**Room Based HVAC Control**

Florida Atlantic University Computer Engineering

Boca Raton, FL

April 29, 2015

Project Engineers:

Jia Qian

Geoffrey Stoner

# Executive Summery

Our design is a multi-room comfort system that regulates temperature of each room in the house rather than having one centralized thermostat that controls the temperature for the entire house. Using this system will help alleviate one room in the house that is cooler or warmer than others in the house and or individual rooms can be raised or lowered for comfort or economy reasons.

The system would require a thermostat unit in each room. Each thermostat unit will contain a display screen as well as temperature adjustment buttons. The thermostat would display room temperature and show set temperature. The operator would adjust the set temperature by using the up and down buttons. If there is a temperature disparity of two degrees the thermostat will request the servo unit to open an electrically operated damper that would be installed in each room’s ventilation supply ducts and supply the room with ventilation. A future expansion would also communicate wirelessly to a main control unit that would replace the houses central thermostat. The control unit will communicate with each rooms thermostat and turn the HVAC unit on or off as necessary. For best efficiency there would be a network of several of the thermostat control units that will also adjust the airflow into each room by adjusting the room’s damper, supplying more airflow to warmer rooms and less or no airflow to colder rooms when operating in AC mode.

# Introduction

This report presents a design of a thermostat based damper control for a central HVAC system. The mockup of our project will include one thermostat unit. It will require a Texas Instruments -MSP430 G2553, simply referred as MSP430. The thermostat units will need a 2x8 LCD display as well as 2 SPST momentary contact switches or touch switches A LM34 temperature sensor as well as various resistors and capacitors will be needed as noted in the circuit diagram (appendix) .

The design objective is as follows. Design a software program that allows the MSP430 to measure temperature, give the user the ability to adjust a desired set temperature, compare desired temperature to room temperature, act as a control until for an LCD display and act as a control unit for a servo motor to control a ventilation damper unit.

# LM34 / LM35 Used as a Temperature SensorTemperature

Our first objective is to measure the ambient room temperature. We connected a LM34 temperature sensor to port 1.0 on the MSP430. The MSP430 G2553 A/D (analog to digital) converter ports start at pin 1.0 and continue up port 1.x until you designate the first port as digital. We discovered that the MSP430 will not allow you to add A/D ports beyond the first port you designate as a digital port. Since we had only one analog sensor we designated pin 1.0 as the port for our analog temp sensor port. The LM34 sensor as noted in the circuit diagram (appendix) was installed VCC through a 100 Ohm resistor to the (+) port on the LM34. The (-) port of the LM34 is connected to project ground and the middle pin is connected to the data line to the MSP430 port 1.0 as well as to a drain capacitor to ground. The LM34 works like a voltage divider and gives a voltage level to the A/D converter that can be converted to a temperature reading. These devices come in (F)arenheit, (C)elcius and (K)elvin, for our project we used the model that is calibrated in degrees Fahrenheit.

Figure 1 example of LM34 temp sensor connection

We setup the MSP430 as follows, we setup the ADC and set P1.0 to read mode:

ConfigureAdc();

P1DIR |= (~BIT0);

We then request the MSP430 read the ADC 5 times and average the results. This helps eliminate any unwanted data from the sensor.

**for**(i=1; i<=5 ; i++) // read all three analog values 5 times each and average

{

ADC10CTL0 &= ~ENC; //ADC10 Enable Conversion

**while** (ADC10CTL1 & BUSY); //Wait while ADC is busy

ADC10SA = (**unsigned**)&ADCReading[0]; //RAM Address of ADC Data, must be reset every conversion

ADC10CTL0 |= (ENC | ADC10SC); //Start ADC Conversion

**while** (ADC10CTL1 & BUSY); //Wait while ADC is busy

temp += ADCReading[2]; // sum all 5 reading

}

temp = temp/5; // Average the 5 reading for the three variables

temproom=((temp\*3.3)/1024)\*100;

we now have our temproom variable loaded into memory.

# Button input for user temp selection

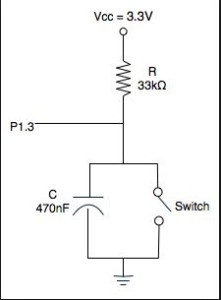
Our second objective is to allow the user to control the desired room temperature. We employed two momentary contact SPST switches so the user could manipulate the SETTEMP variable in the program. We have setup the pushbuttons in a pull-up configuration (see figure 2). For a switch to be used in digital logic, it should provide a 1 or a 0 in either state and opposite in another. In the above when, when the button is not pressed the resistor ties the pins to Vcc, pulling up the voltage to whatever level the board is powered at which is close to 3.3 or 3.6V. When you press the button, it closes the short to ground. Therefore, in our case, when the button is not pressed it gives a logic one, while when it’s pressed it gives a logic 0.  A capacitor is placed in parallel with the switch to counter the de-bouncing effect that occurs with most pushbutton switches. A 33 kΩ resistor helps keep the current draw from the button low. We set the ports 1.4 and 1.5 for input to accept the signal from the switches when they are pressed.

Figure 2 Momentary Contact Switch with debounce protection

P2DIR &= ~BIT4; //for switch button

P2DIR &= ~BIT5;// for switch button

We check the pin to see if the switch is pressed or not. If the switch is not pressed pin 2.4 or pin 2.5 will be high, once the button is pressed on either pin the circuit is shorted to ground and the signal will go low to pin 1.4 or 1.5 depending on which button is pressed. We also have added a flag variable so the MSP will only read 1 press of the button then lock out any further readings until the button is released.

if((P2IN&0x10)!=0x10&&buttonflag1==0)

{

buttonflag1=1;

settempp=settempp+1;

}

if((P2IN&0x10)==0x10)

{

buttonflag1=0;

}

if((P2IN&0x20)!=0x20&&buttonflag2==0)

{

buttonflag2=1;

settempp=settempp-1;

}

if((P2IN&0x20)==0x20)

{

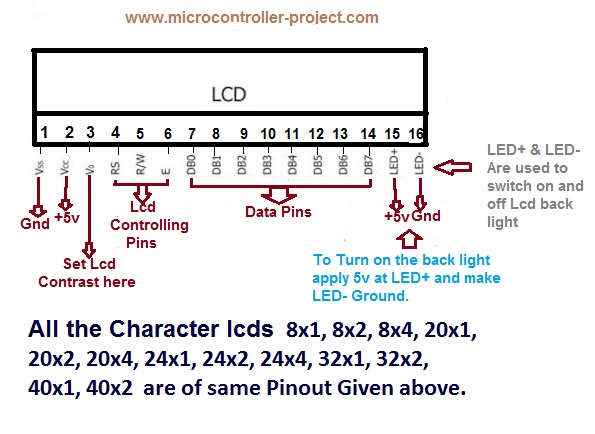
buttonflag2=0;

}

Pressing the appropriate button will also raise or lower the settemp variable. The MSP430 monitors the 4 and 5 bit on P2IN (in LSB notation) to determine if high or low. On low condition (P2IN&0x10)!=0x10) it enters the if statement and changes the settemp variable and the button flag.

# LCD Display

We added a 8x2 LCD display to show the user the current room temperature as well as the desired set temperature for the device in that room. The device we selected uses the HD44780 parallel interface chipset. Our code will work on both 8x2 and 16x2 LCD devices using the HD44780 chipset. We selected the smaller display as we had limited data to show and also wanted our unit to be compact in size. The display will work per specifications up to 7.0 volts but we found it worked just fine on the 3.3V MSP430 system voltage. We did however learn the hard way to use a 10 kΩ POT on the contrast adjustment pin, rather than give it full VCC and burned out our first display in error.

We connected the LCD pinouts as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **MSP** | **LCD** | **LCDPIN** | **Description** |
| **GND** | **Vee** | **1** | **Vee** |
| **VCC** | **Vcc** | **2** | **Vcc** |
| **POT** | **V1** | **3** | **Contrast** |
| **2.4** | **RS** | **4** | **instruction Control** |
| **GND** | **R/W** | **5** | **Low for Write** |
| **2.5** | **E** | **6** | **Control Enable** |
|  | **D0 - D3** | **7-10** | **unused** |
| **2.0** | **D4** | **11** | **Data** |
| **2.1** | **D5** | **12** | **Data** |
| **2.2** | **D6** | **13** | **Data** |
| **2.3** | **D7** | **14** | **Data** |
| **VCC** | **VCC** | **15** | **VCC** |
| **GND** | **GND** | **16** | **GND** |

Figure 3 LCD Pinout

To set the MSP 430 to we must send setup commands to the LCD display, we are using the display in 4 bit mode an we clear the screen and set the curser to upper left corner for later display of information.

**void** **lcd\_init**(**void**)

{

P2DIR |= 0xFF;

//P2DIR |= BIT4|BIT5;

P2OUT &= 0x00;

//maybe i need to do set like the following

P1DIR |= BIT4|BIT5;

P1OUT &= ~BIT4;

P1OUT &= ~BIT5;

send\_command(0x33);

send\_command(0x32);

send\_command(0x28); // 4 bit mode

send\_command(0x0E); // clear the screen

send\_command(0x01); // display on cursor on

send\_command(0x06); // increment cursor

send\_command(0x80); // row 1 column 1

in the main body of the program we send our tempature display information using the sprintf command which converts displays the variables decimal numbers we dump that information into the LCD buffer. We then send the string “Set:” then the buffer contents with the temperature information, then a carriage return (0xC0) and repeat with the “Room:” information.

**sprintf**(buffer1, "%d", displaysettemp);

send\_string("Set: ");

send\_string(buffer1);

send\_command(0xC0);

//send\_string("settemp");

**sprintf**(buffer, "%d", displaytemproom);

send\_string("Room: ");

send\_string(buffer);

# Servo Control

Finally after we have retrieved and displayed temperature information we process it with the servo control of the ventilation system. The servo must have a clock, and step information in the initial setup section of the code. You also setup your PWM steps here.

**#define** MCU\_CLOCK 1000000

**#define** PWM\_FREQUENCY 50 // In Hertz, ideally 50Hz.

**#define** SERVO\_STEPS 180 // Maximum amount of steps in degrees (180 is common)

**#define** SERVO\_MIN 650 // The minimum duty cycle for this servo

**#define** SERVO\_MAX 2700 // The maximum duty cycle

**unsigned** **int** PWM\_Period = (MCU\_CLOCK / PWM\_FREQUENCY); // PWM Period

**unsigned** **int** PWM\_Duty = 0;

In the main section of the code we set pin 1.6 to output as well as setup duty cycle and period of the PWM for the servo as well as servo step values.

TACCTL1 = OUTMOD\_7; // TACCR1 reset/set

TACTL = TASSEL\_2 + MC\_1; // SMCLK, upmode

TACCR0 = PWM\_Period-1; // PWM Period

TACCR1 = PWM\_Duty; // TACCR1 PWM Duty Cycle

P1DIR |= BIT6; // P1.6 = output for servo

P1SEL |= BIT6;

servo\_stepval = ( (SERVO\_MAX - SERVO\_MIN) / SERVO\_STEPS );

servo\_stepnow = SERVO\_MIN;

// Fill up the LUT

**for** (k = 0; k < SERVO\_STEPS; k++) {

servo\_stepnow += servo\_stepval;

servo\_lut[k] = servo\_stepnow;

}

Finally we compare our temperature values and move the servo from position 0, to 45deg to 90 degrees depending on how much fresh air is needed to cool the room.

**if**(temproom>=settempp+4){

TACCR1 = servo\_lut[0];

}

**else** **if** ((temproom<(settempp+4)) &&( temproom>(settempp))){

TACCR1 = servo\_lut[45];

}

**else**

{

TACCR1 = servo\_lut[90];

}

}

# Conclusion

This report has discusses the development of a temperature measurement device that drives a servo to open and close ventilation in a HVAC system. The objectives were to develop the necessary hardware and software to drive the servo motor to control the room temperature in a building. We have met our objective. In the future we would like to add wireless Bluetooth networking capability to the project so multiple devices can communicate with each other to balance temperature throughout the building, eliminating hot and cold rooms. The timeframe and the scope of our knowledge on wireless technology prevented us from continuing to that next step in this project.

# Appendix

## Circuit diagram



## Program flow



## Project full code

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\* CDA 6316 Weekend Class August 11/18, 201 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\* Dr. Bassem Alhalabi and TA Pablo Pastran \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\* A midterm code for students to use and modify \*\*\*\*\*\*\*\*\*\*\*

//Connections

//P2.0 - D4 Pin11

//P2.1 - D5 Pin12

//P2.2 - D6 Pin13

//P2.3 - D7 Pin14

//P1.4 - RS Pin4

//P1.5 - E Pin6 ------ change #define ENABLE\_HIGH and #define ENABLE\_LOW from BIT6 to BIT5

// other pin of lcd

//Pin #16 | GND

//Pin #15 | Vcc

//Pin #3 | Outout from potentiometer for LCD contrast

//Pin #1 | GND

//Pin #2 | Vcc

//R/W Pin5 | GND for writing

**#include** <stdio.h>

**#include** <msp430g2553.h>

**#define** DR P1OUT = P1OUT | BIT4 // define RS high

**#define** CWR P1OUT = P1OUT & (~BIT4) // define RS low

//#define READ P1OUT = P1OUT | BIT5 // define Read signal R/W = 1 for reading,now it is useless,since i ground it

//#define WRITE P1OUT = P1OUT & (~BIT5) // define Write signal R/W = 0 for writing,now it is useless,since i ground it

**#define** ENABLE\_HIGH P1OUT = P1OUT | BIT5 // define Enable high signal

**#define** ENABLE\_LOW P1OUT = P1OUT & (~BIT5) // define Enable Low signal

**unsigned** **int** i;

**unsigned** **int** j;

**#define** MCU\_CLOCK 1000000

**#define** PWM\_FREQUENCY 50 // In Hertz, ideally 50Hz.

**#define** SERVO\_STEPS 180 // Maximum amount of steps in degrees (180 is common)

**#define** SERVO\_MIN 650 // The minimum duty cycle for this servo

**#define** SERVO\_MAX 2700 // The maximum duty cycle

**unsigned** **int** PWM\_Period = (MCU\_CLOCK / PWM\_FREQUENCY); // PWM Period

**unsigned** **int** PWM\_Duty = 0;

**char** buffer[10];

**char** buffer1[10];

**int** temp = 0, temproom = 0;

**int** settempp=73;

**int** buttonflag1=0;

**int** buttonflag2=0;

**int** ADCReading [3];

**void** **ConfigureAdc**(**void**);

**void** **delay**(**unsigned** **int** k)

{

**for**(j=0;j<=k;j++)

{

**for**(i=0;i<100;i++);

}

}

**void** **data\_write**(**void**)

{

ENABLE\_HIGH;

delay(2);

ENABLE\_LOW;

}

**void** **data\_read**(**void**)

{

ENABLE\_LOW;

delay(2);

ENABLE\_HIGH;

}

**void** **check\_busy**(**void**)

{

P2DIR &= ~(BIT3); // make P1.3 as input

**while**((P2IN&BIT3)==1)

{

data\_read();

}

P2DIR |= BIT3; // make P1.3 as output

}

**void** **send\_command**(**unsigned** **char** cmd)

{

check\_busy();

//WRITE;

CWR;

P2OUT = (P2OUT & 0xF0)|((cmd>>4) & 0x0F); // send higher nibble

data\_write(); // give enable trigger

P2OUT = (P2OUT & 0xF0)|(cmd & 0x0F); // send lower nibble

data\_write(); // give enable trigger

}

**void** **send\_data**(**unsigned** **char** data)

{

check\_busy();

//WRITE;

DR;

P2OUT = (P2OUT & 0xF0)|((data>>4) & 0x0F); // send higher nibble

data\_write(); // give enable trigger

P2OUT = (P2OUT & 0xF0)|(data & 0x0F); // send lower nibble

data\_write(); // give enable trigger

}

**void** **send\_string**(**char** \*s)

{

**while**(\*s)

{

send\_data(\*s);

s++;

}

}

**void** **lcd\_init**(**void**)

{

P2DIR |= 0xFF;

//P2DIR |= BIT4|BIT5;

P2OUT &= 0x00;

//maybe i need to do set like the following

P1DIR |= BIT4|BIT5;

P1OUT &= ~BIT4;

P1OUT &= ~BIT5;

send\_command(0x33);

send\_command(0x32);

send\_command(0x28); // 4 bit mode

send\_command(0x0E); // clear the screen

send\_command(0x01); // display on cursor on

send\_command(0x06); // increment cursor

send\_command(0x80); // row 1 column 1

}

//Main Code:

/\* LCD\_own.c

\* Created on: 12-Nov-2013

\* Author: Manpreet

\* In this program we interface the lcd in 4 bit mode. We send strings and display it on the screen.

\*/

**void** **main**(**void**)

{

WDTCTL = WDTPW + WDTHOLD; // stop watchdog timer

TACCTL1 = OUTMOD\_7; // TACCR1 reset/set

TACTL = TASSEL\_2 + MC\_1; // SMCLK, upmode

TACCR0 = PWM\_Period-1; // PWM Period

TACCR1 = PWM\_Duty; // TACCR1 PWM Duty Cycle

P1DIR |= BIT6; // P1.6 = output for servo

P1SEL |= BIT6;

lcd\_init();

ConfigureAdc();

P1DIR |= (~BIT0);

// set all analog values to zero

P2DIR &= ~BIT4; //for switch button

P2DIR &= ~BIT5;// for switch button

settempp=72;

**for**(;;)

{

temp=0;temproom=0;

**for**(i=1; i<=5 ; i++) // read all three analog values 5 times each and average

{

ADC10CTL0 &= ~ENC; //ADC10 Enable Conversion

**while** (ADC10CTL1 & BUSY); //Wait while ADC is busy

ADC10SA = (**unsigned**)&ADCReading[0]; //RAM Address of ADC Data, must be reset every conversion

ADC10CTL0 |= (ENC | ADC10SC); //Start ADC Conversion

**while** (ADC10CTL1 & BUSY); //Wait while ADC is busy

temp += ADCReading[2]; // sum all 5 reading for the three variables

}

temp = temp/5; // Average the 5 reading for the three variables

temproom=((temp\*3.3)/1024)\*100;

if((P2IN&0x10)!=0x10&&buttonflag1==0)

{

buttonflag1=1;

settempp=settempp+1;

}

if((P2IN&0x10)==0x10)

{

buttonflag1=0;

}

if((P2IN&0x20)!=0x20&&buttonflag2==0)

{

buttonflag2=1;

settempp=settempp-1;

}

if((P2IN&0x20)==0x20)

{

buttonflag2=0;

}

**int** displaysettemp;

**int** displaytemproom=temproom;

displaysettemp=1;

displaysettemp=settempp;

**sprintf**(buffer1, "%d", displaysettemp);

send\_string("Set: ");

send\_string(buffer1);

send\_command(0xC0);

//send\_string("settemp");

**sprintf**(buffer, "%d", displaytemproom);

send\_string("Room: ");

send\_string(buffer);

**unsigned** **int** servo\_stepval, servo\_stepnow;

**unsigned** **int** servo\_lut[ SERVO\_STEPS+1 ];

**unsigned** **int** k;

// Calculate the step value and define the current step, defaults to minimum.

servo\_stepval = ( (SERVO\_MAX - SERVO\_MIN) / SERVO\_STEPS );

servo\_stepnow = SERVO\_MIN;

// Fill up the LUT

**for** (k = 0; k < SERVO\_STEPS; k++) {

servo\_stepnow += servo\_stepval;

servo\_lut[k] = servo\_stepnow;

}

**if**(temproom>=settempp+4){

TACCR1 = servo\_lut[0];

}

**else** **if** ((temproom<(settempp+4)) &&( temproom>(settempp))){

TACCR1 = servo\_lut[45];

}

**else**

{

TACCR1 = servo\_lut[90];

}

}

}

**void** **ConfigureAdc**(**void**)

{

ADC10CTL1 = INCH\_2 | CONSEQ\_1; // A2 + A1 + A0, single sequence

ADC10CTL0 = ADC10SHT\_2 | MSC | ADC10ON;

**while** (ADC10CTL1 & BUSY);

ADC10DTC1 = 0x03; // 3 conversions

ADC10AE0 |= (BIT0 | BIT1 | BIT2); // ADC10 option select

}

**#pragma** vector=ADC10\_VECTOR

\_\_interrupt **void** **ADC10\_ISR**(**void**)

{

\_\_bic\_SR\_register\_on\_exit(CPUOFF);

}

# References

"ADM0802A-FSY-YBS/3.3V." *ADM0802A-FSY-YBS/3.3V* (n.d.): n. pag. Web.

Schwartz, Ken. "Electronic Thermometers for Brewers." *Electronic Thermometers for Brewers*. N.p., 9 Aug. 1999. Web. 27 Apr. 2015.

Tushar. "Connecting Buttons to MSP430." *Embedded Projects from around the Web*. Embedds, 13 Dec. 2013. Web. 27 Apr. 2015.