Temperature Monitor & Controlling

Mahesh Prajapati
PDDC Student
L.D. College of Engineering
Ahmedabad, India
maheshec@gmail.com

***Abstract*— Temperature is the most significant parameter so it must be controlled as you desire. Main objective is to display the continous surrounding temperature for different zone and controlling according to desire temperature of specific zone. With some arrangement like if you connect the temperature cooling device like fan it will automatically start with as alarm is on. In this way the temperature can be controlled. Using this we can build data acquisition system which can store the temperature and able to know the temperature at each and every moment of the day or month.**

Keywords—temperature; zone; data acquisition

I. INTRODUCTION

This project is about temperature monitor and controlling. The main objective is to reads real time temperatures of specific zone and based fan on and off or other controlling solenoid valve according desire temperature. In most of the manufacturing industries like chemical, petrochemical, food processing, pharmaceutical etc temperature is one of the main parameter to be controls. Because in these kind of industries some product needs the required temperature to be maintained at highest priority otherwise the product will fail (in case of preparing any medicine in pharmaceutical company the temperature pays vital role. If it is not maintained then medicine may become poison. As mention earlier that the temperature is the most significant parameter so it must be controlled as per desire set temperature. So main objective is to display the continuous all zones temperature.

II. ARCHITECTURE

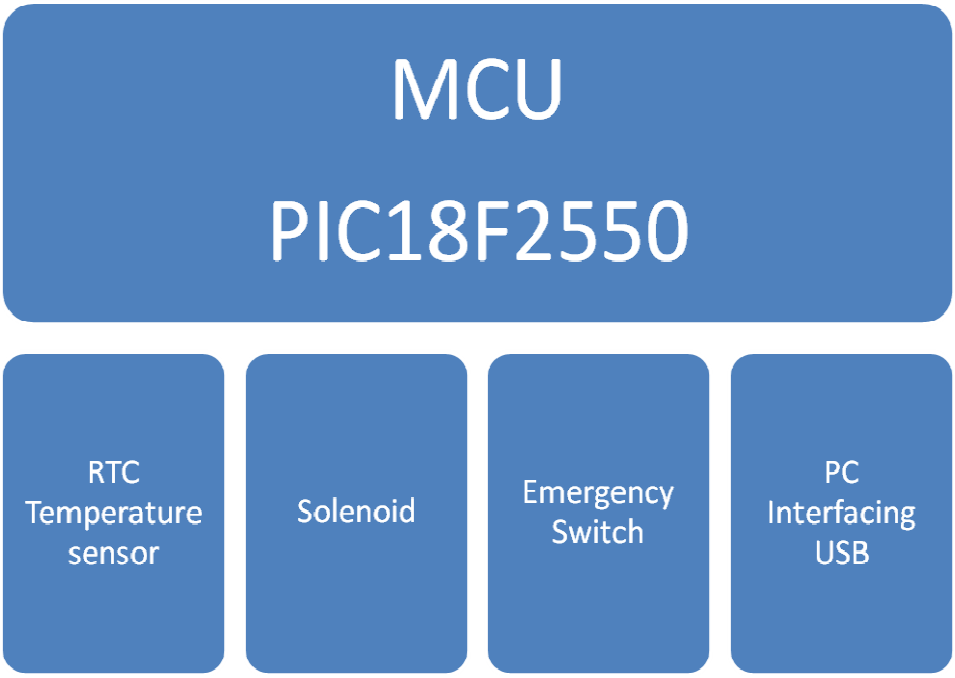

Fig.1: TMC Architecture

A. Temperature

Four channel (Zone) temperature monitor and controlling.

Each channel (Zone) has individual temperature sensor.

All temperature sensor will connect to only ADC channel.

B. Output control (Solenoid)

It is a controlling the fan speed according to temperature or other controlled like solenoid value for oven door look or chamber looks etc.

C. PC Interfacing

TMC unit will sends data to PC or other DATA acquisition card

D. Emergency switch & Rotary DIP

User will open the door by emergency witch at the occurring any emergency

User will set the desire (threshold) temperature of specific zone

III. INTERFACING DIAGRAM

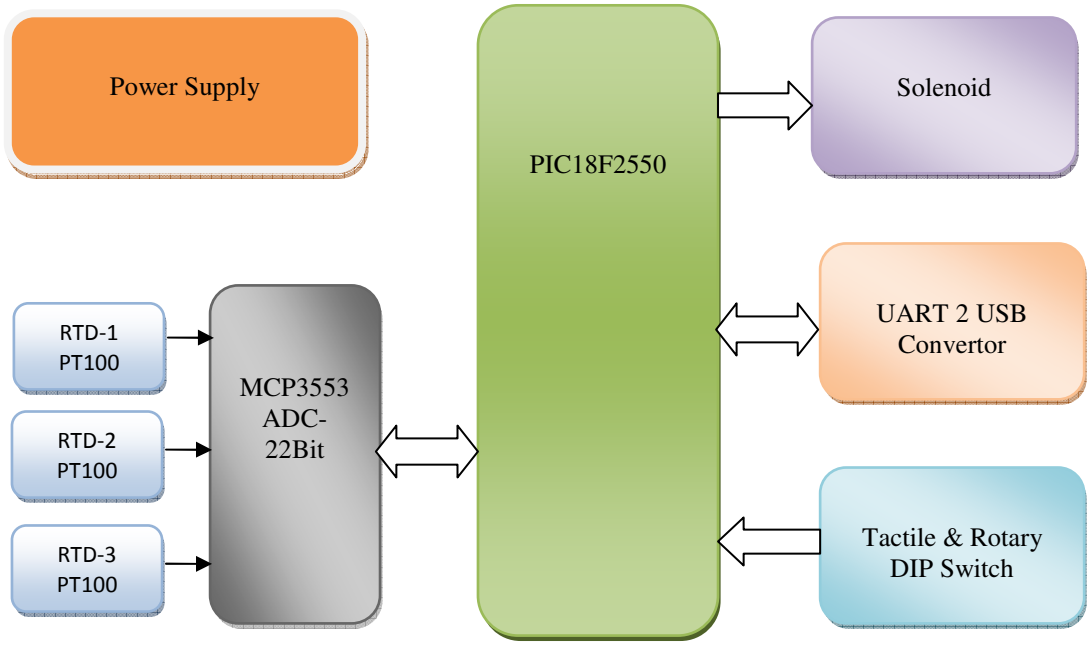

Fig.2: Interfacing diagram

A. RTD calculation

Precision RTD (Resistive Temperature Detector) instrumentation is key for high-performance thermal management applications. This application note shows how to use a high resolution Delta-Sigma Analog-to-Digital Converter, and two resistors to measure RTD resistance ratiometrically. A ±0.1°C accuracy and ±0.01°C measurement resolution can be achieved across the RTD temperature range of -200°C to +800°C with a single point calibration.

This solution uses a common reference voltage to bias the RTD and the ADC which provides a ratio-metric relation between the ADC resolution and the RTD temperature resolution. Only one biasing resistor, RA, is needed to set the measurement resolution ratio Software

$$R_{RTD} = R_A\left(\frac{Code}{2^{n-1} - Code}\right)$$

Where:

Code	=	ADC output code
R_A	=	Biasing resistor
n	=	ADC number of bits (22 bits with sign, MCP3551)

For instance, a 2V ADC reference voltage (VREF) results in a 1 µV/LSb (Least Significant bit) resolution. Setting RA = RB = 6.8 k□ provides 111.6 µV/°C temperature coefficient (PT100 RTD with 0.385□/°C temperature coefficient). This provides 0.008°C/LSb temperature measurement resolution for the entire range of 20□ to 320□ or -200°C to +800°C. A singlepoint calibration with a 0.1% 100□ resistor provides ±0.1°C accuracy.

B. Voltage across RTD

$$V_{RTD} = V_{REF}\left(\frac{R_{RTD}}{R_A + R_{RTD}}\right) = V_{REF}\left(\frac{Code}{2^{n-1}}\right)$$

Where:

V_{RTD} (V)	=	RTD voltage
V_{REF} (V)	=	Reference Voltage
Code	=	ADC output code
n	=	ADC number of bits (22 bits with sign, MCP3551)

C. RTD Resistance & ADC code relations

$$R_{RTD} = R_A\left(\frac{Code}{2^{n-1} - Code}\right)$$

IV. CODE SNIPPET

```
/* File:   main.c
 *Author: Mahesh
 * Created on February 19, 2015, 5:04 PM
 */
#include <stdio.h>
#include <stdlib.h>
#include <p18f2550.h>
#include "gpio.h"
#include "mcp3553.h"
#include "timer.h"
#include "main.h"
#include <usart.h>
   static uint8_t  gFirmwarestat;
   void main(void) {

      SIGNED_DWORD_VAL readadcvalue;
```



```
    int32_t adcAvgValue;
    int32_t ret;
    float_t temperature =0;
    char dataarray[32] = {0};

    gFirmwarestat = POWER_UP_INIT;
    GPIO_vInit();
    TIMERInit();
    MCP3553_Init();
    setFirmwareStat(IDLE_MODE);

    while (1){

    switch(gFirmwarestat)
    {
       case IDLE_MODE:
           MCP3553_msdelay(10);
          break;
       case SAMPLING_MODE:
          /*Toggle at evey sample*/
          ADC_LED_TOGGLE();
          readadcvalue = MCP3553_GetValue();
          ret = MCP3553_SetBuffer(readadcvalue.Val);
          if (BUFFER_SIZE == ret)
          {
             setFirmwareStat(TEMPERATURE_MESURING);
          }else {
             setFirmwareStat(IDLE_MODE);
          }
          break;
       case TEMPERATURE_MESURING:
          adcAvgValue = MCP3553_adcAverage();
          temperature = MCP3553_getTemp(adcAvgValue);

          setFirmwareStat(SOLENOID_CONTROLE);
       break;
    case SOLENOID_CONTROLE:
          MCP3553_ControleLock(temperature);
          setFirmwareStat(IDLE_MODE);
       break;
    case EMERGANCY_OPEN:

       break;

    default:
       break;

     }

   }

}

void  setFirmwareStat (uint8_t state )
{
   gFirmwarestat = state;
}
```



```
uint8_t getFirmwareStat (void)
{
   return gFirmwarestat;

}
```

References

[1] PIC18F2550 Family User's Guide
[2] 22-Bit ADC Datasheet
[3] Synchronous step down convertor user guide
[4] UART to USB Datasheet
[5] 51891a.pdf – reference design for RTD calculation

