Section 1: General Lab Info /0.5 points

Course name: Microprocessor Systems II and Embedded Systems

Course Number: EECE.5520

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

Instructors names: Yan Luo
Ioannis Smanis

Group number: 7

Student name: Poornima Manjunath

Hand in Date: 9/27/2017

Lab Due Date: 10/30/2017

Section 2: Contributions /1 points

1. Group Member 1 – Poornima Manjunath

    - I have done the coding of both the PIC microcontroller and the Intel Galileo Gen2 board.
    - I thoroughly studies about the GPIO pin configuration of the Galileo board as I had not worked on it before.
    - I configured them by referring to the Linux commands provide in the GitHub link. Then I wrote the algorithm for the programs of both the Galileo board and the PIC microcontroller such that the Galileo acts as a master and issues commands and the PIC sends response to it.
    - I felt it to be challenging because the synchronization of the code execution on the Intel Galileo board and the PIC microcontroller was difficult. I invested a lot of efforts on it. Also dividing the ADC result into three packets of 4 bits while sending at the PIC side and then assembling it at the Galileo side was also difficult.
    - I also interfaced the USB with Galileo board. It was really helpful to develop the code in the local machine and transfer it to the Galileo.

2. Group Member 2 – Rinkal Shah

    - She took care of the hardware connections. Connecting the GPIO pins of the Intel Galileo to the I/O pins of the PIC and the sensor circuit connections.
    - She also flashed the Yocto Linux image file to the SD card. She helped in debugging the code as well.

3. Group Member 3 – Akhila Nair
    - She helped in troubleshooting the hardware issues.
    - She also gave good suggestions while debugging the code.

Section 3: Purpose /0.5 points

In the first lab we had designed a sensor circuit that senses the light intensity. In this lab we are interfacing the sensor circuit based on PIC microcontroller to the Intel Galileo Gen2. i.e an embedded computer system. We are also designing a customized bus protocol to send commands from the Intel Galileo board and receive the response back from the PIC microcontroller. In this way we are acquiring data from the sensor circuit using the embedded computer system.

This lab helps us to

1. Get introduced to embedded computer systems like Intel Galileo Gen2 runs an open source Linux operating system (Yocto Linux)
2. Learn embedded application software development in Linux environment.
3. Interfacing sensor circuits with an embedded computer system.
4. Understand the bus protocols and develop a customized bus protocol for communication between the computer system and the sensor circuit.

Section 4: Introduction /0.5 points

Intel Galileo 2 is a microcontroller board based on the Intel® Quark SoC X1000 Application Processor, a 32-bit Intel Pentium-class system on a chip. In this project we are using this embedded computer system to issue commands to the light sensor device and acquire the sensor data. We are designing a customized bus protocol that governs the communication between the Intel Galileo and the sensor device.

The Galileo board functions as the master of the bus protocol, and the PIC-based automation system responds to Galileo as a slave device. On the Galileo board, we develop a user application in C language on Yocto Linux that issues commands to the PIC based sensor device. On the sensor device, the PIC microcontroller responds to commands. The PIC microcontroller on the sensor device is connected to the embedded computer through the GPIO port. Over the GPIO port, five signal lines (D3-D0 and Strobe) are available. The data bus (D3-D0) is 4-bit. The control signal is Strobe driven by the Galileo board to synchronize the information exchange between the sensor device and the embedded system.

Section 5: Materials, Devices and Instruments /0.5 points

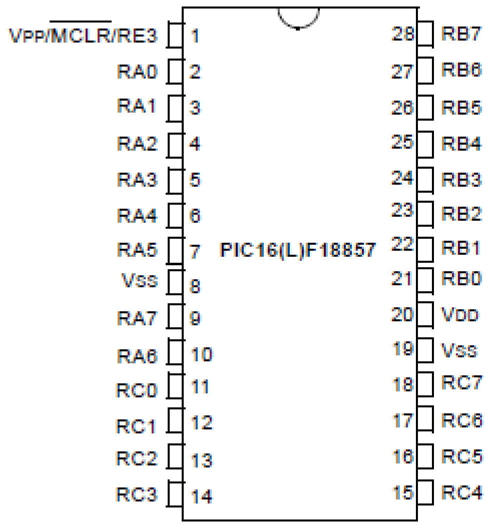
Materials:

1. Breadboard
2. Resistors - 10K , 4.75K
3. Jumper cables
4. Solid core wires
5. LDR
6. LEDs

Devices:

- PIC16F18857 –

- 28 Pin IC
- Operating voltage Range: - 1.8V to 3.6V (PIC16LF18857/77) - 2.3V to 5.5V (PIC16F18857/77)
- 10 bit ADC



- Intel Galileo Gen2 board

- Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to

5V at the I/O pins. This provides support for 5V Uno shields and is the default behavior. By switching the jumper position, the voltage translation can be disabled to provide 3.3V operation at the I/O pins.

- Digital pins 0 to 13 (and the adjacent AREF and GND pins), Analog inputs 0 to 5, the power header, ICSP header, and the UART port pins (0 and 1)

- Micro-SD Card

This 8GB card contains the tailored Yocto Linux* image with many libraries and tools to integrate with the IoTDK.

- USB to 6-pin FTDI serial cable

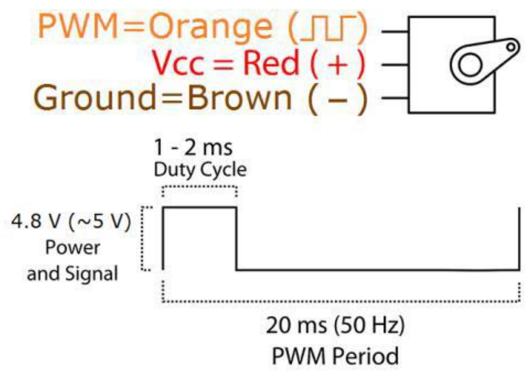
The FTDI cable is a USB to Serial (TTL level) converter which allows for a simple way to connect TTL interface devices to USB. The I/O pins of this FTDI cable are configured to operate at 5V.

- Integrated Wi-Fi* plus Bluetooth* card, including a half to full height PCIe Extender

Intel® Centrino® Wireless-N 135 delivers good Wi-Fi performance with innovative Intel-only features for a richer mobile experience. This single-stream (1x1), single-band, 802.11b/g/n Wi-Fi plus Bluetooth 4.0 product with Wi-Fi Direct* combines good speed (up to 150 Mbps).

- Wi-Fi Antennas

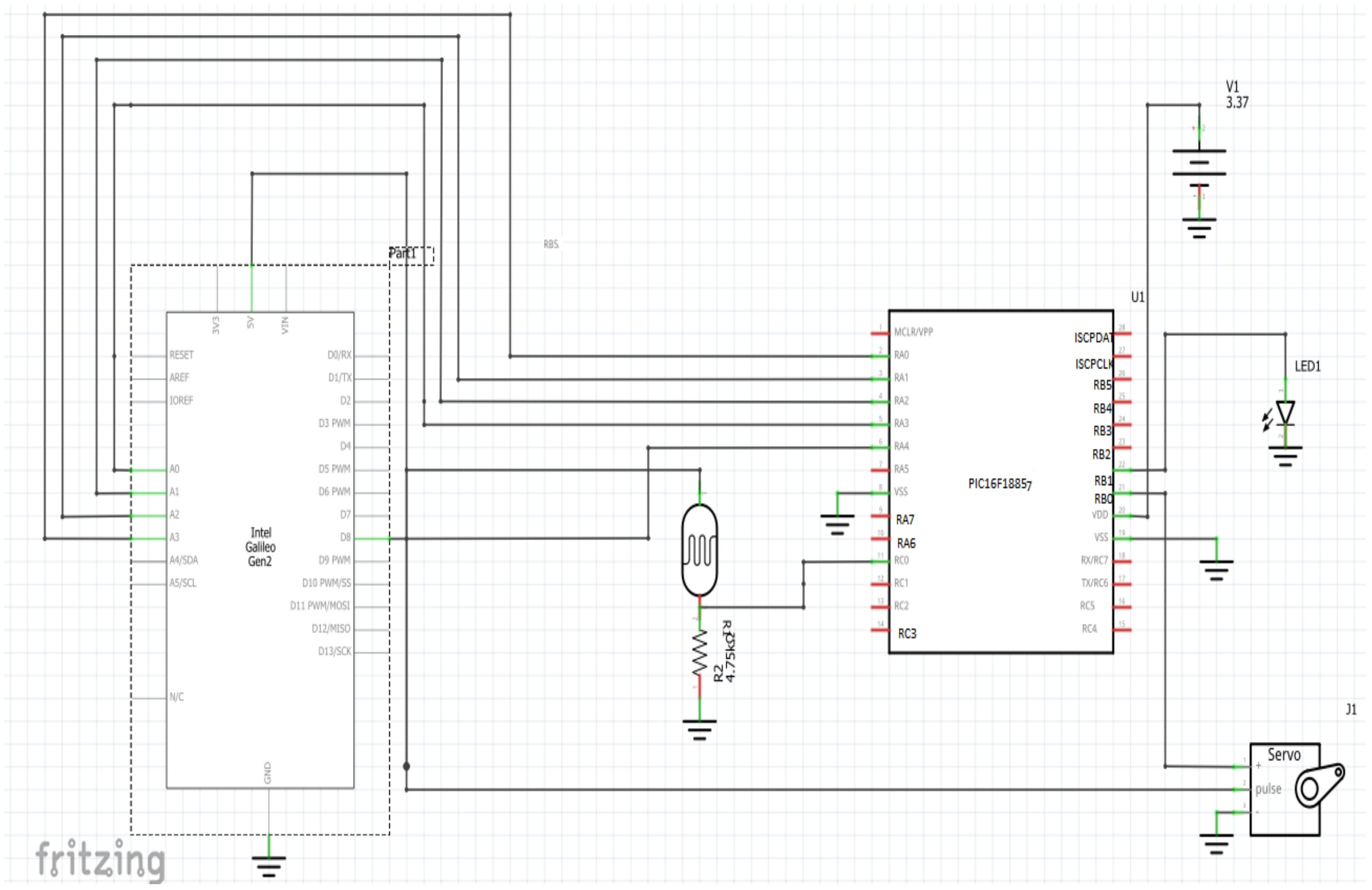
- Servo Motor



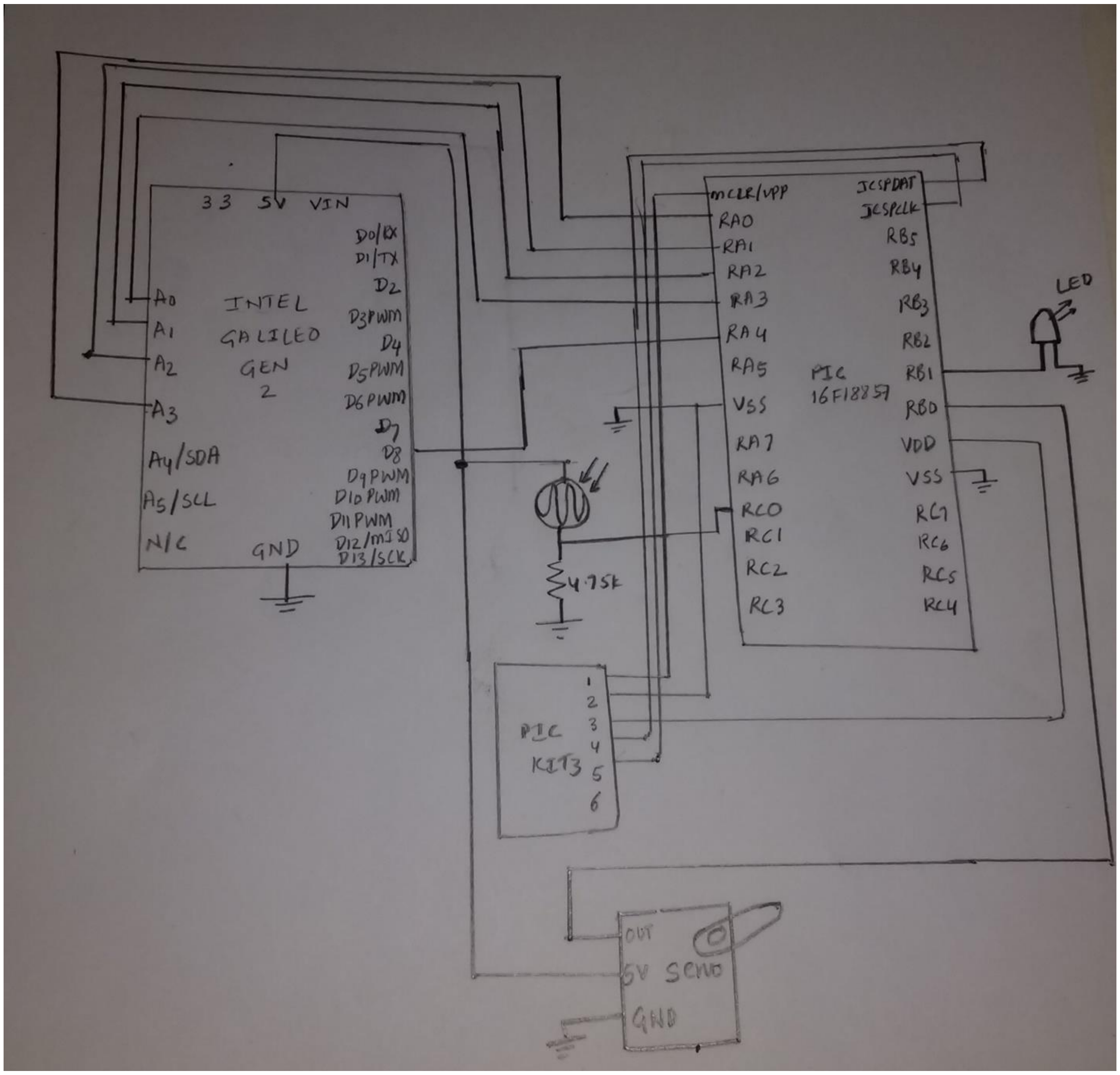
Instruments and Software:

1. PICkit-3
2. 12V regulated DC security power adapter
3. Laptop
4. MPLAB X IDE- V4.01
5. PuTTY

Section 6: Schematics /0.5 points



As I could not show the PIC Kit connection in the fritzing schematic I am using this handmade schematic as well. Please ignore if it seemed to be inappropriate.



Section 7: Lab Methods and Procedure /2 points

Hardware design:

The basic principle of this project is to design a bus protocol for the communication between the Intel Galileo Gen2 board and the PIC based sensor device.
The PIC microcontroller on the sensor device is connected to the embedded computer through the GPIO port. Over the GPIO port, five signal lines (D3-D0 and Strobe) are available. The data bus (D3-D0) is 4-bit. The control signal is Strobe driven by the Galileo board to synchronize the information exchange between the sensor device and the embedded system.

We are using five GPIO pins on the Intel Galileo board. Shield pins IO14, IO15, IO16, IO17 i.e (A0-A3) are used as data lines. Shield pin IO8 is used as Strobe i.e the control signal.
The strobe pin i.e IO8 from the Galileo is connected to the RA4 of the PIC and the data pins(A0-A3) are connected to the PORTA I/O pins (RA3-RA0) of the PIC. PIC monitors the I/O pin RA4 to see the changes in the strobe and monitors the I/O pins (RA3-RA0) to see the changes in the data lines.

PIC and LDR Sensor Connections:
First, connect the LDR to the analog input pin (RC0 in our case) on the PIC. Use a voltage divider configuration to do this. One leg of the LDR is connected to Vref (5V) on the PIC and a 4.75K resistor is also connected to the same leg and other end of the resistor is grounded. Like resistors, LDRs are not polarized. You can insert its pins without regard to polarity.

Connecting the LED to the PIC:
To test the functionality of the sensor device we are turning on and off an LED. LED is connected to the digital output pin of the PIC microcontroller (RB1 in our case). An LED is a polarized component. The longer leg indicates the positive terminal. It must be connected to the digital output pin of the PIC and the other leg must be grounded. LED is forward biased.

Connecting PIC kit3 to the PIC16F18857:

1. Pin 1 of PIC kit 3 i.e MCLR to pin 1 of PIC
2. Pin 2 of PIC kit 3 i.e VDD to pin 20 of PIC
3. Pin 3 of PIC kit 3 i.e VSS to pin 8 and pin 9 of PIC
4. Pin 4 of PIC kit 3 i.e ICSPDAT to pin 28 of PIC
5. Pin 5 of PIC kit 3 i.e ICSPCLK to pin 27 of PIC.

Pin 1 of PIC kit 3 has the arrow pointing to it. A pull up resistor of 10 K is added between MCLR and VDD pins.

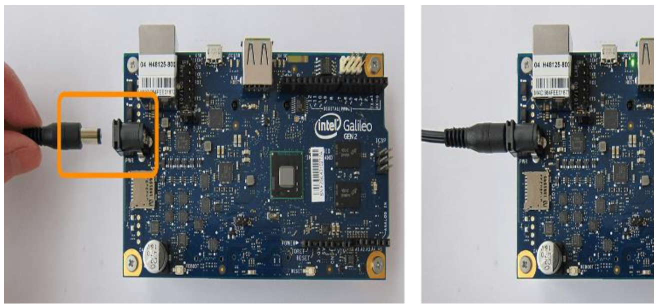
We are powering our PIC IC using PIC Kit 3 and the voltage level is set to 3.375v.

Connecting Servo motor to the PIC:

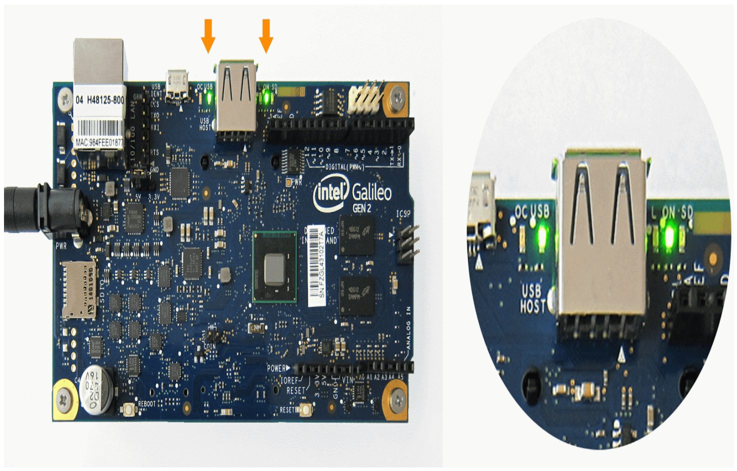
Speaking about interfacing of Servo motor to the PIC microcontroller, Servo Motors have three wires, two of them (RED and BLACK) are used for VCC(+5v) and Ground respectively and the third one is used to give control signals which is connected to digital output pin(RB0) in our case.We are supplying VCC(+5v) from the 5v pin of Galileo board.

Setting up the Galileo board:

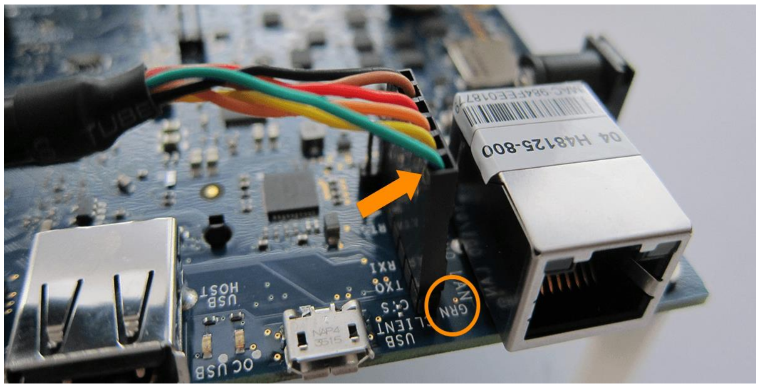
1. Confirm flashed SD card is inserted in your board properly.
2. Plug the DC power supply in to your board. Plug the other end into your wall socket.



3. Wait for two green LEDs to light up.



4. Plug in the 6 pin Serial (FTDI) to USB cable.

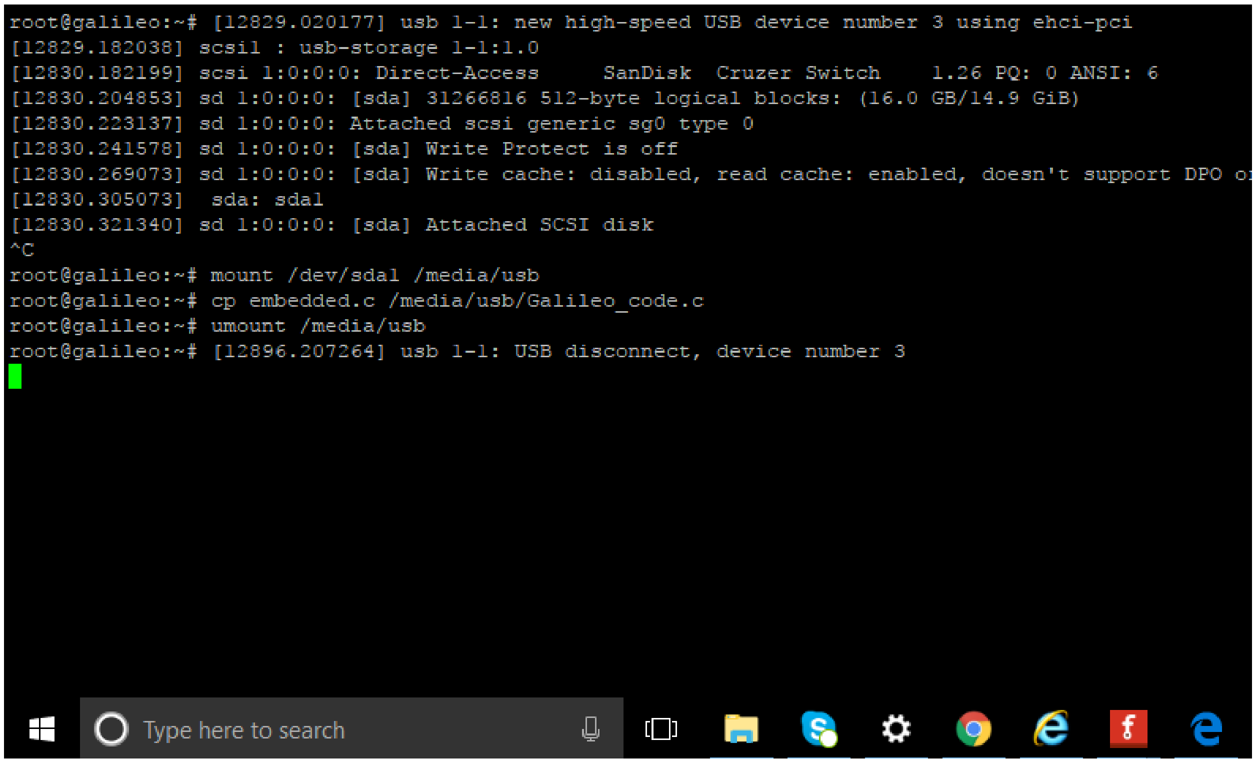


5. Plug the other end of the FTDI cable in to a USB port on your computer.

Setting up the USB interfacing:

Plug in a FAT32 USB stick to the Galileo board. Open the PuTTY console. Find the device name like sda1, sdb1or so.

Use the command mount   /dev/sda1   /media/usb to mount the USB stick.
Use the command umount   /media/usb to disconnect the USB stick.



```
root@galileo:~# [12829.020177] usb 1-1: new high-speed USB device number 3 using ehci-pci
[12829.182038] scsi1 : usb-storage 1-1:1.0
[12830.182199] scsi 1:0:0:0: Direct-Access     SanDisk  Cruzer Switch    1.26 PQ: 0 ANSI: 6
[12830.204853] sd 1:0:0:0: [sda] 31266816 512-byte logical blocks: (16.0 GB/14.9 GiB)
[12830.223137] sd 1:0:0:0: Attached scsi generic sg0 type 0
[12830.241578] sd 1:0:0:0: [sda] Write Protect is off
[12830.269073] sd 1:0:0:0: [sda] Write cache: disabled, read cache: enabled, doesn't support DPO o
[12830.305073]  sda: sda1
[12830.321340] sd 1:0:0:0: [sda] Attached SCSI disk
^C
root@galileo:~# mount /dev/sda1 /media/usb
root@galileo:~# cp embedded.c /media/usb/Galileo_code.c
root@galileo:~# umount /media/usb
root@galileo:~# [12896.207264] usb 1-1: USB disconnect, device number 3
```

Operating voltages:

* We are powering our PIC IC using PIC Kit 3 and the voltage level is set to 3.375v.

* Galileo is designed to support shields that operate at either 3.3V or 5V. The core operating voltage of Galileo is 3.3V. However, a jumper on the board enables voltage translation to 5V at the I/O pins. This provides support for 5V Uno shields and is the default behavior. By setting the jumper pins, we are providing 3.3V operation at the I/O pins.

* 5v pin from the Galileo supplies VCC =+5v which is given to the Vref of LDR and VCC of the Servo motor.

Input and Output pins:

On the Galileo side:

Shield pin IO8 – Strobe i.e the control signal

Shield pin IO14 - Data line to transfer command and receive response to/from the PIC.

Shield pin IO15 – Data line to transfer command and receive response to/from the PIC.

Shield pin IO16 - Data line to transfer command and receive response to/from the PIC.

Shield pin IO17 - Data line to transfer command and receive response to/from the PIC.

On the PIC side:

RA4 – Connected to the strobe or control signal of the Galileo

RA3 – Connected to the data pin A0 of the Galileo

RA2 - Connected to the data pin A1 of the Galileo

RA1 - Connected to the data pin A2 of the Galileo

RA0 – Connected to the data pin A3 of the Galileo.

RC0 -Analog input pin for the voltage input from the LDR.

RB1- Digital Output pin for the LED to check the functionality of light sensor

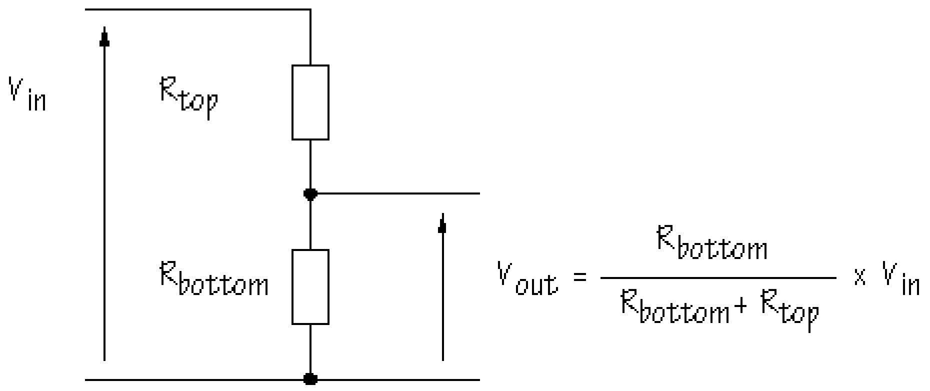
RB0- Digital Output pin for the servo motor.

<u>Circuit theory used</u>:

Voltage divider Rule:

A **light sensor** uses an LDR as part of a voltage divider.

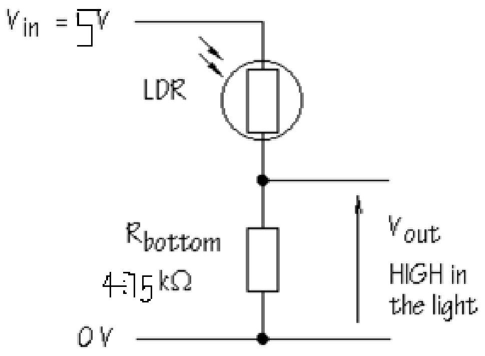
The essential circuit of a voltage divider, also called a **potential divider**, is:



As you can see, two resistors are connected in series with V_{in} , which is often the power supply voltage, connected above R_{top} . The output voltage V_{out} is the voltage across R_{bottom} and is given by:

$$V_{out} = \frac{R_{bottom}}{R_{bottom} + R_{top}} \times V_{in}$$

Here is the voltage divider built with the LDR in place of R_{top}



The resistance of the LDR is less (3K in our case) when measured in light compared to the resistance of LDR in darkness (19.2K). So as per voltage divider rule V_{out} becomes HIGH when the LDR is in the light, and LOW when the LDR is in the shade.

Software design:

GPIO pin configuration on the Galileo side:

We need to configure the GPIO pins of Galileo Gen2 using the Linux commands. In this project we are using shield pin IO8 as strobe. IO14, IO15, IO16, IO17 are used for sending commands to the PIC and receiving response from the PIC. Hence, we need to configure these pins.

Setting IO8(Strobe) as GPIO output:

```
echo  40  >   /sys/class/gpio/export                  //set as GPIO

echo out  >  /sys/class/gpio/gpio40/direction   // set as output

echo  1  >  /sys/class/gpio/gpio40/value    // set as HIGH

echo  0  > /sys/class/gpio/gpio40/value    //set as low
```

While sending the commands from the Galileo to PIC (IO14 – IO17) are set as output pins.

Setting IO14 as GPIO output:

```
echo  48 >   /sys/class/gpio/export                  //set as GPIO

echo out  >  /sys/class/gpio/gpio48/direction   // set as output

echo  1  >  /sys/class/gpio/gpio48/value    // set as HIGH

echo  0  > /sys/class/gpio/gpio48/value    //set as low
```

Setting IO15 as GPIO output:

```
echo  50 >   /sys/class/gpio/export                  //set as GPIO

echo out  >  /sys/class/gpio/gpio50/direction   // set as output

echo  1  >  /sys/class/gpio/gpio50/value    // set as HIGH

echo  0  > /sys/class/gpio/gpio50/value    //set as low
```

Setting IO16 as GPIO output:

```
echo  52 >   /sys/class/gpio/export                 //set as GPIO
```

```
echo out  >  /sys/class/gpio/gpio52/direction   // set as output
```

```
echo  1  >  /sys/class/gpio/gpio52/value    // set as HIGH
```

```
echo  0  > /sys/class/gpio/gpio52/value    //set as low
```

Setting IO17 as GPIO output:

```
echo  54 >   /sys/class/gpio/export                 //set as GPIO
```

```
echo out  >  /sys/class/gpio/gpio54/direction   // set as output
```

```
echo  1  >  /sys/class/gpio/gpio54/value    // set as HIGH
```

```
echo  0  > /sys/class/gpio/gpio54/value    //set as low
```

While receiving the response the GPIO pins(IO14-IO17) are set as input pins.

Setting IO14 as GPIO input:

```
echo in  >  /sys/class/gpio/gpio48/direction   // set as input
```

Setting IO15 as GPIO input:

```
echo in  >  /sys/class/gpio/gpio50/direction   // set as input
```

Setting IO16 as GPIO input:

```
echo in  >  /sys/class/gpio/gpio52/direction   // set as input
```

Setting IO17 as GPIO input:

```
echo in  >  /sys/class/gpio/gpio54/direction   // set as input
```

- C functions used to get and store data in the GPIOs

Sprintf() :

```
int sprintf ( char * str, const char * format, ... );
```

Write formatted data to string

Composes a string with the same text that would be printed if *format* was used on printf, but instead of being printed, the content is stored as a *C string* in the buffer pointed by *str*.

The size of the buffer should be large enough to contain the entire resulting string (see snprintf for a safer version).

A terminating null character is automatically appended after the content.

system() :

string system (string $command [, int &$return_var])

system() is just like the C version of the function in that it executes the given command and outputs the result.

Where:

command

The command that will be executed.

return_var

If the return_var argument is present, then the return status of the executed command will be written to this variable.

- To read the input values from the GPIO pins (IO14-IO17) the following C functions are used.

Open():

Open a file for read or write.

fd = open(name, flag [, mode]);
Where:
char *name;

points to a string containing the name of the file to be opened.

int flag;

indicates various options for the open process.

We are using the flag O_RDONLY - open for reading only.

Read():

#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);

ssize_t read(int *fildes*, void **buf*, size_t *nbyte*);

The read() function shall attempt to read nbyte bytes from the file associated with the open file descriptor, fildes, into the buffer pointed to by buf. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device is unspecified.

Pin configuration on PIC side:

The strobe of the Galileo is connected to RA4 pin of PIC microcontroller. So it is set as an input pin using the piece of code.

```
ANSELAbits.ANSA4 = 0;
TRISAbits.TRISA4 = 1;
PORTAbits.RA4 = 0;
```

IO14 to IO17 data pins from the Galileo board are connected to RA3 – RA0 respectively.

Hence while receiving the command from Galileo these pins(RA3-RA0) on the PIC are set as input pins as follows:

```
ANSELA = 0x00;
TRISA = 0x1F;
PORTA = 0x1F;
```

And while sending the response these pins are set as output pins as follows:

```
ANSELA = 0x00;
TRISA = 0x10;
PORTA = 0x1F;
```

ADC configuration:

PIC16F18857 has 10- bit ADC. If the reference voltage (explained latter) of ADC is 0 to 5v then a 10bit ADC will break it in 1024 divisions so it can measure it accurately up to 5/1024 v= 4.8mV approx.

When configuring and using the ADC the following functions must be considered:

• **Port configuration**

When converting analog signals, the I/O pin should be configured for analog by setting the associated TRIS and ANSEL bits.

We are configuring RC0 pin of PORTC by setting TRISC = 0x01 and ANSELC= 0x01

• Channel selection

We are selecting ANSC0 = 1and selecting channel 0

• **ADC voltage reference selection**

The ADPREF bits of the ADREF register provides control of the positive voltage reference. The positive voltage reference can be:

•VREF+ pin •V DD • FVR 1.024V • FVR 2.048V • FVR 4.096V

The ADNREF bit of the ADREF register provides control of the negative voltage reference. The negative voltage reference can be: •VREF- pin •V SS

We are setting positive voltage reference to VDD and negative voltage reference to VSS by setting ADREF= 0x00.

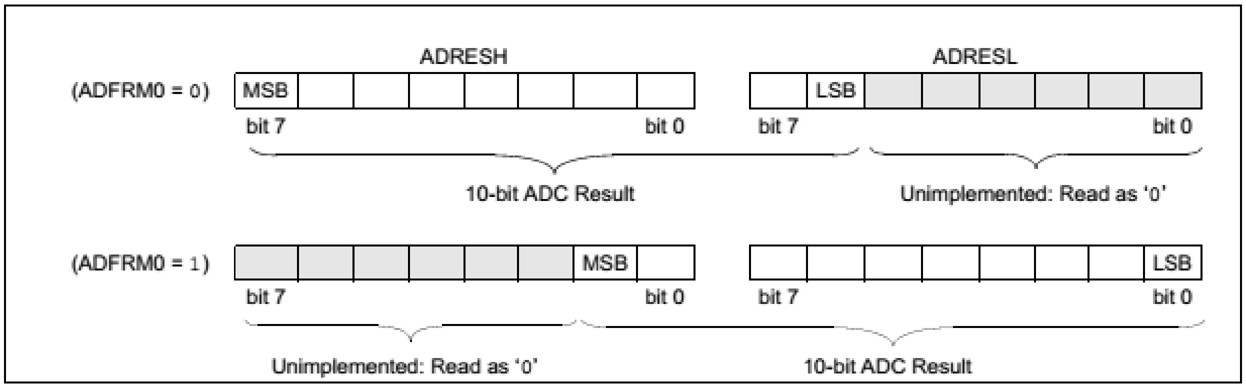
• **ADC conversion clock source**

The source of the conversion clock is software selectable via the ADCLK register and the ADCS bit of the ADCON0 register. There are two possible clock sources: •FOSC/(2*(n+1)) (where n is from 0 to 63), • FRC (dedicated RC oscillator)

We are setting ADCLK = 0x02 which means clock source will be FOSC/6 and ADCS bit of ADCON0 to 0 which means Clock supplied by FOSC, divided according to ADCLK register.

• **Result formatting**

The 10-bit ADC conversion result can be supplied in two formats, left justified or right justified. The ADFRM0 bit of the ADCON0 register controls the output format.



We are setting ADFRM0 and hence our result is right justified.

Starting a conversion

To enable the ADC module, the ADON bit of the ADCON0 register must be set to a '1'. A conversion may be started by setting the ADGO bit of ADCON0 to '1'

Completion of conversion

When any individual conversion is complete, the value already in ADRES is written into ADPREV (if ADPSIS=1) and the new conversion results appear in ADRES.

When the conversion completes, the ADC module will clear the ADGO bit (Unless the ADCONT bit of ADCON0 is set)

PIC PWM implementation:

Handling a GPIO pin and delay functions:

The pulse has been created with the help of delays in the program. The delay duration for an angle is equivalent to the length of the pulse required for the motor to rotate up to that corresponding angle. A pulse of 1500us is required for a rotation up to 90°. We are using a pulse of 1000 us for 30 degree and 1800 for 120 degree rotation.

Delay_us() is used to make delay in microsecond during the program execution.

Explanation of the logic of the ADC_read() function

1) Configure the ADC module by setting the required registers as explained in the section ADC configuration. This will be done by the function ADC_init()

2) Start the ADC conversion by following these steps

* Enable the ADC module by setting ADCON0bits.ADON = 1

* Start the conversion by setting ADCON0bits.ADGO = 1

3) wait for ADC conversion to complete i.e till ADGO bit is cleared. This is done by using a while loop.

```
while(ADCON0bits.ADGO==1);
```

4) ADC results will be in the registers ADRESH and ADRESL. Shift left ADRESH by 8 bits and add the ADRESL to the result in order to get the complete 10 bit result value.

```
lightval = ((ADRESH<<8) |ADRESL);
```

5)after getting the result of ADC conversion disable the ADC by setting ADCON0bits.ADON = 0

7) If the ADC conversion result is more than 900.i.e if the light intensity is more then turn the LED OFF by setting RB1 = 0

6) Else, turn on the LED by setting RB1= 1.

7) Assign it to the global variable light. Later process the global variable light in order to send it in 3 packets of 4 nibbles to the Galileo.

Explanation of piece of code for rotating servo to 90 degree:

We need to generate a PWM of close to 50 Hz frequency. So our time period has to be nearly 20 ms.

* In order to rotate 90 degree, a pulse of 1.5 ms is required. So set the Bit 0 of PORT B = 1 and keep it high for 1500 us which is achieved by

```
PORTBbits.RB0 = 1;
__delay_us(1500);
PORTBbits.RB0 = 0;
__delay_us(18500);
```

These statements are in a for loop so that the servomotor stays in that state for some amount of time.

Similarly, we are rotating the servo motor to 30 degree and 120 degree by varying the pulse width around 1ms on period for 30 degree rotation and 1.8 ms on period for 120 degree rotation.

Explanation of the Galileo Code :

- First set the GPIO pins (IO8 i.e strobe ,IO14,IO15,IO16,IO17(Arudino Shield pins A0-A3) by calling the function initialise() where in the pins are set using the command

  echo xx > /sys/class/gpio/export

  where, xx is the gpio pin number.

- Print the user options:
  1 for PING command
  2 for RESET command
  3 for ADC value read command
  4 for TURN servomotor 30 degrees
  5 for TURN servo motor 90 degrees
  6 for TURN servo motor 120 degrees

- Scan the input command from the user.

- If command is 0x01 i.e MSG_PING then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 3s so that the PIC reads the command.

- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()

- Set the Strobe low and wait for 1s and then read the response using the function readGPIO(). readGPIO() function uses open() and read() system calls.

- Then if the response read is equal to 0x0E(MSG_ACK) then print "PIC is responding" else print "PIC is not responding"

- Else, if the command is 0x00 i.e MSG_RESET i.e 0x00 then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 1s so that the PIC reads the command.

- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()

- Set the Strobe low and wait for 1s and then read the response using the function readGPIO(). readGPIO() function uses open() and read() system calls.

- Then if the response read is equal to 0x0F(MSG_NTHNG) then print "PIC is reset" else print "PIC is not reset".

- Else, if the command is 0x03(TURN_30) then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 5s so that the PIC reads the command. Rotates the servo motor to 30 degrees.

- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()

- Set the Strobe low and wait for 5s and then read the response using the function readGPIO(). readGPIO() function uses open() and read() system calls.

- Then if the response read is equal to 0x06 then print "PIC is rotating the servo to 30 degrees" else print "PIC is not rotating the servo to 30 degrees".

- Else, if the command is 0x04(TURN_90) then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 5s so that the PIC reads the command. Rotates the servo motor to 90 degrees.

- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()
- Set the Strobe low and wait for 5s and then read the response using the function readGPIO(). readGPIO() function uses open() and read() system calls.
- Then if the response read is equal to 0x07 then print "PIC is rotating the servo to 90 degrees" else print "PIC is not rotating the servo to 90 degrees".
- Else, if the command is 0x05(TURN_120) then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 5s so that the PIC reads the command. Rotates the servo motor to 120 degrees.
- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()
- Set the Strobe low and wait for 5s and then read the response using the function readGPIO(). readGPIO() function uses open() and read() system calls.
- Then if the response read is equal to 0x08 then print "PIC is rotating the servo to 120 degrees" else print "PIC is not rotating the servo to 120 degrees".
- Else if the command is 0x02(MSG_GET) then then initialise the GPIO pins(A0-A3) as output using GPIO_init() function. Set the strobe high and then after a 1 s delay write the command to the GPIO pins using writeGPIO() function. Wait for 5s so that the PIC reads the command.
- Set the GPIO pins as input so as to receive the response using the function GPIO_reinit()
- Wait for 6s and then Set the Strobe low and read the first 4-bit packet of ADC value sent by the PIC using the function readADC(). readADC()function uses open() and read() system calls to read the values on the input pins.
- Then wait for 3s and again set the strobe high and wait for 1s and then set the strobe low. After 2s delay read the second 4-bit packet of ADC value sent by the PIC using the function readADC(). readADC()function uses open() and read() system calls to read the values on the input pins.
- Then wait for 3s and again set the strobe high and wait for 1s and then set the strobe low. After 2s delay read the third 4-bit packet of ADC value sent by the PIC using the function

readADC(). readADC()function uses open() and read() system calls to read the values on the input pins.

- Wait for 3s and then set the strobe high. After 5 s reinitialize the GPIO pins as input and set the strobe low. After 5s , read the response from the PIC if the response is equal to 0x0E(MSG_ACK) then it indicates that the PIC has sent the 10 bit ADC result.

- So if response is equal to 0x0E(MSG_ACK) then print “ADC value is read” else print “Error”.

- Combine the three 4 bit packets by shifting the third packet that has the first two bits of the 10-bit ADC result to 6 times to the left. Second packet is left shifted 4 times and the first packet is kept as it is.

- The three packets are then ORed to form the 10 bit ADC result.
  ADC_value = (packet3<<6) |(packet2<<4) |(packet1); //combine the three packets
  ADC value is printed.

<u>Explanation of the PIC code</u>:

- Initialize the PORTA pins as digital input pins. RA4 is connected to the strobe input. And pins RA3-RA0 are connected to the data lines IO14-I015 i.e Arudino Shield pins (A0-A3).

- Also initialize RB0 which is connected to Servo motor output pin as a digital output pin.

- Wait till the strobe input pin i.e RA4 goes high and once it goes high wait for 1.5s and then receive the message in the variable msg using the function receive_msg()

- If the message received i.e command from the Galileo is MSG_PING then reinitialize the pins(RA3-RA0) as output pins using the functions reinit_out() and wait for the strobe to go low.

- Once the strobe is low send 0x0E(MSG_ACK) on the pins RA3- RA0. i.e RA3 = 1, RA2 =1 ,RA1= 1, RA0 = 0. Wait for 1s so that the Galileo reads the response.

- Else if the command received from the Galileo is MSG_RESET(0x00) then then reinitialize the pins(RA3-RA0) as output pins using the functions reinit_out() and wait for the strobe to go low.

- Once the strobe is low send 0x0F(MSG_NOTHING) on the pins RA3- RA0. i.e RA3 = 1, RA2 =1 ,RA1= 1, RA0 = 1.Wait for 3s so that the Galileo reads the response.

- Else if the command received is 0x02 (MSG_GET) then then reinitialize the pins(RA3-RA0) as output pins using the functions reinit_out() .Perform ADC_read() function. Result of ADC conversion will be taken in a global variable called “light”

- Once the strobe goes low, the global variable “light” is processed using the set_and_shift() function and the last 4 bits of the 10 bit ADC result is sent on the pins RA3-RA0 using the function set_pins()

- Wait till the strobe goes high. And do nothing when the strobe is high.

- Wait for the strobe to go low. Once it is set low, send the next 4 bits of ADC result on the pins RA3-RA0 using the functions set_and_shift() and set_pins().

- Wait till the strobe goes high. And do nothing when the strobe is high.

- Wait for the strobe to go low. Once it is set low, send the next packet that consists of the first two bits of 10 bit ADC result on the pins RA3-RA0 using the functions set_and_shift() and set_pins().

- Wait till the strobe goes high. And do nothing when the strobe is high.

- Wait for the strobe to go low. Once it is set low, send the response MSG_ACK on the pins RA3-RA0 i.e RA3=1 ,RA2 =1 RA1 =1 and RA0 = 0 .It indicates that the ADC result reporting has been finished. Wait for 2s so that the Galileo reads the response.

- Else if the command received from the Galileo is 0x03 (TURN_30) then reinitialize the pins RA3-RA0 as output pins and then wait for the strobe to go low. Once the strobe i.e RA4 is low, rotate the servo motor to 30 degrees by setting the pulse width to 1ms as we are using 50Hz PWM signal.

- After the rotation send the response 0x06 on RA3-RA0 indicating the rotation has been done. Wait for 2s so that the Galileo reads the response.

- Else if the command received from the Galileo is 0x04 (TURN_90) then reinitialize the pins RA3-RA0 as output pins and then wait for the strobe to go low. Once the strobe i.e RA4 is low, rotate the servo motor to 90 degrees by setting the pulse width to 1.5ms as we are using 50Hz PWM signal.

- After the rotation send the response 0x07 on RA3-RA0 indicating the rotation has been done. Wait for 2s so that the Galileo reads the response

- Else if the command received from the Galileo is 0x05 (TURN_120) then reinitialize the pins RA3-RA0 as output pins and then wait for the strobe to go low. Once the strobe i.e RA4 is low, rotate the servo motor to 120 degrees by setting the pulse width to 1.8 ms as we are using 50Hz PWM signal.

- After the rotation send the response 0x08 on RA3-RA0 indicating the rotation has been done. Wait for 2s so that the Galileo reads the response.

Section 8: Trouble Shooting /1 points

ISSUE 1:

At first when we booted our Galileo board using the SD card, we were unable to use the Linux commands like cd , gcc .
Because we had not extracted and flashed the .direct file using Win32DiskImager. Instead we had chosen the whole Galileo_Poky_SW_image_20160606.zip in the drop down list of Win32DiskImager because of which SD card was not having the right Yocto*-built Linux image. Later we realized our mistake and loaded the .direct file to the SD card and then we were able to use the Linux libraries and resources.

ISSUE 2:
We were not knowing how to configure the GPIO pins. So when we started working and tried blinking the on board led on and off, we were not successful. We studied the GPIO pin mapping provided in the GitHub Link and also referred to the Intel Community website and then we were able to configure the pins properly.

ISSUE 3:
We had trouble in USB interfacing as we were new to Linux environment. We were referring to the link http://askubuntu.com/questions/37767/how-to-access-a-usb-flash-drive-from-the-terminal-how-can-i-mount-a-flash-driv mentioned in the GitHub to mount our USB device. As per which we had to use 'sudo' command. We were getting error as were the root user. Later we realized it and we used proper commands to mount our USB. Say,

```
mount   /dev/sda1   /media/usb
```

ISSUE 4:
We were having trouble in rotating our servo motor. We were debugging a lot with our code. But when we connected VCC pin of servo motor to the 5V pin of Galileo board it started working properly.

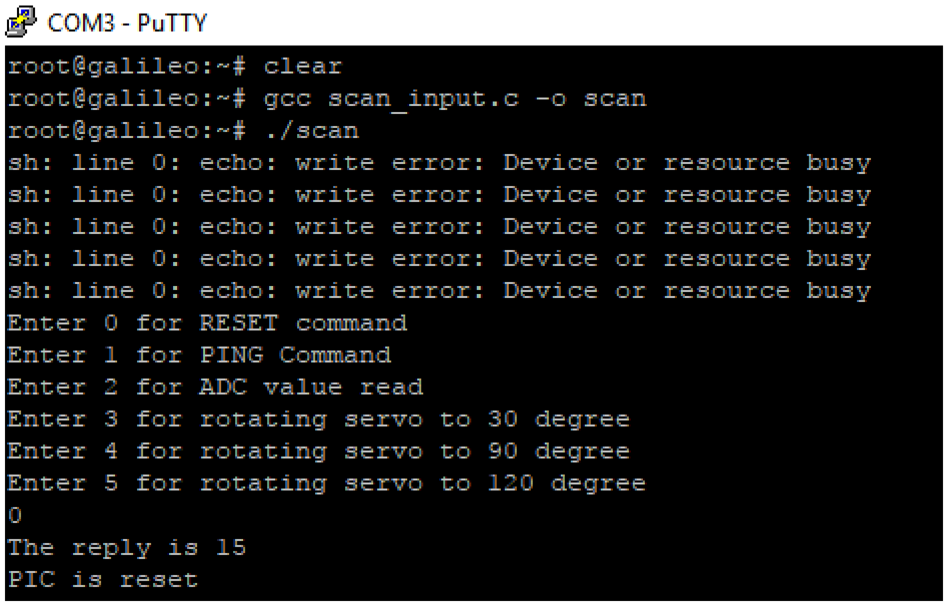
ISSUE 5:

We had trouble in making Galileo and the PIC based sensor device to be synchronized. We had to do lot of manipulation with the delay in both Galileo code and the PIC code so that they both are synchronized.

Section 9: Results *_/0.5 points_*

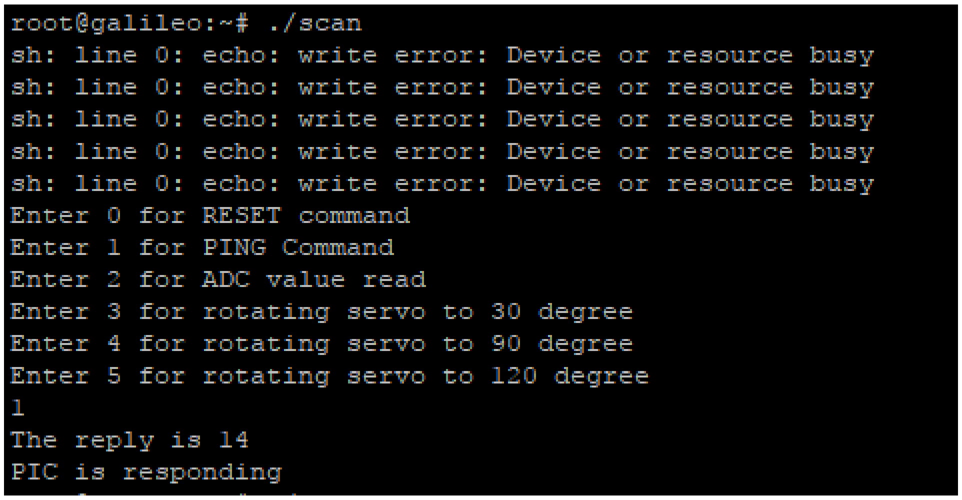
Results screenshots for the User input commands:

Option 1 for RESET command



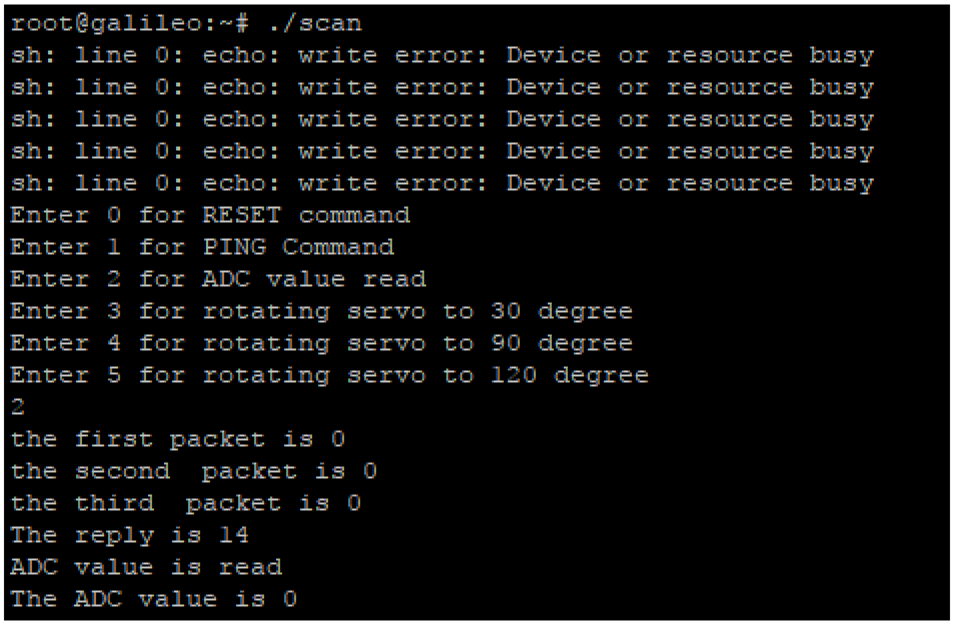
```
root@galileo:~# clear
root@galileo:~# gcc scan_input.c -o scan
root@galileo:~# ./scan
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
0
The reply is 15
PIC is reset
```

Option 2 for PING command



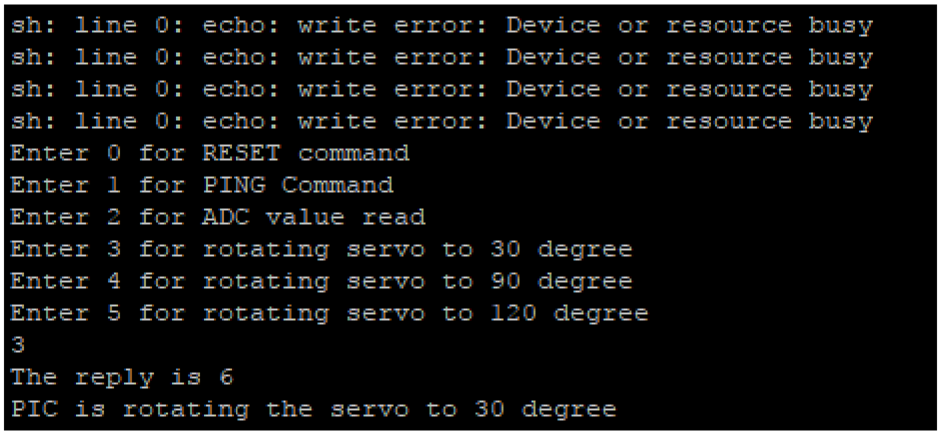
```
root@galileo:~# ./scan
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
1
The reply is 14
PIC is responding
```

Option 3 for MSG_GET i.e ADC read command



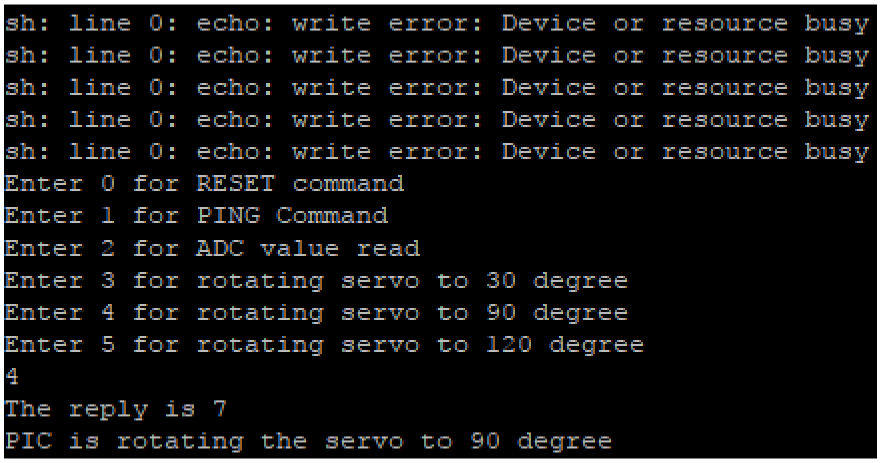
```
root@galileo:~# ./scan
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
2
the first packet is 0
the second  packet is 0
the third  packet is 0
The reply is 14
ADC value is read
The ADC value is 0
```

Option 4 for TURN_30 command (to rotate servo motor to 30 degree)



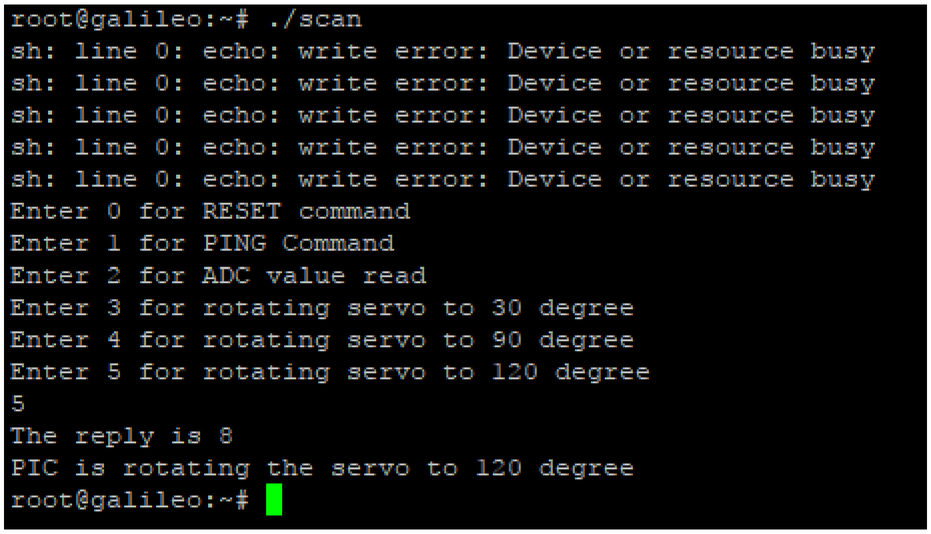
```
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
3
The reply is 6
PIC is rotating the servo to 30 degree
```

Option 5 TURN_90 command (to rotate servo motor to 90 degrees)



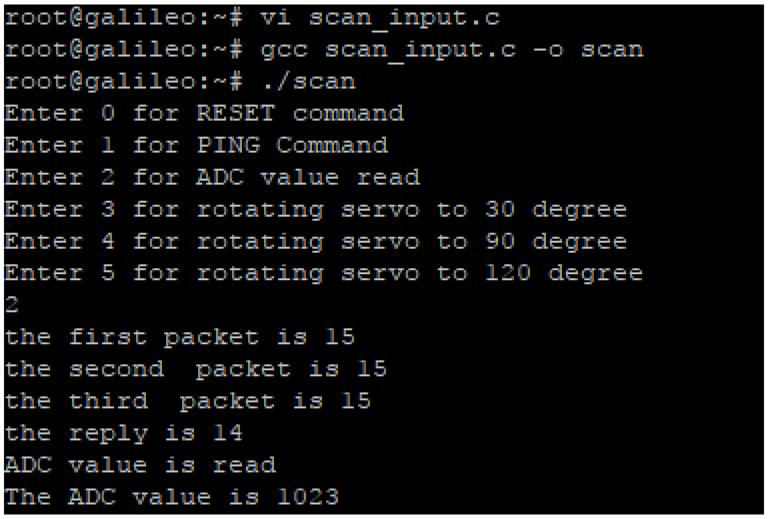
```
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
4
The reply is 7
PIC is rotating the servo to 90 degree
```

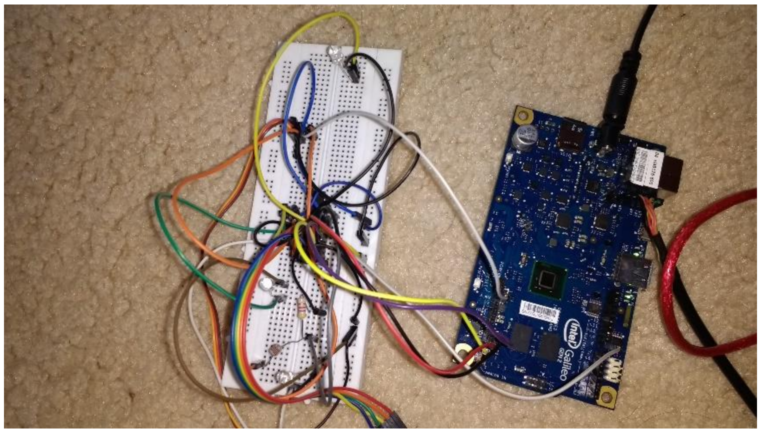
Option 6 for TURN_120 command(to rotate servo motor to 120 degrees)


```
root@galileo:~# ./scan
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
5
The reply is 8
PIC is rotating the servo to 120 degree
root@galileo:~# 
```

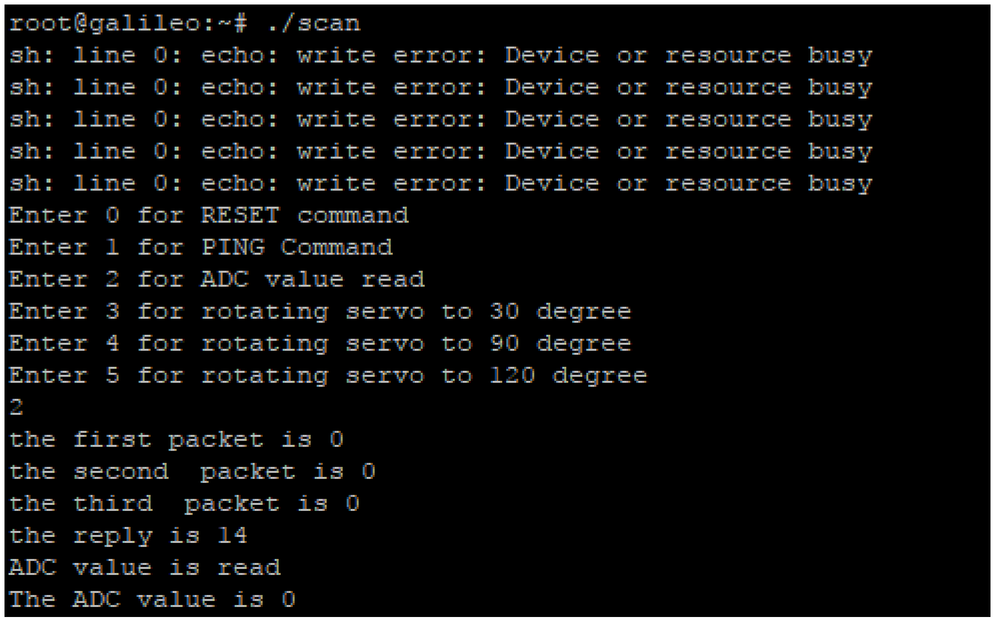
ADC different values depending on the light intensity

When the light intensity is more, the ADC conversion result is high and the LED is tuned off.

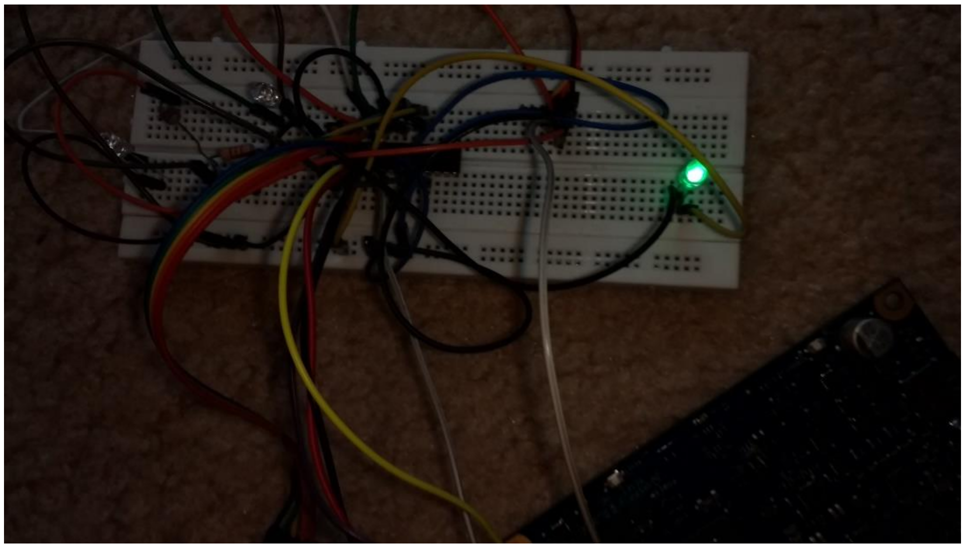

```
root@galileo:~# vi scan_input.c
root@galileo:~# gcc scan_input.c -o scan
root@galileo:~# ./scan
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
2
the first packet is 15
the second  packet is 15
the third  packet is 15
the reply is 14
ADC value is read
The ADC value is 1023
```



When the light intensity is less, ADC result is low and the LED is turned ON.

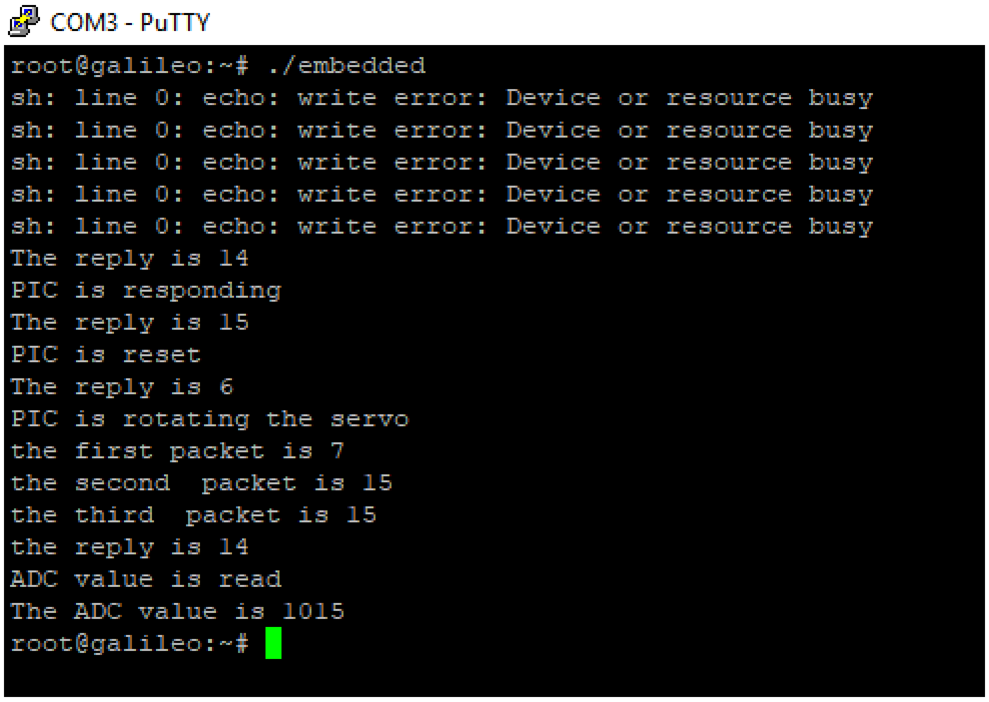


```
root@galileo:~# ./scan
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
Enter 0 for RESET command
Enter 1 for PING Command
Enter 2 for ADC value read
Enter 3 for rotating servo to 30 degree
Enter 4 for rotating servo to 90 degree
Enter 5 for rotating servo to 120 degree
2
the first packet is 0
the second  packet is 0
the third  packet is 0
the reply is 14
ADC value is read
The ADC value is 0
```

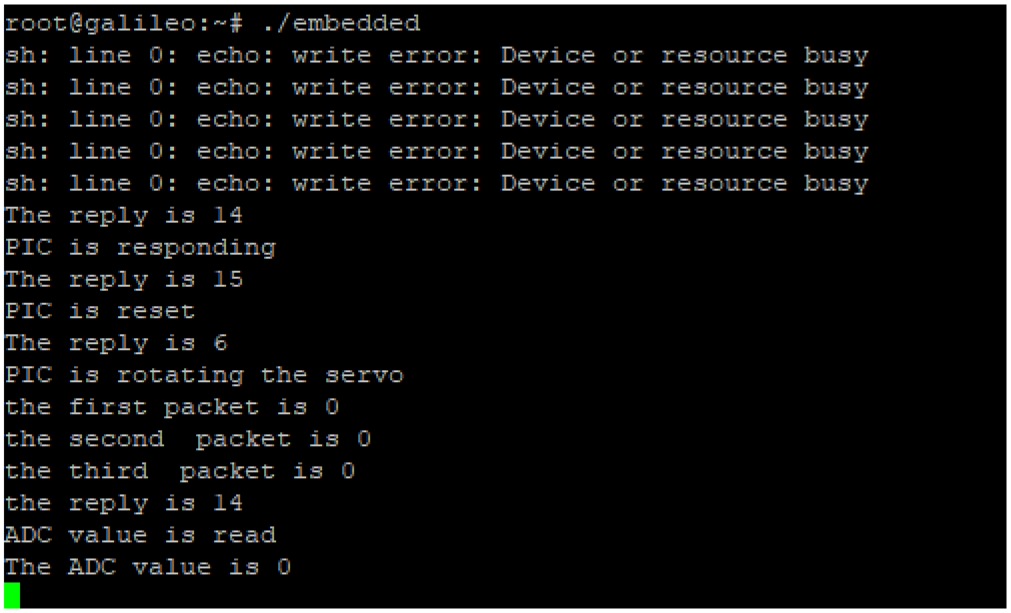


Results for the code with hardwired input commands. (Without scanf function)

Galileo sends command one by one and the PIC performs the necessary actions and sends back the response to each command.



```
root@galileo:~# ./embedded
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
The reply is 14
PIC is responding
The reply is 15
PIC is reset
The reply is 6
PIC is rotating the servo
the first packet is 7
the second  packet is 15
the third  packet is 15
the reply is 14
ADC value is read
The ADC value is 1015
root@galileo:~#
```



```
root@galileo:~# ./embedded
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
sh: line 0: echo: write error: Device or resource busy
The reply is 14
PIC is responding
The reply is 15
PIC is reset
The reply is 6
PIC is rotating the servo
the first packet is 0
the second  packet is 0
the third  packet is 0
the reply is 14
ADC value is read
The ADC value is 0
```

Section 10: Appendix

Note:

Initially we started with the code which has hardwired commands. Say, we had not used any scanf function. Later after testing each command and its response we added the user input scan feature.

A1. Galileo code (without using scanf function)

```
#include <stdlib.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

void initialise()

{

        char buffer[200];

        sprintf(buffer, "echo 40 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 48 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 50 > /sys/class/gpio/export");
        system(buffer);

        sprintf(buffer, "echo 52 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 54 > /sys/class/gpio/export");

        system(buffer);


}

void GPIO_reinit()

{

        char buf[200]={0};

        sprintf(buf, "echo in > /sys/class/gpio/gpio48/direction");

    system(buf);

    sprintf(buf, "echo in > /sys/class/gpio/gpio50/direction");

    system(buf);
    sprintf(buf, "echo in > /sys/class/gpio/gpio52/direction");

    system(buf);

    sprintf(buf, "echo in > /sys/class/gpio/gpio54/direction");

    system(buf);


}

void GPIO_init()

{



    char buf[200];








    sprintf(buf, "echo out > /sys/class/gpio/gpio48/direction");

    system(buf);



    sprintf(buf, "echo out > /sys/class/gpio/gpio50/direction");
    system(buf);



    sprintf(buf, "echo out > /sys/class/gpio/gpio52/direction");
    system(buf);



    sprintf(buf, "echo out > /sys/class/gpio/gpio54/direction");
    system(buf);

}

void writeGPIO(int command)
{
        int x = command;
        int y = 0;

        char buffer[200]={0};
        //sending first lsb bit of the command to gpio54
        y = x & 0x01;
        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio54/value", y);
    system(buffer);


        //sending second lsb bit of the command to gpio52
        x = x>>1;
        y = x & 0x01;


        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio52/value", y);
    system(buffer);


    //sending third lsb bit of the command to gpio50
        x = x>>1;
        y = x & 0x01;


        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio50/value", y);
    system(buffer);


    //sending fourth lsb bit (msb) of the command to gpio54
        x = x>>1;
```

```
        y = x & 0x01;

        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio48/value", y);

    system(buffer);

    return;


}

char readGPIO()

{

        char data = 0;

        char buff1[200]={0};

        char buff2[200]={0};

        char buff3[200]={0};

        char buff4[200]={0};

        int val1 = 0;

        int val2 =0;

        int val3 = 0;

        int val4 = 0;


        int fd1 = -1;

        fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);
```

```
if(fd1!=-1)
{
        read(fd1,buff1,1);
        close(fd1);
}
if(buff1[0]=='0')
{
        val1 = 0;
}
else
{
        val1 = 1;
}

int fd2 = -1;
fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);
if(fd2!=-1)
{
        read(fd2,buff2,1);
        close(fd2);
}
if(buff2[0]=='0')
{
        val2 = 0;
}
else
{
        val2 = 1;
}
int fd3 = -1;
fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);
if(fd3!=-1)
{
        read(fd3,buff3,1);
        close(fd3);
}
if(buff3[0]=='0')
{
        val3 = 0;
}
```

```
else
{
        val3 = 1;
}

int fd4 = -1;
fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);
if(fd4!=-1)
{
        read(fd4,buff4,1);
        close(fd4);
}
if(buff4[0]=='0')
{
        val4 = 0;
}
else
{
        val4 = 1;
}
```

```
    //data = val1;

    data= (val1<<3)|(val2<<2)|(val3<<1)|(val4);

    //printf("%d\t",data);

    return data;

}

unsigned int readADC()

{

    unsigned int packet = 0;

    char buff1[200]={0};

    char buff2[200]={0};

    char buff3[200]={0};

    char buff4[200]={0};

    int val1 = 0;

    int val2 = 0;

    int val3 = 0;

    int val4 = 0;



    int fd1 = -1;

    fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);
```

```
if(fd1!=-1)
{
        read(fd1,buff1,1);
        close(fd1);
}
if(buff1[0]=='0')
{
        val1 = 0;
}
else
{
        val1 = 1;
}

int fd2 = -1;
fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);
if(fd2!=-1)
{
        read(fd2,buff2,1);
        close(fd2);
}

if(buff2[0]=='0')

{

        val2 = 0;

}

else

{

        val2 = 1;

}

int fd3 = -1;

fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);

if(fd3!=-1)

{

        read(fd3,buff3,1);

        close(fd3);

}

if(buff3[0]=='0')

{

        val3 = 0;

}
```

```
else

{

        val3 = 1;

}



int fd4 = -1;

fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);

if(fd4!=-1)

{

        read(fd4,buff4,1);

        close(fd4);

}

if(buff4[0]=='0')

{

        val4 = 0;

}

else

{

        val4 = 1;

}
```

```
	packet= (val4<<3)|(val3<<2)|(val2<<1)|(val1);

	return packet;

}





int main(void)

{

//	printf("hi\n");

	char buffer[200]={0};

	char reply =0 ;

	unsigned int packet1 = 0;

	unsigned int packet2 = 0;

	unsigned int packet3 = 0;

	unsigned int packet4 =0 ;

	unsigned int ADC_value = 0;

	initialise();

	sprintf(buffer, "echo out > /sys/class/gpio/gpio40/direction");

	system(buffer);
```

```
GPIO_init();   //initialise the gpio pins as output and then send the command

//1.Strobe high

sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                    system(buffer);



sleep(1);

//2.write data

writeGPIO(0x01);   //ping the pic micro

sleep(3);



GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

//3.Strobe low

sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                    system(buffer);

sleep(1);



//4.Strobe high
```

```
//      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //                     system(buffer);




                        //2.read data

        reply = readGPIO();

        printf("The reply is %d\n",reply);

        if(reply!= 0x0E)

        {

             printf("PIC is not responding\n");

                        }


                        else

                        {


                                printf("PIC is responding\n");

                                }


                                sleep(1);
```

```
                              //3.Strobe low
         sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                         system(buffer);
//              sleep(1);
            GPIO_init();   //initialise the gpio pins as output and then send the command
        //1.Strobe high
         sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                        system(buffer);


        sleep(1);
        //2.write data
        writeGPIO(0x00);   //reset the pic micro
        sleep(1);
        GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
        //3.Strobe low
        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                        system(buffer);
        sleep(1);


        //4.Strobe high
```

```
//        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
 //                    system(buffer);




                       //2.read data
        reply = readGPIO();
        printf("The reply is %d\n",reply);
        if(reply!= 0x0F)
        {
            printf("PIC is not reset\n");
                       }

                       else
                       {

                              printf("PIC is reset\n");
                              }
                              sleep(1);

                              //3.Strobe low

           sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                           system(buffer);

//         sleep(1);


               GPIO_init();   //initialise the gpio pins as output and then send the command

          //1.Strobe high

          sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                          system(buffer);



          sleep(1);

          //2.write data

          writeGPIO(0x03);   //rotating  the servo

          sleep(5);

          GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

          //3.Strobe low

          sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                          system(buffer);

          sleep(5);
```

```
        //4.Strobe high


//      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //                 system(buffer);




                    //2.read data

        reply = readGPIO();

        printf("The reply is %d\n",reply);

        if(reply!= 0x06)

        {

          printf("PIC is not rotating the servo\n");

                    }


                    else

                    {


                       printf("PIC is rotating the servo\n");
```

```
                              }



                              sleep(3);

                              //3.Strobe low

 sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                    system(buffer);

GPIO_init();   //initialise the gpio pins as output and then send the command

//1.Strobe high

sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                    system(buffer);



sleep(1);

//2.write data

writeGPIO(0x02);   //reading  the adc

sleep(1);

GPIO_reinit();   //initialise the gpio pins as input pins and then read the response




sleep(6);

//3.Stirobe low
```

```
        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);



        //4.Strobe high


//      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
 //                 system(buffer);





                    //2.read data
        packet1  = readADC();


        printf("the first packet is %d\n",packet1);
                            sleep(3);
                            //3.Strobe low
      //   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
      //                 system(buffer);
 packet4 = 14;

   sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                   system(buffer);

     sleep(1);

     sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

     system(buffer);

               sleep(2);

  packet2  = readADC();



 printf("the second  packet is %d\n",packet2);

 sleep(3);

                      //3.Strobe low

//   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

//                   system(buffer);



   sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                   system(buffer);

     sleep(1);

     sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

     system(buffer);
```

```
    sleep(2);


               packet3  = readADC();


  printf("the third  packet is %d\n",packet3);

  sleep(6);

                                //3.Strobe low

//   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

//                    system(buffer);

//

  sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                      system(buffer);

     sleep(1);

     sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

     system(buffer);

    sleep(1);


               reply  = readGPIO();


  printf("the reply is %d\n",reply);
```

```
}

            sleep(3);


            if(reply==0x0E)

           {

                    printf("ADC value is read\n");

           }


            ADC_value = (packet3<<6) |(packet2<<4) |(packet1);

            printf("The ADC value is %d\n", ADC_value);

              sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                              system(buffer);

                sleep(1);

                sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                system(buffer);

               sleep(2);




}
```

A2. PIC code (for using scanned user input commands)

```
/*
 * File:   pic_com.c
 * Author: Family
 */







/*/*/
 /* File:   PIC and Galileo communication
 *
 *
 * simple PIC program example
 * for UMass Lowell 16.480/552
 *
 * Author: Ioann
 * Created on 2017/9/21
 */




#include "mcc_generated_files/mcc.h" //default library



static unsigned int light =0 ;

char P0,P1,P2,P3,P4;

void set_and_shift()

{

            P0 = (light) & 1;             //lsb of the packet

    P1 = (light >> 1) & 1;

            P2 = (light >> 2) & 1;

            P3 = (light >> 3) & 1;

    light = light >> 4;

}


 void set_pins()

 {
  TRISAbits.TRISA3 = 0;

  TRISAbits.TRISA2 = 0;

  TRISAbits.TRISA1 = 0;

  TRISAbits.TRISA0 = 0;




   if (P0==1)

{

    PORTAbits.RA0 = 1;



}

 else

 {

            PORTAbits.RA0 = 0;



 }

        if (P1==1)

  {

         PORTAbits.RA1 = 1;
```

```
     }

    else

     {

PORTAbits.RA1 = 0;



      }

             if (P2==1)

       {

              PORTAbits.RA2 = 1;



                }

               else

                  {

                         PORTAbits.RA2 = 0;



                      }

    if (P3==1)

    {

      PORTAbits.RA3 = 1;
```

```
                }

        else

        {

           PORTAbits.RA3 = 0;



        }

        }




void ADC_Init(void)

{

/*  Configure ADC module  */







 /*----- Set the Registers below:: */

 /* 1. Set ADC CONTROL REGISTER 1 to 0 */



 ADCON1 = 0x00;
```

```
/* 2. Set ADC CONTROL REGISTER 2 to 0 */

ADCON2=0x00;

/* 3. Set ADC THRESHOLD REGISTER to 0 */

ADCON3 =0x00;

/* 4. Disable ADC auto conversion trigger control register */ /* 5. Disable ADACT */

ADACT =0x00;


/* 6. Clear ADAOV ACC or ADERR not Overflowed  related register */

ADSTAT =0x00;


/* 7. Disable ADC Capacitors */

ADCAP = 0x00;


/* 8. Set ADC Precharge time control to 0 */

ADPRE =0x00;


/* 9. Set ADC Clock */

ADCLK = 0x01;


/* 10 Set ADC positive and negative references*/
```

```
 ADREF = 0x00; // VREF+ is connected to VDD and VREF- is connected to analog ground//


 /* 11. ADC chanel - Analog Input */

 ANSELCbits.ANSC0 = 1;

 TRISCbits.TRISC0 = 1;

 PORTCbits.RC0 = 1;


 /* 12. Set ADC result alignment, Enable ADC module, Clock Selection Bit, Disable ADC
Continuous Operation, Keep ADC inactive*/


 ADCON0 = 0x84;          //ADON =1,ADCONT =0,ADCS =1, ADFRM0 = 0i.e right justified,
ADG0 = 0//


 //ADACQ = 0;

 TRISBbits.TRISB1= 0;

PORTBbits.RB1 = 0;

}


int ADC_read()

{
  unsigned int lightval =0;

  // Initialize PIC device

  SYSTEM_Initialize();



  // Initialize the required modules

  ADC_Init();





// **** write your code



  //setting the output port



  ADCON0bits.ADON = 1; //enable the ADC module

  //ADCON0bits.ADCONT=1;//      continuous sampling

        ADCON0bits.ADGO =1; //start the conversion

        /*-----------------------------------------------------------------------------------------*/

        while(ADCON0bits.ADGO==1);  //if the conversion has not been completed then wait



  lightval = ((ADRESH<<8)|ADRESL);

  ADCON0bits.ADON = 0;  //after getting the result of ADC conversion disable the ADC
```

```
   //printf("%d\n",lightval);

                    if(900<=lightval) //if the resistance of the photo resistor is less(light is more) , input
voltage to the analog channel  will be more

                    //1023= 3.3 v corresponds to 3.3v

                    {



                              PORTBbits.RB1 = 0;



                               //180 Degree

                    }

                    else

                    {



                              PORTBbits.RB1 =1;



           //90 Degree5

                    }

   ADCON0bits.ADON = 0;
```

```
    light =  lightval;



}

void reinit_out()

{

    ANSELA = 0x00;

    TRISA = 0x10;

    PORTA = 0x1F;


}

void init_out()

{

    ANSELA = 0x00;

    TRISA = 0x1F;

    PORTA = 0x1F;


}

unsigned char receive_msg()

{
		unsigned char data =0;
	unsigned char data1 =0;
	unsigned char data2 =0;
	unsigned char data3 =0;
	unsigned char data4 =0;


	/*1.wait strobe high*/
//		while(PORTAbits.RA4  != 1)
//		{}
		/*3.read the data */
	data1 = PORTAbits.RA3 << 3;
	data2 = PORTAbits.RA2 << 2;
	data3 = PORTAbits.RA1 << 1;
	data4 = PORTAbits.RA0;
	data = ( data1 | data2 | data3 | data4);


	/*2.wait strobe low */
 //  while(PORTAbits.RA4 != 0)
//		{}
	return data;
```

```
    /*5.return the data*/



}


// Main program

void main (void)

{

    static unsigned char msg = 0;

    int i =0;


    ANSELA = 0x00;

    TRISA = 0x1F;

    PORTA = 0x1F;

    TRISBbits.TRISB0  = 0;
```

```
PORTBbits.RB0 = 0x01;



   /*pinging the PIC*/

      while(PORTAbits.RA4  != 1); //wait till the strobe goes high

   __delay_ms(1500);

   init_out();

               msg = receive_msg();


   if(msg == 1)

   {

      TRISBbits.TRISB3 = 0;

      PORTBbits.RB3 = 1;

      reinit_out();

      while(PORTAbits.RA4  != 0); //wait till the strobe goes low

      //__delay_ms(500);

      PORTAbits.RA3 = 1;

      PORTAbits.RA2 = 1;

      PORTAbits.RA1 = 1;

      PORTAbits.RA0 = 0;
```

```
    __delay_ms(1000);

}

/*resetting the PIC*/

init_out();



while(PORTAbits.RA4  != 1); //wait till the strobe goes high

__delay_ms(1500);

//init_out();

                msg = receive_msg();



if(msg == 0)

{

    //PORTCbits.RC0 = 1;

    reinit_out();

    while(PORTAbits.RA4  != 0); //wait till the strobe goes low

    //__delay_ms(500);

    PORTAbits.RA3 = 1;

    PORTAbits.RA2 = 1;

    PORTAbits.RA1 = 1;

    PORTAbits.RA0 = 1;
    __delay_ms(3000);
}

 /*rotating the servo*/
init_out();

while(PORTAbits.RA4  != 1); //wait till the strobe goes high
__delay_ms(1500);
//init_out();
                msg = receive_msg();

if(msg == 3)
{
    //PORTCbits.RC0 = 1;
    reinit_out();
    while(PORTAbits.RA4  != 0); //wait till the strobe goes low
    __delay_ms(500);

    //__delay_ms(500);
    for(i=0;i<200;i++)   //rotating the servo for 4s to 90 degree
   {
      PORTBbits.RB0 = 1;
      __delay_us(1500);
      PORTBbits.RB0 = 0;
      __delay_us(18500);
    }



    PORTAbits.RA3 = 0;
    PORTAbits.RA2 = 1;
    PORTAbits.RA1 = 1;
    PORTAbits.RA0 = 0;

    __delay_ms(2000);
}



 /*calling the adc*/
init_out();
```

```
while(PORTAbits.RA4  != 1); //wait till the strobe goes high

__delay_ms(1500);

//init_out();

                msg = receive_msg();



if(msg == 2)

{

    TRISBbits.TRISB2 =0;

    PORTBbits.RB2 =1;

    //PORTCbits.RC0 = 1;

    reinit_out();





    int j = 0;



    for(j=0;j<10;j++)

    {

     ADC_read();

    }
```

```
 while(PORTAbits.RA4 != 0);

 set_and_shift();

 set_pins();



while(PORTAbits.RA4 != 1);

if(PORTAbits.RA4 == 1)

{}


 while(PORTAbits.RA4 != 0);

 set_and_shift();

 set_pins();


while(PORTAbits.RA4 != 1);

if(PORTAbits.RA4 == 1)

{}


 while(PORTAbits.RA4 != 0);
 set_and_shift();

 set_pins();


while(PORTAbits.RA4 != 1);

if(PORTAbits.RA4 == 1)

{}


 while(PORTAbits.RA4 != 0);


 TRISAbits.TRISA3 = 0;

 TRISAbits.TRISA2 = 0;

 TRISAbits.TRISA1 = 0;

 TRISAbits.TRISA0 = 0;

 //__delay_ms(500);

 PORTAbits.RA3 = 1;

 PORTAbits.RA2 = 1;

 PORTAbits.RA1 = 1;

 PORTAbits.RA0 = 0;
```

```
        __delay_ms(8000);


      }

    }
```

A3. Galileo code that uses scanf function to scan user input commands

```
#include <stdlib.h>

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#define MSG_RESET  0x00

#define MSG_PING   0x01

#define MSG_GET    0x02

#define MSG_TURN3  0x03

#define MSG_TURN9  0x04

#define MSG_TURN12 0x05

void initialise()

{
        char buffer[200];

        sprintf(buffer, "echo 40 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 48 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 50 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 52 > /sys/class/gpio/export");

        system(buffer);

        sprintf(buffer, "echo 54 > /sys/class/gpio/export");

        system(buffer);


}

void GPIO_reinit()

{

    char buf[200]={0};

    sprintf(buf, "echo in > /sys/class/gpio/gpio48/direction");

    system(buf);

    sprintf(buf, "echo in > /sys/class/gpio/gpio50/direction");

    system(buf);
    sprintf(buf, "echo in > /sys/class/gpio/gpio52/direction");

    system(buf);

    sprintf(buf, "echo in > /sys/class/gpio/gpio54/direction");

    system(buf);


}

void GPIO_init()

{



    char buf[200];









    sprintf(buf, "echo out > /sys/class/gpio/gpio48/direction");

    system(buf);
```

```
   sprintf(buf, "echo out > /sys/class/gpio/gpio50/direction");

   system(buf);




   sprintf(buf, "echo out > /sys/class/gpio/gpio52/direction");

   system(buf);




   sprintf(buf, "echo out > /sys/class/gpio/gpio54/direction");

   system(buf);


}


void writeGPIO(int command)

{

      int x = command;

      int y = 0;


      char buffer[200]={0};

      //sending first lsb bit of the command to gpio54
```

```
    y = x & 0x01;

    sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio54/value", y);

system(buffer);


    //sending second lsb bit of the command to gpio52

    x = x>>1;

    y = x & 0x01;


    sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio52/value", y);

system(buffer);


//sending third lsb bit of the command to gpio50

    x = x>>1;

    y = x & 0x01;


    sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio50/value", y);

system(buffer);


//sending fourth lsb bit (msb) of the command to gpio54

    x = x>>1;
```

```
        y = x & 0x01;

        sprintf(buffer,"echo \"%d\" > /sys/class/gpio/gpio48/value", y);

    system(buffer);

    return;


}

char readGPIO()

{

        char data = 0;

        char buff1[200]={0};

        char buff2[200]={0};

        char buff3[200]={0};

        char buff4[200]={0};

        int val1 = 0;

        int val2 =0;

        int val3 = 0;

        int val4 = 0;



        int fd1 = -1;
```

```
fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);

if(fd1!=-1)

{

        read(fd1,buff1,1);

        close(fd1);

}

if(buff1[0]=='0')

{

        val1 = 0;

}

else

{

        val1 = 1;

}



int fd2 = -1;

fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);

if(fd2!=-1)

{

        read(fd2,buff2,1);
        close(fd2);
}
if(buff2[0]=='0')
{
        val2 = 0;
}
else
{
        val2 = 1;
}
int fd3 = -1;
fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);
if(fd3!=-1)
{
        read(fd3,buff3,1);
        close(fd3);
}
if(buff3[0]=='0')
{
        val3 = 0;
```

```
}
else
{
        val3 = 1;
}


int fd4 = -1;
fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);
if(fd4!=-1)
{
        read(fd4,buff4,1);
        close(fd4);
}
if(buff4[0]=='0')
{
        val4 = 0;
}
else
{
        val4 = 1;
        }



        data= (val1<<3)|(val2<<2)|(val3<<1)|(val4);

        return data;
}
unsigned int readADC()
{
        unsigned int packet = 0;
        char buff1[200]={0};
        char buff2[200]={0};
        char buff3[200]={0};
        char buff4[200]={0};
        int val1 = 0;
        int val2 = 0;
        int val3 = 0;
        int val4 = 0;

        int fd1 = -1;
```

```
fd1= open("/sys/class/gpio/gpio48/value", O_RDONLY);
if(fd1!=-1)
{
	read(fd1,buff1,1);
	close(fd1);
}
if(buff1[0]=='0')
{
	val1 = 0;
}
else
{
	val1 = 1;
}


int fd2 = -1;
fd2= open("/sys/class/gpio/gpio50/value", O_RDONLY);
if(fd2!=-1)
{
	read(fd2,buff2,1);
	close(fd2);
}
if(buff2[0]=='0')
{
	val2 = 0;
}
else
{
	val2 = 1;
}
int fd3 = -1;
fd3 = open("/sys/class/gpio/gpio52/value", O_RDONLY);
if(fd3!=-1)
{
	read(fd3,buff3,1);
	close(fd3);
}
if(buff3[0]=='0')
{
	val3 = 0;
```

```
}
else
{
        val3 = 1;
}


int fd4 = -1;
fd4 = open("/sys/class/gpio/gpio54/value", O_RDONLY);
if(fd4!=-1)
{
        read(fd4,buff4,1);
        close(fd4);
}
if(buff4[0]=='0')
{
        val4 = 0;
}
else
{
        val4 = 1;
```

```
	}

	packet= (val4<<3)|(val3<<2)|(val2<<1)|(val1);
	return packet;
}

int main(void)
{
//	printf("hi\n");
	char buffer[200]={0};
	char reply =0 ;
	char command;
	unsigned int packet1 = 0;
	unsigned int packet2 = 0;
	unsigned int packet3 = 0;
	unsigned int packet4 =0 ;
	unsigned int ADC_value = 0;
	initialise();
```

```
sprintf(buffer, "echo out > /sys/class/gpio/gpio40/direction");

system(buffer);




printf("Enter 0 for RESET command\n");

printf("Enter 1 for PING Command\n");

printf("Enter 2 for ADC value read\n");

printf("Enter 3 for rotating servo to 30 degree\n");

printf("Enter 4 for rotating servo to 90 degree\n");

printf("Enter 5 for rotating servo to 120 degree\n");


scanf("%d",&command);


if(command == 0x01)

{


    GPIO_init();   //initialise the gpio pins as output and then send the command

    //1.Strobe high

    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                system(buffer);
```

```
        sleep(1);

        //2.write data

        writeGPIO(MSG_PING);   //ping the pic micro

        sleep(3);



        GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

        //3.Strobe low

        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                        system(buffer);

        sleep(1);



        //4.Strobe high



//        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //                     system(buffer);






                        //2.read data
```

```
       reply = readGPIO();

       printf("The reply is %d\n",reply);

       if(reply!= 0x0E)

       {

           printf("PIC is not responding\n");

                       }



                       else

                       {



                              printf("PIC is responding\n");

                              }



                              sleep(1);

                              //3.Strobe low

        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                        system(buffer);

}



else if(command == MSG_RESET)
```

```
  {

//              sleep(1);

          GPIO_init();   //initialise the gpio pins as output and then send the command

        //1.Strobe high

         sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                        system(buffer);


        sleep(1);

        //2.write data

        writeGPIO(0x00);   //reset the pic micro

        sleep(1);

        GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

        //3.Strobe low

        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                        system(buffer);

        sleep(1);


        //4.Strobe high
```

```
//        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //                        system(buffer);





                           //2.read data

       reply = readGPIO();

       printf("The reply is %d\n",reply);

       if(reply!= 0x0F)

       {

            printf("PIC is not reset\n");

                           }



                           else

                           {



                                  printf("PIC is reset\n");

                                  }



                                  sleep(1);
```

```
                                //3.Strobe low
          sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                         system(buffer);
//        sleep(1);
    }

 else if(command == 0x03)
          {

             GPIO_init();   //initialise the gpio pins as output and then send the command
        //1.Strobe high
        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                        system(buffer);

        sleep(1);
        //2.write data
        writeGPIO(MSG_TURN3);   //rotating  the servo to 30 degree
        sleep(5);
        GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
        //3.Strobe low
        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);
        sleep(5);


        //4.Strobe high


//       sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
 //                 system(buffer);





                    //2.read data
        reply = readGPIO();
        printf("The reply is %d\n",reply);
        if(reply!= 0x06)
        {
            printf("PIC is not rotating the servo to 30 degree\n");
                    }


                    else
                    {


                              printf("PIC is rotating the servo to 30 degree\n");

                              }



                              sleep(3);

                              //3.Strobe low

         sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                    system(buffer);



}

else if(command == 0x04)

          {



               GPIO_init();   //initialise the gpio pins as output and then send the command

      //1.Strobe high

      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                    system(buffer);



      sleep(1);
```

```
	//2.write data

	writeGPIO(MSG_TURN9);   //rotating  the servo to 90 degree

	sleep(5);

	GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

	//3.Strobe low

	sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

			system(buffer);

	sleep(5);


	//4.Strobe high


//	 sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //			system(buffer);




			//2.read data

	reply = readGPIO();

	printf("The reply is %d\n",reply);

	if(reply!= 0x07)
```

```
        {
             printf("PIC is not rotating the servo to 90 degree\n");
                         }

                         else
                         {

                                   printf("PIC is rotating the servo to 90 degree\n");
                                   }

                                   sleep(3);
                                   //3.Strobe low
          sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                          system(buffer);

}
else if(command == 0x05)
            {

                  GPIO_init();   //initialise the gpio pins as output and then send the command
        //1.Strobe high

        sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                    system(buffer);



        sleep(1);

        //2.write data

        writeGPIO(MSG_TURN12);   //rotating  the servo to 120 degree

        sleep(5);

        GPIO_reinit();   //initialise the gpio pins as input pins and then read the response

        //3.Strobe low

        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");

                    system(buffer);

        sleep(5);



        //4.Strobe high



//      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

 //                 system(buffer);
```

```
                    //2.read data
reply = readGPIO();
printf("The reply is %d\n",reply);
if(reply!= 0x08)
{
     printf("PIC is not rotating the servo to 120 degree\n");
                    }


                    else
                    {


                             printf("PIC is rotating the servo to 120 degree\n");
                             }


                             sleep(3);
                             //3.Strobe low
 sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                     system(buffer);
```

```
}


else if(command == 0x02)

{



    GPIO_init();   //initialise the gpio pins as output and then send the command

    //1.Strobe high

    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");

                    system(buffer);



    sleep(1);

    //2.write data

    writeGPIO(MSG_GET);   //reading  the adc

    sleep(5);

    GPIO_reinit();   //initialise the gpio pins as input pins and then read the response




    sleep(6);

    //3.Stirobe low
```

```
        sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                        system(buffer);



        //4.Strobe high


//      sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
 //                     system(buffer);




                        //2.read data
        packet1  = readADC();


        printf("the first packet is %d\n",packet1);
                                sleep(3);
                                //3.Strobe low
      //   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
      //                 system(buffer);
        packet4 = 14;
```

```
    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                    system(buffer);
      sleep(1);
      sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
      system(buffer);
                 sleep(2);
   packet2  = readADC();


  printf("the second  packet is %d\n",packet2);
  sleep(3);
                            //3.Strobe low
//   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
//                  system(buffer);


    sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                    system(buffer);
      sleep(1);
      sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
      system(buffer);
     sleep(2);
```

```
              packet3  = readADC();


  printf("the third  packet is %d\n",packet3);
  sleep(3);
                            //3.Strobe low
//   sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
//                    system(buffer);
        //1.Strobe high
  sprintf(buffer, "echo 1 > /sys/class/gpio/gpio40/value");
                    system(buffer);



  sleep(5);
  GPIO_reinit();   //initialise the gpio pins as input pins and then read the response
  //3.Strobe low
  sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
                    system(buffer);
  sleep(5);
```

```
                    //2.read data
        reply = readGPIO();


        printf("The reply is %d\n",reply);


        if(reply==0x0E)
        {
                printf("ADC value is read\n");
        }
        else
                 printf("Error");


        ADC_value = (packet3<<6) |(packet2<<4) |(packet1); //combine the three packets
        printf("The ADC value is %d\n", ADC_value);


          sleep(3);
          sprintf(buffer, "echo 0 > /sys/class/gpio/gpio40/value");
          system(buffer);


    }
    }
```

A4. PIC code for scanned user input commands

```
/*
 * File:   pic_com.c
 * Author: Family
 */







/*/*/
 /* File:   PIC and Galileo communication
 *
 *
 * simple PIC program example
 * for UMass Lowell 16.480/552

 *

 * Author: Ioann

 * Created on 2017/9/21

 */




#include "mcc_generated_files/mcc.h" //default library

#define MSG_RESET      0x0 /* reset the sensor to initial state */

#define MSG_PING       0x1 /*check if the sensor is working properly */

#define MSG_GET        0x2 /* obtain the most recent ADC result */

#define MSG_TURN3      0x3 /* turn the servo motor blade 30 degrees */

#define MSG_TURN9      0x4 /* turn the servo motor blade 90 degrees */

#define MSG_TURN12     0x5 /* turn the servo motor blade 120 degrees */


static unsigned int light =0 ;

char P0,P1,P2,P3,P4;

void set_and_shift()

{
```

```
            P0 = (light) & 1;              //lsb of the packet

    P1 = (light >> 1) & 1;

            P2 = (light >> 2) & 1;

            P3 = (light >> 3) & 1;

    light = light >> 4;

}


 void set_pins()

 {



     TRISAbits.TRISA3 = 0;

     TRISAbits.TRISA2 = 0;

     TRISAbits.TRISA1 = 0;

     TRISAbits.TRISA0 = 0;




      if (P0==1)

   {

       PORTAbits.RA0 = 1;
```

```
}

 else

 {

            PORTAbits.RA0 = 0;



 }

        if (P1==1)

  {

          PORTAbits.RA1 = 1;



           }

          else

           {

     PORTAbits.RA1 = 0;



            }

                    if (P2==1)

            {

                     PORTAbits.RA2 = 1;
```

```
                  }
                 else
                   {
                            PORTAbits.RA2 = 0;


                         }
          if (P3==1)
          {
            PORTAbits.RA3 = 1;


                  }
          else
          {
            PORTAbits.RA3 = 0;


          }
          }



void ADC_Init(void)
```

```
{

/*  Configure ADC module  */




 /*----- Set the Registers below:: */

 /* 1. Set ADC CONTROL REGISTER 1 to 0 */


 ADCON1 = 0x00;

 /* 2. Set ADC CONTROL REGISTER 2 to 0 */

 ADCON2=0x00;

 /* 3. Set ADC THRESHOLD REGISTER to 0 */

 ADCON3 =0x00;

 /* 4. Disable ADC auto conversion trigger control register */ /* 5. Disable ADACT */

 ADACT =0x00;


 /* 6. Clear ADAOV ACC or ADERR not Overflowed  related register */

 ADSTAT =0x00;


 /* 7. Disable ADC Capacitors */
```

```
ADCAP = 0x00;


/* 8. Set ADC Precharge time control to 0 */

ADPRE =0x00;


/* 9. Set ADC Clock */

ADCLK = 0x01;


/* 10 Set ADC positive and negative references*/

ADREF = 0x00; // VREF+ is connected to VDD and VREF- is connected to analog ground//



/* 11. ADC chanel - Analog Input */

ANSELCbits.ANSC0 = 1;

TRISCbits.TRISC0 = 1;

PORTCbits.RC0 = 1;


/* 12. Set ADC result alignment, Enable ADC module, Clock Selection Bit, Disable ADC
Continuous Operation, Keep ADC inactive*/
```

```
 ADCON0 = 0x84;          //ADON =1,ADCONT =0,ADCS =1, ADFRM0 = 0i.e right justified,
ADG0 = 0//



 //ADACQ = 0;

 TRISBbits.TRISB1= 0;

PORTBbits.RB1 = 0;

}



int ADC_read()

{

    unsigned int lightval =0;

    // Initialize PIC device

    SYSTEM_Initialize();



    // Initialize the required modules

    ADC_Init();





  // **** write your code
```

```
   //setting the output port



   ADCON0bits.ADON = 1; //enable the ADC module

   //ADCON0bits.ADCONT=1;//      continuous sampling

        ADCON0bits.ADGO =1; //start the conversion

        /*-----------------------------------------------------------------------------------------*/

        while(ADCON0bits.ADGO==1);  //if the conversion has not been completed then wait



   lightval = ((ADRESH<<8)|ADRESL);

   ADCON0bits.ADON = 0;  //after getting the result of ADC conversion disable the ADC

   //printf("%d\n",lightval);

                if(900<=lightval) //if the resistance of the photo resistor is less(light is more) , input
voltage to the analog channel  will be more

                //1023= 3.3 v corresponds to 3.3v

                {



                        PORTBbits.RB1 = 0;



                         //180 Degree

                }
```

```
                    else

                    {



                            PORTBbits.RB1 =1;



            //90 Degree5

                    }

   ADCON0bits.ADON = 0;




   light =  lightval;





}

void reinit_out()

{

   ANSELA = 0x00;

   TRISA = 0x10;

   PORTA = 0x1F;
```

```
}

void init_out()

{

   ANSELA = 0x00;

   TRISA = 0x1F;

   PORTA = 0x1F;


}

unsigned char receive_msg()

{

         unsigned char data =0;

   unsigned char data1 =0;

   unsigned char data2 =0;

   unsigned char data3 =0;

   unsigned char data4 =0;


   /*1.wait strobe high*/

 //      while(PORTAbits.RA4  != 1)

 //      {}

         /*3.read the data */
```

```
    data1 = PORTAbits.RA3 << 3;

    data2 = PORTAbits.RA2 << 2;

    data3 = PORTAbits.RA1 << 1;

    data4 = PORTAbits.RA0;

    data = ( data1 | data2 | data3 | data4);


    /*2.wait strobe low */

  //  while(PORTAbits.RA4 != 0)

 //         {}

    return data;






    /*5.return the data*/




}


// Main program
```

```
void main (void)

{

    static unsigned char msg = 0;

    int i =0;



    ANSELA = 0x00;

    TRISA = 0x1F;

    PORTA = 0x1F;

    TRISBbits.TRISB0  = 0;



    PORTBbits.RB0 = 0x01;







        /*pinging the PIC*/

            while(PORTAbits.RA4  != 1); //wait till the strobe goes high

        __delay_ms(1500);

        init_out();

                        msg = receive_msg();
```

```
if(msg == MSG_PING)    //if the command from the Galileo is PING

{

   TRISBbits.TRISB3 = 0;

   PORTBbits.RB3 = 1;

   reinit_out();

   while(PORTAbits.RA4  != 0); //wait till the strobe goes low

   //__delay_ms(500);

   PORTAbits.RA3 = 1;   //send the response MSG_ACK

   PORTAbits.RA2 = 1;

   PORTAbits.RA1 = 1;

   PORTAbits.RA0 = 0;

   __delay_ms(1000);

}



else if(msg == MSG_RESET)  //if the command from the Galileo is RESET

{

   //PORTCbits.RC0 = 1;

   reinit_out();

   while(PORTAbits.RA4  != 0); //wait till the strobe goes low
```

```
    //__delay_ms(500);

    PORTAbits.RA3 = 1;  //send the response MSG_NTHNG

    PORTAbits.RA2 = 1;

    PORTAbits.RA1 = 1;

    PORTAbits.RA0 = 1;

    __delay_ms(3000);

}




else if(msg == MSG_GET) //if the command from the Galileo is MSG_GET

{

    TRISBbits.TRISB2 =0;

    PORTBbits.RB2 =1;   //to check if the command is received or not

    //PORTCbits.RC0 = 1;

    reinit_out();



    int j = 0;
```

```
 for(j=0;j<10;j++)  //perform ADC_read
 {
  ADC_read();
 }

 while(PORTAbits.RA4 != 0);  //send the  first packet of last 4 bits when the strobe goes
low
 set_and_shift();
 set_pins();

while(PORTAbits.RA4 != 1);  //wait till strobe goes high
if(PORTAbits.RA4 == 1)
{}

 while(PORTAbits.RA4 != 0); //send the next 4 bits when the strobe goes low
 set_and_shift();
 set_pins();

while(PORTAbits.RA4 != 1);  //wait till strobe goes high
```

```
if(PORTAbits.RA4 == 1)

{}



 while(PORTAbits.RA4 != 0); //send the next 4 bits when the strobe goes low



 set_and_shift();

 set_pins();



while(PORTAbits.RA4 != 1);  //wait till the strobe goes high

if(PORTAbits.RA4 == 1)

{}

 TRISAbits.TRISA3 = 0;

 TRISAbits.TRISA2 = 0;

 TRISAbits.TRISA1 = 0;

 TRISAbits.TRISA0 = 0;

 while(PORTAbits.RA4 != 0); //wait till strobe goes low



 PORTAbits.RA3 = 1;  //send the response 0x0E

 PORTAbits.RA2 = 1;

 PORTAbits.RA1 = 1;
```

```
            PORTAbits.RA0 = 0;



            __delay_ms(2000);






        }



        else if(msg == MSG_TURN3)   //if the command from the Galileo is TURN_3 to roate servo
    to 30 degree

        {

            //PORTCbits.RC0 = 1;

            reinit_out();

            while(PORTAbits.RA4  != 0); //wait till the strobe goes low

            __delay_ms(500);



            //__delay_ms(500);



            for(i=0;i<200;i++)   //rotating the servo for 4s to 30 degree

           {
```

```
      PORTBbits.RB0 = 1;

      __delay_us(1000);

      PORTBbits.RB0 = 0;

      __delay_us(19000);

    }



    PORTAbits.RA3 = 0;  //send the response 0x06  to Galileo

    PORTAbits.RA2 = 1;

    PORTAbits.RA1 = 1;

    PORTAbits.RA0 = 0;


    __delay_ms(2000);

}




else if(msg == MSG_TURN9)  //if the command from the Galileo is turn servo to 90 degree

{

    //PORTCbits.RC0 = 1;
```

```
reinit_out();

while(PORTAbits.RA4  != 0); //wait till the strobe goes low

__delay_ms(500);



//__delay_ms(500);



for(i=0;i<200;i++)   //rotating the servo for 4s to 90 degree

{

  PORTBbits.RB0 = 1;

  __delay_us(1500);

  PORTBbits.RB0 = 0;

  __delay_us(18500);

 }




PORTAbits.RA3 = 0;  //send the response 0x07

PORTAbits.RA2 = 1;

PORTAbits.RA1 = 1;

PORTAbits.RA0 = 1;
```

```
    __delay_ms(2000);
}


else if(msg == MSG_TURN12)  //rotating the servo 120 degree
{
    //PORTCbits.RC0 = 1;
    reinit_out();
    while(PORTAbits.RA4  != 0); //wait till the strobe goes low
    __delay_ms(500);


    //__delay_ms(500);


    for(i=0;i<200;i++)   //rotating the servo for 4s to 90 degree
   {
      PORTBbits.RB0 = 1;
      __delay_us(1800);
      PORTBbits.RB0 = 0;
      __delay_us(18200);
    }
```

```
        PORTAbits.RA3 = 1;  //send the response 0x08

        PORTAbits.RA2 = 0;

        PORTAbits.RA1 = 0;

        PORTAbits.RA0 = 0;

        __delay_ms(2000);

    }

}
```

Appendix:4

1. http://www.microchip.com/
2. https://github.com/
3. https://software.intel.com/en-us/get-started-galileo
4. https://www.sparkfun.com/products/retired/13096
5. https://link.springer.com/chapter/10.1007%2F978-1-4302-6838-3_14