ADC & PWM implementations On Samsung S3C6410 ARM-11 Board & Beaglebone Node

Electrical and Computer Engineering Department, Charles W. Davidson College of Engineering
San Jose State University, San Jose, CA95112
Sagar Shah,
Email: sagar.shah@sjsu.edu,

Abstract

This report describes the hardware and software design and development details of validating ADC data acquired using Beaglebone Node to transfer it to Samsung S3C6410 ARM-11 board using UART protocol. Connector CON1 on Samsung board has two channels for ADC input. We are going to use pin 28 to connect to ADC input. The S3C6410 board has an inbuilt power supply circuit, which provides 3.3V DC output. This 3.3V DC is connected to ADC pin on potentiometer. We check output of the ADC by varying potentiometer. Validation of data is done by Fast Fourier Transform and analyzing power spectrum. After validation is done, we calculate P.I.D. values, and apply it to the Motor to generate LSM sensor output on Samsung S3C6410 ARM-11 Board. Work of ADC validation and LSM sensor interface from servo motor toARM-11 board is done by Sagar Shah. Work of UART and PID for servo motor at ARM-11 is done by Sarvesh Harhare. Work of PID for servo motor and ADC at BBB is done by Rajul Gupta. Interfacing of Beaglebone with ARM-11 is done by everyone. Tan was helping in each task little bit.

1. INTRODUCTION

Samsung TINY6410 is an ARM-11 based development board, powered with Samsung S3C6410 System on Chip. It runs a full fledge operating system. and has different types of I/O ports like IR, USB, Audio, Ethernet, Camera, TV-out, 40 pin system bus, 30 pin GPIO, 20 pin SDIO, etc. It operates a Linux operating system. We are going to use UART CON2 pins to interface with the external Beaglebone Node and Node is connected to prototyping board using UART only.

The S3C6410 board has an inbuilt power unit built which gives supply to potentiometer. By varying potentiometer, we can get 0V to 3.3V DC output voltage on V-out which can be connected to ADC pin on Beaglebone Node. Whenever knob of the potentiometer is rotated, we can read the respective digital value on the Beaglebone Node and processed output is displayed on ARM-11 board.

Software program running on ARM board has two components. Its main driver that actually interfaces with ADC pin runs in kernel mode. User level application program continuously reads the digital value of the voltage connected to ADC pin. After values are retrieved, Fast Fourier Transform is calculated. Also power spectrum for the data points is calculated with the help of user application program. After that we can validate the data, and we calculate the proportional (P), Integral(I) and differential (D) values and based on that we give PWM signal to the Motor.

2.METHODOLOGY

Implementation involves interfacing of Samsung ARM-11 board to Beaglebone Node to test the functionality of UART protocol.

2.1 REQUIREMENTS

Hardware Requirements:

1. Samsung ARM-11 TINY S3C6410
2. Power Adapter
3. USB to Serial Connector.
4. Beaglebone Node
5. Servo Motor and Potentiometer

Software Requirements:

1. Linux (Ubuntu)
2. ARM Linux toolchain
3. PuTTY

2.2 DESIGN OBJECTIVES:

The design of the circuit has following objectives:

1. Use the ADC component and the potentiometer on the Beaglebone Node and transfer the data to ARM-11 for validation of ADC.

2. Use the Motor component and on the Beaglebone Node and transfer the data to ARM-11 for PID calculations to drive Motor.

3. Write driver program for ADC, PWM.

4. Write application program to verify data using FFT, power spectrum and drive motor based on PID.

5. Capture waveform on oscilloscope.

3. TECHINAL CHALLENGES

The technical challenges involved in this project are both software and hardware in nature. On the hardware side we interfaced the development board, the prototype board and the Beaglebone Node. The challenge to select the potentiometer to achieve a linear characteristic. Thus the hardware challenge was to establish common ground. On the software side setting up the IDE environment was the main challenge. Also the other challenge was to configure the data and control registers properly to ensure smooth connection and communication. Establishing link between the application program and driver program was a challenge

.Merging all the driver programs and their proper functioning was a challenge.

3.1. PROBLEM FORMULATION AND DESIGN

The problem formulation and design is both for the hardware and software. In the hardware, we developed a wire wrapping prototype board with a power supply which generated 5 Volts dc & 3.3 Volts dc.It also includes an potentiometer serving as sensor output for ADC. The software problem formulation and design involves writing a driver and application to validate the ADC data using FFT.PID values are calculated for PWM testing.PWM output is connected to Motor.

4. HARDWARE DESIGN

Hardware design has three major components, viz., Prototype Board, ARM-11 Board and Beaglebone Node. This section explains implementation details of prototype board and interfacing ADC and PWM pin on Beaglebone Node to transfer to ARM-11 Board using UART.

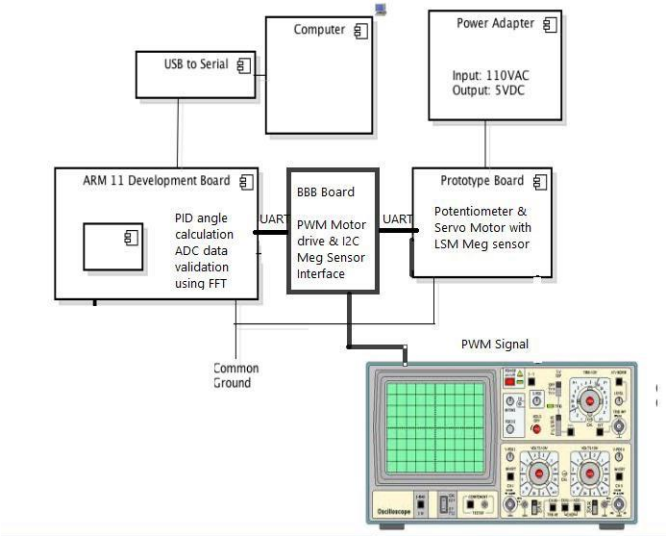


Fig.1. System Architecture Block Diagram

Basic components involved in the design are PC, ARM-11 board, Beaglebone Node, Potentiometer, Servo Motor and power adapter. PC is connected to ARM-11 board using USB to serial connecter and a common ground.

4.1. SAMSUNG ARM-11 BOARD

Samsung ARM-11 Board is powered by Samsung S3C6410 SoC. This board has 30 pin GPIO port. Out of which we are going to use 2 pins viz., ADC (Pin28) and GND (Pin2), for our experiment. ADC pin can read voltage values between 0V to 3.3V DC. GND is connected to common ground. This board runs Linux operating system on it. So to interface with ADC pins, we need to write a device driver module. Section 4 explains no how to write device driver software for the same and user level application program to read and validate ADC data. However, in this section, hardware details of Samsung S3C6410 ARM-11 board are explained. Figure 4 shows the picture of that board. And Figure 5 shows picture of CPU module on that board. As you can see in Figure 5, it has two different types of connectors mounted on it, viz., CON1 and CON2. We are going to use CON1 as it has ADC pins of our interest and UART CON2 to transfer data from and to Beaglebone Node..

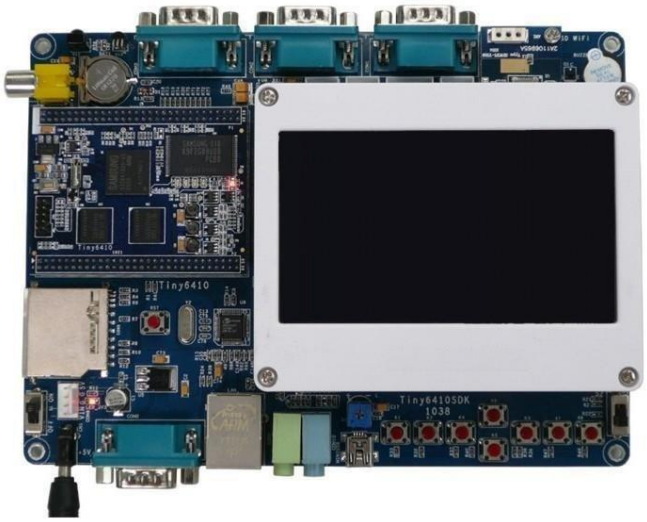


Fig.2. Samsung S3C6410 ARM-11 Board

CPU module on this board is detachable, and detailed picture of CPU module is shown in Fig 3.

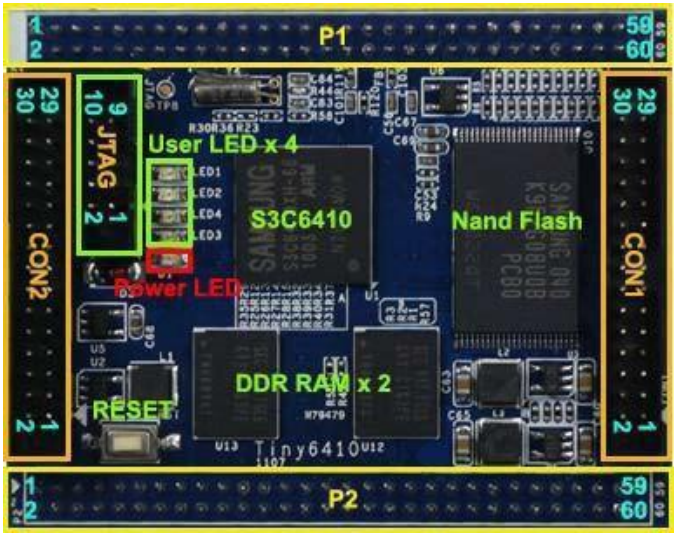


Fig.3. S3C6410 CPU Module

There are two different ADC channels on CON1, channel 2 on pin 28 and channel 3 on pin 29. We are going to use pin 28 for our experiment. Pin 2 which is GND pin, is connected to GND on prototype board. ADC on board can read analog voltage on this ADC pin and can convert it to appropriate digital value. FFT and power spectrum is calculated for retrieved values. After that we calculate P.I.D. values,

and with help of PWM we drive the motor. Software part is explained in detail in the Section 4. This pin configuration for CON1 connector is given in the Table 2

CON1.1	VDD_IO(3.3V)	CON1.2	GND
CON1.3	GPE1	CON1.4	GPE2
CON1.5	GPE3	CON1.6	GPE4
CON1.7	GPM0	CON1.8	GPM1
CON1.9	GPM2	CON1.10	GPM3
CON1.11	GPM4	CON1.12	GPM5
CON1.13	GPQ1	CON1.14	GPQ2
CON1.15	GPQ3	CON1.16	GPQ4
CON1.17	GPQ5	CON1.18	GPQ6
CON1.19	SPICLK0	CON1.20	SPIMISO0
CON1.21	SPICS0	CON1.22	SPIMOSI0
CON1.23	EINT6	CON1.24	EINT9
CON1.25	EINT11	CON1.26	EINT16
CON1.27	EINT17	CON1.28	AIN2
CON1.29	AIN3	CON1.30	DACOUT1

Table 1. Pin Configuration for CON1 Port on CPU Board

4.2 BEAGLEBONE BLACK NODE

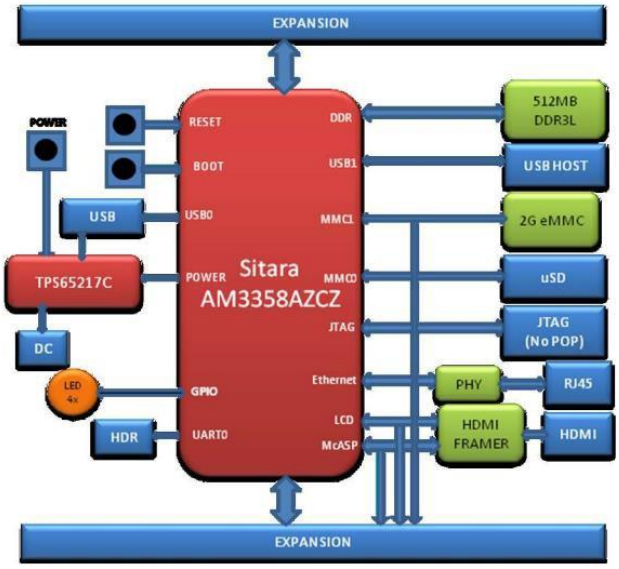


Fig.4. Beaglebone Black Block Diagram

BeagleBone Black is a low-cost, community-supported development platform for developers and hobbyists. Boot Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable.

Fig.5. Actual Beaglebone Board

Serial debug is provided via UART1 on the processor via a single 1x6 pin header. In order to use the interface a USB to TTL adapter will be required. Signals supported are TX and RX. None of the handshake signals are supported. P9-26 UART1_RXD and P9-24 UART1_TXD is used for communication with ARM-11.

1 Processor: AM335x 1GHz ARM® Cortex-A8

- 512MB DDR3 RAM
- 4GB 8-bit eMMC on-board flash storage
- 3D graphics accelerator
- NEON floating-point accelerator
- 2x PRU 32-bit microcontrollers

2 Software Compatibility

- Debian
- Android
- Ubuntu
- Cloud9 IDE on Node.js w/ BoneScript library
- plus much more

3 Connectivity

- USB client for power & communications
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

4.3. SPECIFICATIONS of ARM-11

1. **Dimension:** 180 x 130 mm
   2. EEPROM: 256 Byte (I2C)
   3. Ext. Memory: SD-Card socket
   4. Serial Ports: DB9 connector (RS232), total: 4x serial port connectors)
   5. USB: USB-A Host 1.1, miniUSB Slave/OTG 2.0
   6. mini PCIe
   6. Audio Output: 3.5 mm stereo jack
   7. Audio Input: 3.5mm jack + Condenser microphone
   8. Ethernet: RJ-45 10/100M (DM9000)
   9. RTC: Real Time Clock with battery
   10. Beeper: PWM buzzer
   11. Camera: 20 pin (2.0 mm) Camera interface
   12. TV Output: AV Out
   13. LCD: 40 pin (2.0 mm), 40 pin FFC, 40 pin FFC, 45 pin FFC connector
   14. User Inputs: 8x buttons and 1x A/D pot
   15. Expansion headers (2.0 mm)
   16. Power: regulated 5V

P1.1	DC-5V	P1.31	USB Slave D-
P1.2	GND	P1.32	USB Host D-
P1.3	LCD_R5/GPJ7	P1.33	USB Slave D+
P1.4	LCD_R4/GPJ6	P1.34	USB Host D+
P1.5	LCD_R3/GPJ5	P1.35	TSXP/AIN7
P1.6	LCD_R2/GPJ4	P1.36	TSXM/AIN6
P1.7	LCD_R1/GPJ3	P1.37	TSYP/AIN5
P1.8	LCD_R0/GPJ2	P1.38	TSYM/AIN4
P1.9	LCD_G5/GPI15	P1.39	AIN0
P1.10	LCD_G4/GPI14	P1.40	AIN1
P1.11	LCD_G3/GPI13	P1.41	WiFi_IO/GPP10
P1.12	LCD_G2/GPI12	P1.42	WiFi_PD/GPP11
P1.13	LCD_G1/GPI11	P1.43	SD1_CLK/GPH0
P1.14	LCD_G0/GPI10	P1.44	SD1_CMD/GPH1
P1.15	LCD_B5/GPI7	P1.45	SDI_Ncd/gpn10
P1.16	LCD_B4/GPI6	P1.46	SD1_Nwp/gpl14
P1.17	LCD_B3/GPI5	P1.47	SD1_DAT0/GPH2
P1.18	LCD_B2/GPI4	P1.48	SD1_DAT1/GPH3
P1.19	LCD_B1/GPI3	P1.49	SD1_DAT2/GPH4
P1.20	LCD_B0/GPI2	P1.50	SD1_DAT3/GPH5
P1.21	VDEN/GPJ10	P1.51	DACOUT0
P1.22	PWM1/GPF15	P1.52	PWM0/GPF14
P1.23	LCD/GPJ9	P1.53	XEINT0/GPN0
P1.24	LCD/GPJ8	P1.54	XEINT1/GPN1
P1.25	LCD/GPJ11	P1.55	XEINT2/GPN2
P1.26	GPE0	P1.56	XEINT3/GPN3
P1.27	VBUS	P1.57	XEINT4/GPN4
P1.28	OTGDRV_VBUS	P1.58	XEINT5/GPN5
P1.29	OTGID	P1.59	XEINT19/GPL11
P1.30	EINT8/GPN8	P1.60	XEINT20/GPL12

Table 2. Pin Configuration for P1 Port on CPU Board

2. UART Interface

S3C6410 have four serial port, it is UART0,1,2,3, UART1 is 5-wired serial, the others is 3-wired serial. In Tiny6410SDK board, COM0,1,2 was linked to DB9 interface in RS232, **you can link it to PC.** And the all serial was linked from the board to CON1, CON2, CON3, CON4 in TTL. We are using CON2 of UART for interface with Beaglebone Node.

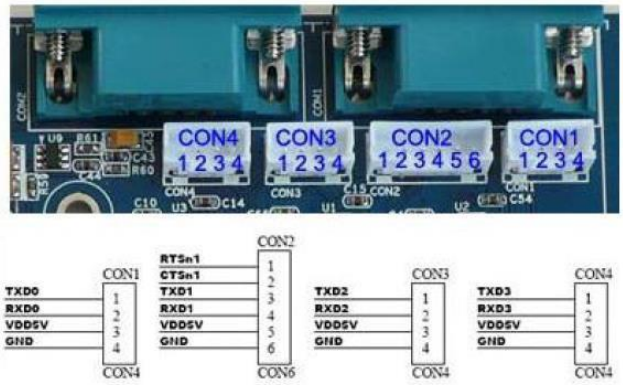


Fig.6. S3C6410 UART Module

CON0,1,2,3 pin signal is as follow:

CON0	Pins signal	CON1	Pin signal	CON2	Pin signal	CON3	Pin signal
1	NC	1	NC	1	NC	1	NC
2	RSRXD0	2	RSRXD1	2	RSRXD2	2	RSRXD3
3	RSTXD0	3	RSRXD2	3	RSRXD2	3	RSRXD3
4	NC	4	NC	4	NC	4	NC
5	GND	5	NC	5	GND	5	NC
6	NC	6	NC	6	NC	6	NC
7	NC	7	RSCTS1	7	NC	7	NC
8	NC	8	RSRTS1	8	NC	8	NC
9	NC	9	NC	9	NC	9	NC

Table 3. Pin Configuration for CON2 Port of UART on CPU Board

5. SOFTWARE DESIGN

Software implementation required for this experiment is divided into three parts, viz., Kernel Driver Module, Beaglebone Module and User Application Program, ARM-11 board runs Linux Operating System on it. So user level applications cannot access hardware directly. Hardware access is allowed to programs running in kernel mode only. So to gain the hardware access, we need to write a kernel device driver module for interfacing with ADC pins. And user level application communicates with this kernel driver via system calls like open, read, etc.

5.1 PWM

Thus the number of samples need to be increased. It can be even be done by padding zeroes
The mid value is chosen as 511.After the ADC values are validated they are subtracted from the midvalue. Thus we get two set of values negative and positive. The user is asked for the sample number. PID value for the chosen value is calculated using the below formulae:
For sample n
P = Kp *e(n)
No-M+1
I = Ki* Σ e(n)
N=0
M=10
D= Kd[(e(n+1)-e(n-1)] \ 2

For differential we have used central difference.Integration is applied over history of data.In this lab we are considering past 10 values.Thus M selected here is 10.If the sample value given by the user is less than 10 then all the values are considered for integral calculations.To calculate the differential value for sample 0 forward difference is considered and for sample 31 backward difference is calculated.

Forward Difference:
D= Kd[e(n+1)-e(n)\2]

Backward Difference
D=Kd[e(n)-e(n-1)]\2

Central Difference
D = [(e(n+1)-e(n-1)] \ 2

After the values of PI and D are calculated,PWM is calculated.
PWM= P+I+D

The output obtained is nothing but the PWM frequency.We can get negative as well as positive frequency value.This frequency is applied to buzzer.But since negative frequent cannot be applied to buzzer we take the modulus or absolute value of the frequency and apply is to the buzzer.To indicate the sign of the frequency an LED is connected to the GPIO pin 3 of port E.If the frequency obtained is negative the lED is turned ON to indicate negative sign else it remains OFF.

5.2 ARM TOOL CHAIN

The steps involved in this process are as follows :
1. Copy all the tar files from the SD Card to the root directory at /opt/FriendlyARM/mini6410/linux and make a copy of it to /tmp/linux
2. Untar the file named arm-linux-gcc to install the ARM toolschain. The command used for this is "$ tar xvzf arm-linux-gcc-4.5.1-v6-vfp-20101103.tgz –C /"
3. The above command creates a new directory as /opt/FriendlyARM/toolschain/4.5.1/ and installs the ARM toolchain
4. Then we need to update the .bashrc file and give the path"exportPATH=$PATH:/opt/FriendlyARM/toolschain/4.5.1/bin"
5. Finally to check whether the toolchain is installed properly, we had to give command as "arm-linux-gcc –v" to get the version of toolchain install. The version should be 4.5.1.

5.3 KERNEL DRIVER MODULE

The steps involved in this process are as follows :
1. Go to the directory/opt/FriendlyARM/mini6410/linux and untar the file by giving the command as "$ tar xvzf linux-2.6.38-20110718.tar.gz
2. This will install the linux kernel into the directory /opt/FriendlyARM/mini6410/linux/linux-2.6.38 and will include all distribution source codes

5.4 BUILD AND COMPILE DRIVER PROGRAM

The steps involved in this process are as follows :
1. Goto mini6410/linux/linux-2.6.38/drivers/char and make a file named mini6410_pwm.c, mini6410_adc.c
2. Write the code to read and write the data onto the GPE pins
3. Edit the Kconfig file
4. Check for the module name by doing $sudo make menuconfig and highlight letter "M".
5. Then do $make modules and look for LD[M] drivers/char/mini6410_pwm.ko, mini6410_adc.ko entry
7. Copy the mini6410_pwm.ko and mini6410_adc.ko file to USB

5.5 BUILD AND COMPILE APP PROGRAM

The steps involved in this process are as follows :
1. Goto mini6410/linux/examples/PID and make a file named pid.c
2. Write the code to build a application drivers
3. Modify the Makefile and include the line as obj-$(CONFIG_pid) += pid.c
4. Build and compile the program by doing $arm-linux-gcc pid.c –o pid
5. Check for the pid file and copy it to the USB

The programs contains a device name. This device name and object file should be added in the 'Makefile' present in the same location.The module name should be added to the 'Makefile' present in the 'driver' directory. We should add this module to the "Kconfig" file present in the "char"

directory to include the tristate value and the CPU it depends upon. Eg:

Config Mini6410_pwm
Tristate "CMPE 242 PWM DRIVER FOR FRIENDLYARM MINI 6410 DEVELOPMENT BOARD"
Depends on CPU_S3C64XX
Help

Config Mini6410_adc
Tristate "CMPE 242 PWM DRIVER FOR FRIENDLYARM MINI 6410 DEVELOPMENT BOARD"
Depends on CPU_S3C64XX
Help

We then have to run the command "make modules" if there are no errors the program successfully complies and build and a "*.ko" file is created. The next step is to go to the directory "linux / linux2.6.38 / " and run "make menuconfig". Here by pressing M we can make it as a loadable module.

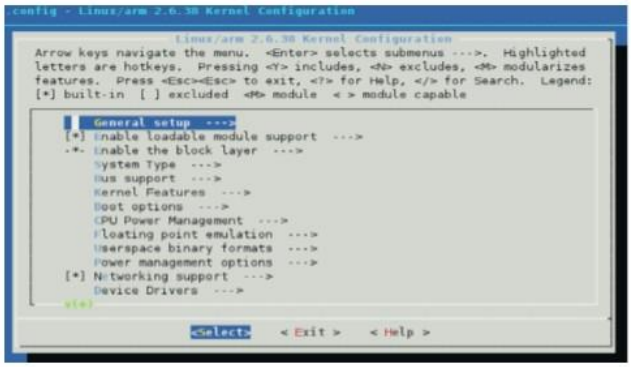


Fig.7. UI of the Kernel Configuration Window

Register	Offset	R/W	Description	Reset Value
TCFG0	0x7F006000	R/W	Timer Configuration Register 0 that configures the two 8-bit Prescaler and DeadZone Length	0x0000_0101

Timer input clock Frequency = PCLK / ({prescaler value + 1}) / {divider value}
{prescaler value} = 1~255
{divider value} = 1, 2, 4, 8, 16, TCLK

TCFG0	Bit	R/W	Description	Initial State
Reserved	[31:24]	R	Reserved Bits	0x00
Dead zone length	[23:16]	R/W	Dead zone length	0x00
Prescaler 1	[15:8]	R/W	Prescaler 1 value for Timer 2, 3 and 4	0x01
Prescaler 0	[7:0]	R/W	Prescaler 0 value for timer 0 & 1	0x01

Table 4. Details of TCFG0 Register

Register	Offset	R/W	Description	Reset Value
TCFG1	0x7F006004	R/W	Timer Configuration Register 1 that controls 5 MUX and DMA Mode Select Bit	0x0000_0000

TCFG1	Bit	R/W	Description	Initial State
Reserved	[31:24]	R	Reserved Bits	0x00
DMA mode	[23:20]	R/W	Select DMA Request Channel Select Bit 0000: No select 0001: INT0 0010: INT1 0011: INT2 0100: INT3 0101: INT4 0110: No select 0111: No select	0x0
Divider MUX4	[19:16]	R/W	Select Mux input for PWM Timer 4 0000:1/1 0001:1/2 0010:1/4 0011:1/8 0100: 1/16 0101: External TCLK1 0110: External TCLK1 0111: External TCLK1	0x0
Divider MUX3	[15:12]	R/W	Select Mux input for PWM Timer 3 0000:1/1 0001:1/2 0010:1/4 0011:1/8 0100: 1/16 0101: External TCLK1 0110: External TCLK1 0111: External TCLK1	0x0
Divider MUX2	[11:8]	R/W	Select Mux input for PWM Timer 2 0000:1/1 0001:1/2 0010:1/4 0011:1/8 0100: 1/16 0101: External TCLK1 0110: External TCLK1 0111: External TCLK1	0x0
Divider MUX1	[7:4]	R/W	Select Mux input for PWM Timer 1 0000:1/1 0001:1/2 0010:1/4 0011:1/8 0100: 1/16 0101: External TCLK0 0110: External TCLK0 0111: External TCLK0	0x0
Divider MUX0	[3:0]	R/W	Select Mux input for PWM Timer 0 0000:1/1 0001:1/2 0010:1/4 0011:1/8 0100: 1/16 0101: External TCLK0 0110: External TCLK0 0111: External TCLK0	0x0

Table 5. Details of TCFG1 Register

Register	Offset	R/W	Description	Reset Value
TCON	0x7F006008	R/W	Timer Control Register	0x0000_0000

TCON	Bit	R/W	Description	Initial State
Reserved	[31:23]	R	Reserved Bits	0x000
Timer 4 Auto Reload on/off	[22]	R/W	0: One-Shot 1: Interval Mode(Auto-Reload)	0
Timer 4 Manual Update	[21]	R/W	0: No Operation 1: Update TCNTB4	0
Timer 4 Start/Stop	[20]	R/W	0: Stop 1: Start Timer 4	0
Timer 3 Auto Reload on/off	[19]	R/W	0: One-Shot 1: Interval Mode(Auto-Reload)	0
Reserved	[18]	R/W	Reserved Bits	0
Timer 3 Manual Update	[17]	R/W	0: No Operation 1:Update TCNTB3,TCMPB3	0
Timer 3 Start/Stop	[16]	R/W	0: Stop 1: Start Timer 3	0
Timer 2 Auto Reload on/off	[15]	R/W	0: One-Shot 1: Interval Mode(Auto-Reload)	0
Reserved	[14]	R/W	Reserved Bits	0
Timer 2 Manual Update	[13]	R/W	0: No Operation 1: Update TCNTB2,TCMPB2	0
Timer 2 Start/Stop	[12]	R/W	0: Stop 1: Start Timer 2	0
Timer 1 Auto Reload on/off	[11]	R/W	0: One-Shot 1: Interval Mode(Auto-Reload)	0
Timer 1 Output Inverter on/off	[10]	R/W	0: Inverter Off 1: TOUT1 Inverter-On	0
Timer 1 Manual Update	[9]	R/W	0: No Operation 1: Update TCNTB1,TCMPB1	0
Timer 1 Start/Stop	[8]	R/W	0: Stop 1: Start Timer 1	0
Reserved	[7:5]	R/W	Reserved Bits	000
Dead zone enable/disable	[4]	R/W	Deadzone Generator Enable/Disable	0
Timer 0 Auto Reload on/off	[3]	R/W	0: One-Shot 1: Interval Mode(Auto-Reload)	0
Timer 0 Output Inverter on/off	[2]	R/W	0: Inverter Off 1: TOUT0 Inverter-On	0
Timer 0 Manual Update	[1]	R/W	0: No Operation 1: Update TCNTB0,TCMPB0	0
Timer 0 Start/Stop	[0]	R/W	0: Stop 1: Start Timer 0	0

Table 6. Details of TCON Register

5.6 KERNEL MODULE ALGORITHM FOR ADC

A. Initialize the ADCCON register in the following manner to select ADC channel 2 and 10 bit resolution.
B. After configuring the pins, reset the ADCDAT1 data register.
C. Define ADC_done function to indicate end of ADC conversion.
D. Define functions like ADC release,open and read
E. Define exit function to deregister the device

5.6.1 ADC DRIVER FLOWCHART

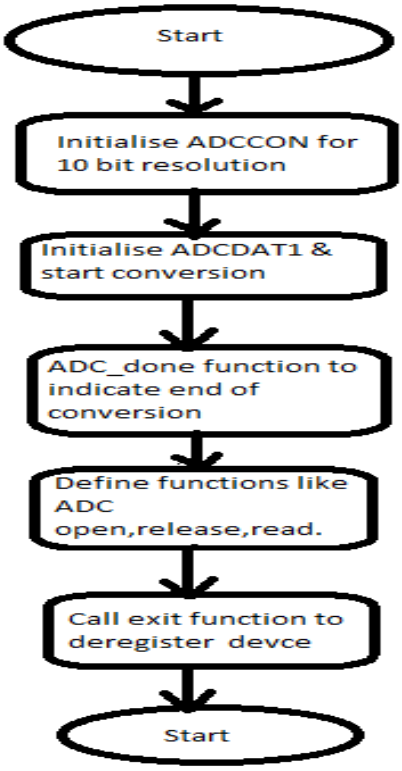


Fig.8. ADC Driver Flowchart

5.7 PWM DRIVER ALGORITHM

A. Configure PWM by initializing and configuring GPIO port E

B. Configure TCON to start the timer

C. Configure TCGF0 to give prescalar value

D. TCFG1 to configure pre scalar value

5.7.1 PWM DRIVER FLOWCHART

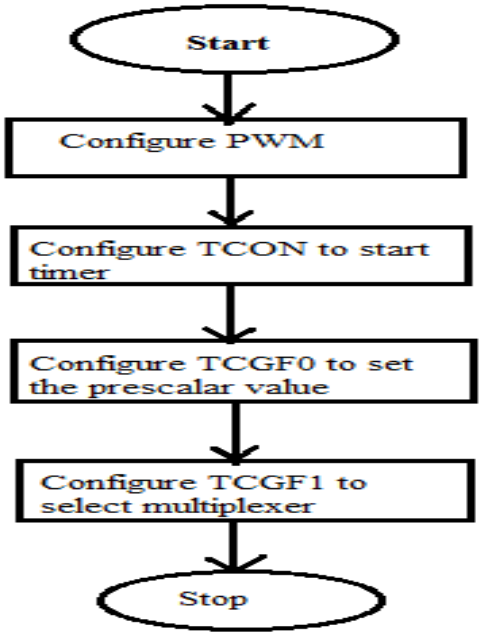


Fig.9. PWM Driver Flowchart

5.8 USER APPLICATION ALGORITHM

User application program reads ADC values from kernel module. it then computes FFT for those values. When we have FFT values, we can calculate power spectrum and find out if the data is valid or not.

If values are valid, P.I.D. values on the FFT values, we introduce an error of 511 in the FFT values so that half of the values have a positive error and half of the values have a negative error. Futher on we calculate the PID values of these samples. For this we use the following formulae:

The values of Kp, Kd, Ki are constants. They are Kp= 15.656, Kd= 0.25 and Ki= 0.25

If the sample is 0 then i.e n=0, we calculate:

P= e(n) i.e.Value of Sample I= e(n) i.e.Value of Sample D= e(n+1)– e(n)

If the sample is 31 then i.e. n=31, calculate:

P = e(n)i.e.Value of Sample

I = [e(n-9) + e(n-8) +…. + e(n)]/10
D = e(n) – e(n-1)

If sample is between 1 and 31 then i.e n= 1, 2...31, we calculate.
I = [e(n-9) + e(n-8) +…. + e(n)]/10
D = ½ (e(n+1) – e(n-1))

Once we have the P.I.D. values we find the PWM value which is obtained by multiplying P.I.D. with Kp, Ki and Kd respectively and adding these values.

PWM= (P x Kp) + (I x Ki) + (D x Kd)

Once the user selects the sample number the equivalent PWM frequency is calculated. This frequency is given to the buzzer. If the frequency is negative, then we start the LED which is connected to GPE port. We take the mod i.e take the absolute value of the negative frequency and apply it to the buzzer.

We also connected oscilloscope to PWM pin of ARM Board. When frequency on PWM changes, waveform changes.

We have written an application program to make all the above said computations. Whenever kernel module is inserted in the system, a new device file is created in "/dev" directory. This device can be opened from the user application. File descriptor FD returned by operating system after opening that device file, is used to do further communication with that

A. Define the file descriptor to open the device driver.Display error if the device doesn't open properly.

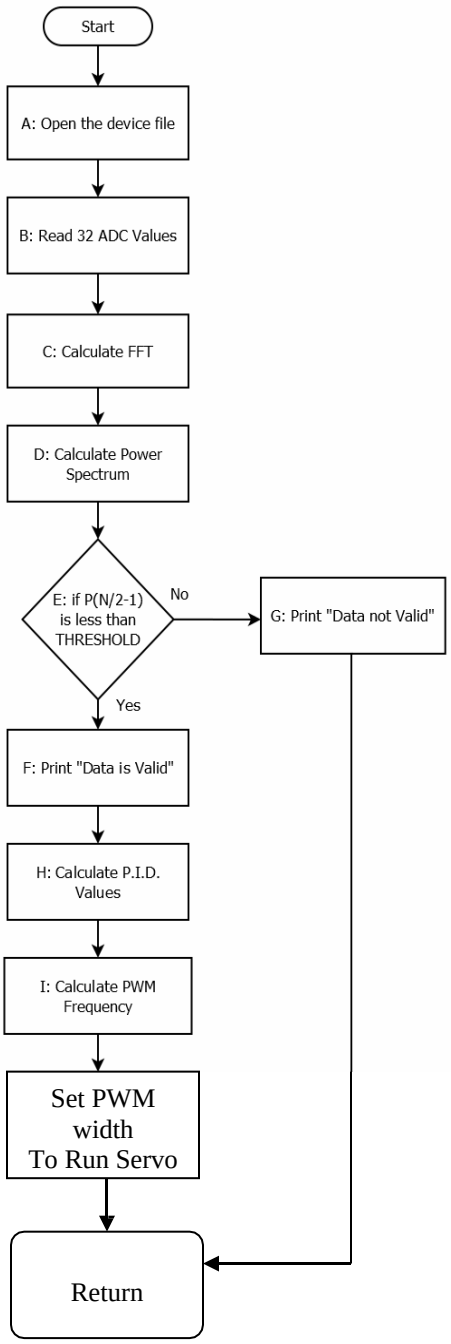
B. Read ADC buffer and copy the value to arr[i].Get keyboard input.Repeat this 32 times.

C. Call FFT function to compute FFT of the obtaines ADC values.Compute power spectrum.

D. Apply PID computation to the sample number given by the user.Obtain PWM value.If value is negative take absolute value and turn the Led ON

E. Apply PWM to the buzzer of the ARM11 board.Observe the waveform on CRO.

5.8.1 USER APPLICATION FLOWCHART



6. PSEUDO CODE

6.1. ADC DRIVER

```
static void iomem *base_addr;
typedefstruct {
wait_queue_head_t wait;
int channel;
intprescale;
} ADC_DEV;
staticint ADC_locked = 0;
static ADC_DEV adcdev;
static volatile intev_adc = 0;
staticintadc_data;
staticstructclk *adc_clock;
DEFINE __ADCREG(name) (*(volatile unsigned long
*)(base_addr + name))
DEFINEADCCON __ADCREG(S3C_ADCCON) // ADC
control
DEFINE ADCTSC __ADCREG(S3C_ADCTSC) // ADC
touch screen control
DEFINE ADCDLY __ADCREG(S3C_ADCDLY) // ADC
start or Interval Delay
DEFINE ADCDAT0 __ADCREG(S3C_ADCDAT0) // ADC
conversion data 0
DEFINE ADCDAT1 __ADCREG(S3C_ADCDAT1) // ADC
conversion data 1
DEFINE ADCUPDN __ADCREG(S3C_ADCUPDN) //
Stylus Up/Down interrupt status
DEFINE PRESCALE_DIS (0 << 14)
DEFINE PRESCALE_EN (1 << 14)
DEFINEPRSCVL(x) ((x) << 6)
DEFINE ADC_INPUT(x) ((x) << 3)
DEFINEADC_START (1 << 0)
DEFINE ADC_ENDCVT (1 << 15)
DEFINE START_ADC_AIN(ch, prescale) \
DO { \
ADCCON = PRESCALE_EN | PRSCVL(prescale) |
ADC_INPUT((ch)) ; \
ADCCON |= ADC_START; \
} WHILE (0)
staticirqreturn_tadcdone_int_handler(intirq, void *dev_id)
{
IF (__ADC_locked) {
adc_data = ADCDAT0 & 0x3ff;
ev_adc = 1;
wake_up_interruptible(&adcdev.wait);
/* clear interrupt */
__raw_writel(0x0, base_addr + S3C_ADCCLRINT);
}
RETURN IRQ_HANDLED;
}
staticssize_t s3c2410_adc_read(struct file *filp, char *buffer,
size_t count, loff_t *ppos)
{charstr[20];
int value;
size_tlen;
IF (mini6410_adc_acquire_io() == 0) {
```

```
__ADC_locked = 1;
START_ADC_AIN(adcdev.channel, adcdev.prescale);
wait_event_interruptible(adcdev.wait, ev_adc);
ev_adc = 0;
DISPLAY("AIN[%d] = 0x%04x, %d\n", adcdev.channel,
adc_data, ADCCON & 0x80 ? 1:0);
value = adc_data;
__ADC_locked = 0;
mini6410_adc_release_io();
} ELSE {
value = -1;
}
len = DISPLAY(str, "%d\n", value);
IF (count >= len) {
int r = copy_to_user(buffer, str, len);
RETURN R?r : len;
} ELSE {
RETURN -EINVAL;
}
}
staticint s3c2410_adc_open(structinode *inode, struct file
*filp)
{init_waitqueue_head(&(adcdev.wait));
adcdev.channel=2;
adcdev.prescale=0xff;
DISPLAY("adc opened\n");
return 0;
}
staticint s3c2410_adc_release(structinode *inode, struct file
*filp)
{DISPLAY("adc closed\n");
return 0;
}
staticint__initdev_init(void)
{int ret;
base_addr = ioremap(SAMSUNG_PA_ADC, 0x20);
IF (base_addr == NULL) {
DISPLAY(KERN_ERR "Failed to remap register block\n");
RETURN -ENOMEM;
}
adc_clock = clk_get(NULL, "adc");
IF (!adc_clock) {
printk(KERN_ERR "failed to get adc clock source\n");
RETURN -ENOENT;
}
clk_enable(adc_clock);
/* normal ADC */
ADCTSC = 0;
ret = request_irq(IRQ_ADC, adcdone_int_handler,
IRQF_SHARED, DEVICE_NAME, &adcdev);
IF (ret) {iounmap(base_addr);
RETURN ret;
}
ret = misc_register(&misc);
DISPLAY (DEVICE_NAME"\tinitialized\n");
RETURN ret;
}
static void __exit dev_exit(void)
{free_irq(IRQ_ADC, &adcdev);
iounmap(base_addr);
IF (adc_clock) {
clk_disable(adc_clock);
clk_put(adc_clock);
adc_clock = NULL;
}
misc_deregister(&misc);
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

6.2. PWM DRIVER

```
static void PWM_Set_Freq( unsigned long freq )
{ unsigned long tcon;
unsigned long tcnt;
unsigned long tcfg1;
unsigned long tcfg0;
struct clk *clk_p;
unsigned long pclk;
unsigned tmp;
tmp = readl(S3C64XX_GPFCON);
tmp&= ~(0x3U << 28);
tmp |= (0x2U << 28);
writel(tmp, S3C64XX_GPFCON);
tcon =_raw_readl(S3C_TCON);
tcfg1 =_raw_readl(S3C_TCFG1);
tcfg0 =__raw_readl(S3C_TCFG0);
//prescaler = 50
tcfg0&= ~S3C_TCFG_PRESCALER0_MASK;
tcfg0 |= (50 - 1);
//mux = 1/16
tcfg1&= ~S3C_TCFG1_MUX0_MASK;
tcfg1 |= S3C_TCFG1_MUX0_DIV16;
CALL__raw_writel(tcfg1,   S3C_TCFG1);
CALL__raw_writel(tcfg0,   S3C_TCFG0);
clk_p = clk_get(NULL, "pclk");
pclk = clk_get_rate(clk_p);
tcnt = (pclk/50/16)/freq;
CALL_raw_writel(tcnt, S3C_TCNTB(0));
CALL_raw_writel(tcnt/2, S3C_TCMPB(0));
tcon&= ~0x1f;
tcon |= 0xb; //disable deadzone, auto-reload, inv-off, update
TCNTB0&TCMPB0, start timer 0
raw_writel(tcon, S3C_TCON);
tcon&= ~2; //clear manual update bit
raw_writel(tcon, S3C_TCON);
}
void PWM_Stop( void )
{
unsigned tmp;
tmp = readl(S3C64XX_GPFCON);
tmp&= ~(0x3U << 28);
```

```
writel(tmp, S3C64XX_GPFCON);
}
static int s3c64xx_pwm_open(struct inode *inode, struct file
*file)
{
IF (!down_trylock(&lock))
RETURN 0;
ELSE
RETURN -EBUSY;
}
static int s3c64xx_pwm_close(struct inode *inode, struct file
*file)
{
CALLup(&lock);
RETURN 0;
}
static long s3c64xx_pwm_ioctl(struct file *filep, unsigned int
cmd, unsigned long arg)
{
switch (cmd) {
case PWM_IOCTL_SET_FREQ:
IF (arg == 0)
RETURN -EINVAL;
CALL PWM_Set_Freq(arg);
break;
case PWM_IOCTL_STOP:
default:
PWM_Stop();
break;
}
RETURN 0;
}
static int __init dev_init(void)
{
int ret;
CALL sema_init(&lock, 1);
ret = misc_register(&misc);
DISPLAY (DEVICE_NAME"\tinitialized\n");
RETURN ret;
}
static void __exit dev_exit(void)
{
CALL misc_deregister(&misc);
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
MODULE_DESCRIPTION("S3C6410 PWM Driver");
```

6.3. APPLICATION PROGRAM

```
DEFINE Kp 1
DEFINE Kd 0.25
DEFINE Ki 0.25
DEFINE SET_LED_OFF 0
DEFINE SET_LED_ON 1
DEFINE GET_SWITCH 3
int M = 5;
int N;
struct Complex
{ double a; //Real Part
double b; //Imaginary Part
} X[33];
static int fd = -1;
static int fd1= -1;
static void close_buzzer(void);
static void open_buzzer(void)
{
fd = open("/dev/harita_pwm", 0);
IF (fd < 0)
{
perror("open pwm_buzzer device");
exit(1);
}
fd1 = open("/dev/harita_leds0", 0);
IF (fd1 < 0)
{
fd1 = open("/dev/harita_leds", 0);
}
IF (fd1 < 0)
{
perror("open device leds");
exit(1);
}
// any function exit call will stop the buzzer
atexit(close_buzzer);
}
static void close_buzzer(void)
{
IF (fd >= 0) {
ioctl(fd, PWM_IOCTL_STOP);
IF (ioctl(fd, 2) < 0) {
perror("ioctl 2:");
}
close(fd);
close(fd1);
fd = -1;
fd = -1;
}
}
static void set_buzzer_freq(int freq)
{
// this IOCTL command is the key to set frequency
int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, freq);
IF (ret < 0) {
perror("set the frequency of the buzzer");
exit(1);
}
}
static void stop_buzzer(void)
{
int ret = ioctl(fd, PWM_IOCTL_STOP);
IF (ret < 0) {
perror("stop the buzzer");
```

```
exit(1);
}
IF (ioctl(fd, 2) < 0) {
perror("ioctl 2:");
}
}
```

7. TESTING AND VERIFICATION

The steps involved in doing the testing and verification are as follows:

1. Connect the ARM11 board and Beaglebone Node to laptop using the USB to Serial Connector
2. Configure settings in PuTTY as baud rate 115200, 8N1 with none parity.
3. Turn on the development board and BB Board and insert the USB
4. On the terminal, mount the USB and go into the directory
5. Run the device driver by giving the insmod command
6. Run the application driver by giving the command as $./adc //on ARM Server and $python adc.py //on BBB.
7. Once both the program are running, keep changing the pot value and obtain the ADC value on the Beaglebone Node.
8. If the ADC value is less than threshold data is valid or else data is not valid.
9. Do FFT and Validation at the ARM-11 server and plot the graph.
10. The Compensation Function obtained experimentally is **angle = 0.04 * PID - 90;**

SN	Input Voltages	ADC	H'[n]
1	0	0	3186 + 0j
2	0.1	26	122.813 +1450.22j
3	0.2	53	80.163 + 1237.65j
4	0.3	81	50.659 +1182.45j
5	0.4	118	18.814 + 1048.23j
6	0.5	150	-154.546 + 991.64j
7	0.6	182	-240.642+931.727j
8	0.7	208	-298.485 + 815.447j
9	0.8	247	-325.195 + 712.253j
10	0.9	277	-417.746 + 648.294j
11	1.0	301	-498.813+594.163j
12	1.1	330	-527.565 + 526.142j
13	1.2	371	-558.246 + 481.235j
14	1.3	398	-592.436 + 406.132j
15	1.4	422	-626.146 + 351.234j
16	1.5	453	-642.753 + 124.346j
17	1.6	492	-658 + 0j
18	1.7	524	-658 + 0j
19	1.8	555	-642.753 + 124.346j
20	1.9	581	-626.146 + 351.234j
21	2.0	615	-592.436 + 406.132j
22	2.1	639	-558.246 + 481.235j
23	2.2	665	-527.565 + 526.142j
24	2.3	701	-498.813+594.163j
25	2.4	733	-417.746 + 648.294j
26	2.5	766	-325.195 + 712.253j
27	2.6	795	-298.485 + 815.447j
28	2.7	832	-240.642+931.726j
29	2.8	865	-154.546 + 991.64j
30	2.9	894	18.814 + 1048.23j
31	3.0	924	50.659 +1182.45j
32	3.2	980	80.163 + 1237.65j
33	3.3	1020	122.813-1450.219j

Experimental setup for ADC:

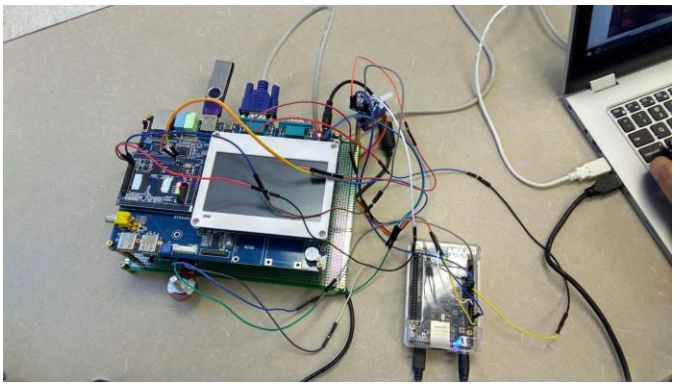


Fig.12. Screenshot of ADC Setup

ADC POWER SPECTRUM GRAPH

We have done the ADC offline for 32 values and got the below Power Spectrum.

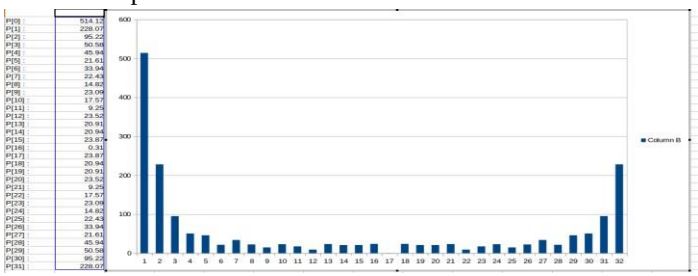


Fig.13. Power Spectrum Graph of ADC

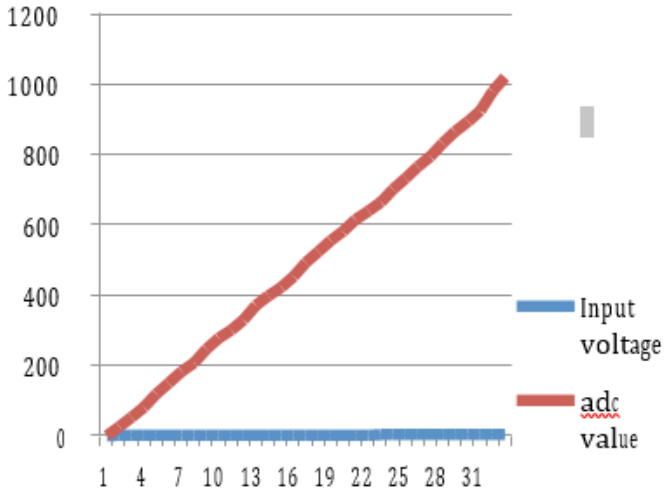


Fig.14. ADC vs I/P Voltage Graph

Servo Motor using PID and PWM:

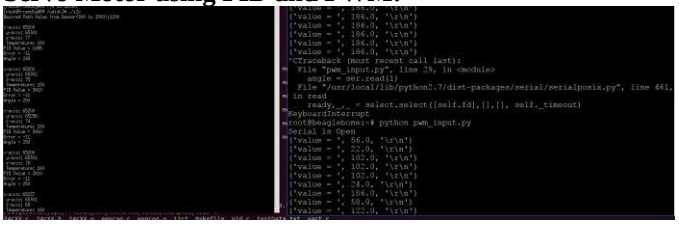


Fig.15. Screenshot of PID Result

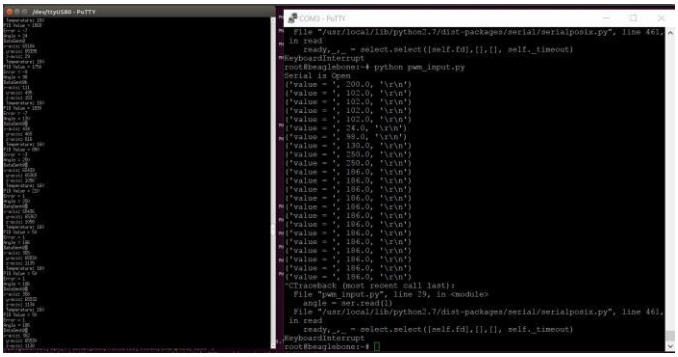


Fig.16. Screenshot of PID Result

Setup:

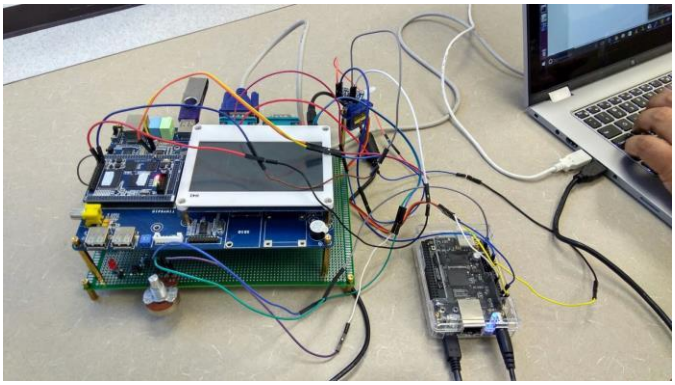


Fig.17. Screenshot of PID Setup

PID GRAPH:

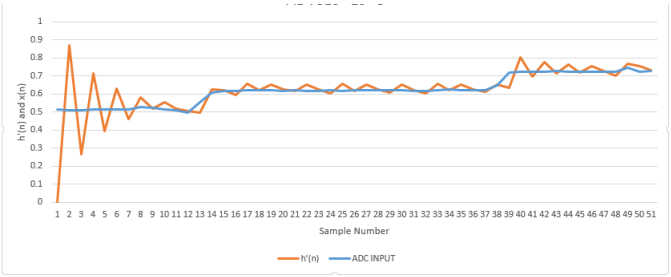


Fig.18. PID Motion Trajectory Graph by Motor

8. CONCLUSION

ADC validation experiment is performed successfully with the help of potentiometer interfaced on BeagleboneBlack Board as Node and processing code for PID and FFT Algorithm running on ARM 11 Board. Also, output from ADC is captured and linear nature of ADC is observed. Servo Motor was ran successfully with the help of BBB PWM and calculations of PID at BBB and ARM-11 server. UART communication was established for exchanging the data between ARM server and BBB Node.

9. ACKNOWLEDGMENT

The work described in this paper was made possible and achievable by the contribution of Dr. Harry Li. The electrical and electronic components were made available by Amazon marketplace and Mouser Electronics.

10. REFERENCES

[1] Datasheet for S3C6410
[2] Datasheet for LM7805
[3] Datasheet for BeagleboneBlack
[3] Dr. Harry Li, Lecture Notes of CMPE 242 – Embedded Hardware System Design

APPENDIX

1. PWM DRIVER CODE

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <asm/irq.h>
#include <asm/io.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/hardware.h>
#include <plat/regs-timer.h>
#include <mach/regs-irq.h>
#include <asm/mach/time.h>
#include <linux/clk.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/miscdevice.h>
#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>
#include <plat/gpio-cfg.h>
#include <mach/gpio-bank-e.h>
#include <mach/gpio-bank-f.h>
#include <mach/gpio-bank-k.h>
#define DEVICE_NAME "mmini6410_pwm"
#define PWM_IOCTL_SET_FREQ 1
#define PWM_IOCTL_STOP 0
static struct semaphore lock;
/* freq: pclk/50/16/65536 ~ pclk/50/16
* if pclk = 50MHz, freq is 1Hz to 62500Hz
* human ear : 20Hz~ 20000Hz
*/
static void PWM_Set_Freq( unsigned long freq )
{
unsigned long tcon;
unsigned long tcnt;
unsigned long tcfg1;
unsigned long tcfg0;
```

```
struct clk *clk_p;
unsigned long pclk;
unsigned tmp;
tmp = readl(S3C64XX_GPFCON);
tmp &= ~(0x3U << 28);
tmp |= (0x2U << 28);
writel(tmp, S3C64XX_GPFCON);
tcon = raw_readl(S3C_TCON);
tcfg1 = __raw_readl(S3C_TCFG1);
tcfg0 = __raw_readl(S3C_TCFG0);
//prescaler = 50
tcfg0 &= ~S3C_TCFG_PRESCALER0_MASK;
tcfg0 |= (50 - 1);
//mux = 1/16
tcfg1 &= ~S3C_TCFG1_MUX0_MASK;
tcfg1 |= S3C_TCFG1_MUX0_DIV16;
__raw_writel(tcfg1, S3C_TCFG1);
__raw_writel(tcfg0, S3C_TCFG0);
clk_p = clk_get(NULL, "pclk");
pclk = clk_get_rate(clk_p);
tcnt = (pclk/50/16)/freq;
__raw_writel(tcnt, S3C_TCNTB(0));
__raw_writel(tcnt/2, S3C_TCMPB(0));
tcon &= ~0x1f;
tcon |= 0xb; //disable deadzone, auto-reload,  inv-off,
update TCNTB0&TCMPB0, start timer 0
__raw_writel(tcon,  S3C_TCON);
tcon &= ~2; //clear manual update bit
__raw_writel(tcon, S3C_TCON);
}
void PWM_Stop( void )
{
        unsigned tmp;
tmp = readl(S3C64XX_GPFCON);
tmp  &= ~(0x3U << 28);
writel(tmp, S3C64XX_GPFCON);
}
static int s3c64xx_pwm_open(struct inode *inode, struct
file *file)
{
if (!down_trylock(&lock))
return 0;
else
return -EBUSY;
}
static int s3c64xx_pwm_close(struct inode *inode, struct
file *file)
{
up(&lock);
return 0;
}
static long s3c64xx_pwm_ioctl(struct file *filep, unsigned
int cmd, unsigned long arg)
{
switch (cmd) {
case PWM_IOCTL_SET_FREQ:
if (arg == 0)
return -EINVAL;
PWM_Set_Freq(arg);
break;
case PWM_IOCTL_STOP:
default:
PWM_Stop();
break;
}
return 0;
}
static struct file_operations dev_fops = {
.owner = THIS_MODULE,
.open = s3c64xx_pwm_open,
.release = s3c64xx_pwm_close,
.unlocked_ioctl = s3c64xx_pwm_ioctl,
};
static struct miscdevice misc = {
.minor = MISC_DYNAMIC_MINOR,
.name = DEVICE_NAME,
.fops = &dev_fops,
};
static int __init dev_init(void)
{
int ret;
sema_init(&lock, 1);
ret = misc_register(&misc);
printk (DEVICE_NAME"\tinitialized\n");
return ret;
}
static void __exit dev_exit(void)
{
misc_deregister(&misc);
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("sagar");
MODULE_DESCRIPTION("S3C6410 PWM Driver");
```

2. ADC DRIVER C CODE on ARM Server

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clk.h>
#include <linux/wait.h>
#include <linux/sched.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>
#include <asm/io.h>
#include <asm/irq.h>
#include <asm/uaccess.h>
```

```
#include <mach/map.h>
#include <mach/regs-clock.h>
#include <mach/regs-gpio.h>
#include <plat/regs-timer.h>
#include <plat/regs-adc.h>
#undef DEBUG
//#define DEBUG
#ifdef DEBUG
#define DPRINTK(x...) {printk(__FUNCTION__"(%d): ",__LINE__);printk(##x);}
#else
#define DPRINTK(x...) (void)(0)
#endif
#define DEVICE_NAME "mini6410_adc"
static void __iomem *base_addr;
typedef struct {
wait_queue_head_t wait;
int channel;
int prescale;
} ADC_DEV;
static inline int mini6410_adc_acquire_io(void) {
return 0;
}
static inline void mini6410_adc_release_io(void) {
/* Nothing */
}
//#endif
static int ADC_locked = 0;
static ADC_DEV adcdev;
static volatile int ev_adc = 0;
static int adc_data;
static struct clk *adc_clock;
#define __ADCREG(name) (*(volatile unsigned long *)(base_addr + name))
#define ADCCON __ADCREG(S3C_ADCCON) // ADC control
#define ADCTSC __ADCREG(S3C_ADCTSC) // ADC touch screen control
#define ADCDLY __ADCREG(S3C_ADCDLY) // ADC start or Interval Delay
#define ADCDAT0 __ADCREG(S3C_ADCDAT0) // ADC conversion data 0
#define ADCDAT1 __ADCREG(S3C_ADCDAT1) // ADC conversion data 1
#define ADCUPDN ADCREG(S3C_ADCUPDN) // Stylus Up/Down interrupt status
#define PRESCALE_DIS (0 << 14)
#define PRESCALE_EN (1 << 14)
#define PRSCVL(x) ((x) << 6)
#define ADC_INPUT(x) ((x) << 3)
#define ADC_START (1 << 0)
#define ADC_ENDCVT (1 << 15)
#define START_ADC_AIN(ch, prescale) \
do { \
ADCCON = PRESCALE_EN | PRSCVL(prescale) | ADC_INPUT((ch)) ; \
ADCCON |= ADC_START; \
} while (0)
static irqreturn_t adcdone_int_handler(int irq, void *dev_id)
{
if (__ADC_locked) {
adc_data = ADCDAT0 & 0x3ff; // ADC data is in 0-9 bits of ADCDAT0
ev_adc = 1;
wake_up_interruptible(&adcdev.wait);
/* clear interrupt */
__raw_writel(0x0, base_addr + S3C_ADCCLRINT);
}
return IRQ_HANDLED;
}
static ssize_t s3c6410_adc_read(struct file *filp, char *buffer, size_t count, loff_t *ppos)
{
char str[20];
int value;
size_t len;
if (mini6410_adc_acquire_io() == 0) {
ADC_locked = 1;
START_ADC_AIN(adcdev.channel, adcdev.prescale); // Start ADC
wait_event_interruptible(adcdev.wait, ev_adc); // Wait till adc interrupt is generated
ev_adc = 0;
DPRINTK("AIN[%d] = 0x%04x, %d\n", adcdev.channel, adc_data, ADCCON & 0x80 ? 1:0);
value = adc_data;
ADC_locked = 0;
mini6410_adc_release_io();
} else {
value = -1;
}
len = sprintf(str, "%d\n", value);
if (count >= len) {
int r = copy_to_user(buffer, str, len);
return r ? r : len;
} else {
return -EINVAL;
}
}
static int s3c6410_adc_open(struct inode *inode, struct file *filp)
{
init_waitqueue_head(&(adcdev.wait));
adcdev.channel=2;
adcdev.prescale=0xff;
DPRINTK("adc opened\n");
return 0;
}
static int s3c6410_adc_release(struct inode *inode, struct file *filp)
{
DPRINTK("adc closed\n");
return 0;
```

```
}
static struct file_operations dev_fops = {
owner: THIS_MODULE,
open: s3c6410_adc_open,
read: s3c6410_adc_read,
release: s3c6410_adc_release,
};
static struct miscdevice misc = {
.minor = MISC_DYNAMIC_MINOR,
.name = DEVICE_NAME,
.fops = &dev_fops,
};
static int __init dev_init(void)
{
int ret;
base_addr = ioremap(SAMSUNG_PA_ADC, 0x20);
if (base_addr == NULL) {
printk(KERN_ERR "Failed to remap register block\n");
return -ENOMEM;
}
adc_clock = clk_get(NULL, "adc");
if (!adc_clock) {
printk(KERN_ERR "failed to get adc clock source\n");
return -ENOENT;
}
clk_enable(adc_clock);
/* normal ADC */
ADCTSC = 0;
ret = request_irq(IRQ_ADC, adcdone_int_handler,
IRQF_SHARED, DEVICE_NAME, &adcdev);
if (ret) {
iounmap(base_addr);
return ret;
}
ret = misc_register(&misc);
printk (DEVICE_NAME"\tinitialized\n");
return ret;
}
static void __exit dev_exit(void)
{
free_irq(IRQ_ADC, &adcdev);
iounmap(base_addr);
if (adc_clock) {
clk_disable(adc_clock);
clk_put(adc_clock);
adc_clock = NULL;
}
misc_deregister(&misc);
}
module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("sagar");
```

3.I2C DRIVER CODE FOR LSM SENSOR

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <linux/fs.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <assert.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <sys/stat.h>
#include "24cXX.h"

#define CRA_REG_M                       (0x00)
#define CRB_REG_M                       (0x01)
#define MR_REG_M                        (0x02)
#define OUT_X_H_M                       (0x03)
#define OUT_X_L_M                       (0x04)
#define OUT_Z_H_M                       (0x05)
#define OUT_Z_L_M                       (0x06)
#define OUT_Y_H_M                       (0x07)
#define OUT_Y_L_M                       (0x08)
#define SR_REG_M                        (0x09)
#define TEMP_OUT_H_M                            (0x31)
#define TEMP_OUT_L_M                            (0x32)
#define LSM_303_ACCEL_ADDR                      (0x19)
#define LSM_303_MAGNET_ADDR                     (0x1E)
#define usage_if(a) do { do_usage_if( a ,_LINE_); }
while(0);
static inline s32 i2c_smbus_access(int file, char
read_write, __u8 command,
        int size, union i2c_smbus_data *data)
{
          struct i2c_smbus_ioctl_data args;


args.read_write = read_write;
args.command = command;
args.size = size;
args.data = data;
return ioctl(file,I2C_SMBUS,&args);
}
static inline_s32  i2c_smbus_write_quick(int  file,_u8
value)
{         return
i2c_smbus_access(file,value,0,I2C_SMBUS_QUICK,NUL
L);
}
static inline __s32 i2c_smbus_read_byte(int file)
{
          union i2c_smbus_data data;
```

```
        if
(i2c_smbus_access(file,I2C_SMBUS_READ,0,I2C_SMBU
S_BYTE,&data))
return -1;
        else
                return 0x0FF & data.byte;
}
static inline_s32 i2c_smbus_write_byte(int file, _u8
value)
{
        return
i2c_smbus_access(file,I2C_SMBUS_WRITE,value,
                        I2C_SMBUS_BYTE,NULL);
}

static inline __s32 i2c_smbus_read_byte_data(int file,
__u8 command)
{
        union i2c_smbus_data data;
        if
(i2c_smbus_access(file,I2C_SMBUS_READ,command,
                        I2C_SMBUS_BYTE_DATA,&data))
                return -1;
        else
                return 0x0FF & data.byte;
}

static inline __s32 i2c_smbus_write_byte_data(int file,
__u8 command,
                                __u8 value)
{
union i2c_smbus_data data;
        data.byte = value;
        return
i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                        I2C_SMBUS_BYTE_DATA,
&data);
}
static inline __s32 i2c_smbus_read_word_data(int file,
__u8 command)
{
        union i2c_smbus_data data;
        if
(i2c_smbus_access(file,I2C_SMBUS_READ,command,

I2C_SMBUS_WORD_DATA,&data))
                return -1;
        else
                return 0x0FFFF & data.word;
}

static inline __s32 i2c_smbus_write_word_data(int file,
__u8 command,
                                __u16 value)
{
        union i2c_smbus_data data;
        data.word = value;
        return
i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                        I2C_SMBUS_WORD_DATA,
&data);
}

static inline s32 i2c_smbus_process_call(int file, u8
command, __u16 value)
{
        union i2c_smbus_data data;
        data.word = value;
        if
(i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                        I2C_SMBUS_PROC_CALL,&data))
                return -1;
        else
                return 0x0FFFF & data.word;
}
/* Returns the number of read bytes */
static inline __s32 i2c_smbus_read_block_data(int file,
u8 command,
                                __u8 *values)
{
        union i2c_smbus_data data;
        int i;
        if
(i2c_smbus_access(file,I2C_SMBUS_READ,command,

I2C_SMBUS_BLOCK_DATA,&data))
return -1;
        else {
                for (i = 1; i <= data.block[0]; i++)
                        values[i-1] = data.block[i];
                return data.block[0];
        }
}

static inline __s32 i2c_smbus_write_block_data(int file,
u8 command,
                                __u8 length, __u8 *values)
{
        union i2c_smbus_data data;
        int i;
        if (length > 32)
                length = 32;
        for (i = 1; i <= length; i++)
                data.block[i] = values[i-1];
        data.block[0] = length;
        return
i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
I2C_SMBUS_BLOCK_DATA, &data);
}
#define CHECK_I2C_FUNC( var, label ) \
        do {    if(0 == (var & label)) { \
                fprintf(stderr, "\nError: " \
                        #label " function is required.
Program halted.\n\n"); \
```

```
exit(1); } \
        } while(0);
int lsm303_open(char *dev_fqn, int addr, struct lsm303*
e)
{
        int funcs, fd, r;
        e->fd = e->addr = 0;
        e->dev = 0;

        fd = open(dev_fqn, O_RDWR);
        if(fd <= 0)
        {
fprintf(stderr, "Error lsm303_open: %s\n",
strerror(errno));
                return -1;
        }
// get funcs list
        if((r = ioctl(fd, I2C_FUNCS, &funcs) < 0))
        {
fprintf(stderr, "Error ioctl I2C_FUNCS: %s\n",
strerror(errno));
                return -1;
        }
// check for req funcs
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_READ_BYTE );
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_WRITE_BYTE );
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_READ_BYTE_DATA );
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_WRITE_BYTE_DATA );
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_READ_WORD_DATA );
CHECK_I2C_FUNC( funcs,
I2C_FUNC_SMBUS_WRITE_WORD_DATA );
// set working device
        if( ( r = ioctl(fd, I2C_SLAVE, addr)) < 0)
        {
fprintf(stderr, "Error eeprom_open: %s\n",
strerror(errno));
                return -1;
        }
        e->fd = fd;
        e->addr = addr;
        e->dev = dev_fqn;
        return 0;
}
int lsm303_close(struct lsm303 *e)
{
close(e->fd);
e->fd = -1;
e->dev = 0;
        return 0;
}

#if 0
int eeprom_24c32_write_byte(struct lsm303 *e, u16
mem_addr, __u8 data)
{
        __u8 buf[3] = { (mem_addr >> 8) & 0x00ff,
mem_addr & 0x00ff, data };
        return i2c_write_3b(e, buf);
}
int eeprom_24c32_read_current_byte(struct lsm303* e)
{
        ioctl(e->fd, BLKFLSBUF); // clear kernel read
buffer
        return i2c_smbus_read_byte(e->fd);
}
int eeprom_24c32_read_byte(struct lsm303* e, u16
mem_addr)
{
        int r;
        ioctl(e->fd, BLKFLSBUF); // clear kernel read
buffer
        u8 buf[2] = { (mem_addr >> 8) & 0x0ff,
mem_addr & 0x0ff };
        r = i2c_write_2b(e, buf);
        if (r < 0)
                return r;
        r = i2c_smbus_read_byte(e->fd);
        return r;
}
#endif

#if 0
int lsm303_read_current_byte(struct lsm303* e)
{
        ioctl(e->fd, BLKFLSBUF); // clear kernel read
buffer
        return i2c_smbus_read_byte(e->fd);
}

int lsm303_read_byte(struct lsm303* e, __u16 mem_addr)
{
        int r;
        ioctl(e->fd, BLKFLSBUF); // clear kernel read
buffer
        if(e->type == EEPROM_TYPE_8BIT_ADDR)
        {
                __u8 buf = mem_addr & 0x0ff;
                r = i2c_write_1b(e, buf);
        } else if(e->type ==
EEPROM_TYPE_16BIT_ADDR) {
                u8 buf[2] = { (mem_addr >> 8) &
0x0ff, mem_addr & 0x0ff };
                r = i2c_write_2b(e, buf);
        } else {
                fprintf(stderr, "ERR: unknown eeprom
type\n");
                return -1;
        }
        if (r < 0)
```

```
                return r;
        r = i2c_smbus_read_byte(e->fd);
        return r;
}
int lsm303_write_byte(struct lsm303 *e, u16 mem_addr, __u8 data)
{
        if(e->type == EEPROM_TYPE_8BIT_ADDR) {
                __u8 buf[2] = { mem_addr & 0x00ff, data };
                return i2c_write_2b(e, buf);
        } else if(e->type == EEPROM_TYPE_16BIT_ADDR) {
                __u8 buf[3] =
                        { (mem_addr >> 8) & 0x00ff, mem_addr & 0x00ff, data };
                return i2c_write_3b(e, buf);
        }
        fprintf(stderr, "ERR: unknown eeprom type\n");
        return -1;
}

#endif

void do_usage_if(int b, int line)
{
        const static char *lsm303_usage =
                "I2C-LSM303 Program, ONLY FOR TEST!\n";

        if(!b)
                return;

        fprintf(stderr, "%s\n[line %d]\n", lsm303_usage, line);
        exit(1);
}
#define die_if(a, msg) do { do_die_if( a , msg, __LINE__); } while(0);

void do_die_if(int b, char* msg, int line)
{
        if(!b)
                return;
        fprintf(stderr, "Error at line %d: %s\n", line, msg);
        fprintf(stderr, " sysmsg: %s\n", strerror(errno));
        exit(1);
}
#if 0
static int read_from_lsm303(struct lsm303 *e, int addr, int size)
{
        int ch, i;
        for(i = 0; i < size; ++i, ++addr)
        {
                die_if((ch = lsm303_read_byte(e, addr)) < 0, "read error");
                if( (i % 16) == 0 )
                {
                        printf("\n %.4x| ", addr);
                }
                else if( (i % 8) == 0 )
                {
                        printf("  ");
                }

                printf("%.2x ", ch);
                fflush(stdout);
        }

        fprintf(stderr, "\n\n");
        return 0;
}
static int write_to_lsm303(struct lsm303 *e, int addr)
{
        int i;
        for(i=0, addr=0; i<256; i++, addr++)
        {
                if( (i % 16) == 0 )
                        printf("\n %.4x| ", addr);
                else if( (i % 8) == 0 )
                        printf("  ");
                printf("%.2x ", i);
                fflush(stdout);
                die_if(lsm303_write_byte(e, addr, i), "write error");
        }
        fprintf(stderr, "\n\n");
        return 0;
}

#endif
/*************************************************************************
 Main entry point

*************************************************************************/

int magread()
{
   struct lsm303 e;
   int  ret=0, i=0;
   int recvData[8]={0}, xaxis=0,yaxis=0,zaxis=0,temper=0;

/************************Magnetometer*************************************/
ret             = lsm303_open("/dev/i2c/0", LSM_303_MAGNET_ADDR, &e);
   if(ret < 0)
   {
```

```
    printf("Unable to open Magnetometer device file \n");
    return;
  }
ret = i2c_smbus_write_byte_data(e.fd, CRA_REG_M, 0x94);
  if(ret<0)
  {
    printf("Error writing data %d\n",__LINE_);
    return;
  }

  ret = i2c_smbus_write_byte_data(e.fd, MR_REG_M, 0x00);
  if(ret<0)
  {
    printf("Error writing data %d\n",__LINE_);
    return;
  }

  // Read data from Magnetometer
recvData[0] = i2c_smbus_read_byte_data(e.fd, OUT_X_H_M);
recvData[1] = i2c_smbus_read_byte_data(e.fd, OUT_X_L_M);
recvData[2] = i2c_smbus_read_byte_data(e.fd, OUT_Y_H_M);
recvData[3] = i2c_smbus_read_byte_data(e.fd, OUT_Y_L_M);
recvData[4] = i2c_smbus_read_byte_data(e.fd, OUT_Z_H_M);
recvData[5] = i2c_smbus_read_byte_data(e.fd, OUT_Z_L_M);
recvData[6] = i2c_smbus_read_byte_data(e.fd, TEMP_OUT_H_M);
recvData[7] = i2c_smbus_read_byte_data(e.fd, TEMP_OUT_L_M);


xaxis = recvData[0] * 256 + recvData[1];
yaxis = recvData[2] * 256 + recvData[3];
zaxis = recvData[4] * 256 + recvData[5];
temper = (recvData[6] * 256 + recvData[7]);

   printf("x-axis: %d \n y-axis: %d \n z-axis: %d \n Temperature: %d \n",twos(xaxis),twos(yaxis),twos(zaxis),temper);
          lsm303_close(&e);
          return twos(zaxis);
}

int twos(int a){
  int result;
  result = 0xFFFF & a;
  result = (((result&0x8000)?(0<<15):(1<<15))|((result&0x4000)?(0<<14):(1<<14))
|((result&0x2000)?(0<<13):(1<<13))|((result&0x1000)?(0<<12):(1<<12))

|((result&0x0800)?(0<<11):(1<<11))|((result&0x0400)?(0<<10):(1<<10))

|((result&0x0200)?(0<<9):(1<<9))|((result&0x0100)?(0<<8):(1<<8))

|((result&0x0080)?(0<<7):(1<<7))|((result&0x0040)?(0<<6):(1<<6))

|((result&0x0020)?(0<<5):(1<<5))|((result&0x0010)?(0<<4):(1<<4))

|((result&0x0008)?(0<<3):(1<<3))|((result&0x0004)?(0<<2):(1<<2))

|((result&0x0002)?(0<<1):(1<<1))|((result&0x0001)?0:1));
  return result=result+1;
}
```

4. PID APPLICATION C CODE AT ARM-11

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <linux/fs.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <assert.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <sys/stat.h>
#include "24cXX.h"
#include <termios.h>        // POSIX terminal control definitionss
#include <sys/time.h>   // time calls
#include <string.h>

int main(int argc, char** argv)
{
  int fdu;// n;
  char *DeviceName;
  struct termios port_settings;
  struct lsm303 e;

  if(argc>1)
  {
    DeviceName = (char*)malloc(strlen(argv[1])+1);
    strcpy(DeviceName, argv[1]); //Lay tham so nhap vao
  //printf("Device Name=%s\n", DeviceName);
```

```
   }
   else
     DeviceName = "/dev/ttySAC1";
//Open com port
fdu = open(DeviceName, O_RDWR | O_NOCTTY |
O_NDELAY);
if(fdu<0)
{
   printf("Open com port %s failed\n", DeviceName);
   return fdu;
}
fcntl(fdu, F_SETFL, FNDELAY);    /* Configure port
reading */

   int error[4];
   int prev_value  = 0,  value,Kp = 20  , Ki= 15  , Kd   = 10,
PID = 0;

   int angle = 0;
   char str2[10],str[10];

//Cau hinh tham so com port
//baudrate 9600, 8N1
cfsetispeed(&port_settings, B9600);
cfsetospeed(&port_settings, B9600);

port_settings.c_cflag &= ~PARENB; //Set no parity
port_settings.c_cflag &= ~CSTOPB; //Set 1 stop bit
port_settings.c_cflag &= ~CSIZE;     //Set 8 bit  data using
mask bit
port_settings.c_cflag |= CS8;
port_settings.c_cflag  &=  ~CRTSCTS;     //No  hadware
handshaking

tcsetattr(fdu, TCSANOW, &port_settings); // apply the
settings to the port
printf("Desired Path Value from Sensor(0 to 5000):");
scanf("%d", &prev_value);
printf("\r\n");

error[0] = 0;
error[1] = 0;
error[2] = 0;

while(1){

        error[3] = ((magread()) - prev_value) / 100;

        PID = Kp * error[1] + Ki * (error[3] * error[3] +
error[2]  *  error[2]  +  error[1]  *  error[1]  +  error[0]  *
error[0]) + Kd * (error[1] - error[0]);
        printf("PID Value = %d \r\n",PID);
        error[0] = error[1];
        error[1] = error[2];
        error[2] = error[3];
        angle = 0.04 * PID - 90;
        if(angle > 270) angle = angle;
        if(angle <-90)   angle = 0;
        //printf("Error = %d\r\n", error[3]);
        printf("Angle = %d\r\n",(angle & 0xFF));
        //write(fdu, (angle & 0xFF), 2);
        //write(fdu, (angle & 0xFF), 1);
        //read(fdu, str2, 2);
        str2[0] = (angle & 0xFF);
        str2[1] = 0x00;
        write(fdu,str2,1);
        read(fdu,str,8);
        //str2[5] = 0x00;
        printf("%s \r\n",str);
        //write(fdu, angle>>8, 1);
        //printf("angle = %d \r\n", angle);
        sleep(1);
}
close(fdu);
return 0;
}
```

5. **SERVO MOTOR APPLICATION PYTHON CODE AT BEAGLEBONE BOARD (RAJUL)**

```
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.UART as UART
import serial
from time import sleep

#sleep(5)
servo_pin = "P9_14"

UART.setup("UART1")
ser = serial.Serial(port = "/dev/ttyO1", baudrate = 9600)
ser.close()

#minimum duty-cycle for servo control is 1ms i.e. 5% for
50Hz freq
duty_min = 4.5

#maximum duty-cycle for servo control is 2ms  i.e.  10%
for 50Hz freq
duty_max = 10.5

duty_span = duty_max - duty_min

#setting the servo at 0 degree initial position
PWM.start(servo_pin, duty_min, 50.0, 0)
ser.open()
if ser.isOpen():
        print "Serial is Open"
sleep(5)
#angle = 'q'
while (1):
   angle = ser.read(1)
   ser.write("DataSent")
   if angle == 'x':
```

```
        PWM.stop(servo_pin)
        PWM.cleanup()
        break
    angle_f = float(ord(angle))
    print("value = ", angle_f, "\r\n")
    duty = ((angle_f / 180) * duty_span + duty_min)
    PWM.set_duty_cycle(servo_pin, duty)

ser.close()
```

6. ADC USER APPLICATION C CODE FOR ARM-11

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <linux/fs.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <assert.h>
#include <string.h>
#include <getopt.h>
#include <errno.h>
#include <sys/stat.h>
#include <termios.h>        // POSIX terminal control definitionss
#include <sys/time.h>   // time calls
#include <string.h>
#include <math.h>


struct Complex
{       double a;       //Real Part
        double b;       //Imaginary Part
}X[11];

void FFT(void)
{
        int M=5,i=1,j=1,k=1,LE=0,LE1=0,IP=0;
        struct Complex U, W, T, Tmp;
        int N=pow(2.0,M);
        for(k=1;k<=M;k++)
        {
                LE=pow(2.0,M+1-k);
                LE1 = LE / 2;
                U.a = 1.0;
                U.b = 0.0;
                W.a = cos(M_PI / (double)LE1);
                W.b = -sin(M_PI / (double)LE1);
                for (j = 1; j <= LE1; j++)
                {
                        for (i = j; i <= N; i = i + LE)
                        {
                                IP = i + LE1;
                                T.a = X[i].a + X[IP].a;
                                T.b = X[i].b + X[IP].b;
                                Tmp.a = X[i].a - X[IP].a;
                                Tmp.b = X[i].b - X[IP].b;
                                X[IP].a = (Tmp.a * U.a) - (Tmp.b * U.b);
                                X[IP].b = (Tmp.a * U.b) + (Tmp.b * U.a);
                                X[i].a = T.a;
                                X[i].b = T.b;
                        }
                        Tmp.a = (U.a * W.a) - (U.b * W.b);
                        Tmp.b = (U.a * W.b) + (U.b * W.a);
                        U.a = Tmp.a;
                        U.b = Tmp.b;
                }
        }
        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.a = X[j].a;
                T.b = X[j].b;
                X[j].a = X[i].a;
                X[j].b = X[i].b;
                X[i].a = T.a;
                X[i].b = T.b;
TAG25: K = NV2;
TAG26: if (K >= j) goto TAG30;
                j = j - K;
                K = K / 2;
                goto TAG26;
TAG30: j = j + K;
        }
}

int main(int argc, char** argv)
{
        //sleep(20);
        int i=0;
        unsigned int arr1[100];
        unsigned int arr2[100];
        float power[100];
        float mean;
        fprintf(stderr, "press Ctrl-C to stop\n");
    int fdu;// n;
    char *DeviceName;
    struct termios port_settings; //Cau truc de  luu  tru cau hinh uart
         if(argc>1)
    {
```

```
    DeviceName = (char*)malloc(strlen(argv[1])+1);
    strcpy(DeviceName, argv[1]); //Lay tham so nhap vao
   //printf("Device Name=%s\n", DeviceName);
     }
    else
     DeviceName = "/dev/ttySAC1"; //Cong com mac dinh
    //Open com port
    fdu = open(DeviceName, O_RDWR | O_NOCTTY | O_NDELAY);
    if(fdu<0)
    {
      printf("Open com port %s failed\n", DeviceName);
      return fdu;
    }
    for(i = 0; i < 64; i++)
         arr1[i] = 0;

    fcntl(fdu, F_SETFL, FNDELAY); /* Configure port reading */
    //baudrate 9600, 8N1
    cfsetispeed(&port_settings, B9600);
    cfsetospeed(&port_settings, B9600);

    port_settings.c_cflag &= ~PARENB; //Set no parity
    port_settings.c_cflag &= ~CSTOPB; //Set 1 stop bit
    port_settings.c_cflag &= ~CSIZE;      //Set 8 bit data using mask bit
    port_settings.c_cflag |= CS8;
    port_settings.c_cflag &= ~CRTSCTS; //No hadware handshaking

    tcsetattr(fdu, TCSANOW, &port_settings); // apply the settings to the port
         while(1){
         write(fdu, "ReadyToSent",11);
         for (i=0; i<64; i++)
           read(fdu,&arr1[i],1);

         write(fdu, "DataSent",8);
         for(i = 0; i<64;i++){
         printf("\r\nUART Values %d: %d ",i,arr1[i]);
         }
         for (i=0; i <=32;i++){
                  arr2[i] = 0;
         }

         arr2[0] = 0;
         //sleep(2);
         for(i = 1;i<=32;i++)
         {
                  arr2[i] = (256 * arr1[(2*i)-2]) + arr1[(2*i)-1];
         }


         for (i=1; i <=32;i++){
                  printf("arr %d: %d \r\n",i,arr2[i]);

         }
         for (i = 1; i <= 32; i++)
         {
                  X[i].a = arr2[i];
                  X[i].b = 0.0;
         }
         printf ("*********Before*********\n");
         for (i = 1; i <= 32; i++)
                  printf ("X[%d]:real == %f imaginary == %f\n", i, X[i].a, X[i].b);

         FFT();

         for (i = 1; i <= 32; i++)
         {
                  X[i].a = X[i].a/32;
                  X[i].b = X[i].b/32;
         }
         printf ("\n\n**********After*********\n");
         for (i = 1; i <= 32; i++)
                  printf ("X[%d]:real == %f imaginary == %f\n", i, X[i].a, X[i].b);
         for (i = 1; i <= 32; i++)
         {
                  power[i]= sqrt(((X[i].a*X[i].a)+(X[i].b*X[i].b)));
                  printf("power spectrum value of power[%d] is = %f \n",i,power[i]);
         }
         mean=0;
         for(i=20;i<=29;i++)
         {
                  mean= mean+ power[i];
         }
         mean/=10;
         printf("mean is %f \n",mean);
         if(mean<(0.5*power[1]))
                  printf("values are valid \n");
         else
                  printf("values are not valid \n");
         for(i=0;i<=32; i++)
         {
                  X[i].a = 0;
                  X[i].b = 0;
         }
         sleep(2);

}
}
```

7. ADC PYTHON USER APPLICATION CODE AT BEAGLEBONE NODE

```
import Adafruit_BBIO.ADC as ADC
import Adafruit_BBIO.UART as UART
import serial
import time

UART.setup("UART1")

ser = serial.Serial(port = "/dev/ttyO1",baudrate =
9600)
ser.close()

ser.open()
if ser.isOpen():
    print "Serial is open!"
ADC.setup()
value = []
for i in range(32):
    a = int(ADC.read_raw("P9_40"))
    import sys

    high = int('{0:016b}'.format(a)[:8],2)
    low = int('{0:016b}'.format(a)[8:],2)
    print(a,high,low)
    #value = ADC.read_raw("AIN1")
    value.append(high)
    value.append(low)
    print(value)
    #ser.write(value)
    #value.pop()
    #value.pop()
    time.sleep(0.5)
while (1):
    print("sending....")
    print(value)
    str1 = ser.read(11)
    ser.write(value)
    str2 = ser.read(8)
    print(str1)
    print(str2)
    time.sleep(0.1)

ser.close()
```