***ESD-RTS MINI PROJECT REPORT***

**REAL TIME THERMAL MONITORING AND LOGGING SYSTEM**

**Sixth Semester Electronics and Communication Engineering**

*Submitted by*

**Student Name.: Manya Singh**

**Reg. No.: 170907394**

**Student Name.: Zaki Sadique**

**Reg. No.: 170907584**

**Student Name.: Sushant Lenka**

**Reg. No.: 170907474**

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**MANIPAL INSTITUTE OF TECHNOLOGY**

(A Constituent Institution of Manipal Academy of Higher Education)

MANIPAL – 576104, KARNATAKA, INDIA

**MARCH/APRIL 2019**

1. **ABSTRACT**

Thermal power plants have a multitude of systems that need a very specific range of temperature and humidity in order to function. A single deviation into suboptimal temperatures can disrupt the entire plant’s yield. This is a real time system that helps constantly keep check on a fluctuating industrial unit, by allowing all systems to work only when the fluctuations are below a specified threshold. This involves the use of a dynamic prioritising software which can change the order in which it processes tasks based on the need of the time and temperature conditions. A simple function que-scheduling approach is used, with queue-based priority line up. A handy addition to this project is the logging of temperature and humidity data to a website which enables maintenance of these plants by the touch of a button, remotely from anywhere in the world

1. **BLOCK DIAGRAM**

Insert image here

1. **COMPONENTS INTERFACED**
2. ARM Microcontroller
3. Temperature and humidity Sensor (LM35)
4. Wi-Fi Module
5. LEDs
6. DC Motor
7. Buzzer
8. Hardware interrupt buttons
9. **METHODOLOGY**

* The first area of this project involves dealing with the temperature and humidity sensors. After being interfaced with STM32 they give a binary value of the actual temperature and humidity values. The accepted values we’re looking at is any fluctuations between *2.5 to 5 degrees Celsius, and 8-12 atmospheric bars*. This task of checking the conditions and comparing it with the threshold then becomes the task of highest priority. No other task can be performed in the power plant if the fluctuations are above these specifications.
* Once it clears this, we reach the main power plant systems. To model these systems, we have created a simple three task equivalent where, on pressing a specific button you can activate the respective task i.e.- LED up count, LED down count, motor rotation and a buzzer. Four buttons have been installed to enable these tasks.
* These tasks although have a chain of priorities which they must follow as well. There are 3 main chains:  
  *> buttons pressed between 0 and 5 seconds* – (LED up, LED down, Motor, Buzzer)  
  > *buttons pressed between 5 and 10 seconds* – (LED down, Motor, Buzzer, LED up)  
  > *buttons pressed between 10 and 15 seconds* – (Motor, Buzzer, LED up, LED Down)
* The data of the temperature and humidity as is being sensed and compared is also parallelly being uploaded to a website where we can access it. The status of all the devices i.e LED, Buzzers etc can also be monitored on the website through the ease of access GUI made. The Wi-Fi module which is interfaced to STM32 allows the upload and downloading of this data.
* The issue we faced while simulating this was the addition of multiple priorities and how to allow the temp and humidity sensors to maintain highest priority while only creating dynamics for the lower priorities. Another issue was making sure all data is sensed, uploaded and priorities are set within the time frame of a single delay (one second) before it changes the next second, and making sure the respective button’s task also gets carried out.

1. **CODE**

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

#define int time=0

**// creating the node for the queue**

struct node

{

int data;

struct node \*next;

};

struct node\* front = NULL;

struct node\* rear = NULL;

struct node\* temp;

**// Functions to implement function queue scheduling**

int isEmpty()

{

if(rear == front == -1)

return 1;

else

return 0

}

int isFull()

{

if(front = (rear+1)%MAX)

return 1;

else return 0;

}

**// Used to add the functions to the queue**

int enQ(int \*p)

{

if (isEmpty()){

temp = (struct node\*)malloc(sizeof(struct node));

temp->data =(\*p);

temp->next = NULL;

front = temp;

rear = temp;

else

temp =(struct node\*) malloc (sizeof( struct node));

temp -> data = (\*p);

temp -> next = NULL;

rear -> next = temp;

rear = temp;

return;

}

}

**// Deciding which set of priorities to use based on time frame of button interrupt**

int setPriority(int t)

{

if(t%15 <5){ // First set of priorities

enQ(\*fn\_motorclock);

enQ(\*fn\_motoranticlock);

enQ(\*fn\_LED);

enQ(\*fn\_buzzer);

}

else if (5<t%15<10){ //Second set of priorities

enQ(\*fn\_motoranticlock);

enQ(\*fn\_LED);

enQ(\*fn\_buzzer);

enQ(\*fn\_motorclock);

}

else { // Third set of priorities

enQ(\*fn\_LED);

enQ(\*fn\_buzzer);

enQ(\*fn\_motorclock);

enQ(\*fn\_motoranticlock);

}

}

**// delay funtion**

void delay(uint32\_t dly){

while(dly--);

}

**// To drive motor clockwise**

void drivingmotorclock(){

HAL\_GPIO\_WritePin(GPIOC,GPIO\_PIN\_0,GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(GPIOC,GPIO\_PIN\_1,GPIO\_PIN\_RESET);

delay(4000000\*5);

HAL\_GPIO\_TogglePin(GPIOC,GPIO\_PIN\_0);

}

**// To drive motor anticlockwise**

void drivingmotoranticlock(){

HAL\_GPIO\_WritePin(GPIOC,GPIO\_PIN\_2,GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(GPIOC,GPIO\_PIN\_3,GPIO\_PIN\_SET);

delay(4000000\*5);

HAL\_GPIO\_TogglePin(GPIOC,GPIO\_PIN\_3);

}

**// To make the LEDs blink**

void ledblink(){

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_10,GPIO\_PIN\_SET);

delay(4000000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_10,GPIO\_PIN\_RESET);

delay(4000000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_10,GPIO\_PIN\_SET);

delay(4000000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_10,GPIO\_PIN\_RESET);

delay(4000000);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_10,GPIO\_PIN\_SET);

delay(4000000);

}

**// Driver code to make LEDs on/off**

void LEDONOFF(int number){

if(number&0x01)

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,GPIO\_PIN\_SET);

else

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_0,GPIO\_PIN\_RESET);

if(number&0x02)

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,GPIO\_PIN\_SET);

else

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_1,GPIO\_PIN\_RESET);

if(number&0x04)

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,GPIO\_PIN\_SET);

else

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_2,GPIO\_PIN\_RESET);

if(number&0x08)

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_3,GPIO\_PIN\_SET);

else

HAL\_GPIO\_WritePin(GPIOA,GPIO\_PIN\_3,GPIO\_PIN\_RESET);

}

**// To start the built in ADC**

HAL\_ADC\_Start\_DMA (&hadc1, &adc\_buf, 1);

**// variables to be used for temperature sensing**

uint32\_t adc\_buf; // default ADC buffer

int temp; // temperature

**// To get temperature from the sensor**

void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef\* hadc)

{

temp = adc\_buf\*.322; // convert value from adc buffer to temperature in Centigrade

}

**// To start the built in ADC**

HAL\_ADC\_Start\_DMA (&hadc1, &adc\_buf, 1);

**// variables to be used for temperature sensing**

uint32\_t adc\_buf; // default ADC buffer

int temp; // temperature

**// To get temperature from the sensor**

void HAL\_ADC\_ConvCpltCallback(ADC\_HandleTypeDef\* hadc)

{

temp = adc\_buf\*.322; // convert value from adc buffer to temperature in Centigrade

}

int main(void)

{

int number=15;

**// To reset of all peripherals, Initializes the Flash interface and the Systick**

HAL\_Init();

**// To configure the system clock**

SystemClock\_Config();

To initialize all configured peripherals

MX\_GPIO\_Init();

while (1)

{

delay();

time++;

void interrupt vHandleLED(void)

{

if (GPIO\_PIN\_0 == 0)

{

setPriority(time);

}

}

void interrupt vHandleBuzzer(void)

{

if (GPIO\_PIN\_1 ==0)

{

setPriority(time);

}

}

void interrupt vHandleMotor\_clock(void)

{

if (GPIO\_PIN\_2 ==0)

{

setPriority(time);

}

}

void interrupt vHandleMotor\_anticlock(void)

{

if (GPIO\_PIN\_3 ==0)

{

setPriority(time);

}

}

for(int i=0;i<=15;i++){

LEDONOFF(i);

delay(3000000);

}

}

}

**// Code required for microcontroller to function**

// @brief System Clock Configuration

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

// To initialize the CPU, AHB and APB busses clocks

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

// Tonitializes the CPU, AHB and APB busses clocks

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSI;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

// GPIO Ports Clock Enable

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

// To configure GPIO pin Output Level

HAL\_GPIO\_WritePin(GPIOA, GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(GPIOC,GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(GPIOB,GPIO\_PIN\_0,GPIO\_PIN\_RESET);

// To configure GPIO pins : PA0 PA1 PA2 PA3

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

// To configure GPIO pins: PC0 PC1 PC2 PC3

GPIO\_InitStruct.Pin = GPIO\_PIN\_0|GPIO\_PIN\_1|GPIO\_PIN\_2|GPIO\_PIN\_3;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);

// To configure GPIO pins: PB10

GPIO\_InitStruct.Pin = GPIO\_PIN\_10;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

}

**// To send post request to the server**

#include <stdio.h> /\* printf, sprintf \*/

#include <stdlib.h> /\* exit \*/

#include <unistd.h> /\* read, write, close \*/

#include <string.h> /\* memcpy, memset \*/

#include <sys/socket.h> /\* socket, connect \*/

#include <netinet/in.h> /\* struct sockaddr\_in, struct sockaddr \*/

#include <netdb.h> /\* struct hostent, gethostbyname \*/

void error(const char \*msg) { perror(msg); exit(0); }

int main(int argc,char \*argv[])

{

**/\* first what are we going to send and where are we going to send it? \*/**

int portno = 80;

char \*host = "api.somesite.com";

char \*message\_fmt = "POST /apikey=%s&command=%s HTTP/1.0\r\n\r\n";

struct hostent \*server;

struct sockaddr\_in serv\_addr;

int sockfd, bytes, sent, received, total;

char message[1024],response[4096];

if (argc < 3) { puts("Parameters: <apikey> <command>"); exit(0); }

**/\* fill in the parameters \*/**

sprintf(message,message\_fmt,argv[1],argv[2]);

printf("Request:\n%s\n",message);

**/\* create the socket \*/**

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0) error("ERROR opening socket");

**/\* lookup the ip address \*/**

server = gethostbyname(host);

if (server == NULL) error("ERROR, no such host");

**/\* fill in the structure \*/**

memset(&serv\_addr,0,sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(portno);

memcpy(&serv\_addr.sin\_addr.s\_addr,server->h\_addr,server->h\_length);

**/\* connect the socket \*/**

if (connect(sockfd,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)) < 0)

error("ERROR connecting");

**/\* send the request \*/**

total = strlen(message);

sent = 0;

do {

bytes = write(sockfd,message+sent,total-sent);

if (bytes < 0)

error("ERROR writing message to socket");

if (bytes == 0)

break;

sent+=bytes;

} while (sent < total);

**/\* receive the response \*/**

memset(response,0,sizeof(response));

total = sizeof(response)-1;

received = 0;

do {

bytes = read(sockfd,response+received,total-received);

if (bytes < 0)

error("ERROR reading response from socket");

if (bytes == 0)

break;

received+=bytes;

} while (received < total);

if (received == total)

error("ERROR storing complete response from socket");

**/\* close the socket \*/**

close(sockfd);

**/\* process response \*/**

printf("Response:\n%s\n",response);

return 0;

}

1. **REFERENCES**1. https://github.com/smart-storm/storm-iot-node

2. https://github.com/maxgerhardt/pio-stm32-with-esp8266-dht11

3. <https://components101.com/lm35-temperature-sensor>