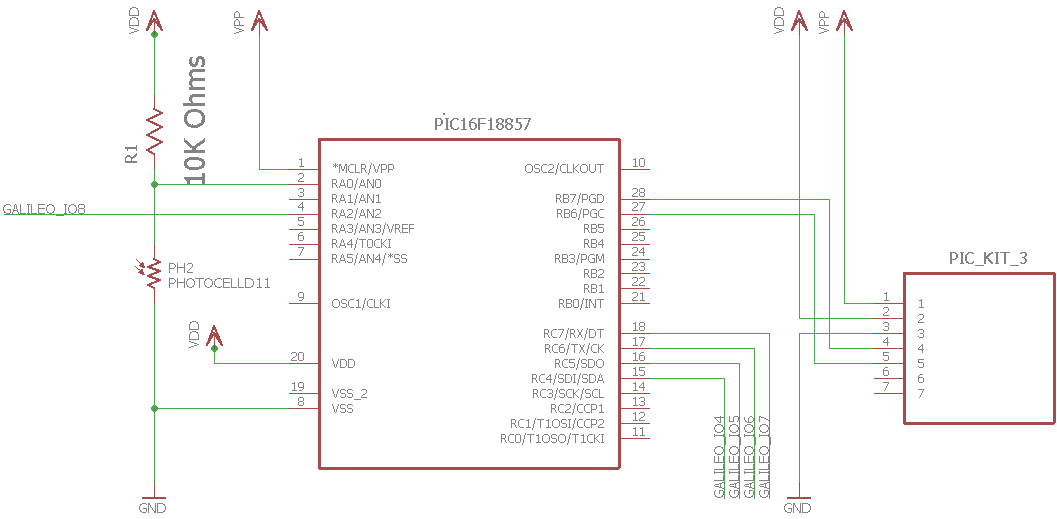


Figure 1 – Schematic of whole lab.

**Lab Methods and Procedure:**

**https://lh3.googleusercontent.com/-XHAAOn2gFok/WCAGmYgj9UI/AAAAAAAAAq8/R-UBJHgm0-s7W0jdIwNBSdyVSjweBrdPgCL0B/h1039/2016-11-06.png**

Figure 2 – program flowchart for Galileo.

https://lh3.googleusercontent.com/-81J-n-aVQZM/WCAGlpaS04I/AAAAAAAAAq8/7Yji5ZAlbUg02KzM8j1X2v_VaUXSLfuLQCL0B/h922/2016-11-06.png

Figure 2 – program flowchart for PIC.

**Galileo Code Explanation:**

1. Set up GPIO ports
   1. Export GPIO
   2. Set GPIO direction
   3. Set GPIO voltage
   4. Set GPIO direction
2. readGPIO function
   1. reads in file value of GPIO
   2. returns value
3. writeGPIO function
   1. writes value given to file directory
4. send function
   1. opens all GPIO
   2. strobes high
   3. writes all 4 GPIO pins
   4. strobes low
5. receive function
   1. opens all GPIO
   2. strobes high
   3. reads first bit, then shifts
      1. does same with all 4 bits
   4. strobes low
   5. returns 4 bit message
6. Main program loop
   1. Asks for command
   2. Reset
      1. Calls send with reset message code
      2. Then sleeps
      3. Listens for ack
   3. Ping
      1. Sends ping message
      2. sleeps
      3. Waits for response
   4. Get
      1. sends get message
      2. receives 3 4 bit nibbles of data
      3. sleeps
      4. waits for ack
   5. quit
      1. Quits program

**Pic code explanation:**

1. Set\_receive function
   1. Sets RC2,3,4,5 to input
2. Set\_send function
   1. Sets RC2,3,4,5 to output
3. Receive\_msg function
   1. Calls set\_receive
   2. Waits for strobe high
   3. Reads data
   4. Waits for strobe low
   5. Returns data
4. Send\_message function
   1. Waits for strobe high
   2. Calls set\_send
   3. Clears PORTC
   4. Shifts message into PORTC
   5. Wait for strobe low
5. Main program loop
   1. Set ra2 as input
   2. Set up adc
   3. Set voltage = to adres register
   4. Read received message
   5. MSG\_RESET
      1. Reset everything
      2. Reply with ack
   6. MSG\_PING
      1. Send Ack
   7. MSG\_GET
      1. Send voltage data
   8. Restart adc

The Galileo program revolved around opening and editing files in the GPIO directory. GPIO works by editing a file with the desired trait and changing the value for the Galileo. The openGPIO function did this, and then the writeGPIO function changed the value in the ‘value’ file to write. The readGPIO function changed the value in the ‘value’ file to read.

The pic program revolved around reading the input from RC and checking it against each of the defined messages it was looking for. It would send out a then prefabricated response. For a get message, it would send out the current voltage at the adc.

**Troubleshooting:**

* Problem: many GPIO pins were not working when turned on.
  + Solution: Used numerous system commands to turn on other pins. Tried using file read/write and system command combinations. Turned out that all pins except pins 7 and 8 can output a high signal for some reason – STILL NOT FIXED.
* Problem: PIC has no response.
  + Solution: forgot to turn watchdog off, unsure if this fixed the problem as we could not get a response due to the GPIO error.
* Problem: programming on Galileo difficult.
  + Solution: Installed vim on Galileo, causing it to crash, installed many other packages in an attempt to create an easier development environment.
* Problem: itoa and atoi were causing seg faults.
  + Solution: removed them from code, used typecasting instead

**Results:**

* Galileo assumed to be broken, GPIO associated with pins did not cause pins to go high.
* This made it difficult to see if anything else works

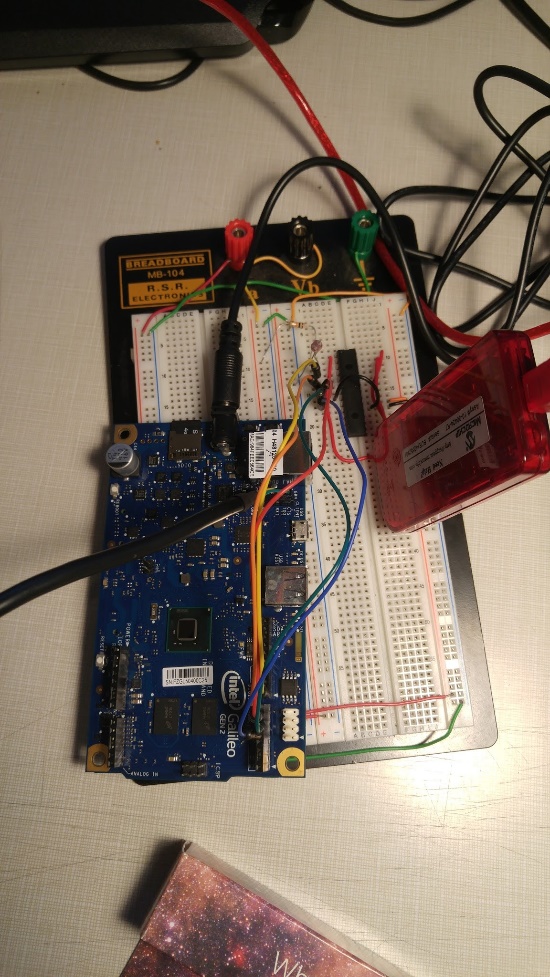


Figure 4 – Picture of lab configuration

**Contributions to this lab:**

* Eric Craaybeek
  + Created pic program
  + Troubleshooted Galileo program
  + Troubleshooted pic program
  + Created flow charts
* Munib Elkhatib
  + Created pic program
  + Troubleshooted pic program
  + Created circuit
* Sean Webster
  + Created Galileo program
  + Troubleshooted Galileo program
  + Created schematic
  + Troubleshooted pic program

**Appendix:**

/\*

\* File: PIC and Galileo communication

\*

\*

\* simple Galileo program example

\* for UMass Lowell 16.480/552

\*

\* Author: Sean Webster,Eric Craaybeek, Munib Elkhatib

\*

\* Created on 2014/9/13

\* Code from class example and from black-swift.com used

\*/

#include <stdlib.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

/\* user commands \*/

#define MSG\_RESET 0x0 // reset the sensor to the intitial state

#define MSG\_PING 0x1 // check if the sensor is working properly

#define MSG\_GET 0x2 // obtain the most recent ADC result

#define MSG\_ACK 0xE // ack message obtained from pic

#define Strobe (40) // IO8

#define GP\_4 (6) // IO4

#define GP\_5 (0) // IO5

#define GP\_6 (4) // IO6

#define GP\_7 (38) // IO7

#define GPIO\_DIRECTION\_IN (1)

#define GPIO\_DIRECTION\_OUT (0)

#define ERROR (-1)

int fileHandleGPIO\_4;

int fileHandleGPIO\_5;

int fileHandleGPIO\_6;

int fileHandleGPIO\_7;

int fileHandleGPIO\_S;

//open GPIO and set the direction

int openGPIO(int gpio, int direction )

{

char buffer[256];

int fileDirectory; // file location to be opened

int fileMode; // file mode

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1.set the GPIO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// system command example:

// system("echo -n "27" > /sys/class/gpio/export");

// Open file to write GPIO being used

fileDirectory = open("/sys/class/gpio/export", O\_WRONLY);

// Close and return error if error

if(ERROR == fileDirectory)

{

puts("Error opening /sys/class/gpio/export");

return ERROR;

}

// Write GPIO ports to buffer

sprintf(buffer, "%d", gpio);

// write buffer to file

write(fileDirectory, buffer, strlen(buffer));

close(fileDirectory);

//\*\*\*\*\*\*\*\*\*\* 2.set the direction\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// system command example:

// system("echo -n "out" > /sys/class/gpio/gpio6/direction");

sprintf(buffer, "/sys/class/gpio/gpio%d/direction", gpio);

fileDirectory = open(buffer, O\_WRONLY);

if(ERROR == fileDirectory)

{

puts("Unable to open file:");

puts(buffer);

return ERROR;

}

// Direction = out

if (direction == GPIO\_DIRECTION\_OUT)

{

write(fileDirectory, "out", 3);

}

// Direction = in

else

{

write(fileDirectory, "in", 2);

}

close(fileDirectory);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.set the voltage\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// system command example:

// system("echo -n "strong" > /sys/class/gpio/gpio40/drive");

sprintf(buffer, "/sys/class/gpio/gpio%d/drive", gpio);

fileDirectory = open(buffer, O\_WRONLY);

if(ERROR == fileDirectory)

{

puts("Unable to open file:");

puts(buffer);

return ERROR;

}

if(direction == GPIO\_DIRECTION\_OUT)

{

write(fileDirectory, "strong", 6);

}

else

{

write(fileDirectory, "pulldown", 8);

}

close(fileDirectory);

//\*\*\*\*\*\*\*\*\*4.Open GPIO for Read / Write\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

sprintf(buffer, "/sys/class/gpio/gpio%d/value", gpio);

fileDirectory = open(buffer, O\_WRONLY);

if(ERROR == fileDirectory)

{

puts("Unable to open file:");

puts(buffer);

return(-1);

}

write(fileDirectory, "%d", fileMode);

//Return memory address to use in other parts

return fileDirectory;

}

int readGPIO (int fileDirectory, char value, int size)

{

read(fileDirectory, value, size);

int temp = value - '0';

return temp;

}

//write values GPIO

int writeGPIO(int fileDirectory, int value)

{

// previous way: cat /sys/class/gpio/gpio40/value

write(fileDirectory,value ,1);

return 0;

}

void send(int message)

{

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);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*1.Strobe high\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

writeGPIO(fileHandleGPIO\_S, 1);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*2.write data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

writeGPIO(fileHandleGPIO\_4, (message && 0b0001));

writeGPIO(fileHandleGPIO\_5, (message && 0b0010) >> 1);

writeGPIO(fileHandleGPIO\_6, (message && 0b0100) >> 2);

writeGPIO(fileHandleGPIO\_7, (message && 0b1000) >> 3);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.Strobe low\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

writeGPIO(fileHandleGPIO\_S, 0);

}

char receive(void)

{

int message = 0;

char currBit;

fileHandleGPIO\_S = openGPIO(Strobe, GPIO\_DIRECTION\_OUT);

fileHandleGPIO\_4 = openGPIO(GP\_4, GPIO\_DIRECTION\_IN);

fileHandleGPIO\_5 = openGPIO(GP\_5, GPIO\_DIRECTION\_IN);

fileHandleGPIO\_6 = openGPIO(GP\_6, GPIO\_DIRECTION\_IN);

fileHandleGPIO\_7 = openGPIO(GP\_7, GPIO\_DIRECTION\_IN);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*1.Strobe high\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

writeGPIO(fileHandleGPIO\_S, 1);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*2.read data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

readGPIO(fileHandleGPIO\_4, currBit, 1);

message = message << 1;

message |= currBit;

readGPIO(fileHandleGPIO\_5, currBit, 1);

message = message << 1;

message |= currBit;

readGPIO(fileHandleGPIO\_6, currBit, 1);

message = message << 1;

message |= currBit;

readGPIO(fileHandleGPIO\_7, currBit, 1);

message = message << 1;

message |= currBit;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3.Strobe low\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

writeGPIO(fileHandleGPIO\_S, 0);

return message;

}

//main

int main(void)

{

int value;

unsigned char msg;

unsigned char temp;

char cmd;

while(1)

{

printf("enter a command: ");

scanf("%c", &cmd);

switch (cmd) {

case 'r' :

printf("resetting\n");

msg = 0x0;

send(msg);

usleep(1000);

if(receive() != MSG\_ACK)

printf("error: no ack\n");

break;

case 'p' :

printf("pinging\n");

msg = 0x1;

send(msg);

sleep(1);

temp = receive();

printf("response: %x\n", temp);

if(temp != MSG\_ACK)

printf("error: no ack\n");

break;

case 'g' :

printf("getting\n");

msg = 0x2;

send(msg);

int i;

for(i = 0; i < 3; i++)

{

usleep(1000);

temp = receive();

value = value << 4;

value |= temp;

}

usleep(1000);

if(receive() != MSG\_ACK)

printf("error: no ack\n");

break;

break;

case 'q' :

printf("quitting\n");

return;

default : printf("command not recognized\n");

}

}

return 0;

}

PIC CODE:

/\*

\* File: PIC and Galileo communication

\*

\*

\* simple PIC program example

\* for UMass Lowell 16.480/552

\*

\* Author: Munib Elkhatib, Sean Webster, Eric Craaybeek

\*

\* 10/31/2016

\*/

//

/\*

STROBE RA2

D0 RC2

D1 RC3

D2 RC4

D3 RC5

\*/

#pragma config WDTE = OFF

#define \_XTAL\_FREQ 500000

#include <stdio.h>

#include <stdlib.h>

#include <xc.h>

#include <pic16f18857.h>

#define MSG\_RESET 0x00 // reset the sensor to intial state

#define MSG\_PING 0x01 //check if the sensor is working

#define MSG\_GET 0X02 //obtain the most recent ADC result

#define MSG\_ACK 0x0E //acknowledgment to the commands

//#define MSG\_NOTHING 0x0F //reserved

unsigned int voltage;

void set\_receive()

{

//SET RC2,3,4,5 TO INPUT

TRISC = 0b00111100;

//SET RC2,3,4,5

ANSELC = 0x00;

return;

}

void set\_send()

{

//set RC2, 3,4,5 to Output

TRISC = 0x00;

ANSELC = 0x00;

return;

}

unsigned char receive\_msg()

{

set\_receive();

//wait until strobe high

while(!PORTAbits.RA2)

{

\_\_delay\_ms(1);

}

//read the data

unsigned char message;

message =0x0;

message = PORTC >> 2;

//wait strobe low

while(PORTAbits.RA2)

{

\_\_delay\_ms(1);

}

//return the data

return message;

}

void send\_message( unsigned char msg)

{

while(!PORTAbits.RA2)

{

\_\_delay\_ms(1);

}

set\_send();

PORTC &= 0x00;

PORTC = msg << 2;

//wait for galileo to start reading

//wait until strobe low

while(PORTAbits.RA2)

{

\_\_delay\_ms(1);

}

//wait for galileo to read message

return;

}

// Main program

void main (void)

{

//SET RA2 AS INPUT

TRISA = 0b10100000;

ANSELA = 0b10000000;

//set up ADC

ADCON1 = 0b11110000;

ADCON0 = 0b10010001;

ADPCH = 0b00000000;

ADREF = 0b00000000;

unsigned char msg;

while(1)

{

//read adc

voltage = (ADRESH<<8) + ADRESL;

msg = receive\_msg();

if (msg = MSG\_RESET)

{

//reset sensor to initial state

voltage = 0;

ADRESH = 0;

ADRESL = 0;

msg = MSG\_ACK;

send\_message(msg);

}

else if(msg = MSG\_PING)

{

msg = MSG\_ACK;

send\_message(msg);

}

else if(msg = MSG\_GET)

{

msg = voltage >> 8;

msg &= 0x03;

send\_message(msg);

msg = voltage >> 4;

msg &= 0x0F;

send\_message(msg);

msg = voltage;

msg &= 0x0F;

send\_message(msg);

}

//restart ADC

ADCON0bits.ADGO = 1;

}

}