ADC Data Validation, Interprocessor Communication and PID Controller

Ifunanya Nnoka ,Harshitha Bura, Sushma Macha
Computer Engineering Department, College of Engineering
San Jose State University, San Jose, CA 94303
E-mail:, *ifunanya.nnoka@sjsu.edu*, harshitha.bura@sjsu.edu, *sushma.macha@sjsu.edu*

Abstract

The objective of this lab is to perform ADC data validation by giving a varying input voltage to the ADC in LPC1769 through a potentiometer and sending the converted digital values to FriendlyARM development kit, which has Samsung S3C6410 processor, using UART protocol. After the data is received in the FriendlyARM development kit, data validation is performed using FFT and finding the power spectral density. The next objective of this lab is to implement a PID controller by interfacing LSM303 magnetometer, accelerometer and Temperature sensor to LPC1758, obtaining data from the sensor and controlling a servo motor according to the frequency and duty cycle obtained from the PID program written for driving the vehicle to a desired location based on the sensor input.

1. Introduction

The Tiny6410 development board is based on Samsung S3C6410, a 32 – bit ARM 11 RISC processor which delivers up to 667MHz of processing performance. For serial communication S3C6410 supports SPI, I2C, I2S, and USB protocols. The LPC1758 is an ARM Cortex-M3 based microcontroller for embedded applications. It consumes less power and operates at CPU frequencies of up to 48MHz. For serial communication LPC1758 supports SPI, I2C, CAN and UART protocols. The ADC is interfaced to As LSM303 module supports I2C, the I2C protocol is used for serial communication. The potentiometer is connected to ADC0 channel 5 of LPC1758 (P1.31). The LSM module is interfaced to the I2C (P0.10 (SDA) and P0.11 (SCL)) of LPC1758. A servo motor is connected to the PWM1.2 output of LPC1758 (P2.1). LPC1758 and FriendlyARM development kit communicate through UART protocol. So the Rx and Tx pins of LPC1769 are connected to the Tx and Rx pins of the FriendlyARM development kit respectively and the block diagram of the system is shown in Figure 1. A user application program is written in the FriendlyARM development platform to receive data using the ttySAC1 UART driver and also perform validation on the data usng FFT and finding the power spectral density of the data.

Tiny6410 development board is connected to the PC using an USB – RS232 serial interface. The board runs Qtopia (QT Extended) operating system. User programs and Kernel modules can be developed and built on a PC and shifted to Tiny6410 using an SD card or pen drive to run and install respectively. LPC1758 (SJOne board) is connected to the PC via a USB cable.

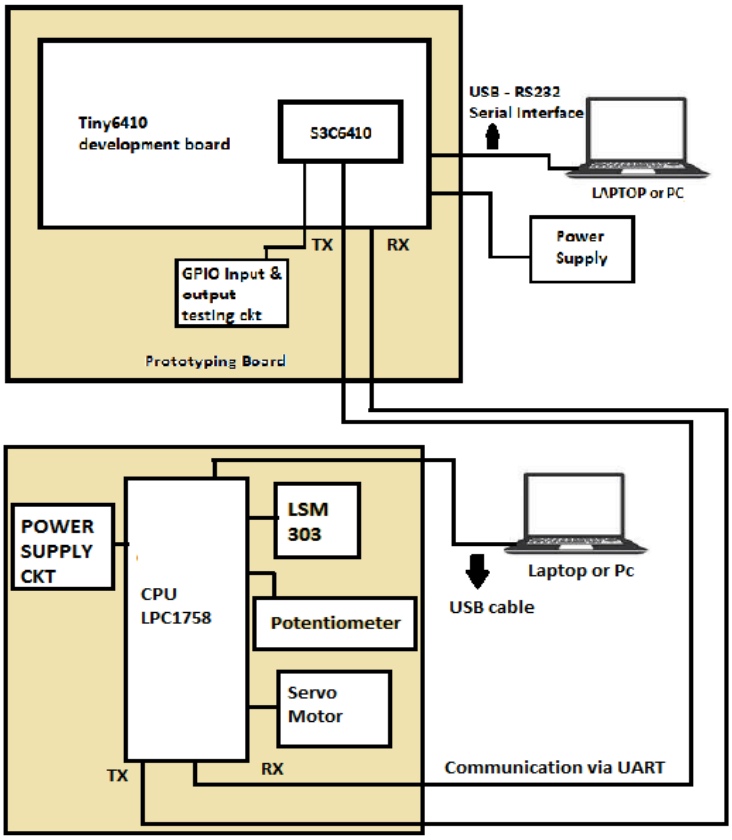


Figure 1.1 System Block Diagram

2. Methodology

The entire circuit is built and soldered on the prototyping board. The FriendlyARM development kit is connected to the PC via a USB – RS232 serial cable for accessing through Putty. Programs are written on the PC and compiled with the help of arm-linux-gcc compiler for FriendlyARM kit, then they are transferred to Tiny6410 with a USB stick and run with terminal commands sent serially from the PC. LPC1758 is compiled with the help of arm gcc compiler and Eclipse is the IDE used for development. It is connected to the PC using a USB cable and programmed with the help of Hyperload.

2.1. Design Objectives

The following are the design objectives

1. Fix the Tiny6410 development board and SJOne board which contains LPC1758 processor to the prototyping board. Solder the LSM303 module, servo and potentiometer onto the board.
2. Interface the LSM303, servo, potentiometer to LPC1758 by making appropriate connections.
3. Develop a program for configuring ADC, PWM and UART in LPC1758.
4. Develop a user application program to obtain the ADC data via UART protocol in Tiny6410 and also to perform validation.
5. Transfer the output file into the Tiny6410 and run it with the help of terminal commands to get the ADC data and perform validation.
6. Take input from the sensor and verify the functionality for the PID control program.

2.2. Hardware Requirements

The first thing to be done for the hardware is building a testing circuit and soldering LSM303, potentiometer and servo motor on to the board and making appropriate connections with LPC1758 and Tiny6410. The testing circuit built for Tiny6419 is shown in Figure 2. All the components required are shown in Table 1.

Description	Quantity
Tiny6410 Development Board	1
LSM303 module	1
Prototyping Board	1
Wires	As Required
Resistors for Testing circuit	2
Push Button switch for testing circuit	1
LEDs	1
USB – RS232 serial cable	1
LPC1758	1
Potentiometer	1
Servo motor	1
USB Cable	1

Table 2.1: Components required for building the circuit

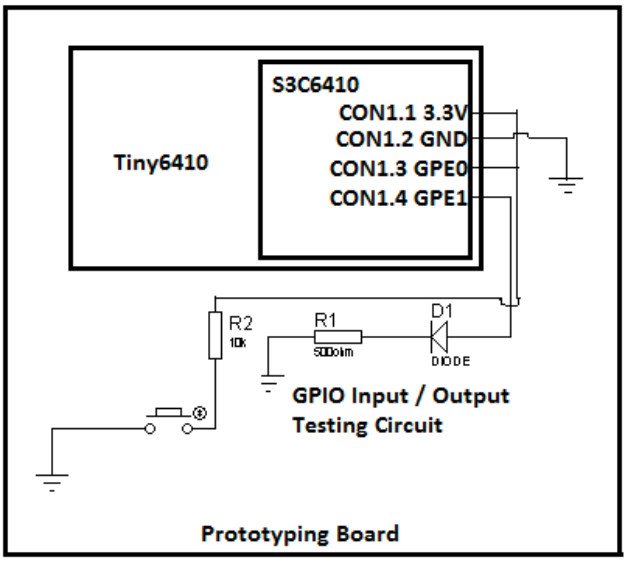


Figure 2.1 Testing Circuit on Tiny6410

2.2.1 LSM303 (Compass + Accelerometer)

The LSM303 breakout board combines a compass and a triple axis accelerometer. The I2C interface is compatible with both 3.3v and 5v processors and the two pins can be shared by other devices. The sensor consists of micro machined structures on a silicon wafer. These are structures designed to measure the acceleration and magnetic fields in X, Y and Z directions. The output of the sensor is n two's complement form.

Figure 2.2 LSM303 Module

2.2.2 Tower Pro Micro Servo Motor

Figure 2.3 Tower Pro Micro Servo

Tis servo motor is tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction). It can make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

2.3 Software Requirements

For Tiny6410, programing can be done using any text editor of our choice in Linux. The code is compiled with the help of arm-linux-gcc compiler. For this first we must install arm-linux-gcc. We also need an embedded Linux operating system for writing drivers and building kernel modules. For LPC1758 we need an ARM cross C compiler, Eclipse IDE, Hyperload for loading the code and Hercules for viewing the output from LPC1758. So the required software components are:

1. Arm Linux GCC
2. Embedded Linux 2.6.38
3. Qtopia OS installed into Tiny6410
4. ARM cross C compiler
5. Eclipse IDE
6. Hyperload
7. Hercules

3. Circuit for Interfacing the LSM303 Module, Servo, Potentiometer and connections for Interprocessor communication

The entire circuit is built on a prototyping board. It consists of the LSM303, Tiny6410, SJOne board, Servo motor, Potentiometer and testing circuit.. The Tiny6410 development board is connected to a Laptop using a USB – RS232 cable. The entire design and connections are shown in Figure 5. Table 2 gives a description of the connections. Figure 6 shows the prototyping board after soldering all the components. Figure 7 shows the connection between a laptop and Tiny6410 and SJOne board.

LSM303	LPC1758	Description
3V3	No Connection	3.3v supply
Vin	CON2.1 3.3v	Input Voltage
GND	CON2.2 GND	Ground
DRDY	No connection	Data Ready
I1	No connection	Interrupt 1
I2	No connection	Interrupt 2
SDA	P0.10 SDA	I2C Serial Data
SCL	P0.11 SCL	I2C Serial Clock
GND	J6-1 (GND)	GROUND

Table 3.1 Pin Connections between LPC1758 and LSM303

Tower Pro Micro	LPC1758	Description

Servo Motor		
Vin	3.3v	3.3v supply
GND	J6-1	Ground
PWM In	P2.1	PWM1.2

Table 3.2 Pin Connections between LPC1758 and Servo Motor

Tiny6410	LPC1758	Description
TXD1	RXD2	Transmit connected to Receive
RXD1	TXD2	Receive connected to Transmit
GND	GND	Ground

Table 3.3 Connections between LPC1758 and Tiny6410

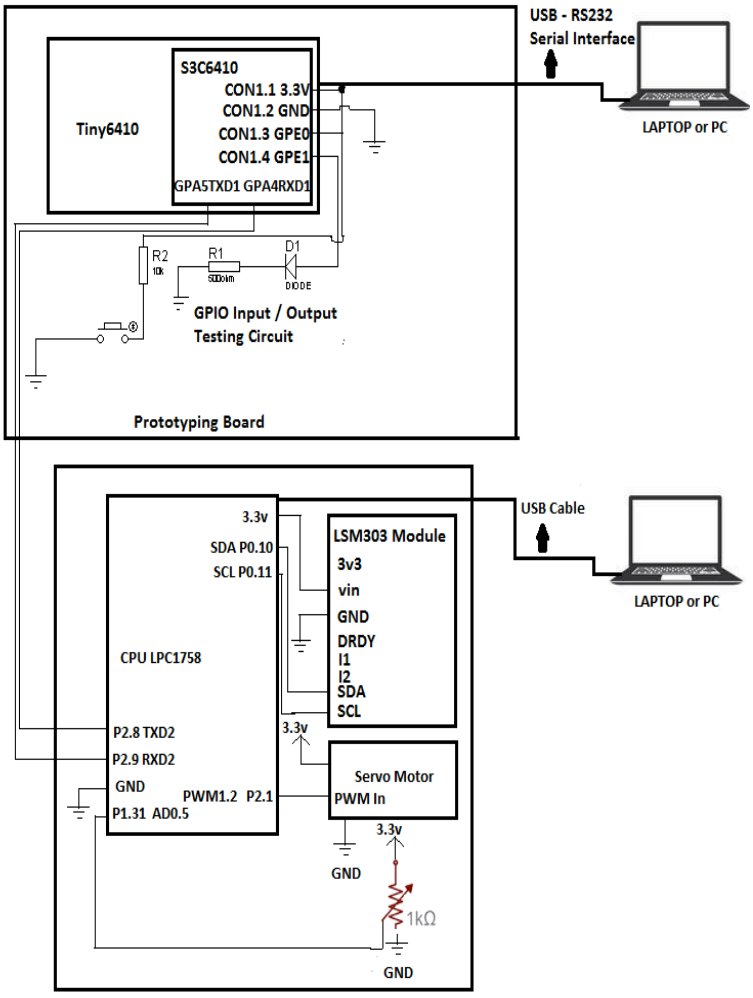


Figure 3.1 Circuit Diagram for the entire setup

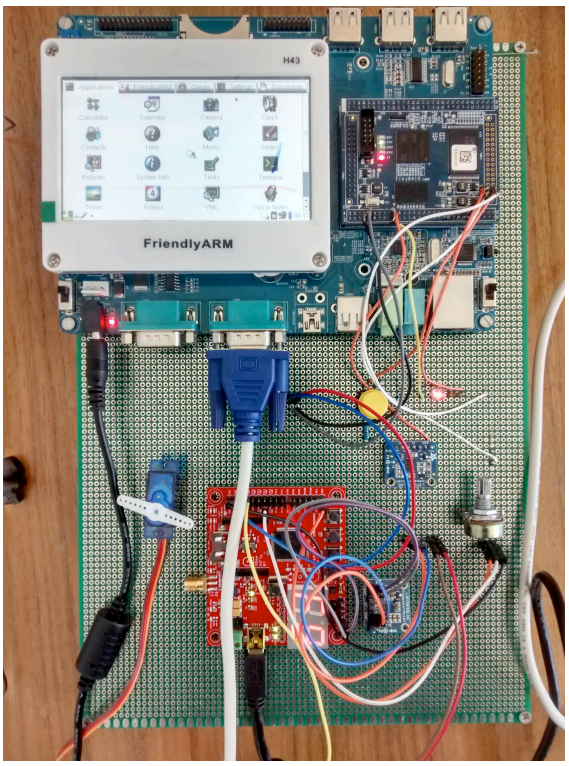


Figure 3.2 Setup (a)

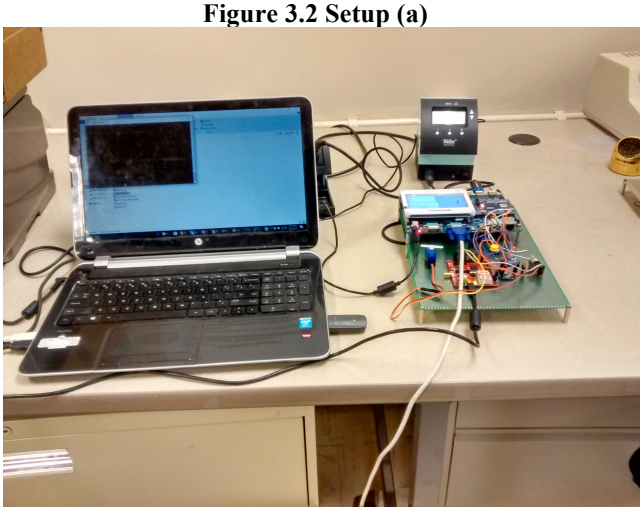


Figure 3.3 Setup (b)

4. Algorithm for the UART Sensor Reading

A. Initialize and configure the ADC and UART in LPC1758
B. Initialize the UART in Tiny6410.
C. Set the start "conversion bit" for the ADC
D. Continuously send values converted into digital form to Tiny6410.
E. In Tiny 6410 continuously read 10 values from LPC1758.

5. ADC Calibration

To calibrate the ADC, we take 10 data points corresponding to 10 analog inputs, from 0 to 3.3V where ΔV = 0.3V. We then find the characteristic curve and subsequently the compensation formula, this will then be integrated into our ADC program to compensated ADC data.

To find the characteristic curve equation we took 10 sample of digital data at 0V, and 3.3V. Next, we take the mean of each 10 samples and apply an interpolation equation to find the characteristic curve.

Interpolation:

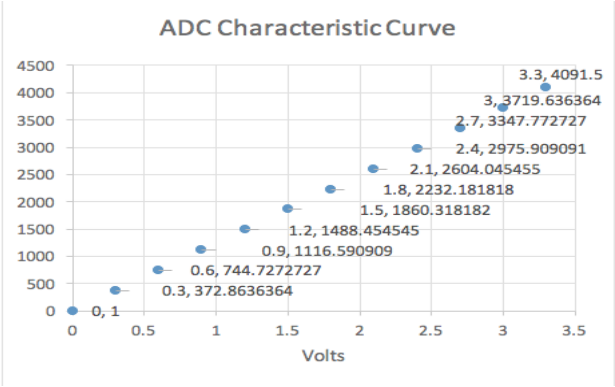
Interpolation Formula: $y = y1 + (x\text{-}x1)*\frac{y2-y1}{x2-x1}$

Excel Interpolation Formula:
NewY=FORECAST(*NewX*,OFFSET(*KnownY*, MATCH(*NewX*,*KnownX*,1)-1,0,2), OFFSET(*KnownX*,MATCH(*NewX*,*KnownX*,1)-1,0,2)

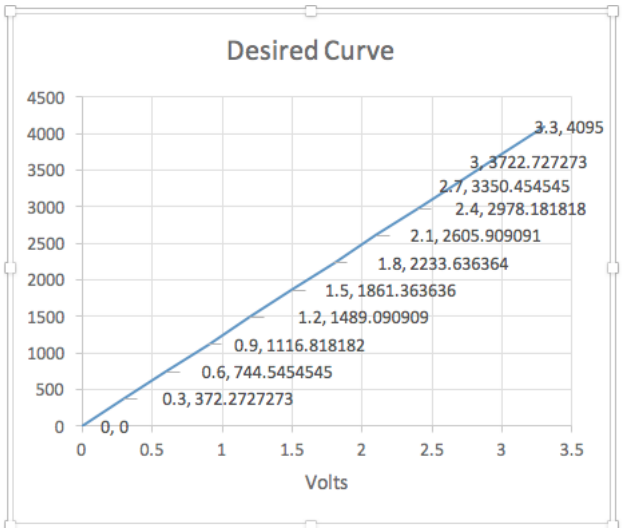
Characteristic Curve Data points

Volts	ADC value (12 bit resolution)
0	1
0.3	372.8636364
0.6	744.7272727
0.9	1116.590909
1.2	1488.454545
1.5	1860.318182
1.8	2232.181818
2.1	2604.045455
2.4	2975.909091
2.7	3347.772727
3.0	3719.636364
3.3	4091.5

Characteristic curve equation:
$f(x) = 1239.5x + 1$



Graph 1 ADC Characteristic Curve.



Graph 2 ADC Desired Curve.

Desired ADC curve equation:

$$Digital\ Val\ (y) = \frac{Resolution}{Vref} * AnalogVal(x)$$

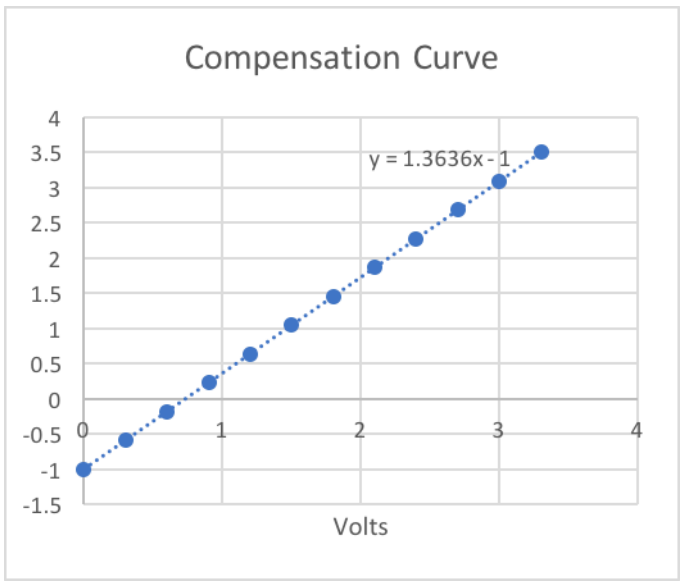
$$\therefore f(x)_{desired} = \frac{4095}{3.3} * x$$

<u>Compensation</u>:
The compensation curve can be derived by getting the difference between the ADC characteristic curve and the desired ADC curve.

$$\therefore f(x)_{comp} = f(x)_{desired} - f(x)$$

$$\therefore f(x)_{comp} = \left(\frac{4095}{3.3} * x\right) - (1239.5x + 1)$$

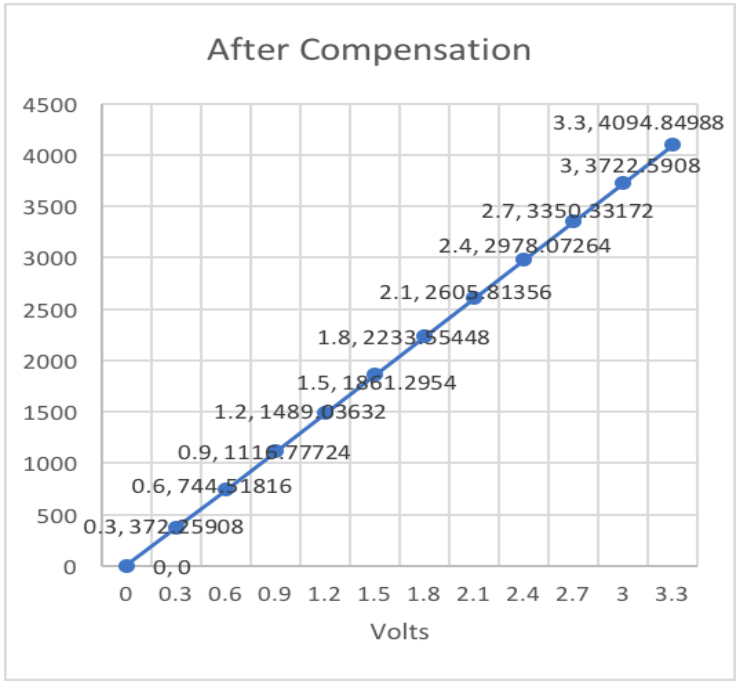
$$\therefore f(x)_{comp} = (1.3636x) - 1$$



Graph 3 ADC Compensation Curve.

$$ax = f(x) + f(x)_{comp}$$

$$\therefore ax = 1240.8636x$$



Graph 4 ADC After Compensation Curve.

Therefore, If DigitalVal[] is an array of digital values received from the LPCNOD, we compensate the data using the equation: $ax = f(x) + f(x)_{comp}$ after which we convert to volts.

6. Pseudo Code Data Compensation

```
float digital_comp = Digital[j]  + ((1.3636 * Volts[j]) -1);
float volt_comp = (Vref/Resolution) * digital_comp;
```

7. ADC Data validation

To Validate the ADC data, we first perform a D.F.T on the discrete data points, find the power spectrum and then to validate, we check to see if $f_{sampling} \geq 2f_{max}$.

Discrete Fourier Transform:
By definition, the formula for a D.F.T is:

$$X(m) = \frac{1}{N}\sum_{n=0}^{n=N-1} x(n)e^{-j2\pi\frac{mn}{N}}$$

where N = 10 (number of discrete value points)
m = 0,1,2,4,…..,N-1

Next we perform fft() on Volts[j] and store the values in an array $X(m)$.

Power Spectrum:
For data to be valid: $f_{sampling} \geq 2f_{max}$
To check if data is valid we need to get the power spectrum of $X(m)$ and then find the mean of the central positive band region, and if this mean is less than 5% of power spectrum at frequency index 0 (max power), then it is valid. If the data is not valid we adjust the sampling frequency of our ADC.

Formula for Power Spectrum:

$$P[m] = \sqrt[2]{(Re[X(m)^2] + Im[X(m)^2]}$$

$$Power\ Spectrum\ Density = \frac{no\ of\ ADC\ samples}{2} + 1$$

Therefore, if X[m] is an array of discrete values then:

```
P[m] = pow((X[m].rl),2)+pow((X[m].im),2);
```

Assuming 16 data samples, we find the mean of P[m] from P[6] to P[9].
If mean < 0.05(P[0]) then data is valid.

8. Algorithm for the ADC data validation

A. Read ADC data array into the fft discrete val array X.rl. Place zeros in all indexes of X.im
B. Perform FFT.
C. Find the power for all derived values in the freq. domain, for m = 0,…N-1.
D. Find Power spectrum density
E. Find the mean for central low power band region
F. If mean < 5% of P[0] then data is valid

9. Flow Chart for ADC Validation

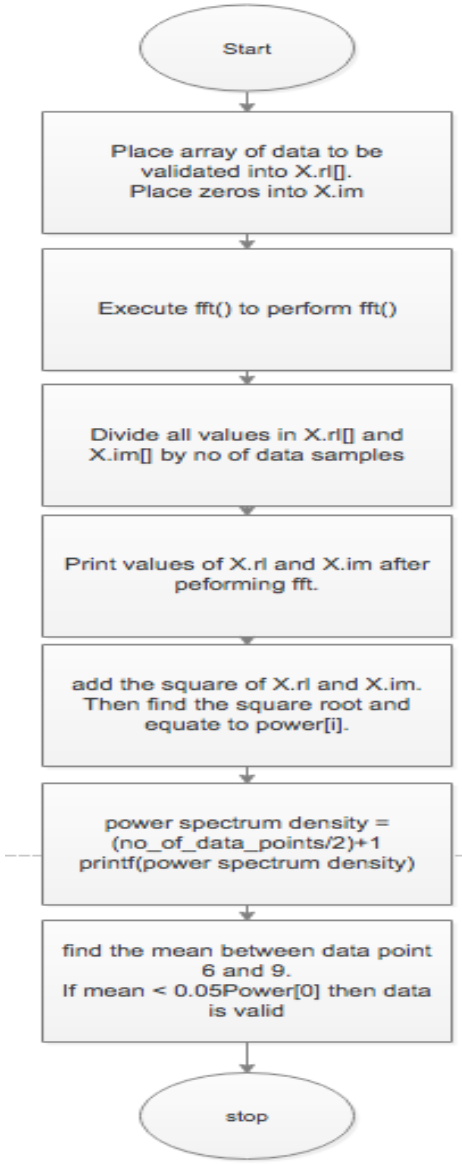


Figure 9.1 Flow Chart ADC data Validation

10. Pseudo Code ADC Validation

```
1.for(i=1; I < no_of_data_points_inarray_to_validate;i++)
      X[i].real_component = data[i];
      X[i].imaginary_component = 0.0;
}
2. for (i < no of data points)
print("x[%d]:%.2f\n",i,data[i]);//print values before fft
3. perform_fft();
4. for (i < no of data points)
      X[i].real = X[i].real/no_of_data_points;
      X[i].imaginary = X[i].imaginary/no_of_data_points;
}
5. for (i < no of data points)
print("X[i].real,X[i].imaginary);//print values after fft
6. for (i=1; i<=no of data points; i++) {
      Power[i]= sqrt((X[i].rl*X[i].rl)+(X[i].im*X[i].im));
      print(Power[i]);
   }
7. power spectrum density = (no of data points/2)+1;
   print("Power Spectrum");
8.   mean = 0;
   for (i between 4 and 11) {
      mean = mean +Power[i];
   }
   mean/= no_of_point_in_region;
   printf("\n");
   if (mean <(5% of P[1]))
      print("Data is valid ");
   else
      print("Data is not valid ");
```

11. Source Code ADC Validation

```
void validate_data (float *Volts)
{
   int psd;
   float P[17];
   float mean;
   int i;
   for (i=1; i<=ADC_NO_OF_SAMPLES; i++) {
      X[i].rl = Volts[i];
      X[i].im = 0.0;
   }
   printf("\n\nADC Values:\n");
   for (i=1; i<=ADC_NO_OF_SAMPLES; i++)
      printf("x[%d]:%.2f\n",i,X[i].rl);
   FFT();
   for (i=1; i<=ADC_NO_OF_SAMPLES; i++) {
      X[i].rl = X[i].rl/ADC_NO_OF_SAMPLES;
      X[i].im = X[i].im/ADC_NO_OF_SAMPLES;
   }
   printf("/n/nFFT Values:\n");
   for (i=1; i<=ADC_NO_OF_SAMPLES; i++)
         printf("X[%d]:real == %fimaginary == %f\n",i-
1,X[i].rl,X[i].im);

   //power spectrum
 printf("\n\n****************Power
Spectrum*****************\n");
   for (i=1; i<=ADC_NO_OF_SAMPLES; i++) {
      P[i] = pow((X[i].rl),2)+pow((X[i].im),2);
      //P[i]= sqrt(((X[i].rl*X[i].rl)+(X[i].im*X[i].im)));
      printf("P[%d]:%.2f\n",i-1,P[i]);
   }
   psd = (ADC_NO_OF_SAMPLES/2)+1;
   printf("Power Spectrum Density == %f\n",P[psd+1]);
   mean = 0;
   for (i = 4; i<=11; i++) {
      mean = mean +P[i];
   }
   mean/=8;
   printf("\n");
   if (mean <(0.05*P[1])) {
      printf("Data is valid.\n");
   }
   else
   {
      printf("Data is not valid.\n");
   }
}
```

12. Space time diagram for sending one byte of data to LSM303 using I2C protocol

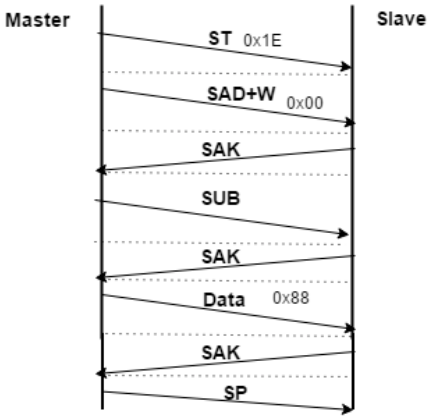


Figure 12.1 Space Time Diagram

ST – Start
SAD + W –Sub address + Write
SAK – Slave Acknowledge
SUB – Incremented sub Address
SP – Stop

13. Flow Chart for Communicating with the LSM303 and reading the sensor data.

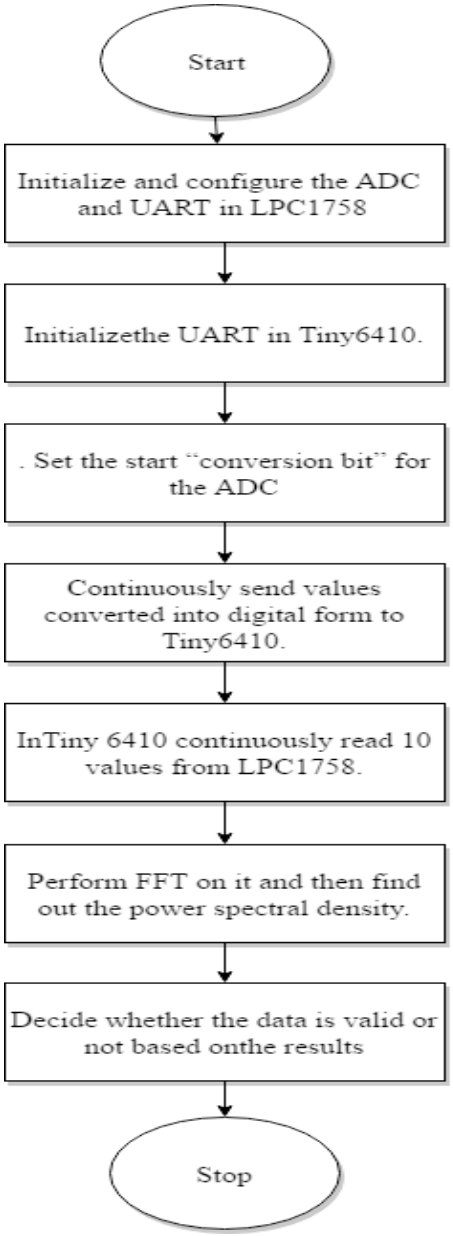


Figure 13.1 Flow Chart for Interprocessor Communication.

14. Theory behind the PID Control

A PID controller is used to calculate and correct error continuously. It is a control system with feedback mechanism i.e., it continuously generates a feedback based on the error which is used to correct the error.

A PID controller is used to control error in real time in systems where input and output is changing every instant and so is the error. To correct error in continuous time we need mechanisms like PID

A PID controller consists of three components

Proportional Component : It is a component that is directly proportional to the error i.e., it is used reduce the error in a magnitude specified by the user

Integral Component: This is used to calculate the previous errors and correct them along with reducing the error at the current instant. It is very efficient when there is a constant error. The error will add up with time and it can be corrected even after the process variable reaches the assumed set point

Derivative Component: The derivative component is used to reduce the slope of the magnitude of change in error with respect to time. It is helpful in avoiding the overshoot and helps in stabilizing the error to follow a smooth curve

A PID controller can be designed such that it can be a combination of any two combinations instead of all the three components depending on the requirements, but PID is the most effective way of controlling and reducing the error

15. Algorithm for the PID Control

A. Assume a set displacement the vehicle needs to make

B. The current location of the vehicle will be captured from the coordinates sent by LSM303 Magnetometer sensor.

C. These coordinates will provide the current angle of the vehicle

D. The Error is the difference between the displacement made by the vehicle and the expected displacement

E. The SumError is the sum of Integral, Derivative and the proportional components of the error. The SumError is assumed to be directly proportional to the angle in which the motor should rotate

F. The PWM driver is configured with the frequency of 50 Hz and duty cycle equal to the the angle necessary to correct the sensor to move in a given direction

G. The displacement of the vehicle is calculated based on the speed and the angle

H. Hence the sensor is rotated in such a way that it corrects its angle in every iteration till the error reaches zero

16. Flow Chart for the PID Control

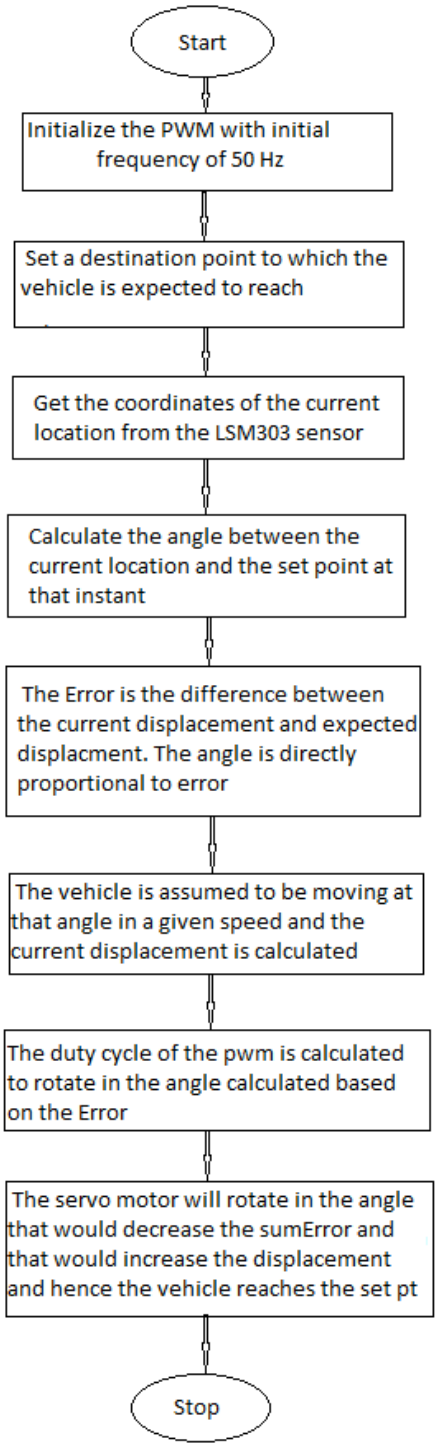


Figure 16.1 Flow Chart the PID Controller

17. Individual Contributions

Harshitha Bura -- Initialized and configured **UART** and **ADC** on LPC1758 to send ADC values, Initialized and configured **I2C** on LPC Node to receive coordinates from LSM303. Wrote Application program to perform 2's complement on the raw data to receive the coordinates

Sushma Macha – **PID** Control based on the LSM303 sensor values, **PWM** initialization and configuration, Servo Motor configuration. Wrote **Application program** on ARM side to receive the UART data.

Ifunanya Nnoka – Used potentiometer to send varying data and used it as input to ADC peripheral in LPC Node. Wrote application program to perform **ADC validation** on the continuously varying data using Fourier Transforms and by analyzing the power spectrum, Performed **Interpolation** and **compensation** on the received ADC values.

18. Pseudo Code

18.1 ADC_PID.c

Function: Initialize_ADC
Parameters: None
Return values:None

```
LPC_PINCON->PINSEL3 |= (3<<30);
//Setting p0.31 to act as an ADC pin
```

Function: Initialize_UART
Parameters: None
Return values:None

```
LPC_PINCON->PINSEL4 &=~ (3<<16)|(3<<18);
//Set P2.8 as TxD2
LPC_PINCON->PINSEL4   |=   (2<<16)|(2<<18);
//Set P2.9 as RxD2
```

Function: Initialize_PWM
Parameters: None
Return values:None

```
LPC_PINCON->PINSEL4 |= (1<<2); // P2.1
setting value to 01 for PWM1.2
```

Function: Initialize_UART
Parameters: None
Return values:None

```
LPC_PINCON->PINSEL4 &=~ (3<<16)|(3<<18);
//Set P2.8 as TxD2
LPC_PINCON->PINSEL4 |= (2<<16)|(2<<18);
//Set P2.9 as RxD2
```

Function:PID
Parameters: None
Return Values:None
If n=0

```
        Current_location=destination_location+x
        Error[n] = atan((destloc[1]-
        curloc[1])/(destloc[0]-curloc[0]))*180/PI;
i=n
for j=n to j>=n-3
IError[n]=IError[n]+(Error[j]*Error[j]);
```

```
BDError[n]=BDError[n]+(Error[n]-Error[n-
1]);
sumError[n]=
Kp*alpha1*Error[n]+Ki*alpha2*IError[n]+Kd*
alpha3*BDError[n];
n_pwm[n] = (sumError[n]/40)+50;
Ang[n] =
atan((destloc[1]curloc[1])/(destloc[0]-
curloc[0]))*(180/PI);
disVe[n]
=(1/5)*n_pwm[n]*2*PI*1*sin(Ang[n])*(0.33);

      //current displacement at 3 samples
per second
if(n-1 >= 0)
    disVe[n] = disVe[n]+disVe[n-1];
n_pwm[n] = 250;
msTcMax = 48000000/(n_pwm[n]);
prcnt = Ang[n]/180*100;
valueNeeded = (prcnt * msTcMax) / 100;
LPC_PWM1->LER |= (1 << 2);

      LPC_PWM1->MR2 = valueNeeded;
Error[n+1] = (Ang[n]);
```

Function: ADC
Parameters: None
Return Values: none

```
uint16_t output = (LPC_ADC->ADDR5>>4)&(0xfff);
UARTSend(send, sizeof(send));
```

Function: I2C Sensor
Parameters: None
Parameters: None

```
I2C2::getInstance().writeReg(0x3C, 0x00,
0x08); //Write 0x08 into register 00h to
set to 3Hz

I2C2::getInstance().writeReg(0x3C, 0x02,
0x00); //write 0x00 into register 02h to
set continuous conversion
I2C2::getInstance().readRegisters(Addr,
0x03, &x[0], count);

      I2C2::getInstance().readRegisters(Ad
dr, 0x04, &x[1], count);

      X=(0xFFFF^((x[0]<<8|x[1])))+1;
                  printf("x:%d\n",X);

      I2C2::getInstance().readRegisters(Ad
dr, 0x05, &y[0], count);

      I2C2::getInstance().readRegisters(Ad
dr, 0x06, &y[1], count);

      Y=(0xFFFF^((y[0]<<8|y[1])))+1;
                  printf("y:%d\n",Y);

      I2C2::getInstance().readRegisters(Ad
dr, 0x07, &z[0], count);

      I2C2::getInstance().readRegisters(Ad
dr, 0x08, &z[1], count);

      Z=(0xFFFF^((z[0]<<8|z[1])))+1;
            printf("z:%d\n",Z);
            location[0] = X;
            location[1] = Y;
            location[2] = Z;
//**************************************
**************************************//
Adc_validation.c
```

18.2 adc_validation.c

```
fd= open("/dev/ttySAC1", 0);
read(fd, &buffer,1);
int voltage=atoi(buffer2);
                  Voltage[j]=voltage;
                  Volts[j] =
(Vref/Resolution)*Voltage[j];
for (i=1; to ADC_NO_OF_SAMPLES;
{
        X[i].rl = Volts[i];
        X[i].im = 0.0;
}
        FFT();
for (i=1; to ADC_NO_OF_SAMPLES) {
X[i].rl =X[i].rl/ADC_NO_OF_SAMPLES;
X[i].im =X[i].im/ADC_NO_OF_SAMPLES;
}
for i=1; to ADC_NO_OF_SAMPLES;
P[i]=
sqrt(((X[i].rl*X[i].rl)+(X[i].im*X[
i].im)));
psd = (ADC_NO_OF_SAMPLES/2)+1;
for i=4; to 11
mean = mean +P[i];
mean/=8;
if mean is less than(0.05*P[1])
print(Data is valid)
else
print(Data is not valid)
```

19. Testing and Verification

All the connections on the board are checked with the help of a digital multi meter which is set to be in the connectivity mode.

The GPIO testing program is run for testing. The program for interprocessor communication and ADC data validation, and the program for the PID control have been run and checked. The outcome of running these programs is presented below.

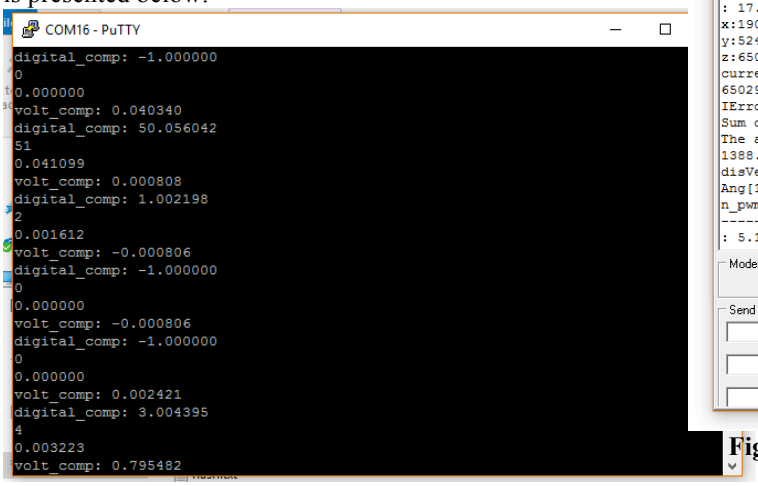


Figure 19.1 Data received on Tiny6410 from LPC1758

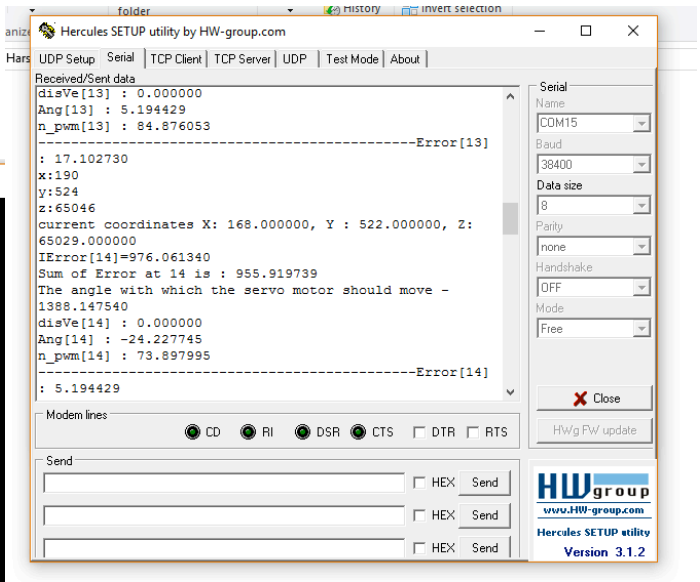


Figure 19.4 Error values from the PID program

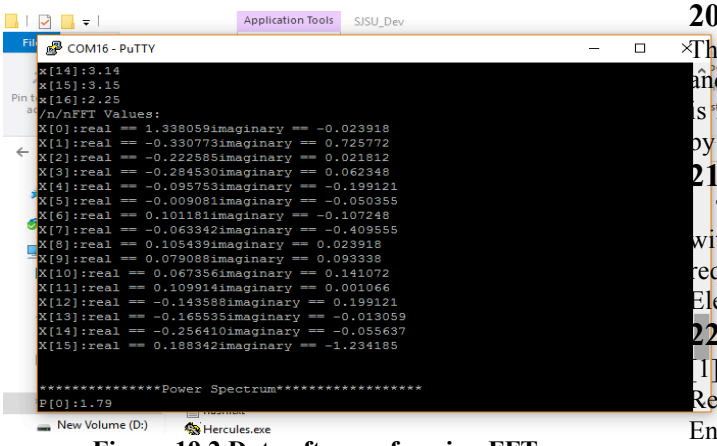


Figure 19.2 Data after performing FFT

Figure 19.3 Data is Valid

20. Conclusion

The ADC data sent from LPC1758 was received properly and it was valid. The program written for the PID control is running as desired and correcting the error in the path by taking input from the LSM303 sensor.

21. Acknowledgements

The work described in this paper has been achieved with the help of Dr. Harry Li. All the components required for the design were acquired from HSC Electronics and Amazon.

22. References

[1] H. Li, "Author Guidelines for CMPE 146/242 Project Report", *Lecture Notes of CMPE 146/242*, Computer Engineering Department, College of Engineering, San Jose State University, March 6, 2006, pp. 1.

[2] S3C6410 Datasheet. http://www.datasheetspdf.com/PDF/S3C6410/632072/1

[3] https://www.adafruit.com/product/1120

[4] http://www.micropik.com/PDF/SG90Servo.pdf

[5]https://github.com/hualili/CMPE242-Embedded-Systems-/blob/master/S3C6410X.pdf

23. Appendix

[A] Install ARM Linux GCC

A. Arm Linux GCC must be downloaded and installed in the Linux development environment.

B. Embedded Linux -2.6.38 must be downloaded and installed for writhing drivers and building kernel modules.

C. Eclipse with Arm cross compiler must be installed for LPC1758.

The detailed steps to do A and B can be found at https://github.com/dhruvkakadiya/mini6410_Samsung

[B] ADC_PID.c

```
#include "tasks.hpp"
#include "examples/examples.hpp"
#include <stdio.h>
#include <utilities.h>
#include <io.hpp>
#include "lpc_sys.h"
#include "lpc17xx.h"
#include "uart0_min.h"
#include "semphr.h"
#include <string.h>
#include <algorithm>
#include <storage.hpp>
#include <stdio.h>
#include <stdlib.h>
//#include <unistd.h>
#include <math.h>
#include <stdint.h>
//#include <pid.h>
#include <queue.h>
#include <printf_lib.h>

int once = 1;
#define BUFSIZE 77

#define PI 3.14159265

#define number_IError    4     // number of integration terms for IError[n]
#define Kp               30   // gain proportional
#define Kd               10    // gain derivative
#define Ki               10    // gain integral
#define alpha1           1  // weighting for kp
#define alpha2           1  // weighting for ki
#define alpha3           1  // weighting for kd
#define minError         0// min error limit
#define maxError         180 // max error limit
#define itrtns                   20
float Error[itrtns+1];           // for error
float DError[itrtns];          // derivative of error
float FDError[itrtns];         // forward difference computation for the derivative of error
float BDError[itrtns+1];         // backward difference computation for the derivative of error
float CDError[itrtns];         // central difference computation for the derivative of error
float IError[itrtns];          // integral of error (squared each individual error)
float sumError[itrtns];        // summation of pid errors
int n = 0;
float n_pwm[itrtns];
float contl[itrtns];
float Ang[itrtns];
float disVe[itrtns];
float SpeedVe;
float h[2]={1,-1};
int i,j;
uint8_t UART2Buffer[BUFSIZE];
uint32_t UART2Count=0;
uint32_t UART2TxEmpty=1;
uint32_t msTcMax;
uint16_t percent;
uint16_t valueNeeded;
int prcnt;
float location[3];

QueueHandle_t coordinates = xQueueCreate(1,sizeof(location));

bool pwm_init(void)
        {
              u0_dbg_printf("enter init\n");

                    LPC_SC->PCONP |= (1 << 6); //PWM1 power/clock control bit
                    msTcMax = 960000;
                    //Peripheral clock selection for PWM1 12 and 13 bit
                    LPC_SC->PCLKSEL0 &= ~(3 <<12); // Clear clock Bits
                    LPC_SC->PCLKSEL0 |= (1 << 12); // CCLK / 1

                    //Enable Counters,PWM module
                    LPC_PWM1->MCR |= (1<<1);//Reset on PWMMR0, reset TC if it matches MR0
                    LPC_PWM1->MR0 = msTcMax;//set PWM
```

```
cycle(Ton+Toff)=100)(1400microseconds),
Reset on PWMMR0: the PWMTC will be reset
if PWMMR0 matches it.
                    //uint16_t valueNeeded
=   0.1*msTcMax; // set pulse to 10% of
total time period
                    LPC_PWM1->TCR =
(1<<0);
                    LPC_PWM1->CTCR &=
~(0xF << 0); //the TC is incremented when
the Prescale Counter matches the Prescale
Register

                    //using PWM1.2, GPIO
PIN - 2.1
                LPC_PINCON->PINSEL4 &=
~(3 <<2); // P2.1 clearing to 00 value
                    LPC_PINCON->PINSEL4 |=
(1<<2); // P2.1 setting value to 01 for
PWM1.2
                    LPC_PWM1->PCR
|=(1<<10);//The PWM2 output enabled.
                    u0_dbg_printf("exit
init\n");
            return true;
        }

class PID : public scheduler_task
{
    public:
        PID(uint8_t priority) :
scheduler_task("PID", 20000, priority)
        {
            /* Nothing to init */
        }
        bool init(void)
        {
             return true;
        }
        bool run(void *p)
        {
             float curloc[3],
destloc[3],setpoint[3];

             while(n<=itrtns)
                    {


      xQueueReceive(coordinates,curloc,por
tMAX_DELAY);

      u0_dbg_printf("\ncurrent coordinates
X: %f, Y : %f, Z:
%f\n",curloc[0],curloc[1],curloc[2]);

                    if(n == 0)
                    {
                           pwm_init();
                           contl[n] = 0;
                           Error[n] = 1;
                           Ang[n] =
atan(curloc[1]/curloc[0])*(180/PI);

                    }

                    i=n;

                    IError[n]=0;
                    for(j=n;j>=n-3;j--)
                    {
                           if(j>0)

      IError[n]=IError[n]+(Error[j]*Error[
j]);

                    }

      u0_dbg_printf("IError[%d]=%f\n",n,IE
rror[n]);

      for(j=n+1;j<=itrtns;j++)
                    {
                           Error[j]=0;
                    }

                    for(j=2;j<=itrtns-
1;j++)
                    {
                           h[j]=0;
                    }
                    BDError[n]=0;
                    if(n>0)

      BDError[n]=BDError[n]+(Error[n]-
Error[n-1]);
                    sumError[n] =
Kp*alpha1*Error[n]+Ki*alpha2*IError[n]+Kd*
alpha3*BDError[n];
                    u0_dbg_printf("Sum of
Error at %d is : %f\n",n,sumError[n]);

                    n_pwm[n] = 50;

                                  //linear
equation from (50,0) and (200,6000)

                    Angc =
(sumError[n]/10)*(PI/180);
                                  //current
angle
```

```
	if(atan(curloc[1]/curloc[0])*(180/PI
)+Angc < 90)
			Ang[n] = Angc;

				//change
in the angle needed to get the output
			u0_dbg_printf("The
angle with which the servo motor should
move %f\n",Ang[n]);

			disVe[n] =
1.38*sin(Ang[n])*(0.33);
				//current
displacement at 3 samples per second
			if(n-1 >= 0)
			disVe[n] =
disVe[n]+disVe[n-1];

	u0_dbg_printf("disVe[%d] : %f\n", n,
disVe[n]);

	//u0_dbg_printf("contl[%d] : %f\n",
n, contl[n]);
			u0_dbg_printf("Ang[%d]
: %f\n", n, Ang[n]);

	u0_dbg_printf("n_pwm[%d] : %f\n", n,
n_pwm[n]);
			if(n_pwm[n]>250)
				n_pwm[n] = 250;
			msTcMax =
48000000/(n_pwm[n]);
			prcnt =
(Ang[n]*(180/PI))/180*100;
				//max
angle of servo motor is 180

			/*Load the duty cycle
of servo motor with the required angle*/
			{

	valueNeeded = (prcnt*10 * msTcMax) /
100;

	LPC_PWM1->LER |= (1 << 2);

	LPC_PWM1->MR2 = valueNeeded;
			}

			if(n < itrtns)
			{
				if(n>0)
				Error[n+1] = 1-
disVe[n];
			}
			u0_dbg_printf("-------
----------------------------------------
Error[%d] : %f\n", n, Error[n]);
			vTaskDelay(1000);
			n++;

		}



		return true;
	}
};

class UART2 : public scheduler_task
{
    public:
	UART2(uint8_t priority) :
scheduler_task("UART", 2000, priority)
	{
	    /* Nothing to init */
	}
	bool init(void)
	{
		LPC_SC->PCONP|=(1<<24);
//Activate UART2 in PCONP register
		LPC_SC->PCLKSEL1 &=~(3<<16);
//Reset the clock
		LPC_SC->PCLKSEL1 |= (1<<16);
//Set clock as 48MHz
		LPC_UART2->LCR = (1<<7);
//Set DLAB bit
		LPC_UART2->LCR |= (3<<0);
//Set 8-bit character length
		LPC_UART2->DLM=0;
//Set divisor latch MSB as 0
		LPC_UART2->DLL =
(sys_get_cpu_clock() / (16 * 115200))-
0.041666667;		//set divisor
latch LSB
		LPC_UART2->LCR &= ~(1<<7);
//Reset the DLAB bit
		LPC_UART2->FCR |=(1<<0);
//Enable the FIFO
		LPC_PINCON->PINSEL4 &=~
(3<<16)|(3<<18);
//Set P2.8 as TxD2
		LPC_PINCON->PINSEL4 |=
(2<<16)|(2<<18);
//Set P2.9 as RxD2
		LPC_UART2-
>FCR|=(1<<1)|(1<<2);
//Clear all bytes in Rx and Tx FIFO
```

```
            NVIC_EnableIRQ(UART2_IRQn);
//Enable IRQ
            LPC_UART2->IER |=
(1<<0)|(1<<1)|(1<<2);
//Enable RBR, THRE, and Rx Line Status
interrupts
           return true;
        }
        bool run(void *p)
        {
             return true;
        }
};
class Motor_PWM : public scheduler_task
{
    public:
        Motor_PWM(uint8_t priority) :
scheduler_task("Motor_PWM", 2000,
priority)
        {
            /* Nothing to init */
        }
        bool init(void)
        {
             printf("enter init\n");
                   //msTcMax = 2400000;//
for 50Hz frequency assuming a 120MHz clock
                   LPC_SC->PCONP |= (1 <<
6); //PWM1 power/clock control bit

                   //Peripheral clock
selection for PWM1 12 and 13 bit
                   LPC_SC->PCLKSEL0 &=
~(3 <<12); // Clear clock Bits
                   LPC_SC->PCLKSEL0 |=
(1 << 12); // CCLK / 1

                   //Enable Counters,PWM
module
                   LPC_PWM1->MCR |=
(1<<1);//Reset on PWMMR0, reset TC if it
matches MR0
                   LPC_PWM1->MR0 =
msTcMax;//set PWM
cycle(Ton+Toff)=100)(1400microseconds),
Reset on PWMMR0: the PWMTC will be reset
if PWMMR0 matches it.
                   //uint16_t valueNeeded
=   0.1*msTcMax; // set pulse to 10% of
total time period
                   LPC_PWM1->TCR =
(1<<0);
                   LPC_PWM1->CTCR &=
~(0xF << 0); //the TC is incremented when
the Prescale Counter matches the Prescale
Register

                   //using PWM1.2, GPIO
PIN - 2.1
                    LPC_PINCON->PINSEL4
&= ~(3 <<2); // P2.1 clearing to 00 value
                    LPC_PINCON->PINSEL4
|= (1<<2); // P2.1 setting value to 01 for
PWM1.2
                    LPC_PWM1->PCR
|=(1<<10);//The PWM2 output enabled.
                    printf("exit
init\n");
              return true;
        }
        bool run(void *p)
        {
             u0_dbg_printf("enter run\n");
                   valueNeeded = (5 *
msTcMax) / 100; //for 1 ms
                   //valueNeeded = (10 *
msTcMax) / 100; //for 2 ms
                   LPC_PWM1->LER |= (1 <<
2);
                   LPC_PWM1->MR2 =
valueNeeded;
                   vTaskDelay(330);
                   valueNeeded = (50 *
msTcMax) / 100; //for 2 ms
                   LPC_PWM1->LER |= (1 <<
2);
                   LPC_PWM1->MR2 =
valueNeeded;
                   vTaskDelay(330);
                   u0_dbg_printf("exit
run\n");
             return true;
        }
};
class I2C_Sensor : public scheduler_task
{
    public:
        I2C_Sensor(uint8_t priority) :
scheduler_task("I2C_Sensor", 2000,
priority)
        {
            /* Nothing to init */
        }
        bool init(void)
        {

            return true;
        }
        bool run(void *p)
```

```
        {
              unsigned char buffer; int
count=1; int Addr=0x3C; int X,Y,Z; uint8_t
x[2],y[2],z[2];
              int i,bit;
              if(once==1)
              {
                    for (int addr = 2;
addr <= 254; addr += 2) {

if
(I2C2::getInstance().checkDeviceResponse(a
ddr)) {

printf("I2C device responded to address
%#4x\n", addr);

if(addr==0x3C)

{

       printf("Discovered sensor LSM303");

       break;

}

}

       I2C2::getInstance().writeReg(0x3C,
0x00, 0x08); //Write 0x08 into register
00h to set to 3Hz

       I2C2::getInstance().writeReg(0x3C,
0x02, 0x00); //write 0x00 into register
02h to set continuous conversion
                                    once=0;
                    }
              }


       I2C2::getInstance().readRegisters(Ad
dr, 0x03, &x[0], count);

       I2C2::getInstance().readRegisters(Ad
dr, 0x04, &x[1], count);

       X=(0xFFFF^((x[0]<<8|x[1])))+1;
                    printf("x:%d\n",X);

       I2C2::getInstance().readRegisters(Ad
dr, 0x05, &y[0], count);

       I2C2::getInstance().readRegisters(Ad
dr, 0x06, &y[1], count);

       Y=(0xFFFF^((y[0]<<8|y[1])))+1;
                    printf("y:%d\n",Y);

       I2C2::getInstance().readRegisters(Ad
dr, 0x07, &z[0], count);

       I2C2::getInstance().readRegisters(Ad
dr, 0x08, &z[1], count);

       Z=(0xFFFF^((z[0]<<8|z[1])))+1;
              printf("z:%d\n",Z);
              location[0] = X;
              location[1] = Y;
              location[2] = Z;

       xQueueSend(coordinates,location,port
MAX_DELAY);
              vTaskDelay(330);

              return true;
        }
};

class ADC : public scheduler_task
{
    public:
        ADC(uint8_t priority) :
scheduler_task("ADC", 2000, priority)
        {
            /* Nothing to init */
        }
        bool init(void)
        {
              LPC_SC->PCONP|=(1<<12);
//Enable PCONP register for the ADC
                    LPC_ADC-
>ADCR|=(1<<21);
//Enable the PDN bit
                    LPC_ADC->ADCR &=
~(0XFF);                    //Clear the
enable bits of all the channels
                    LPC_ADC->ADCR |=
(1<<5);                     //Enable
conversions for channel 5
                    LPC_SC-
>PCLKSEL0&=~(3<<24);
//Clear the system clock selection bits
                    LPC_SC-
>PCLKSEL0|=(1<<24);
//Select (system clock/1)=120MHz
                    LPC_ADC->ADCR &=
~(0xFF<<8);                 //Clearing
the pre-scalar bits in the ADC Control
Register
```

```
                    LPC_ADC->ADCR |=
(10<<8);                    //Setting the
pre-scalar to 11
                    LPC_PINCON->PINSEL3 &=
~(3<<30);               //Clearing the
PINSEL for p0.31
                    LPC_PINCON->PINSEL3 |=
(3<<30);                //Setting p0.31 to
act as an ADC pin
                    return true;
        }
        void UARTSend(char *BufferPtr,
uint32_t Length )
        {
                    UART2TxEmpty=1;
                    for(uint32_t
i=0;i<Length;i++)
//Transmit a string of length "Lenghth"
                    {
                          vTaskDelay(1);
                          while (
!(UART2TxEmpty & 0x01) );
//Wait while the UART2 Tx FIFO is full.
Proceeeds when it becomes empty.
                          LPC_UART2->THR =
BufferPtr[i];                 //Put the
data in the THR register
                          UART2TxEmpty =
0;
//Indicate that Tx FIFO is not empty,
contains valid data.
                    }
// UART2TxEmpty is made 1 by the interrupt
handler when TX FIFO becomes empty
        }
        bool run(void *p)
        {
             LPC_ADC->ADCR|= (1<<24);
             char send[5]="";
                    while(!(LPC_ADC-
>ADDR5&(1<<31)));
                    uint16_t output =
(LPC_ADC->ADDR5>>4)&(0xfff);
                    if(output<10)
                    {
                          sprintf(send,
"000%d\n", output);
                    }
                    else
if(output>=10&&output<100)
                    {
                          sprintf(send,
"00%d\n", output);
                    }
                    else
if(output>=100&&output<1000)
                    {
                          sprintf(send,
"0%d\n", output);
                    }
                    else
                    {
                          sprintf(send,
"%d\n", output);
                    }
                    u0_dbg_printf("ADC :
%s",send);
                    //char *send="Hello";
                    vTaskDelay(330);
                    UARTSend(send,
sizeof(send));
              return true;
        }
};

extern "C"
{
void UART2_IRQHandler(void)
//UART Interrupt handler
{
       uint8_t IIRValue, LSRValue;
       uint8_t Dummy;
       IIRValue=LPC_UART2->IIR;
//Read the IIR register (Interrupt
Identification register)
       IIRValue>>=1;
//Shift right by one bit
       IIRValue &= 0x07;
//perform AND operation on last three bits
[3:1] with 1 to read their values
       LSRValue=LPC_UART2->LSR;
//Read the LSR register
       if ( LSRValue &
((1<<1)|(1<<2)|(1<<3)|(1<<7)|(1<<4)) )
//Check if the error bits are set in
LSRValue
              {
                Dummy = LPC_UART1->RBR;
//Read the RBR register and return. (Only
way to clear the error)
                return;
              }
       if ( LSRValue & (1<<0) )
//If there is valid data in the Receive
buffer read it and print it
       {
              UART2Buffer[UART2Count] =
LPC_UART2->RBR;
```

```
	//printf("%c",UART2Buffer[UART2Count
]);
			UART2Count++;
			if ( UART2Count == BUFSIZE )
			{
				UART2Count = 0;
			}
		}
		if ( IIRValue == 0x01 )
//If THRE interrupt is enabled
		{
			LSRValue = LPC_UART2->LSR;
			if ( LSRValue & (1<<5) )
//If the Tx FIFO is empty return
UART2TxEmpty=1
			{
				UART2TxEmpty = 1;
			}
			else
			{
				UART2TxEmpty = 0;
			}
		}
}
}
int main()
{
	scheduler_add_task(new
terminalTask(PRIORITY_HIGH));
	//scheduler_add_task(new
UART2(PRIORITY_LOW));
	//scheduler_add_task(new
ADC(PRIORITY_LOW));
	scheduler_add_task(new
I2C_Sensor(PRIORITY_MEDIUM));
	scheduler_add_task(new
PID(PRIORITY_LOW));
	scheduler_start();
	return -1;
}
```

[B]ADC_Validation.c

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define ADC_NO_OF_SAMPLES 16
#define Vref              3.3
#define Resolution        4095

struct Complex
{    double rl;         //Real Part
    double im;        //Imaginary
Part
}  X[33], U, W, T, Tmp;

void FFT(void)
{
    int M = 4;
    int N = pow(2.0, M);

    int i = 1, j = 1, k = 1;
    int LE = 0, LE1 = 0;
    int IP = 0;

    for (k = 1; k <= M; k++)
    {
        LE = pow(2.0, M + 1 - k);
        LE1 = LE / 2;

        U.rl = 1.0;
        U.im = 0.0;

        W.rl = cos(M_PI /
(double)LE1);
        W.im = -sin(M_PI/
(double)LE1);

        for (j = 1; j <= LE1; j++)
        {
            for (i = j; i <= N; i =
i + LE)
            {
                IP = i + LE1;
                T.rl = X[i].rl +
X[IP].rl;
                T.im = X[i].im +
X[IP].im;
                Tmp.rl = X[i].rl -
X[IP].rl;
                Tmp.im = X[i].im -
X[IP].im;
                X[IP].rl = (Tmp.rl
* U.rl) - (Tmp.im * U.im);
                X[IP].im = (Tmp.rl
* U.im) + (Tmp.im * U.rl);
                X[i].rl = T.rl;
                X[i].im = T.im;
            }
```

```
            Tmp.rl = (U.rl * W.rl)
- (U.im * W.im);
            Tmp.im = (U.rl * W.im)
+ (U.im * W.rl);
            U.rl = Tmp.rl;
            U.im = Tmp.im;
        }
    }

    int NV2 = N / 2;
    int NM1 = N - 1;
    int K = 0;

    j = 1;
    for (i = 1; i <= NM1; i++)
    {
        if (i >= j) goto TAG25;
        T.rl = X[j].rl;
        T.im = X[j].im;

        X[j].rl = X[i].rl;
        X[j].im = X[i].im;
        X[i].rl = T.rl;
        X[i].im = T.im;
    TAG25: K = NV2;
    TAG26: if (K >= j) goto TAG30;
        j = j - K;
        K = K / 2;
        goto TAG26;
    TAG30: j = j + K;
    }
}
int main()
{
     int fd = 0;
     char buffer,buffer2[5];int
i=0,j=0;int start=0;int
Voltage[16];
     float Volts[32];
     int adcVal_raw[32];
     int psd;
     float P[33];
     float mean;
     fd= open("/dev/ttySAC1", 0);
     if (fd < 0) {
          fd =
open("/dev/ttySAC1", 0);
     }
     if (fd < 0) {
          perror("open device
leds");
          exit(1);
     }
     while(j<16)
     {
          read(fd, &buffer,1);
          if(buffer=='\n')
          {
               start=1;
          }
          while(start==1)
          {
          read(fd, &buffer,1);
          if(buffer!='\n')
          {
               buffer2[i]=buffer;
               i++;
          }
          else
          {
               i=0;
               int
voltage=atoi(buffer2);
               Voltage[j]=voltage;
               Volts[j] =
(Vref/Resolution)*Voltage[j];
           float volt_comp =
(Volts[j]+ ((Voltage[j]+1)/1.3636);
           int digital_comp =
volt_comp*1240.8636;
           printf("volt_comp:
%f\n",volt_comp);
           printf("digital_comp:
%d\n",digital_comp);

     printf("%d\n",Voltage[j]);

     printf("%f\n",Volts[j]);
               //printf("string:
%s\n",buffer2);
               start=0;
               j++;
          }
          }

     }
     for (i=1;
i<=ADC_NO_OF_SAMPLES; i++) {
       X[i].rl = Volts[i];
       X[i].im = 0.0;
    }
       printf("\n\nADC
Values:\n");
```

```
        for (i=1;
i<=ADC_NO_OF_SAMPLES; i++)
printf("x[%d]:%.2f\n",i,X[i].rl);
        FFT();
        for (i=1;
i<=ADC_NO_OF_SAMPLES; i++) {
            X[i].rl =
X[i].rl/ADC_NO_OF_SAMPLES;
            X[i].im =
X[i].im/ADC_NO_OF_SAMPLES;
        }
        printf("/n/nFFT
Values:\n");
        for (i=1;
i<=ADC_NO_OF_SAMPLES; i++)
            printf("X[%d]:real ==
%fimaginary == %f\n",i-
1,X[i].rl,X[i].im);

            //power spectrum

printf("\n\n***************Power
Spectrum*****************\n");
            for (i=1;
i<=ADC_NO_OF_SAMPLES; i++) {
                P[i] =
pow((X[i].rl),2)+pow((X[i].im),2);
                //P[i]=
sqrt(((X[i].rl*X[i].rl)+(X[i].im*X[
i].im)));

printf("P[%d]:%.2f\n",i-1,P[i]);
            }
            psd =
(ADC_NO_OF_SAMPLES/2)+1;
            printf("Power Spectrum
Density == %f\n",P[psd+1]);
            mean = 0;
            for (i = 4; i<=11; i++)
{  //12 -> 19
                mean = mean +P[i];
            }
            mean/=8;
            printf("\n");
            if (mean <(0.05*P[1]))
{
                printf("Data is
valid.\n");
            }
            else
            {
                printf("Data is not
valid.\n");
            }
//ExitFinal:
    return 0;
     close(fd);
     return 0;
}
```