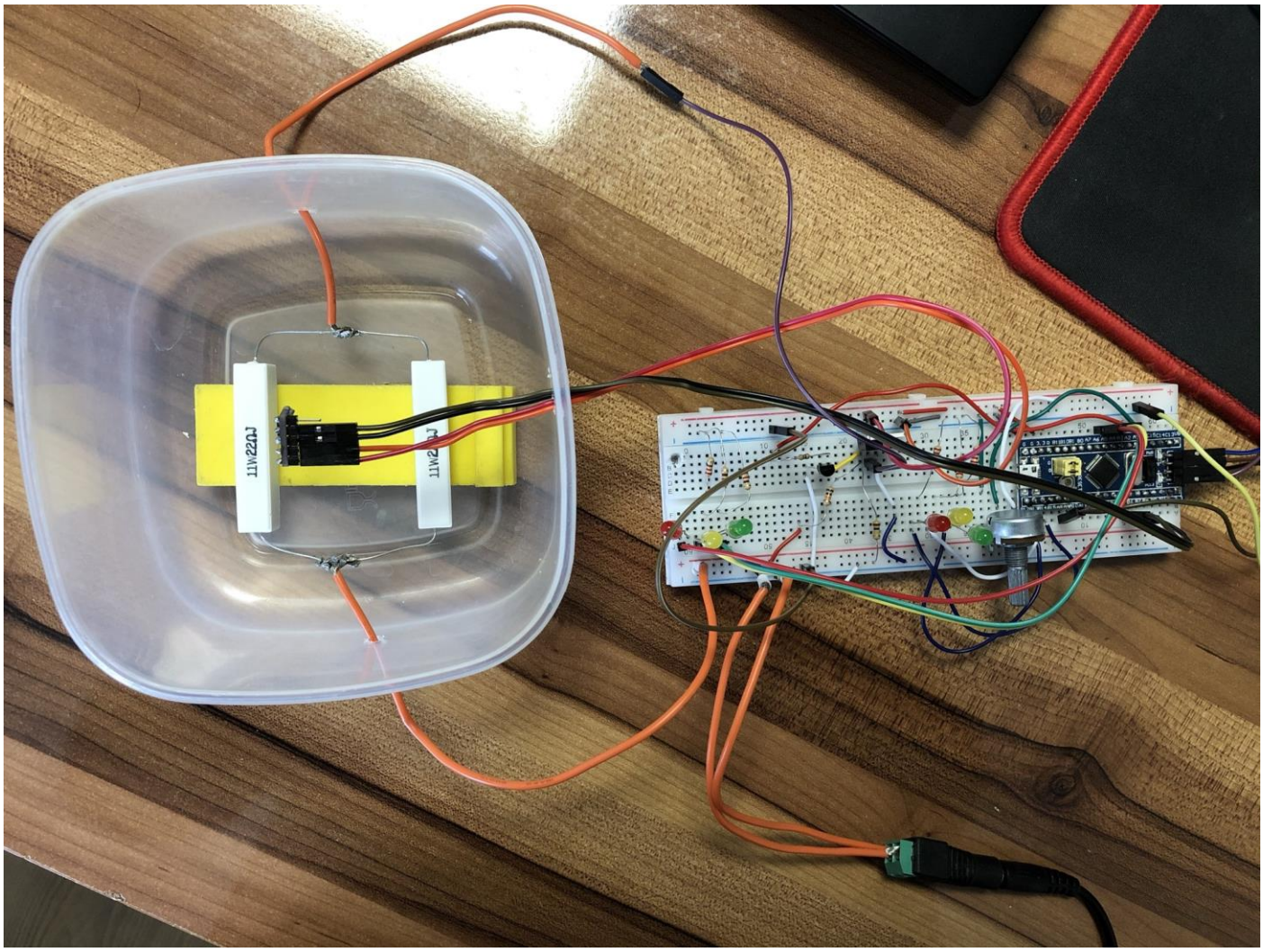
Control & Automation Engineering Department

KON309E Microcontroller Systems

FINAL PROJECT

Name of the project	Implementation of a temperature control system.
Prepared by:	Mehmet Ali ARSLAN 040170402 Control and Automation Engineering

PHOTO OF CIRCUIT：It seems complicated but connections are true.



YOUTUBE LINK: https://youtu.be/Wg32paVgFhA

→ Configurations:

- **<u>TIMER and its interrupt</u>** → I used TIM2 for my timer and PWM purpose. With my parameters I have got 5Hz = 0.2 second of period for interrupt. Also I have a TIM2_CCR2 for giving PWM input to the system, 50000 (equal to period which means max) in default because when we give signal to the system, the current goes to ceramic resistors is zero so its reverse. My pwm output pin is A1.

```
void TIM2_config(void)    // TIMER configuration for TIM2
{
  TIM_TimeBaseStructure.TIM_Period = 49999;
  TIM_TimeBaseStructure.TIM_Prescaler = 287;            // 72M /288*50K = 5Hz = 0.2 second period.
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  TIM_Cmd(TIM2, ENABLE); //Enabling the timer

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; // Our PWM for input to the system
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

  TIM_OCInitStructure.TIM_Pulse = 50000;      // When input is max, the current goes to the ceramic resistors = 0.
  TIM_OC2Init(TIM2, &TIM_OCInitStructure); // for input voltage (pin A1)

}
```

- In timer interrupt, I toggle the sendtime variable for usage in while loop and NVIC is as follows:

```
void TIM2_IRQHandler(void){    //TIMER Function for sending data to pc as period of 0.2 seconds.

     if((TIM_GetITStatus(TIM2, TIM_IT_Update) == SET) ){  // 5 Hz = 0.2 seconds of period.

       sendtime=!sendtime; //this variable changes every 0.2 second.

     TIM_ClearITPendingBit(TIM2, TIM_IT_Update);    //we need to clear line pending bit manually
   }
    }
void NvicConfig(void)  //NVIC Configuration
{
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn; //Choosing timer2 for NVIC
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_Init(&NVIC_InitStructure);
}
```

- **<u>USART Configuration</u>** → I configurated the USART as follows and my TX pin is A9.

```
void USART_config(void)    // USART configuration
{
  // USART settings
  USART_InitStructure.USART_BaudRate = 9600;      //Our Baud rate.
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);
  USART_Cmd(USART1, ENABLE);
}
void UART_Transmit(char *string){  //Our data processing function for sending temp value to PC.
  while(*string)
  {
  while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
  USART_SendData(USART1,*string++);
  }
}
```

- **<u>ADC Configuration</u>** → ADC configuration is for the potatentiometer as our reference temperature value. ADC pin is A0 pin.

```
void ADC_config(void)    // ADC configuration
{
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;    //For continious conversation of pot.Value.
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_7Cycles5);
  ADC_Cmd(ADC1, ENABLE);

  ADC_ResetCalibration(ADC1);
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
  // Start the conversion
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
```

- **<u>I2C Configuration and Temperature sensor function</u>** → We should config the I2C for our temperature sensor as follows:

```
void I2C_config(void)    // I2C configuration
{
  // I2C configuration
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_ClockSpeed = 100000;
  I2C_Init(I2C1, &I2C_InitStructure);
  I2C_Cmd(I2C1, ENABLE);
}
```

- Also my temperature read function as follows:

```
float Temp_read(void)  //Our temperature reading function.
{
  uint8_t dataBuffer[2]={0,0};    // This is for temperature.
  uint16_t Temperature;

  // Wait if busy
  while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

  // Generate START condition
  I2C_GenerateSTART(I2C1, ENABLE);
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB));

  // Send device address for read
  I2C_Send7bitAddress(I2C1, 0x90, I2C_Direction_Receiver);
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

  // Read the first data
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
  dataBuffer[0] = I2C_ReceiveData(I2C1);

  // Disable ACK and generate stop condition
  I2C_AcknowledgeConfig(I2C1, DISABLE);
  I2C_GenerateSTOP(I2C1, ENABLE);

  // Read the second data
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
  dataBuffer[1] = I2C_ReceiveData(I2C1);

  // Disable ACK and generate stop condition
  I2C_AcknowledgeConfig(I2C1, DISABLE);
  I2C_GenerateSTOP(I2C1, ENABLE);

  // Calculate temperature value in Celcius
  Temperature = (uint16_t)(dataBuffer[0] << 8) | dataBuffer[1];
  return(float)(Temperature >> 7)*0.5; //For reading 0.5 C resolution.
}
```

- **Getting reference value and temperature sending algorithm in while loop →**

```
potValue = ADC_GetConversionValue(ADC1); // getting the value from our potentiometer. Max pot. value is 4096, i checked from value viewer of stmstudio.
TempReference = potValue/68.2;  // Calibrated for when potentiometer is max, its value is 60. I checked from value viewer of stmstudio.

if(sendtime==true){  //This is for sending data to PC algorithm. sendtime=true for interval of 0.2 seconds but we need to assume it false at the end of the process.
  Temp = Temp_read();  //Reading the temperature data from our sensor with function.
  sprintf(data,"%0.2f\r",Temp);
  UART_Transmit(data);
  sendtime=false;
}
```

- **PIN (GPIO) Configuration →**

```
void GPIO_config(void)      //GPIO configuration
{
  // Configure analog input
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;  // Configuring pin A0 for analog input (potentiometer).
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  // configure REFERENCE leds' output
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;  //REFERENCE LEDS
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Push-pull mode
  GPIO_Init(GPIOA, &GPIO_InitStructure); //A port

  // configure OVERSHOOT leds' output
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;  //OVERSHOOT LEDS
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Push-pull mode
  GPIO_Init(GPIOB, &GPIO_InitStructure); //B port

  // configure input to system
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;  //input to the system pin
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // Alternate Function Push-pull mode
  GPIO_Init(GPIOA, &GPIO_InitStructure); //A port

  // Configue UART TX - UART module's RX should be connected to this pin
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  // Configure pins (SCL, SDA)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;  //Our pins for recieve data from temperature sensor.
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;   // AF open drain
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}
```

A3 → Green Reference Led A4 → Yellow Reference Led A5 → Red Reference Led

B13 → Green O.S. Led B14 → Yellow O.S. Led B15 → Red O.S. Led

- **System of implementation:** I used switch cases for each situtation according to **LOW =30 (30-40 degrees, state=1)**, **MEDIUM=40 (40-50 degrees, state=2)** and **HIGH=50 (50-60 degrees, state=3)** reference values.

Default state → case:0 → All leds are off and PWM max. (pwr. to R is 0)
Low ref. state → case:1 → Green ref. led is on. And until Reference-2.5 degrees,PWM is 0 for give max power to Resistors. If reference=temperature, steadyreached variable =1 and if temperature goes down due to opening the box, pwm max, else pwm min.
Medium ref. state → case:2 → Yellow ref. led is on. And until Reference-2.5 degrees,PWM is 0 for give max power to Resistors. If reference=temperature, steadyreached variable =1 and if temperature goes down due to opening the box; pwm max, else pwm min.

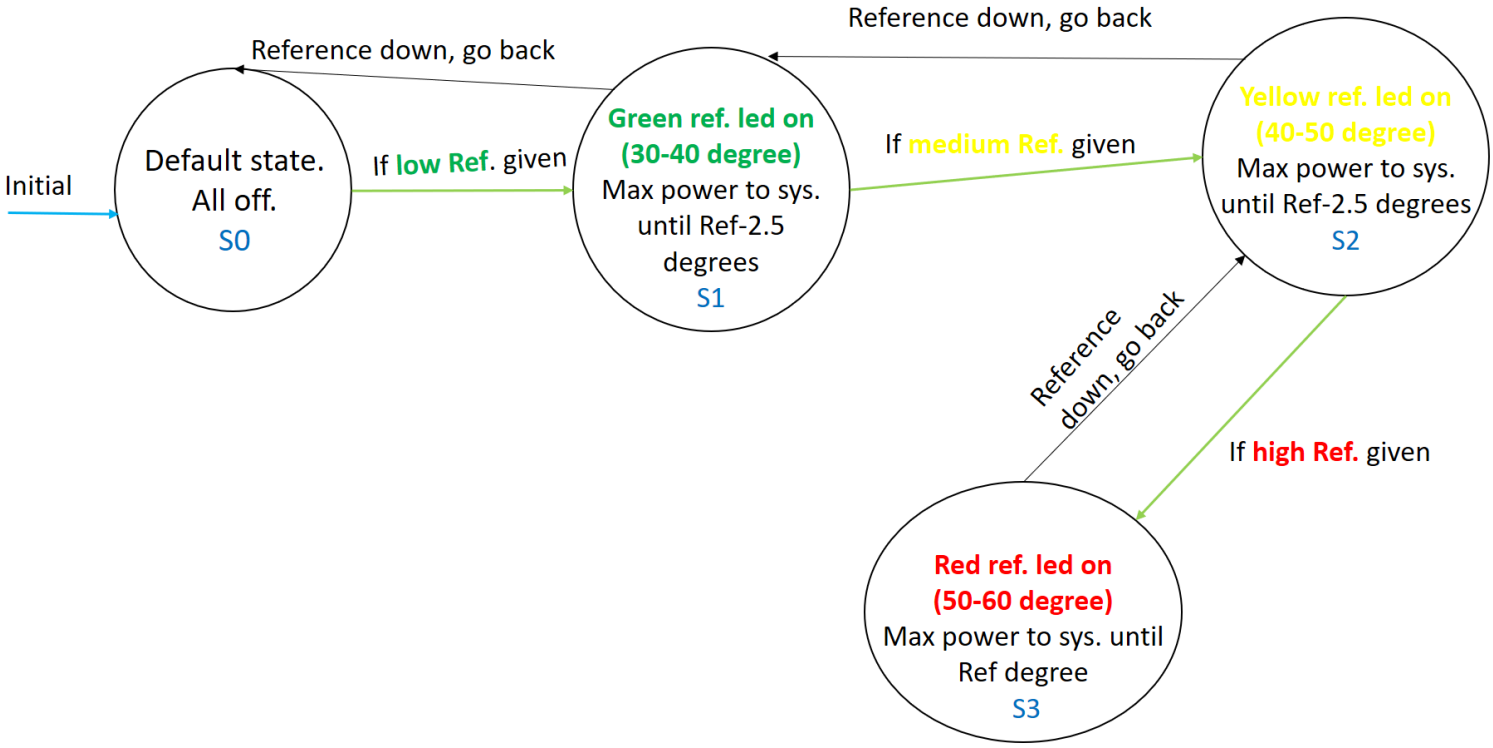
High ref. state → case:3 → Red ref. led is on. And until Reference degree, PWM is 0 for give max power to Resistors. If reference=temperature, steadyreached variable =1 and if temperature goes down due to opening the box, pwm max, else pwm min.

Common spec. for every case : If system has O.S. until 0.02*Ref, Green led is on

If system has an overshoot between 2% to 10%, yellow LED will be ON.

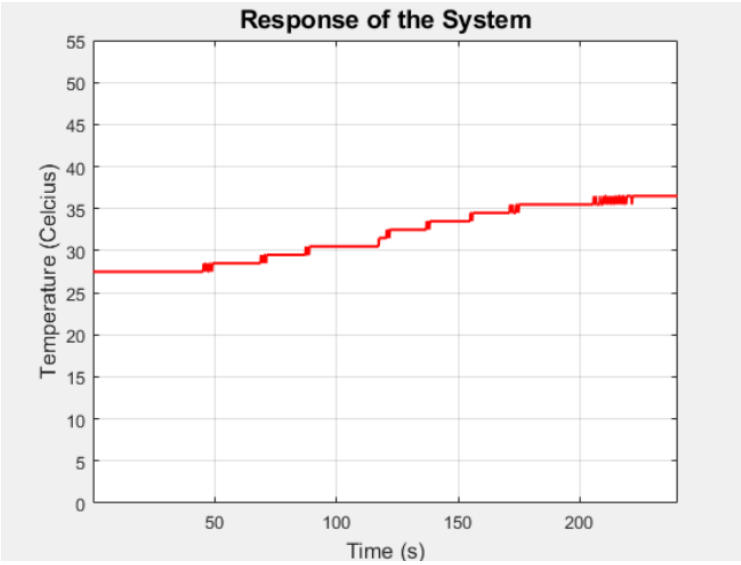
If system has an overshoot more than 10%, red LED will be ON.

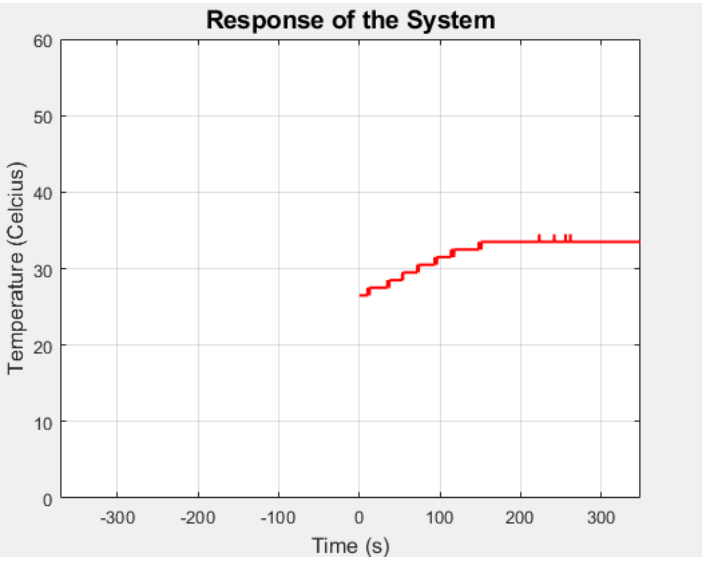
Finite state machine diagram:



- **APPLICATION EXAMPLES:**

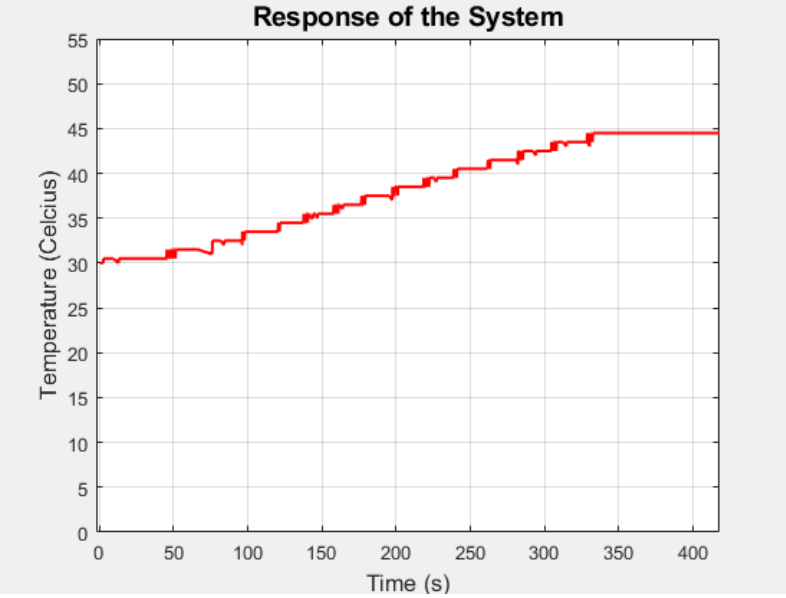
→ System output when LOW ref. given as 36 degree and 34 degree (checked from STMStudio) as follows:





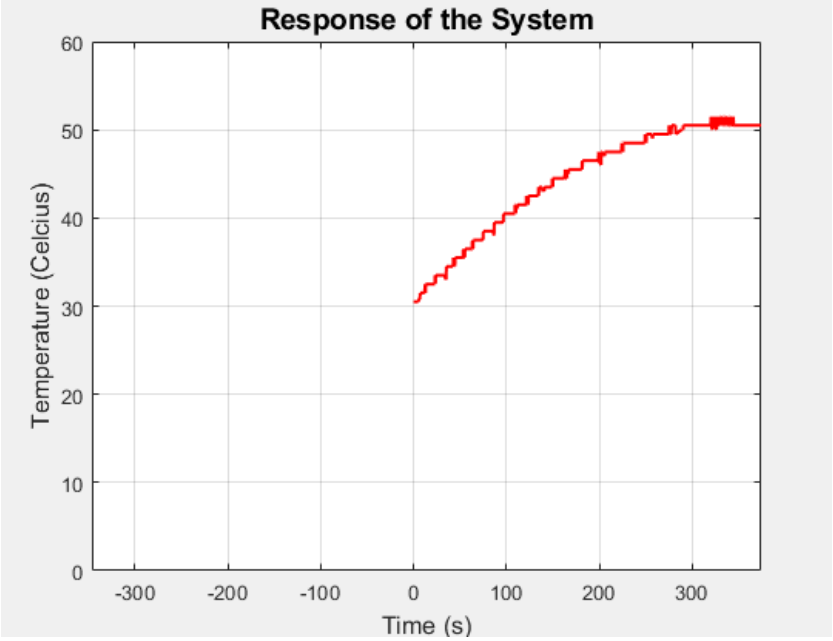
You can see that nearly zero overshoot. Green O.S. led on. But settling time is very high because of this is temperature application and resistors cant heat the plastic container quickly. I am cutting the given power to the resistors before 2.5 degrees to reference as resistors are still heating the plastic container.

→ System output when MEDIUM ref. given as 45≅ 44.5 degree (checked from STMStudio) as follows:



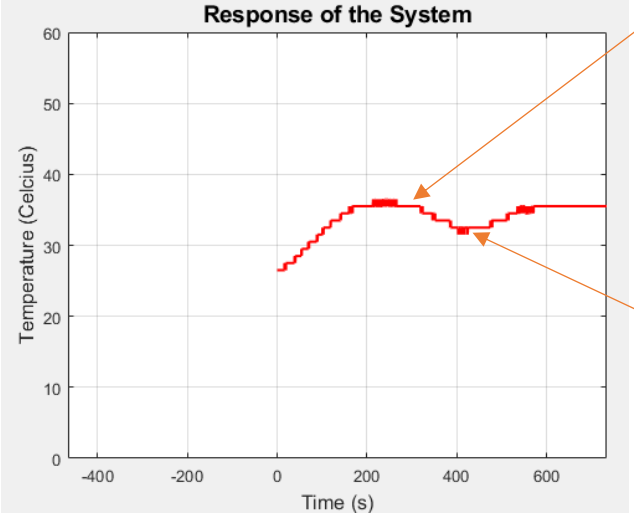
You can see that there is no overshoot and system steady at **44.5≅ 45 degree.** And Green O.S. led is on.

→ System output when HIGH ref. given as 51≅ 51.5 degree (checked from STMStudio) as follows:



Here, I put the temperature sensor very close to the resistors and the settling time is reduced as expected. Green O.S. led on and system is nearly steady at **51≅ 51.5 degree.**

→ **Now lets give some disturbance to the system as opening the cover for a while and close again for LOW Ref. 35 ≅ 35.5 degrees as follows:**

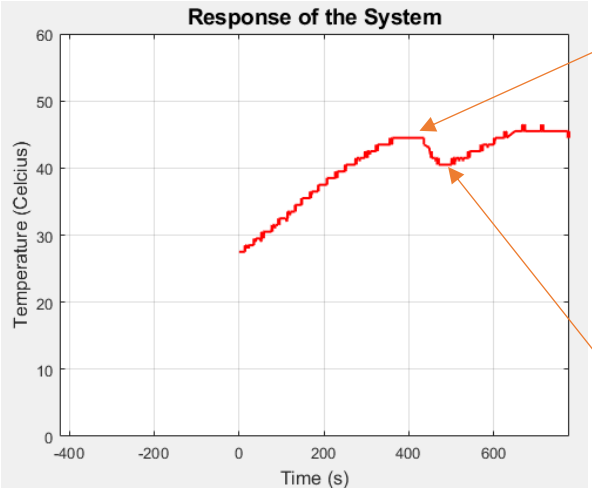


I opened the cover of the box here.

You can see that my system works correctly.

I closed the cover of the box here.

→ **Now lets give some disturbance to the system as opening the cover for a while and close again for MEDIUM Ref. 45 ≅ 45.5 degrees as follows:**

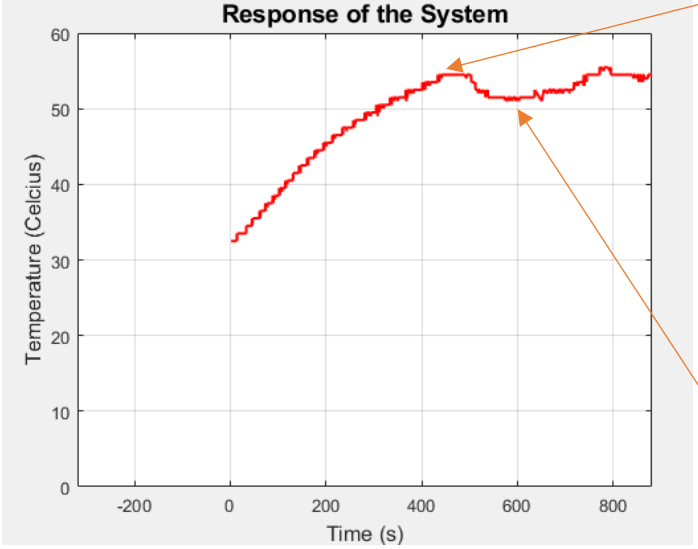


I opened the cover of the box here.

You can see that my system works correctly.

I closed the cover of the box here.

→ **Now lets give some disturbance to the system as opening the cover for a while and close again for HIGH Ref. 55 ≅ 55.5 degrees as follows:**



I opened the cover of the box here.

You can see that my system works correctly.

I closed the cover of the box here.

→ In addition, the other and correct way to implementing this control system is giving the 1V input with PWM to the system and see the response of system so this is step response. Then we could get zeta and wn values from probably second order transfer function response. We could create the transfer function model of this system using this way. Moreover we could find one of the appropriate P-PI-PD-PID controller parameters and implement this controller in discrete time or continious time in Keil with C language. Finally we could give the control signal to our system according to reference input. So we could control this system with analytical way.

→ **ADDITIONAL EXPLANATION :** In my youtube video, for medium and high reference parts, I opened the plastic container's cover and when temperature decreases after steady state, max current and voltage (10V) went to my resistors as you can see in my code. But naturally, as the air goes up when the lid is opened, even if the resistors work at full capacity, they could not heat my environment and the temperature dropped a little. I am sure that my algorithm works correctly because i checked with multimeter. In video I didn't close the cover again but in above applications, I closed. **Other code explanations are in my code as a comment.**

MAIN CODE:

```
#include "stm32f10x.h"
#include "delay.h"
#include <stdbool.h>
#include <stdio.h>

void UART_Transmit(char *string){  //Our data processing function for sending temp value to PC.
  while(*string)
  {
  while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
  USART_SendData(USART1,*string++);
  }
}

float Temp_read(void)  //Our temperature reading function.
{
  uint8_t dataBuffer[2]={0,0};    // This is for temperature.
  uint16_t Temperature;

  // Wait if busy
  while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));

  // Generate START condition
  I2C_GenerateSTART(I2C1, ENABLE);
  while (!I2C_GetFlagStatus(I2C1, I2C_FLAG_SB));

  // Send device address for read
  I2C_Send7bitAddress(I2C1, 0x90, I2C_Direction_Receiver);
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

  // Read the first data
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
  dataBuffer[0] = I2C_ReceiveData(I2C1);

  // Disable ACK and generate stop condition
  I2C_AcknowledgeConfig(I2C1, DISABLE);
  I2C_GenerateSTOP(I2C1, ENABLE);

  // Read the second data
  while (!I2C_CheckEvent(I2C1, I2C_EVENT_MASTER_BYTE_RECEIVED));
  dataBuffer[1] = I2C_ReceiveData(I2C1);

  // Disable ACK and generate stop condition
  I2C_AcknowledgeConfig(I2C1, DISABLE);
  I2C_GenerateSTOP(I2C1, ENABLE);

  // Calculate temperature value in Celcius
  Temperature = (uint16_t)(dataBuffer[0] << 8) | dataBuffer[1];
  return(float)(Temperature >> 7)*0.5;        //For reading 0.5 C resolution.
}
```

```
GPIO_InitTypeDef GPIO_InitStructure; // Peripheral libraries
EXTI_InitTypeDef EXTI_InitStructure; //External interrupt library
NVIC_InitTypeDef NVIC_InitStructure; //NVIC Library
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //Timer library
TIM_OCInitTypeDef TIM_OCInitStructure; //Oc library
ADC_InitTypeDef ADC_InitStructure; //ADC library
USART_InitTypeDef USART_InitStructure; //USART Library
I2C_InitTypeDef I2C_InitStructure;  //I2C Library

void GPIO_config(void);    //My configuration functions
void ADC_config(void);
void TIM2_config(void);
void NvicConfig(void);
void USART_config(void);
void I2C_config(void);
float Temp_read(void);

int LOW = 30;   //LOW STATE IS BETWEEN 30-40 DEGREES.
int MEDIUM = 40;  // MEDIUM STATE IS BETWEEN 40-50 DEGREES.
int HIGH = 50;   // HISH STATE IS BETWEENN 50-60 DEGREES.

static int state=0;     //Our state variable.

static int steadyreached = 0;     //Knowing the steady reached.

uint32_t potValue;  //Our potentiometer value.

static int TempReference;   // Our setted temperature reference

float Temp;   //Our temperature value.

bool sendtime = false;    //Sending time value

char data[20];          // This is for ADC.



void TIM2_IRQHandler(void){   //TIMER Function for sending data to pc as period of 0.2 seconds.

    if((TIM_GetITStatus(TIM2, TIM_IT_Update) == SET) ){  // 5 Hz = 0.2 seconds of period.

      sendtime=!sendtime; //this variable changes every 0.2 second.

    TIM_ClearITPendingBit(TIM2, TIM_IT_Update);   //we need to clear line pending bit manually
  }
   }
int main(void) {

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE); //A port clock enabled
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE); //B port clock enabled
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,  ENABLE); //AFIO clock enabled
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); // Timer clock enabled for send data
  RCC_ADCCLKConfig(RCC_PCLK2_Div6); // Setting Adc clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);  // ADC clock
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE); // USART CLOCK enabled
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE); //I2C Clock enabled

  delayInit(); //delay initialize

  GPIO_config(); //Init. of configurations.
  ADC_config();
  NvicConfig();
  TIM2_config();
  USART_config();
  I2C_config();
```

```
120     while(1)
121     {
122
123       potValue = ADC_GetConversionValue(ADC1); // getting the value from our potentiometer. Max pot. value is 4096, i checked from value viewer of stmstudio.
124       TempReference = potValue/68.2;  // Calibrated for when potentiometer is max, its value is 60. I checked from value viewer of stmstudio.
125
126       if(sendtime==true){  //This is for sending data to PC algorithm. sendtime=true for interval of 0.2 seconds but we need to assume it false at the end of the process.
127         Temp = Temp_read();  //Reading the temperature data from our sensor with function.
128         sprintf(data,"%0.2f\r",Temp);
129         UART_Transmit(data);
130         sendtime=false;
131       }
```

```
133         switch (state)
134         {
135           case 0:      //THIS IS DEFAULT STATE ALL SYSTEMS ARE OFF.
136             GPIO_ResetBits(GPIOA,GPIO_Pin_3 | GPIO_Pin_4| GPIO_Pin_5);
137             GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_14| GPIO_Pin_15);
138             TIM2->CCR2 = 50000;           // When input is max, the current goes to the ceramic resistors = 0.
139             if(TempReference>=LOW)    //If reference >=LOW (30) go to low state = 1
140             {
141               steadyreached=0;  //this variable is resetted.
142               state=1;  //go to low temp. state
143               break;
144             }
147         case 1:                                             //THIS IS LOW TEMPERATURE STATE 30-40 degrees.
148           GPIO_ResetBits(GPIOA,GPIO_Pin_3 | GPIO_Pin_4| GPIO_Pin_5);
149           GPIO_SetBits(GPIOA,GPIO_Pin_3);                   //GREEN  REFERENCE LED IS ON
150
151           TIM2->CCR2 = 50000;    //When other conditions are not satisfied. Current goes to resistors is zero.
152
153           if(TempReference-2.5>=Temp)    //Heating the resistors until the reference-2.5 degree.
154           {
155            TIM2->CCR2 = 0;
156           }
157           if(TempReference+0.5 == Temp)    //Knowing Steady state reached
158           {
159             steadyreached = 1;
160           }
161           if((steadyreached==1)&&(TempReference>Temp))      //If steady state reached and our temperature is lower than reference, give max power to resistors.
162           {
163             TIM2->CCR2 = 0;
164           }
165           if((steadyreached==1)&&(TempReference<Temp))       //If steady state reached and out temprature is higher than reference, give min power to resistors.
166           {
167             TIM2->CCR2 = 50000;
168           }
169
170           if(TempReference*0.02>=Temp-TempReference && Temp-TempReference>=0)
171           {
172              GPIO_ResetBits(GPIOB,GPIO_Pin_14| GPIO_Pin_15); //other leds are off
173              GPIO_SetBits(GPIOB,GPIO_Pin_13);                //GREEN O.S. LED ON
174           }
175           if(TempReference*0.1>=Temp-TempReference && Temp-TempReference>TempReference*0.02)
176           {
177             GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_15); //other leds are off
178             GPIO_SetBits(GPIOB,GPIO_Pin_14);                //YELLOW O.S. LED ON
179           }
180           if(TempReference*0.1<Temp-TempReference)
181           {
182           GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_14); //other leds are off
183           GPIO_SetBits(GPIOB,GPIO_Pin_15);                //RED O.S. LED ON
184           }
185
186           if(TempReference>=MEDIUM)
187           {
188             steadyreached=0; //Resetting this variable.
189             state=2;  //go to medium temp. state
190             break;
191           }
192           if(TempReference<LOW)
193           {
194             steadyreached=0; //Resetting this variable.
195             state=0;  //go to default state
196             break;
197           }
198         break;
200         case 2:                                             //THIS IS MEDIUM TEMPERATURE STATE  40-50 degrees.
201
202           GPIO_ResetBits(GPIOA,GPIO_Pin_3 | GPIO_Pin_4| GPIO_Pin_5);
203           GPIO_SetBits(GPIOA,GPIO_Pin_4);                   //YELLOW REFERENCE LED IS ON
204
205           TIM2->CCR2 = 50000;                     //When other conditions are not satisfied. Current goes to resistors is zero.
206
207           if(TempReference-2.5>=Temp)       //Heating the resistors until the reference-2.5 degree.
208           {
209            TIM2->CCR2 = 0;
210           }
211           if(TempReference+0.5 == Temp)     //Knowing Steady state reached
212           {
213             steadyreached = 1;
214           }
215           if((steadyreached==1)&&(TempReference>Temp))    //If steady state reached and our temperature is lower than reference, give max power to resistors.
216           {
217             TIM2->CCR2 = 0;
218           }
219           if((steadyreached==1)&&(TempReference<Temp))     //If steady state reached and out temprature is higher than reference, give min power to resistors.
220           {
221             TIM2->CCR2 = 50000;
222           }
```

```
            if(TempReference*0.02>=Temp-TempReference && Temp-TempReference>=0)
            {
               GPIO_ResetBits(GPIOB,GPIO_Pin_14| GPIO_Pin_15); //other leds are off
               GPIO_SetBits(GPIOB,GPIO_Pin_13);                                        //GREEN O.S. LED ON
            }
            if(TempReference*0.1>=Temp-TempReference && Temp-TempReference>TempReference*0.02)
            {
              GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_15); //other leds are off
              GPIO_SetBits(GPIOB,GPIO_Pin_14);                                         //YELLOW O.S. LED ON
            }
            if(TempReference*0.1<Temp-TempReference)
            {
            GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_14); //other leds are off
            GPIO_SetBits(GPIOB,GPIO_Pin_15);                                         //RED O.S. LED ON
            }

            if(TempReference>=HIGH)
            {
              steadyreached=0; //Resetting this variable.
              state=3; //go to high temp. state
              break;
            }
            if(TempReference<MEDIUM)
            {
              steadyreached=0; //Resetting this variable.
              state=1; //go to low temp. state
              break;
            }
          break;
        case 3:                                                 //THIS IS HIGH TEMPERATURE STATE  50-60 degrees.

          GPIO_ResetBits(GPIOA,GPIO_Pin_3 | GPIO_Pin_4| GPIO_Pin_5);
          GPIO_SetBits(GPIOA,GPIO_Pin_5);                       //RED REFERENCE LED IS ON

          TIM2->CCR2 = 50000;              //When other conditions are not satisfied. Current goes to resistors is zero.

          if(TempReference>=Temp)      //Heating the resistors until reference temperature value.
          {
           TIM2->CCR2 = 0;
          }
          if(TempReference+0.5 == Temp)           //Knowing Steady state reached
          {
            steadyreached = 1;
          }
          if((steadyreached==1)&&(TempReference>Temp))    //If steady state reached and our temperature is lower than reference, give max power to resistors.
          {
            TIM2->CCR2 = 0;
          }
          if((steadyreached==1)&&(TempReference<Temp))    //If steady state reached and out temprature is higher than reference, give min power to resistors.
          {
            TIM2->CCR2 = 50000;
          }

          if(TempReference*0.02>=Temp-TempReference && Temp-TempReference>=0)
          {
             GPIO_ResetBits(GPIOB,GPIO_Pin_14| GPIO_Pin_15); //other leds are off
             GPIO_SetBits(GPIOB,GPIO_Pin_13);                       //GREEN O.S. LED ON
          }
          if(TempReference*0.1>=Temp-TempReference && Temp-TempReference>TempReference*0.02)
          {
            GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_15); //other leds are off
            GPIO_SetBits(GPIOB,GPIO_Pin_14);                        //YELLOW O.S. LED ON
          }
          if(TempReference*0.1<Temp-TempReference)
          {
          GPIO_ResetBits(GPIOB,GPIO_Pin_13 | GPIO_Pin_14); //other leds are off
          GPIO_SetBits(GPIOB,GPIO_Pin_15);                        //RED O.S. LED ON
          }

          if(TempReference<HIGH)
          {
            steadyreached=0; //Resetting this variable.
            state=2;
            break;
          }
        break;


      } //Closing switch
  } //Closing while
} //Closing main
 void GPIO_config(void)     //GPIO configuration
{
  // Configure analog input
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;  // Configuring pin A0 for analog input (potentiometer).
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  // configure REFERENCE leds' output
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_5;  //REFERENCE LEDS
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Push-pull mode
  GPIO_Init(GPIOA, &GPIO_InitStructure); //A port

  // configure OVERSHOOT leds' output
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;  //OVERSHOOT LEDS
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; // Push-pull mode
  GPIO_Init(GPIOB, &GPIO_InitStructure); //B port

  // configure input to system
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;  //input to the system pin
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz; //clock Speed
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; // Alternate Function Push-pull mode
  GPIO_Init(GPIOA, &GPIO_InitStructure); //A port
```

```
  // Configue UART TX - UART module's RX should be connected to this pin
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  // Configure pins (SCL, SDA)
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;  //Our pins for recieve data from temperature sensor.
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_OD;   // AF open drain
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}

void NvicConfig(void)  //NVIC Configuration
{
  NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn; //Choosing timer2 for NVIC
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x00;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x00;
  NVIC_Init(&NVIC_InitStructure);
}

void ADC_config(void)   // ADC configuration
{
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;   //For continious conversation of pot.Value.
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_7Cycles5);
  ADC_Cmd(ADC1, ENABLE);

  ADC_ResetCalibration(ADC1);
  while(ADC_GetResetCalibrationStatus(ADC1));
  ADC_StartCalibration(ADC1);
  while(ADC_GetCalibrationStatus(ADC1));
  // Start the conversion
  ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}
void TIM2_config(void)    // TIMER configuration for TIM2
{
  TIM_TimeBaseStructure.TIM_Period = 49999;
  TIM_TimeBaseStructure.TIM_Prescaler = 287;            // 72M /288*50K = 5Hz = 0.2 second period.
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

  TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
  TIM_Cmd(TIM2, ENABLE); //Enabling the timer

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; // Our PWM for input to the system
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;

  TIM_OCInitStructure.TIM_Pulse = 50000;      // When input is max, the current goes to the ceramic resistors = 0.
  TIM_OC2Init(TIM2, &TIM_OCInitStructure); // for input voltage (pin A1)

}

void USART_config(void)   // USART configuration
{
  // USART settings
  USART_InitStructure.USART_BaudRate = 9600;     //Our Baud rate.
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Tx;
  USART_Init(USART1, &USART_InitStructure);
  USART_Cmd(USART1, ENABLE);
}

void I2C_config(void)   // I2C configuration
{
  // I2C configuration
  I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStructure.I2C_OwnAddress1 = 0x00;
  I2C_InitStructure.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
  I2C_InitStructure.I2C_ClockSpeed = 100000;
  I2C_Init(I2C1, &I2C_InitStructure);
  I2C_Cmd(I2C1, ENABLE);
}
```

THE END OF THE REPORT. THANK YOU VERY MUCH FOR EVERYTHING…