**Project Report**

*EMBEDDED SYSTEMS*

**Automated Household Temperature Control System**

Kamran Ali 13100174

## Aim

The aim of the project was to observe and control the temperature of a chamber by implementing PID (Proportional, Integral and Differential) feedback control system (shown in *figure 1*) using **Mini-6410**. The GUI for temperature control and monitoring was developed through QTcreator. Moreover, a separate driver module was written for interfacing Mini-6410 with the hardware, consisting mainly of the temperature measurement through ADC and fan/heater control.

## Hardware Setup

The temperature sensor that we have used is **LM35**. The AC lament lamp is used for heating purpose whereas the DC fan is used for cooling. We have used the **PID** controller to control this setup.

We are using **LM35** as temperature sensors for the purpose of our project. The potential difference across this device changes with the temperature.

We have used Analog to Digital Converter on mini-6410 to convert the voltage (corresponding to the existing temperature) in to digital signal, so that it could be fed to mini-6410 for further processing.



Figure 1

## Overview of the System:

## Driver Interface

Following is the driver we wrote for interfacing mini-6410 ADC with the chamber’s temperature sensor and communication of the PID control (running on QT application) to the chamber’s digital input for turning heater/fan ON & OFF.

#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/gpio-cfg.h>

#include <mach/gpio-bank-q.h>

#include <mach/gpio-bank-e.h>

//#include <mach/map.h>

#include <plat/regs-timer.h>

#include <plat/regs-adc.h>

static void \_\_iomem \* base\_addr;

static struct clk \*adc\_clock;

int value;

int temp1, temp2, temp3, temp4,tmp;

int t = 0;

#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 DEVICE\_NAME "adc\_dev"

static int adc\_init()

{

unsigned int preScaler = 0XFF;

ADCCON = (1<<14)|(preScaler<<6)|(1<<3)|(0<<2);

ADCCON |= ADC\_START;

return 0;

}

static int adc\_open(struct inode \*inode ,struct file \*filp)

{

adc\_init();

return 0;

}

static int adc\_release(struct inode \*inode,struct file \*filp)

{

return 0;

}

static ssize\_t adc\_read(struct file \*filp,char \_\_user \*buff,size\_t size,loff\_t \*ppos)

{

ADCCON |= ADC\_START;

while(ADCCON & 0x01);//check if Enable\_start is low

while(!(ADCCON &0x8000));

// printk("%d\n",ADCDAT0 & 0x3ff);

return (ADCDAT0 & 0x3ff);

}

static ssize\_t MyWrite (struct file \*filp, const char \* buf, size\_t count, loff\_t \*f\_pos) {

char buffer[100];

int ret;

int a,b;

ret = copy\_from\_user(&buffer,buf,count);

\*f\_pos+=count;

sscanf(buffer,"%d",&value);

//printk("%d\n",value);

//if (buffer[count-1]!='\n') printk("\n");

temp2 = readl(S3C64XX\_GPQDAT);

temp4 = readl(S3C64XX\_GPEDAT);

b = (value << 1) & 0x0000000E;

a = (value >> 2) & 0x0000001E;

t = a;

t = ((t&(1<<3)) << 1) | ((t&(1<<4)) >> 1);

t = (a&(6)) | t;

a = t;

temp2 &= (a|0xFFFFFFE1);

temp2 |= a;

writel(a,S3C64XX\_GPQDAT);

temp4 &= (b|0xFFFFFFF1);

temp4 |= b;

writel(b,S3C64XX\_GPEDAT);

return count;

}

static struct file\_operations dev\_fops =

{

.owner = THIS\_MODULE,

.open = adc\_open,

.release = adc\_release,

.read = adc\_read,

.write = MyWrite,

};

static struct miscdevice misc =

{

.minor = MISC\_DYNAMIC\_MINOR,

.name = DEVICE\_NAME,

.fops = &dev\_fops,

};

static int \_\_init dev\_init()

{

int ret;

base\_addr =ioremap(SAMSUNG\_PA\_ADC,0X20);

if(base\_addr == NULL)

{

printk(KERN\_ERR"failed to remap\n");

return -ENOMEM;

}

adc\_clock = clk\_get(NULL,"adc");

if(!adc\_clock)

{

printk(KERN\_ERR"failed to get adc clock\n");

return -ENOENT;

}

clk\_enable(adc\_clock);

// Pins

// Initiallizing the registers

tmp = readl(S3C64XX\_GPECON); //32 bits

tmp = (tmp & 0xFFF1111F); //GPE 1-4

tmp|= (0x00011110);

writel(tmp, S3C64XX\_GPECON);

tmp = readl(S3C64XX\_GPEDAT);

tmp &= (0xFFFFFFE1);// 1110 0001

writel(tmp, S3C64XX\_GPEDAT);

tmp = readl(S3C64XX\_GPQCON); //32 bits

tmp = (tmp & 0xFFFFF555); //clear GPQ 1-4

tmp|= (0x00000555);

writel(tmp, S3C64XX\_GPQCON);

tmp = readl(S3C64XX\_GPQDAT);

tmp &= (0xFFFFFFE1);// 1110 0001

writel(tmp, S3C64XX\_GPQDAT);

ret = misc\_register(&misc);

printk("dev\_init return ret:%d\n",ret);

return ret;

}

static void \_\_exit dev\_exit()

{

iounmap(base\_addr);

if(adc\_clock)//disable 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("Kamran");

## Connections of Analog and Digital Terminals

We have used ‘***A1***’ as input to the *Analog to Digital Converter* of mini 6410, and ‘***D1***’ was connected to the output of mini6410, which was used to turn-on/turn-off the Bulb.

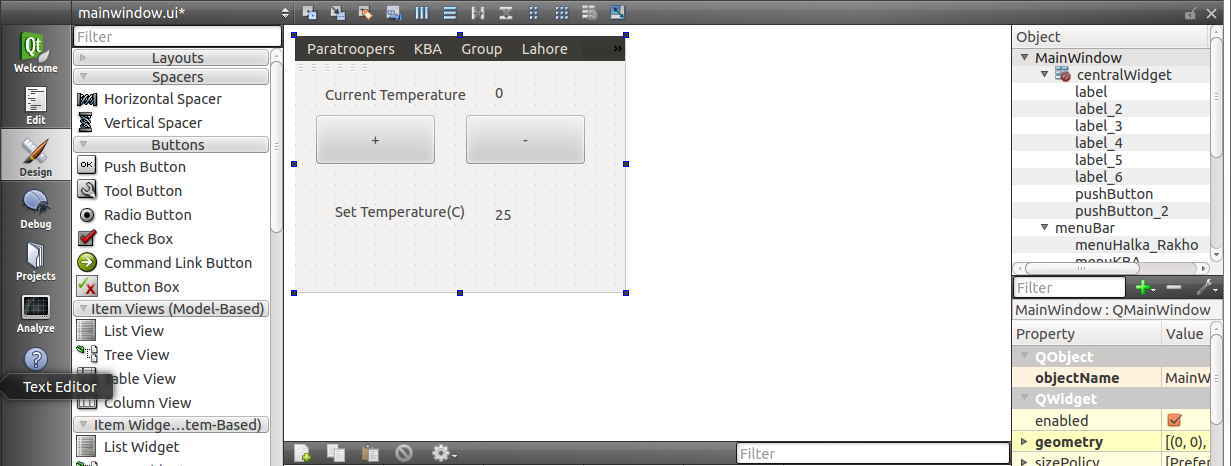
When we provide 0V to the system then it turn-off the bulb and when we apply 5V to the system then it turns on the bulb.

**QT application**

The user is able to set the required temperature through our interactive user friendly QT application and this application then controls the temperature of the room by turning on the fan if the room is too hot and would turn on the bulb/heater if the room is cold. Our system monitors the temperature at regular intervals and makes the decisions (whether to turn on the fan or bulb) based on the temperature sensed.

Depending on the difference between the desired temperature (entered by the user) and the actual temperature, we minimize the overshoot or undershoot, using the PID controller.

Here is the snapshot of the GUI we made on QTcreator:



## PID coefficient estimates using MATLAB:

Matlab provides support for estimating PID coefficients. To find these coefficients, Matlab gives certain stimuli to the system and based on the response it identifies the system and associates with it a system function. Based on this system function it gives estimates of PID coefficients. These coefficients were then tested using Matlab by interfacing our model with a computer hosting Matlab through a DAC card. Following is the code used:

clc

clear all

AI = analoginput('nidaq', 'Dev9');

addchannel(AI, 1);

set(AI,'InputType', 'singleended');

AO = analogoutput('nidaq', 'Dev9');

addchannel(AO, 1);

reftemp = zeros(1,20000);

currtemp = zeros(1,20000);

error = zeros(1,20000);

control = zeros(1,20000);

reftemp(1:6000) = 55;

reftemp(6001:20000) = 45;

for i =1:1:20000

currtemp(i) = 100\*getsample(AI);

error(i) = reftemp(i) - currtemp(i);

if (i>1)

control(i) = -1\*( 10\*error(i)+9.5\*error(i-1)+control(i-1) );

else

control(i) = 10\*error(i);

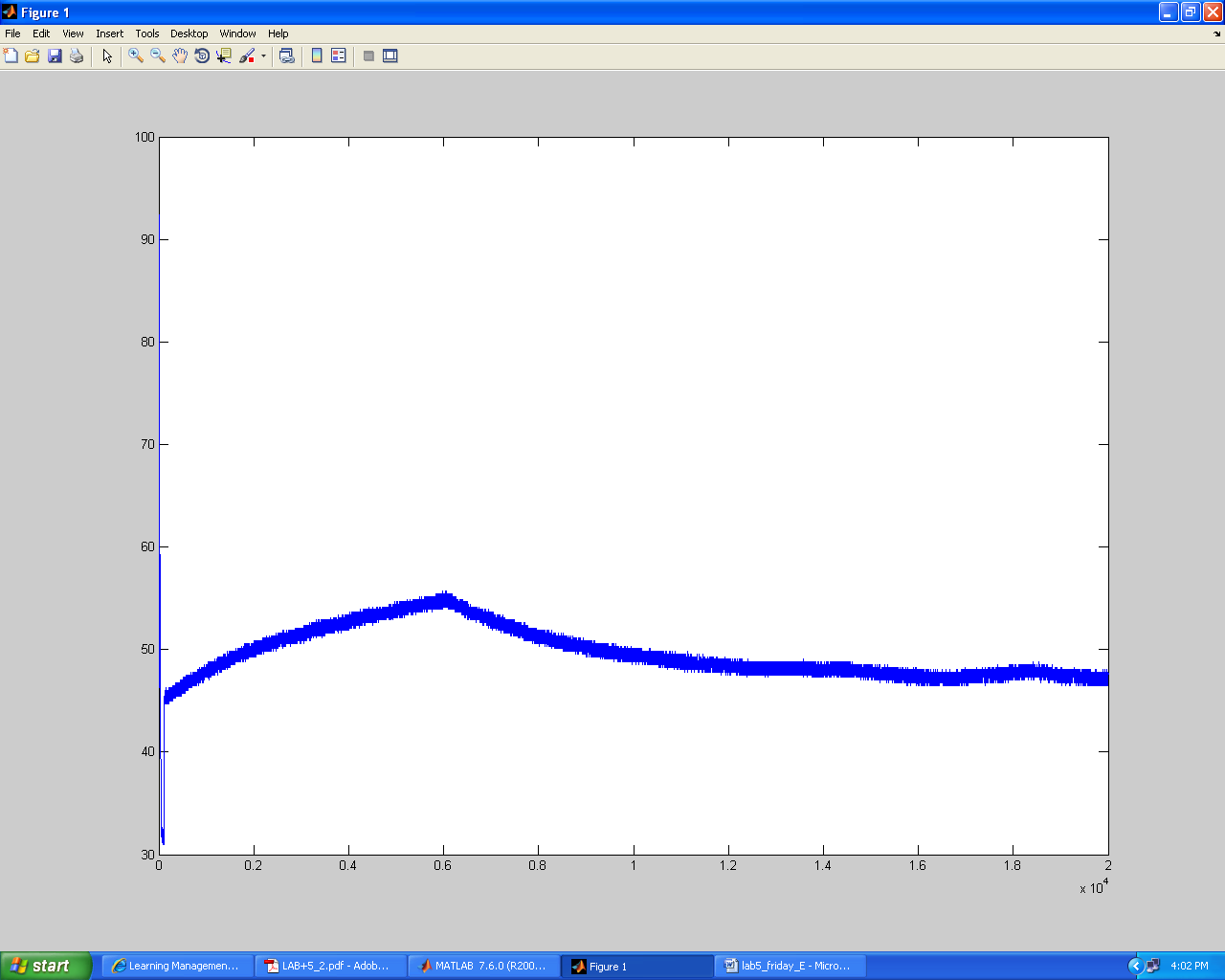
end

putsample(AO,control(i));

end

plot(currtemp);

Following was the response of the system:



However, since these coefficients were calculated using a DAC card and Matlab, the coefficients were not accurate for MINI6410. But they did provide us with an interval within which we could find the coefficients that worked best for our system.

## Final PID Control implementation on QT

After using the values of Proportional, Integral and Differential control coefficients from the Matlab simulation system we implemented the PID control on QT using C++. Following is the C++ code:

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

fd = open("/dev/adc\_dev",O\_RDWR); // Read write enable

ui->setupUi(this);

timer = new QTimer(this);

connect(timer, SIGNAL(timeout()), this, SLOT(update()));

timer->start(100);

reftemp = 45;

curr\_temp = 0;

curr\_temp\_prev = 0;

error = 0;

error\_prev = 0;

control = 0;

control\_prev = 0;

set\_temp = 55;

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::update()

{

// Temperature control

curr\_temp = 330\*read(fd,NULL,0)/1023;

error = reftemp-curr\_temp;

control = -1\*(10\*error+9.5\*error\_prev+control\_prev);

//output control signal here

curr\_temp\_prev = curr\_temp;

error\_prev = error;

control\_prev = control;

if (curr\_temp > set\_temp)

{

sprintf(str,"%u",0xFFFFFFFF);

write(fd,&str,10);

}

else

{

sprintf(str,"%u",0x00000000);

write(fd,&str,10);

}

sprintf(str,"%u",curr\_temp);

ui->label\_6->setText(str);

printf("%u\n",curr\_temp);

}

void MainWindow::on\_pushButton\_2\_clicked() // increase temperature

{

set\_temp++;

sprintf(str,"%u",set\_temp);

ui->label\_2->setTelxt(str);

}

void MainWindow::on\_pushButton\_clicked() // decrease temperature

{

set\_temp--;

sprintf(str,"%u",set\_temp);

ui->label\_2->setText(str);

}

## Conclusion

The actual behavior of the PID control using mini-6410 matched with the simulations run on Matlab using the same values of P, I and D marking the successful completion of our project. The project can be used for controlling HVAC **(**[**heating**](https://en.wikipedia.org/wiki/Heating)**,**[**ventilation**](https://en.wikipedia.org/wiki/Ventilation_(architecture))**, and**[**air conditioning**](https://en.wikipedia.org/wiki/Air_conditioning)**)** systems of Educational Institutions, Industrial zones, Factories and Apartments.