



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES

Habtamu Ashengo

# THE DESIGN OF TEMPERATURE AND HUMIDITY REMOTE MONITORING

## PREFACE

I would like to thank and express my sincere gratitude for my supervisors, Mr. Jani Ahvonen and Mr. Timmo Rinne for giving me a chance to work on this interesting thesis topic. Their knowledge, hard work and dedication have been first class.

Secondly, I would like to thank Mr. Seppo Makinen, Information Technology Department head for facilitating the project and all my teachers for making me a better student throughout my studies.

This thesis would not have happen had it not been for Technobothnia laboratory, hence

I would like to thank VAMK for providing all the necessary equipment for the thesis to be done.

Finally, I would to thank my wife and my parents for your valuable advice and commitments in all aspects during my stay in Finland.

Vaasa, Finland 27/04/2016

Habtamu Gebrewold Ashengo



VAASAN AMMATTIKORKEAKOULU  
UNIVERSITY OF APPLIED SCIENCES  
Bachelor of Engineering

## ABSTRACT

Author	Habtamu Ashengo
Title	The Design of Temperature and Humidity Remote Monitor
Year	2015
Language	English
Pages	35 + 1 Appendices
Name of Supervisor	Jani Ahvonen

---

Currently we are in a time where all equipment have to be tested for different circumstances. For example, if you take a laptop the extreme conditions in which it functions properly need to be known. These extreme conditions could be high temperature and humidity or low temperature and humidity. To test any equipment like a phone, a laptop or any device, a temperature and humidity chamber is needed.

In this thesis the communication between a control box and a main PC was studied. All the command bits sent from both directions were well documented. This communication happened through a RS-232 DB-9 cable and had its own communication protocols. In addition, a remote system in which the operator could monitor the chamber remotely was essential, hence, this thesis covers two software applications in which one reads the temperature and humidity values which are exchanged between the main PC and the control box and displays those values in a graph. The other one is for the outside sensor, DS18B20, which is connected to the GPIO's of the Raspberry Pi and records the values in to a database and displays those values using a web page.

This experiment was done by using a Temperature and Humidity Chamber (EUT), RS-232 cable, Raspberry Pi, a DS18B20 temperature sensor and also by using the Linux operating system.

## CONTENTS

### PREFACE

1	INTRODUCTION .....	10
1.1	Structure of the Thesis .....	10
1.2	Objective of the Project .....	10
1.3	Benefits of the Project.....	10
2	BACKGROUND .....	13
2.1	Raspberry Pi.....	13
2.1.1	History of Raspberry Pi.....	13
2.1.2	Hardware and Software Specifications of Raspberry Pi.....	14
2.1.3	Raspberry Pi GPIO's.....	14
2.2	RS-232 Serial Adapter .....	15
2.2.1	Introduction .....	15
2.2.2	Asynchronous Communication.....	17
2.3	EUT Chamber .....	18
2.3.1	Purpose of Chamber .....	18
3	SNIFFING OF RS-232 COMMUNICATION BETWEEN OLD PC AND CONTROL BOX .....	19
3.1	Communication Protocols.....	19
3.2	How to capture of messages made.....	21
3.3	Interpretation of the Captured Messages .....	23
4	REMOTE MONITORING USING RASPBERRY PI.....	26
4.1	Control Box Reading .....	26
5	CONTROL BOX READING USING QT APPLICATION.....	27
5.1	Programming Flow Chart Reading Control Box Using C++ .....	27
6	TESTING FOR TEMPERATURE AND HUMIDITY BY USING EXTERNAL SENSORS .....	30
6.1	Programming Flow Chart for Reading External Sensor .....	33
7	CONTROL BOX READING USING PYTHON SCRIPT.....	34
7.1	Programming Flow Chart for Reading from Control Box.....	34

8	CONCLUSION .....	35
	8.1 Review of the Project.....	35
	8.2 Future Improvements .....	36
9	ANNEXATION.....	36
10	REFERENCES .....	53

## LIST OF FIGURES AND TABLES

<b>Figure 1</b> Main diagram of the project design (Technobothnia Electrical Department).....	12
<b>Figure 2</b> Raspberry Pi Model B+ 512 MB (Linux User and Developer).....	13
<b>Figure 3</b> Hardware and Software specification of Raspberry Pi model B+(GE Tech) .....	14
<b>Figure 4</b> GPIO layout of model B+ RPi (Raspberry Pi Foundation) .....	15
<b>Figure 5</b> RS-232 to USB Converter (Technobothnia Electrical Department) .....	16
<b>Figure 6</b> Shows the view of connector pins (Zytrax Info) .....	17
<b>Figure 7</b> EUT chamber .....	19
<b>Figure 8</b> Main PC from above is connected to control box with black cover by using RS-232 DB-9 Cable.....	20
<b>Figure 9</b> user panel on the screen of the Chamber .....	26
<b>Figure 10</b> program flow charts for reading temperature values from control box	29
<b>Figure 11</b> GUI which receives the temperature and humidity .....	30
<b>Figure 12</b> DS18B20 digital Temperature Sensor (Spark Fun Electronics).....	31
<b>Figure 13</b> Circuit Diagram for Connection of sensor to GPIO's of PI (REUK)..	32
<b>Figure 14</b> Program flow chart for reading temperature from external sensor.....	33
<b>Figure 15</b> Program flow chart for reading temperature from internal sensor .....	34
<b>Figure 16</b> Temperature and Humidity versus time graph .....	35
<b>Table 1</b> Shows Pin Configuration of RS-232 Cable (Zytrax Info).....	16
<b>Table 2</b> Communication protocols between Main PC and Control Box .....	20
<b>Table 3</b> Transmitted message from the main PC to the Control Box.....	22
<b>Table 4</b> Transmitted messages from Control Box to main PC .....	22
<b>Table 5</b> Command bits sent by Control Box in ASCII codes .....	23
<b>Table 6</b> Transmitted data bits from Main PC to Control Box .....	24

**LIST OF ABBREVIATIONS**

RPi	Raspberry Pi
SD	Secure Digital
ARM	Annotated Reference Manual
GPIO	General Purpose Input/Output
HDMI	High Definition Multimedia Interface
USB	Universal Serial Bus
CPU	Central Processing Unit
GPU	Graphics Processing Unit
SoC	System on Chip
DC	Direct Current
I/O	Input/Output
LAN	Local Area Network
LED	Light Emmiting Diode
DCE	Data Communications Equipment
DTE	Data Transmitting Equipment
PC	Personal Computer
Tx	Transmit Messages

Rx	Received Messages
PID	Proportional Integral Derivative
ASCII	American Standard Code for Information Interchange
COM	Communication Port

# 1 INTRODUCTION

The main contents of the introduction is the structure of this thesis, the objective of the project and an explanation why the client wants this thesis.

## 1.1 Structure of the Thesis

This project is to develop a web based application to remotely monitor the Temperature and Humidity chamber at Technobothnia Electrical Department. EUT chamber is used to test various equipment based on the given program testing parameters. There are temperature and humidity sensors residing inside the chamber and the sensors are controlled by a PC software program.

## 1.2 Objective of the Project

This project has several phases, the first phase is the sniffing of the RS-232 communication between the main PC and the internal control box. The second phase is testing temperature and humidity sensors by remote monitoring using Raspberry Pi. Then a Python program which shows temperature and humidity as a function of time using graph will be implemented. The program has two main parts, one of them will record the temperature values from the internal sensor and save the data in the database of a particular file in Raspberry Pi, while the other program will display the values on to a web page of a remote computer or a laptop. The application could be installed in the Raspberry Pi and remote monitoring could be established using LAN.

## 1.3 Benefits of the Project

Working design of the project can be used in customer service; Electrical Department in Technobothnia. When various test are made in the chamber the inspector can remotely monitor the results from anywhere. The project work design is given in Figure 1.



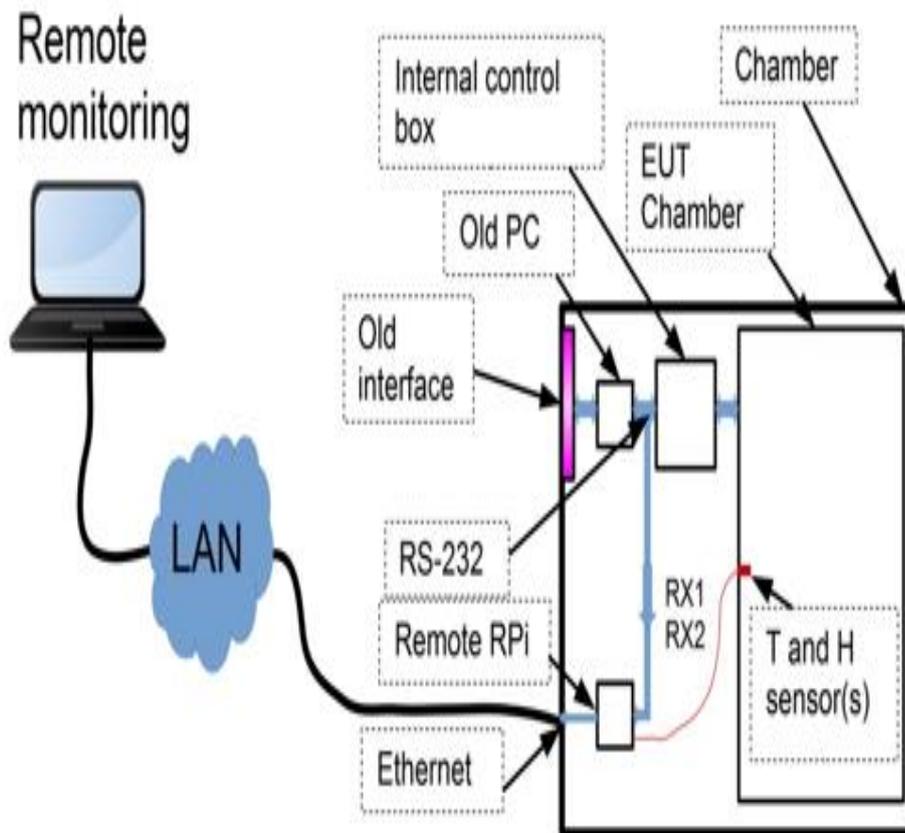


Figure 1 Main diagram of the project design (Technobothnia Electrical Department)

## 2 BACKGROUND

This chapter discusses the different equipment used in the project, the technologies used and briefly point out the main working designs of the project.

### 2.1 Raspberry Pi

Raspberry Pi is a credit card sized single board computer developed in the UK by Raspberry Pi Foundation.

#### 2.1.1 History of Raspberry Pi

Raspberry Pi Foundation was established in 2006 by Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft based at the University of Cambridge's computer laboratory. The new Raspberry Pi model B+ was used in this project and it has an ARM 1176J2F-S 700 MHz processor. It supports different operating systems such as Linux (Debian), and RISC OS (Raspberry Pi Foundation).



Figure 2 Raspberry Pi Model B+ 512 MB (Linux User and Developer)

### 2.1.2 Hardware and Software Specifications of Raspberry Pi

<b>Chip</b>	Broadcom BCM2835 SoC
<b>Core architecture</b>	ARM11
<b>CPU</b>	700 MHz Low Power ARM1176JZFS core
<b>GPU</b>	Dual Core VideoCore IV
	Open GL ES 2.0, hardware-accelerated OpenVG, 1080p30 H.264 high-profile decode
	Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
<b>Memory</b>	512MB SDRAM
<b>Operating System</b>	Boots from Micro SD card, running a version of the Linux operating system
	Supports Debian GNU/Linux, Fedora, Arch Linux, RISC OS and More
<b>Power</b>	Micro USB socket 5V, 2A
<b>Ethernet</b>	10/100 BaseT Ethernet socket
<b>Video Output</b>	HDMI (rev 1.3 & 1.4)
	Composite RCA (PAL and NTSC)
<b>Audio Output</b>	3.5mm jack, HDMI
<b>USB</b>	4 x USB2.0 Ports with up to 1.2A output
<b>GPIO Interface</b>	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip
	Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
<b>Camera Interface</b>	15-pin MIPI Camera Serial Interface (CSI-2)
<b>JTAG</b>	Not populated
<b>Display Interface</b>	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
<b>Memory Card Slot</b>	SDIO

Figure 3 Hardware and Software specification of Raspberry Pi model B+(GE Tech)

### 2.1.3 Raspberry Pi GPIO's

Pins are physical interfaces between Pi and the outside world, in this project the pins are used as input/output device for DS18B20 sensors. As shown in Figure 3, this model of Raspberry Pi has 40 pins. You can program the pins and communicate with the outside world. Inputs do not have to come from a physical switch, it could be input from a sensor or a signal from another computer or a device. The output can also do anything, from turning a LED on to sending a signal or data to another device. If the RPi is on a network, you can control devices that are attached to it.

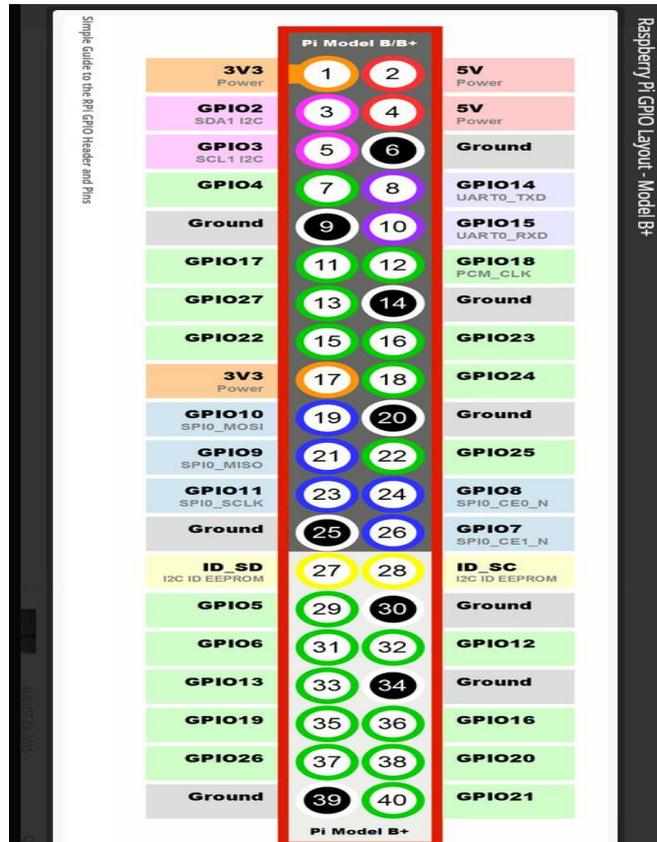


Figure 4 GPIO layout of model B+ RPi (Raspberry Pi Foundation)

## 2.2 RS-232 Serial Adapter

One of the universal parts of the PC is the serial port. The serial port is used to connect a mouse, a printer, a USB, and many other possible devices.

### 2.2.1 Introduction

Serial communication is a process of sending data bits by bits through a channel or devices. There are two types of serial communication, asynchronous and synchronous communication. In this project we have used the DB-9 RS-232 cable and USB to RS232 converter cable are used as shown in Figure 4.



Figure 5 RS-232 to USB Converter (Technobothnia Electrical Department)

9-Pin	Pin	Definition	Direction(PC view)
1	DCD	Data Carrier Detect	Input
2	Rx	Receive Data	Input
3	Tx	Transmit Data	Output
4	DTR	Data Terminal Ready	Output
5	GND	Signal Ground	-
6	DSR	Data Set Ready	Input
7	RTS	Request to Send	Output
8	CTS	Clear to Send	Input
9	RI	Ring Indicator	Input

Table 1 Shows Pin Configuration of RS-232 Cable (Zytrax Info)

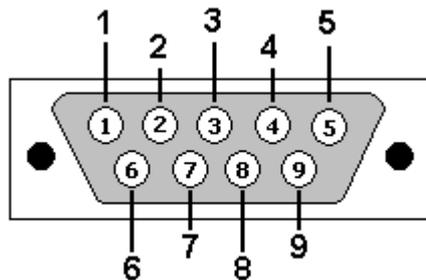


Figure 6 The view of connector pins (Zytrax Info)

### 2.2.2 Asynchronous Communication

In asynchronous communication, the interface does not include a clock line. Instead each computer or device provide its own clock as a time reference. In this project asynchronous communication is established by using a RS-232 cable between the main PC and the control box. The computers must agree on a clock frequency, the actual frequency within each computer matched within a few percent. A transmitted start bit synchronizes the transmitter's and receiver's clocks.

There are some parameters which both devices should have:

- **Baud Rate:** It specifies the rate at which bits are transmitted
- **Data Bits:** number of data bits to be transmitted
- **Parity Bit:** a bit which checks if the receiver get the right message or not  
It could be Even, Odd or None
- **Start Bit:** The starting bit
- **Stop Bit:** the stop bit

In this project the parameters given below are set between the two devices and the Raspberry pi to capture the messages. The configuration is done by following the protocols in which the control box and the main PC communicate.

Baud Rate: 9600, 1 Start bit, No parity, 1 Stop bit

When a packet of data is sent, it follows the above parameters in which 8 bits of data is passed at a time. When a transmission is not sending anything we check if the logical state of the wire is “1” or “-15V”. So, when sending a message or character the voltage is changed to +15 V indicating the logical state of “0” when using RS-232 DB9 cables.

### **2.3 EUT Chamber**

EUT chamber is an equipment which is used to test different types of devices. It consists of two main sensors that are the temperature and humidity sensors. These sensor are used to measure the internal temperature and humidity of the chamber.

#### **2.3.1 Purpose of Chamber**

The chamber is widely used to test various equipment such as cell phones, laptops, circuits, and electrical cables. The testing procedure is done in the following steps. The equipment is placed to the racks of the chamber and a program is set in which the test is designed. Most of the time such tests are made to fully understand the working range of an equipment, that is within what range of temperature and humidity that particular equipment could work. The chamber used for this project is a climate chamber. It is manufactured by FINERO and is shown in Figure 7.



Figure 7 EUT chamber

### **3 SNIFFING OF RS-232 COMMUNICATION BETWEEN OLD PC AND CONTROL BOX**

#### **3.1 Communication Protocols**

This is the first step which was done in order to replace the old monitor from chamber. A lot of simulation and data recording was done in order to identify how the communication protocols work between the main PC and the control box.

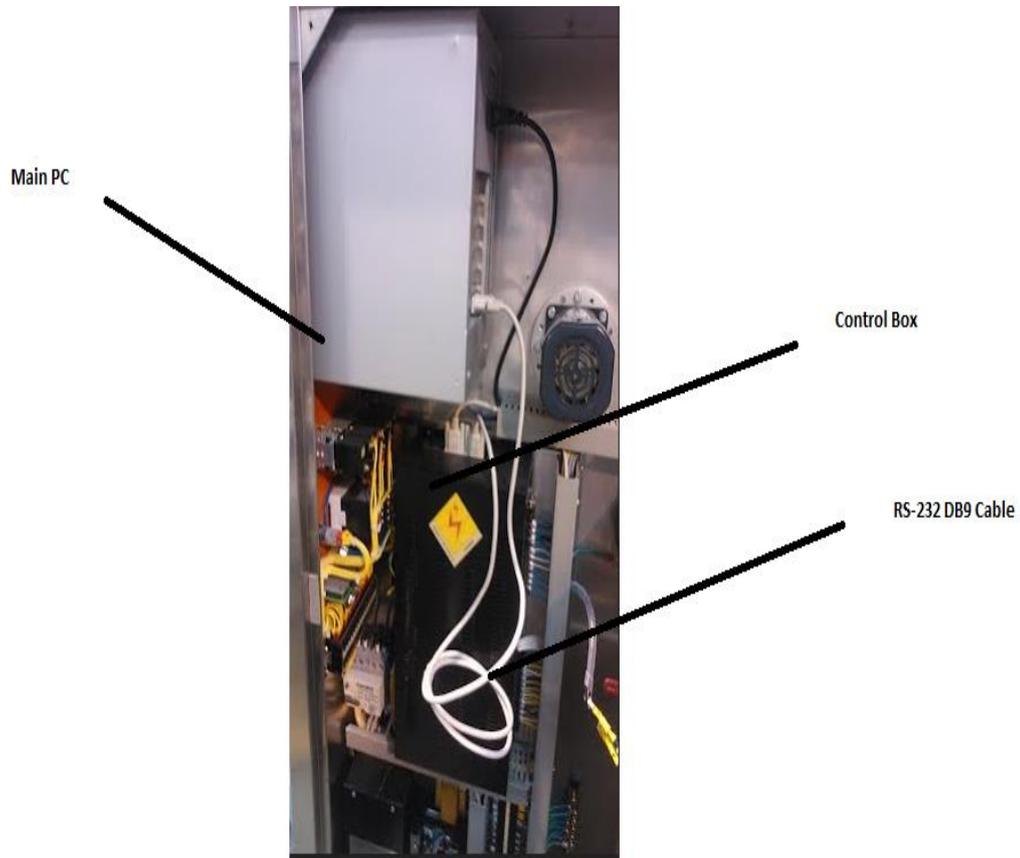


Figure 8 The main PC from above is connected to the control box with a black cover by using a RS-232 DB-9 Cable

The communication protocol is defined below, showing what one packet of data message could look like:

	STX	ID1	ID2	COMMAND	EXT	CHKDSK	END
HEX values	02	30	30	message	03	Check sum	0D

Table 2 Communication protocols between main PC and control box

STX: it is start bit its hexadecimal value is 02

ID1: the ID number of the message which is 30

ID2: The ID number of the message which is 30

COMMAND: This is the message we sent to the other device

EXT: End of the text or message

CHKDSK: check sum where it checks if there is error or not

END: it is the stop bit 0D

Hence all the message which is sent from the Main PC to the control box or vice versa have the same protocols.

### **3.2 How to capture of messages made**

Real term software is used for capturing data streams from the old PC to control box and vice versa in a Linux machine and Cutecom in Raspberry Pi. This software is a terminal program in which two COM ports are opened and capture the transmit data bits from main PC to control box and transmit data bits from control box to main PC. The captured data bits are either in hexadecimal or ASCII code. (Source Forge)

In Tables 3 and 4, the transmission of the messages from the control box to the main PC and vice versa are captured using software tools and then they are studied thoroughly.







C1: Compressor 1  
C2: Compressor 2  
V1: Refrigerant loop valve 1  
V2: Refrigerant loop valve 2  
V3: Refrigerant loop valve 3  
V4: Refrigerant loop valve 4  
LNV: Extra cooler gas insertion valve  
T1: Temperature heater 1  
T2: Temperature heater 2  
H1: Number 2 Humidifier Heater  
H2: Number 2 Humidifier Heater  
P1: Number 1 Time Signal  
P2: Number 2 Time Signal  
P3: Number 3 Time Signal

For example from one column of the above figure, it can be observed that T2, T1, C1, V2/C2, and FAN are the active devices. This implies that the temperature values are the one which is increase or decrease. Whenever a program is executed the PID calculates which devices should be on and which ones to be off. To simplify this this active devices determine the target temperature and humidity. From the above figure, one can understand those binary numbers which show a bit of 1 are considered to be an active device.

As Table 6 demonstrates, when a particular data packets were sent to the control box by using hexadecimal or ASCII codes, the result shows which of the active devices are selected from the panel. It is shown by pointing a rectangular selector on to the active device on the screen.

## UNIT B-1 MONITOR

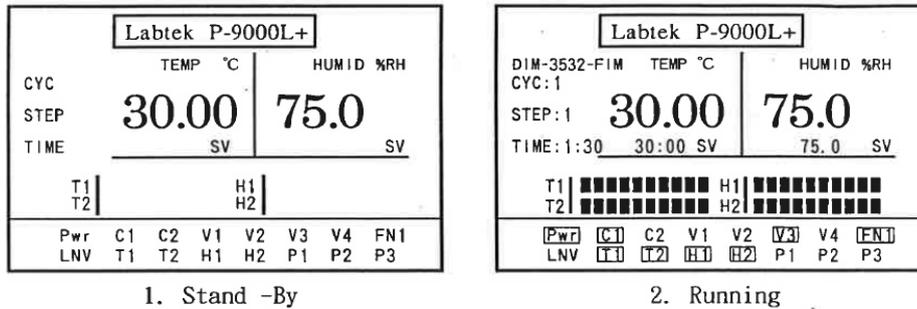


Figure 9 User panel on the screen of the Chamber

The main PC user panel is shown on Figure 9, it shows the values of temperature and humidity. During stand by it shows the current temperature and humidity, while running there is already a program to be tested.

## 4 REMOTE MONITORING USING RASPBERRY PI

A RS-232 to a USB converter cable is tapped between the main PC and control box and a USB cable is plugged to the RPi. Cutecom is used as software to capture all the readings from the control box and the main PC. (Source Forge).

### 4.1 Control Box Reading

Two COM ports were open from the terminal and started reading the internal temperature and humidity values from the control box, capturing transmit messages (Tx) from the control box and the transmit messages from the main PC.

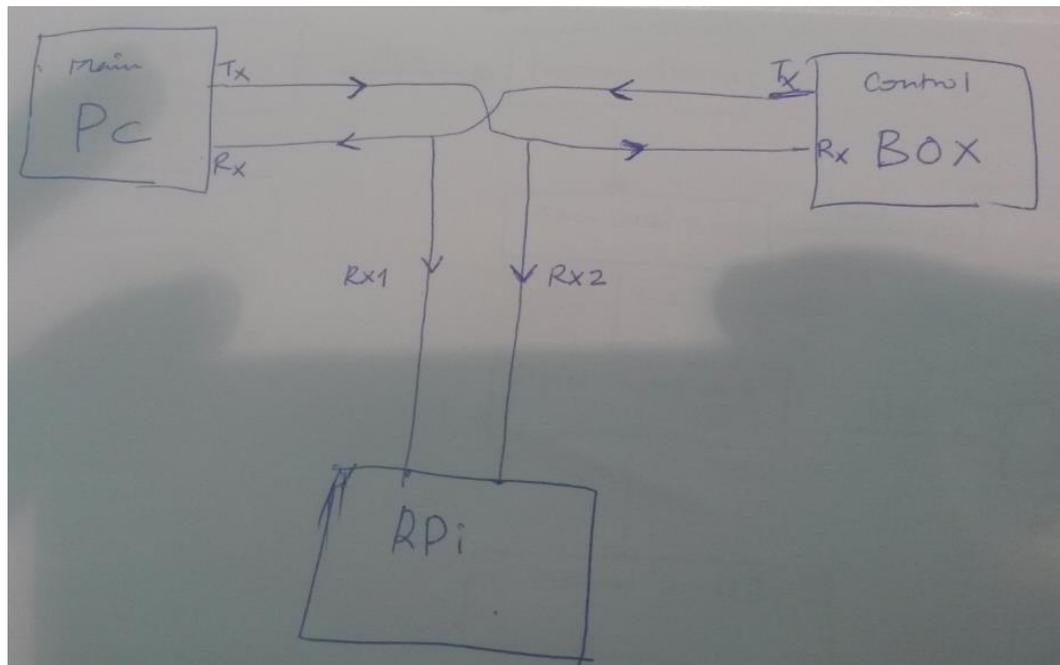


Figure 9 Simulation of reading from control box

Here we have used a QT, which is a cross platform application used for developing software applications that can be run on various hardware or software environments. This experiment was done on a Linux machine, hence there is a need to implement this application in to Raspberry Pi.

## 5 CONTROL BOX READING USING QT APPLICATION

A QT platform application to start to listen and display communication between the control box and the main PC is used in this project. This application is done in a Linux machine by using C++ programming.

### 5.1 Programming Flow Chart Reading Control Box Using C++

Main Functions:

ReadSerial function is used to set all the parameters to send or receive any data from one device to the other. The following parameters on the main functions are

set. These parameters are imported from the QT library and their importance is given below.

**SetPortName:-** there are often two or more serial ports on any computers, hence the name of the serial port has to be set in which communication started. This method will make sure that the right port is selected.

**SetBaudRate:-** is used to control the number of bits transmitted through the medium. It could be selected as 9600/11700. In this program the DCE and the DTE should have the same baud rate, otherwise the information could not be correctly transmitted as there will be a different timing of the messages. In this program both the DTE and DCE are set to be 9600.

**SetDataBits:-** is used to determine how many data bits are to be transmitted through the media. In the program eight data bits are transmitted at a time.

**SetFlowControls:-** this method controls the rate of data transmission between two nodes.

**SetParityBits:-** is used for error checking, to help detect data corruption during transmission.

**SetStopBits:-** one packet of data should have one stop bit.

### **Other Functions:**

**ReadSerial:** method is used to receive all the data bytes from the control box and it splits each data in an order to show the temperature and humidity values.

**Update Temperature:** method is used to update the screen when the first bytes of data are received on the LCD object. It manages to update the value of the displayed temperature values on the lcdNumber object.

**Update Humidity:** method is used to update the screen when the first bytes of data are received on the LCD object. It manages to update the value of the displayed humidity values on the lcdNumber object.

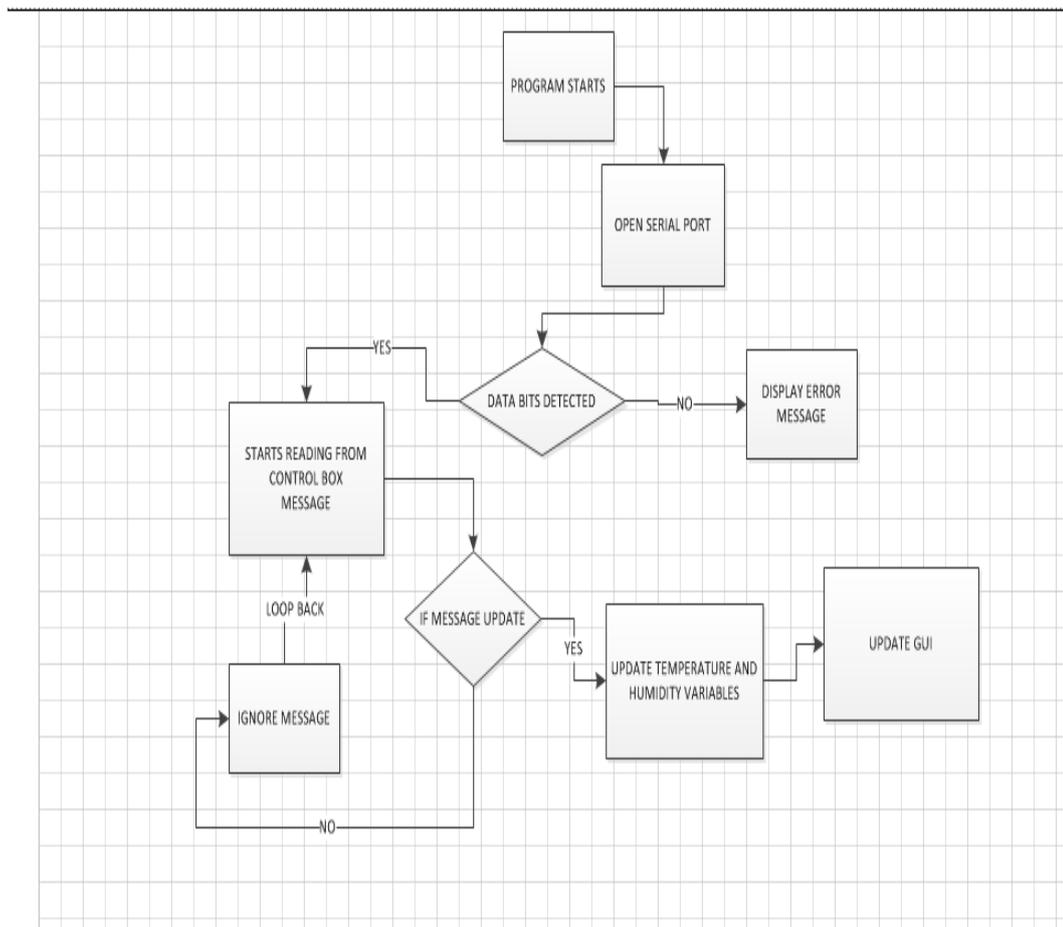


Figure 10 program flow charts for reading temperature values from control box

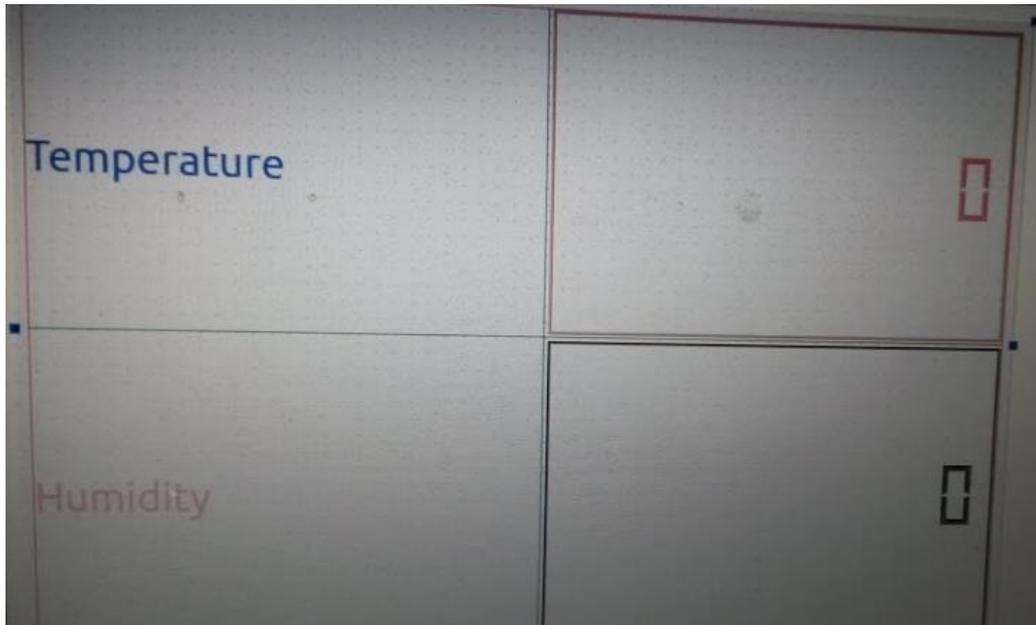


Figure 11 GUI which receives the temperature and humidity

In Figure 10, the user interface for the reading of the control box is given. Every time a new value of temperature and humidity is received, it updates the latest readings and displays it on the panel.

## **6 TESTING FOR TEMPERATURE AND HUMIDITY BY USING EXTERNAL SENSORS**

In the project external digital temperature sensor DS18B20 was used, it is used to measure the temperature values inside the chamber. It is plugged in to the GPIO's of the Raspberry Pi and is shown in the following diagram:

3.3 V power is connected to the pin 3 of the sensor, the resistor in this setup is used as a pull up for the data line, and should be connected between the DQ and the VDD line. It ensures that the one wire data line is at a definite logic level, and limits interference from the electrical noise if one pin is left floating.

**DS18B20 Temperature Sensor:**

- Manufacturer: Dallas
- Manufacturer Part No: DS18B20
- Programmable resolution
- 1 wire digital thermometer sensor
- Output type : digital
- Sensing Temperature:  $-55^{\circ}\text{C} \sim 125^{\circ}\text{C}$
- Voltage Supply:  $3\text{V} \sim 5.5\text{V}$



Figure 12 DS18B20 digital Temperature Sensor (Spark Fun Electronics)

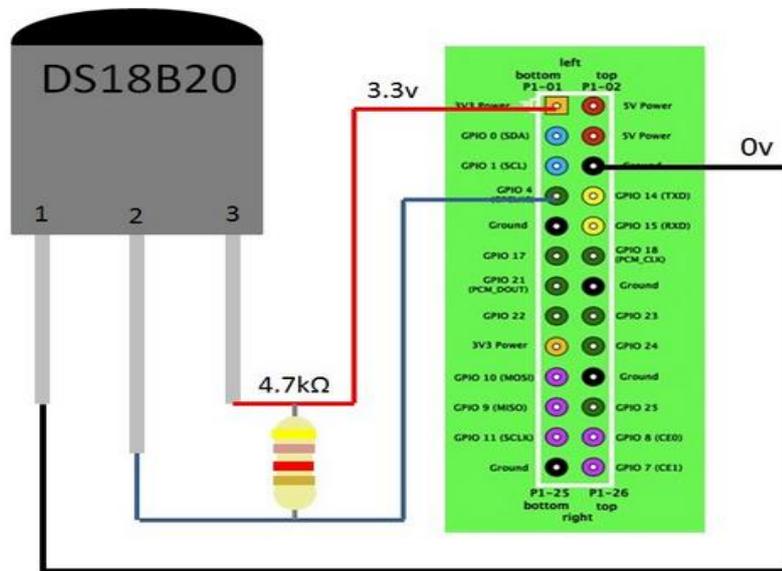


Figure 13 Circuit Diagram for Connection of sensor to GPIO's of PI (REUK)

The remote monitoring consists of two parts: a Python script which measures the temperature at specific date and time intervals, and the second part is the web page which displays the external temperature inside the chamber into a webpage. The first script `chamber_monitor.py` is triggered by a Crontab utility or function. It is possible at what year, month, date, hour, or min in which the first Python script could be called and executed. The minimum amount of the time interval which can be measured is 1 minute or 60 seconds. So, every one minute, it reads the temperature from DS18B20 sensor which is connected to the GPIO pins of the Raspberry Pi, and it stores the readings in a database. In Raspberry Pi sqlite3 database utility have already been installed from the terminal of the RPi. The other script, `guiSenRead.py` executes when it is required by the Apache web server. We have also installed the Apache 2 server for Raspberry Pi from the terminal. This script queries the database and displays the readings formatted in HTML. The temperature values from the chamber are displayed in java script chart generated by code from Google charts.

### 6.1 Programming Flow Chart for Reading External Sensor

The Python program which monitors the external temperature of the chamber by plugging the external sensors into the chamber could be designated is shown in Figure 14.

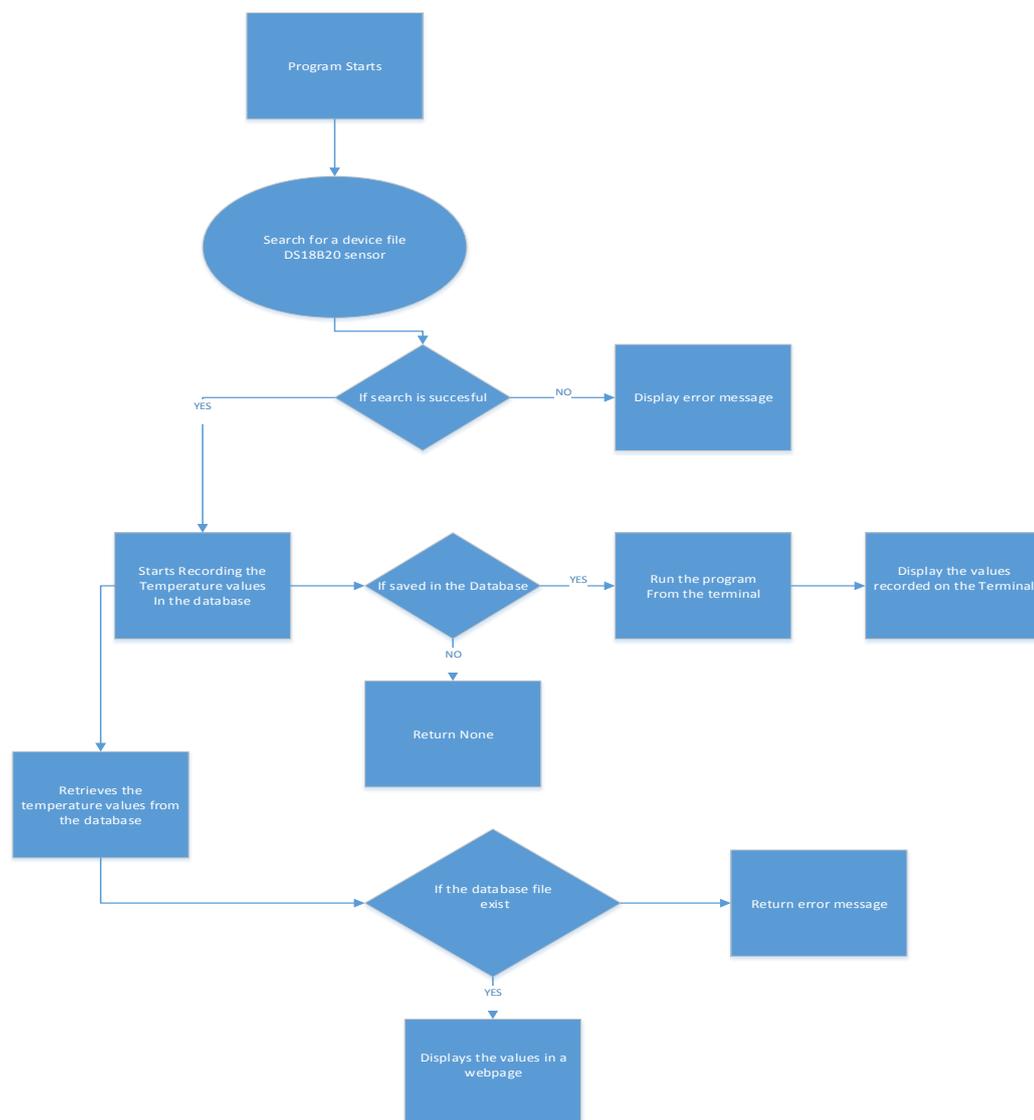


Figure 14 Program flow chart for reading temperature from external sensor

## 7 CONTROL BOX READING USING PYTHON SCRIPT

The testing parameters used which are configured in all the programs are the same, they are the baud rate, parity bits, start bits, flow control and stop bit. There are two Python programs, the first one reads the values of the temperature and humidity from the control box and display values when it is run from the terminal of the PI. The second script is used to display the readings of chamber as a graph in to an apache webserver. In addition, it has a user interface in which one could select the time interval of the data. In Figure 15, the algorithm of the program is shown in detail.

### 7.1 Programming Flow Chart for Reading from Control Box



Figure 15 Program flow chart for reading temperature from internal sensor

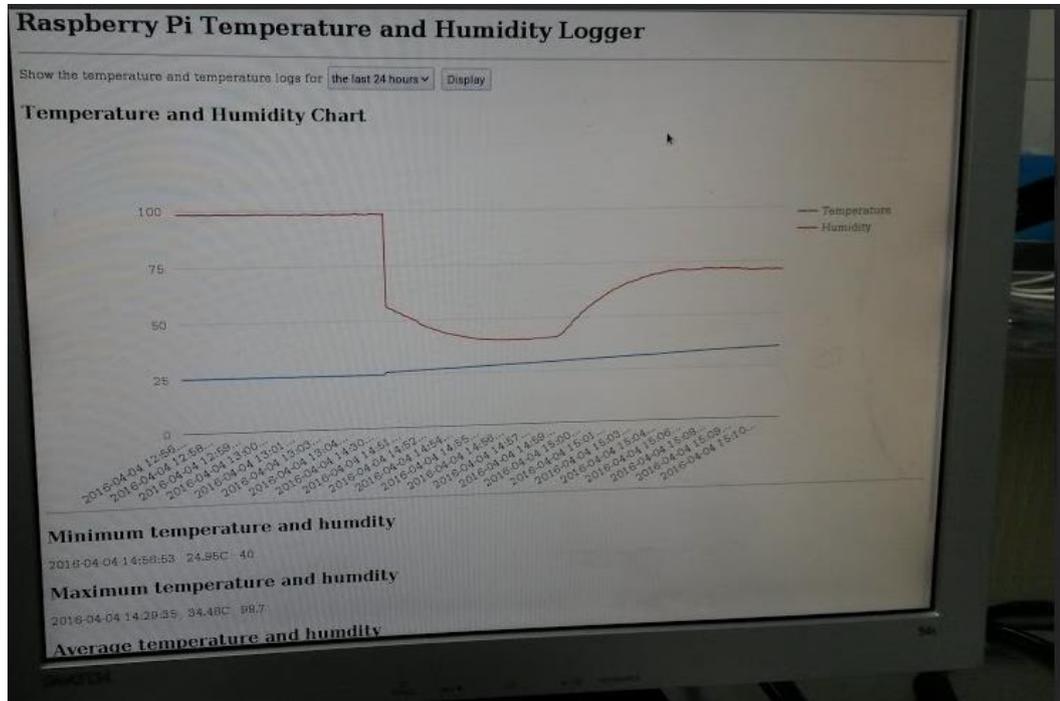


Figure 16 Temperature and Humidity versus time graph

## 8 CONCLUSION

### 8.1 Review of the Project

This project was correctly discussed and it has achieved and accomplished the primary objective of the work; the establishment of remote monitoring of the chamber using Raspberry PI was achieved. To be specific, the user or the chamber operator must be able to view, analyze and monitor the program during the program execution remotely. The system must be able to generate or draw temperature and humidity to time function graphs. The user must be able to save the readings to a database and search for values from the database files. Overall this application helps the user to thoroughly investigate the temperature and humidity readings changes through graphical presentation when he is at any place or time. He can also set a date, time or hour for the sensor to be triggered and start measuring

the internal temperature of the chamber when planning the testing equipment. The application which was made to measure the external temperature has an operator manual.

## **8.2 Future Improvements**

When doing the project, the temperature sensor DS18B20 was used and it works in the range of -50 °C to +125 °C on to the GPIO's of the Raspberry PI. However, a humidity sensor within that range was not found. If the humidity sensor is found it would be possible to use the same application for attaining the humidity values from the chamber. Hence, there is not external humidity sensor tested in the project.

## **9 ANNEXATION**

In the project four python script programs are used. The program codes are attached below:

```

D:\AllProjectreal_One_read.py 11. toukokuuta 2016 12:33
#!/usr/bin/env python
#When this program is executed, it receives the values of the
# temperature and humidity from the chamber and save it in to the database
#this program is used for the internal sensor readings from the control box

import serial

import sqlite3

import os
import time
import glob

#Global Variables

ser = 0
serialString = " test string"
temperature = ""
humidity = ""

# chamber temperature and humidity stored in the database
dbname='/var/www/html/real_DB.db'

#Function to Initialize the Serial Port
def init_serial():

    global ser          #Must be declared in Each Function
    ser = serial.Serial()
    ser.baudrate = 9600
    ser.parity = serial.PARITY_NONE
    ser.stopbits = serial.STOPBITS_ONE
    ser.bytesize = serial.EIGHTBITS

    #assign to the folder of the opening port
    ser.port = '/dev/ttyUSB1'

    #Specify theTimeOut in seconds, so that SerialPort
    #Doesn't hangs
    #ser.timeout = 0.2
    ser.open()          #Opens SerialPort

#Call the Serial Initilization Function
init_serial()

# store the temperature in the database
#receives two parameters temperature and humidity
def chamber_log_temperature(mytemp, humid):

    conn=sqlite3.connect(dbname) #define connection object
    curs=conn.cursor()
    #query inserts the temp and humidity values w/c receives from the bufferData function
    curs.execute("INSERT INTO mytemps values(datetime('now'), (?), (?))", (mytemp,humid))

```

```

    # commit the changes
    conn.commit()

    conn.close()
#displays the temp and humid readings from the database
def display_rpi_data():

    conn=sqlite3.connect(dbname)
    curs=conn.cursor()

    for row in curs.execute("SELECT * FROM mytemps"):
        print str(row[0])+" "+ str(row[1]) + " " + str(row[2])

    conn.close()

#function receives the bytes and parse the temperature and humidity readings

def bufferData(ser):
    value = ""
    endl = b'\r' #end of line which is denoted by HEX 0D
    tdata = ser.read(26) # read the given amount of bytes
    time.sleep(5)
    data_left = ser.inWaiting()
    tdata += ser.read(data_left) #reads data one by one
    value += tdata[0:len(tdata)-1]

    if value.endswith(endl):
        ser.flush()
    else:

        if "0A" in tdata: #it finds the message with ID2 0A
            buffer_split = tdata.split(' ') # it splits the message by checking the
space between bytes
            if len(buffer_split) < 3: # if there is at least 2 spaces in one packet of
data
                return
            else:
                newData = tdata
                if newData[2] == "A": #check for index no 2
                    temp = newData[3:10] #index range of the temp value in a data
                    hum = newData[19:23] #index of the hum value

                    print "Temperature = " + temp, "Humidity = " + hum #print
temperature and humidity readings on the terminal
                    chamber_log_temperature(temp, hum) #send the temp and hum value to
database function

while 1:
    bufferData(ser) #while everything is ok execute this function

```

```

D:\AllProject\gui_int.py 11. toukokuuta 2016 12:33
#!/usr/bin/env python
# This is the gui representation of the reading from the control box of the chamber
# directly from the panel of the chamber. When this program is called
# from the apache2 webserver. It shows both temperature
# and humidity in a graph by retrieving the data which was saved in the database.

import sqlite3
import sys
import cgi
import cgitb

# global variables

dbname='/var/www/html/real_DB.db'

# print the HTTP header
def printHTTPHeader():
    print "Content-type: text/html\n\n"

# print the HTML head section
# arguments are the page title and the table for the chart
def printHTMLHead(title, table):
    print "<head>"
    print "    <title>"
    print title
    print "    </title>"

    print_graph_script(table)

    print "</head>"

# get data from the database
# if an interval is passed,
# return a list of records from the database
def get_data(interval):

    conn=sqlite3.connect(dbname)
    curs=conn.cursor()

    if interval == None:
        curs.execute("SELECT * FROM mytemps")
    else:
        curs.execute("SELECT * FROM mytemps WHERE timestamp>datetime('now','- %s hours') "
% interval)
        #curs.execute("SELECT * FROM mytemps WHERE timestamp>datetime('2016-01-03
16:20:12','- %s hours') AND timestamp<=datetime('2016-01-03 16:21:06') " % interval)

    rows=curs.fetchall()

    conn.close()

```

```

    return rows

# convert rows from database into a javascript table
def create_table(rows):
    chart_table=""

    for row in rows[:-1]:
        rowstr="['{0}', {1}, {2}],\n".format(str(row[0]),str(row[1]),str(row[2]))
        chart_table+=rowstr

    row=rows[-1]
    rowstr="['{0}', {1}, {2}]\n".format(str(row[0]), str(row[1]),str(row[2]))
    chart_table+=rowstr

    return chart_table

# print the javascript to generate the chart
# pass the table generated from the database info
def print_graph_script(table):

    # google chart snippet
    chart_code=""
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("visualization", "1.1", {packages:["corechart"]});
        google.setOnLoadCallback(drawChart);
        function drawChart() {
            var data = google.visualization.arrayToDataTable([
                ['Time', 'Temperature', 'Humidity'],
                %s
            ]);
            var materialOptions = {
            chart: {
                title: 'Chamber measurement'
            },
            series: {
                0: {axis: 'Temperature'},
                1: {axis: 'Humidity'}
            },
            axes: {
                y: {

                    Temperature: {label: 'Temperature'},
                    Humidity: {label: 'Humidity'}
                }
            }
            };
            var chart = new
google.visualization.LineChart(document.getElementById('chart_div'));
            chart.draw(data, materialOptions);
        }
    </script>""

```



```

    print "<hr>"

    print "<h2>In the last hour:</h2>"
    print "<table>"
    print
    "<tr><td><strong>Date/Time</strong></td><td><strong>Temperature</strong></td><td><strong>Humidity</strong></td></tr>"

    rows=curs.execute("SELECT timestamp, mytemp, humid FROM mytemps WHERE
timestamp>datetime('new', '-1 hour') AND timestamp<=datetime('new')")
    #rows=curs.execute("SELECT * FROM mytemps WHERE timestamp>datetime('2016-01-03
16:20:12', '-1 hour') AND timestamp<=datetime('2016-01-03 16:21:06')")
    for row in rows:
        rowstr="<tr><td>{0}</td><td>{1}</td><td>{2}</td></tr>".format(str(row[0]),str(row
[1]),str(row[2]))
        print rowstr
    print "</table>"

    print "<hr>"

    conn.close()

def print_time_selector(option):

    print ""<form action="/cgi-bin/gui_int_sensor.py" method="POST">
    Show the temperature and temperature logs for
    <select name="timeinterval">""

    if option is not None:

        if option == "6":
            print "<option value=\"6\" selected=\"selected\">the last 6 hours</option>"
        else:
            print "<option value=\"6\">the last 6 hours</option>"

        if option == "12":
            print "<option value=\"12\" selected=\"selected\">the last 12 hours</option>"
        else:
            print "<option value=\"12\">the last 12 hours</option>"

        if option == "24":
            print "<option value=\"24\" selected=\"selected\">the last 24 hours</option>"
        else:
            print "<option value=\"24\">the last 24 hours</option>"

    else:
        print ""<option value="6">the last 6 hours</option>
        <option value="12">the last 12 hours</option>
        <option value="24" selected="selected">the last 24 hours</option>""

    print "" </select>

```

```
<input type="submit" value="Display">
</form>"""

# check that the option is valid
# and not an SQL injection
def validate_input(option_str):
    # check that the option string represents a number
    if option_str.isalnum():
        # check that the option is within a specific range
        if int(option_str) > 0 and int(option_str) <= 24:
            return option_str
        else:
            return None
    else:
        return None

#return the option passed to the script
def get_option():
    form=cgi.FieldStorage()
    if "timeinterval" in form:
        option = form["timeinterval"].value
        return validate_input (option)
    else:
        return None

# main function
# This is where the program starts
def main():

    cgitb.enable()

    # get options that may have been passed to this script
    option=get_option()

    if option is None:
        option = str(24)

    # get data from the database
    records=get_data(option)

    # print the HTTP header
    printHTTPHeader()

    if len(records) != 0:
        # convert the data into a table
        table=create_table(records)
    else:
        print "No Data Found"
        return

    # start printing the page
    print "<html>"
    # print the head section including the table
    # used by the javascript for the chart
```

```
printHTMLHead("Temperature and Humidity Logger",table)

# print the page body
print "<body>"
print "<h1>Raspberry Pi Temperature and Humidity Logger</h1>"
print "<hr>"
print_time_selector(option)
show_graph()
show_stats(option)
print "</body>"
print "</html>"

sys.stdout.flush()

if __name__ == "__main__":
    main()
```

```

D:\AllProject\ext_read.py 11. toukokuuta 2016 12:33
#!/usr/bin/env python
# This is the program which is used to measure
# values of the temperature by the external sensor from the chamber
# and store those values in a database
#
import sqlite3

import os
import time
import glob

# global variables

dbname= '/var/www/html/ext_temp_DB.db'

# store the temperature in the database
def log_temperature(chamber_temp):

    conn=sqlite3.connect(dbname)
    curs=conn.cursor()

    curs.execute("INSERT INTO chamber_temps values(datetime('now'), (?))", (chamber_temp
,))

    # commit the changes
    conn.commit()

    conn.close()

# display the contents of the database
def display_data():

    conn=sqlite3.connect(dbname)
    curs=conn.cursor()

    for row in curs.execute("SELECT * FROM chamber_temps"):
        print str(row[0])+" "+str(row[1])

    conn.close()

# get temperature
# returns None on error, or the temperature as a float
def get_temp(devicefile):

    try:
        fileobj = open(devicefile,'r')
        lines = fileobj.readlines()
        fileobj.close()
    except:
        return None

    # get the status from the end of line 1
    status = lines[0][-4:-1]

    # is the status is ok, get the temperature from line 2

```

```
if status=="YES":
    print status
    tempstr= lines[1][:-1]
    tempvalue=float(tempstr)/1000
    print tempvalue
    return tempvalue
else:
    print "There was an error."
    return None

# main function
# This is where the program starts
def main():

    # enable kernel modules
    os.system('sudo modprobe wl-gpio')
    os.system('sudo modprobe wl-therm')

    # search for a device file that starts with 28
    devicelist = glob.glob('/sys/bus/wl/devices/28*')
    if devicelist=='':
        return None
    else:
        # append /wslave to the device file
        wldevicefile = devicelist[0] + '/wl_slave'

# while True:

# get the temperature from the device file
temperature = get_temp(wldevicefile)
if temperature != None:
    print "temperature="+str(temperature)
else:
    # Sometimes reads fail on the first attempt
    # so we need to retry
    temperature = get_temp(wldevicefile)
    print "temperature="+str(temperature)

    # Store the temperature in the database
    log_temperature(temperature)

    # display the contents of the database
    display_data()

    #time.sleep(1)

if __name__=="__main__":
    main()
```

```

D:\AIIPProject\gui_ext.py 11. toukokuuta 2016 12:33
#!/usr/bin/env python
# This is the gui representation of the reading from the external sensor DS18B20
# directly from the panel of the chamber. When this program is called
# from the apache2 webserver. It shows only temperature
# in a graph by retrieving the data which was saved in the database.

import sqlite3
import sys
import cgi
import cgitb

# global variables

dbname='/var/www/html/ext_temp_DB.db'

# print the HTTP header
def printHTTPHeader():
    print "Content-type: text/html\n\n"

# print the HTML head section
# arguments are the page title and the table for the chart
def printHTMLHead(title, table):
    print "<head>"
    print "    <title>"
    print title
    print "    </title>"

    print_graph_script(table)

    print "</head>"

# get data from the database
# if an interval is passed,
# return a list of records from the database
def get_data(interval):

    conn=sqlite3.connect(dbname)
    curs=conn.cursor()

    if interval == None:
        curs.execute("SELECT * FROM chamber_temps")
    else:
        curs.execute("SELECT * FROM chamber_temps WHERE timestamp>datetime('now','-%s
hours')" % interval)
    #     curs.execute("SELECT * FROM chamber_temps WHERE timestamp>datetime('2015-06-12
13:20:02','-%s hours') AND timestamp<=datetime('2015-06-12 13:20:02')" % interval)

    rows=curs.fetchall()

    conn.close()

```

```

    return rows

# convert rows from database into a javascript table
def create_table(rows):
    chart_table=""

    for row in rows[:-1]:
        rowstr="['{0}', {1}]\n".format(str(row[0]),str(row[1]))
        chart_table+=rowstr

    row=rows[-1]
    rowstr="['{0}', {1}]\n".format(str(row[0]),str(row[1]))
    chart_table+=rowstr

    return chart_table

# print the javascript to generate the chart
# pass the table generated from the database info
def print_graph_script(table):

    # google chart snippet
    chart_code=""
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript">
        google.load("visualization", "1", {packages:["corechart"]});
        google.setOnLoadCallback(drawChart);
        function drawChart() {
            var data = google.visualization.arrayToDataTable([
                ['Time', 'Temperature'],
                %s
            ]);
            var options = {
                title: 'Temperature'
            };
            var chart = new
google.visualization.LineChart(document.getElementById('chart_div'));
            chart.draw(data, options);
        }
    </script>""

    print chart_code % (table)

# print the div that contains the graph
def show_graph():
    print "<h2>Temperature Chart</h2>"
    print '<div id="chart_div" style="width: 1000px; height: 600px;"></div>'

# connect to the db and show some stats

```



D:\AllProject\gui\_ext.py

11. toukokuuta 2016 12:33

```

    for row in rows:
        rowstr="<tr><td>{0}&emsp;&emsp;</td><td>{1}C</td></tr>".format(str(row[0]),str(
row[1]))
        print rowstr
    print "</table>"

    print "</hr>"

    conn.close()

def print_time_selector(option):
#we should provide the current script name to be printed in a web server
    print "<<form action="/cgi-bin/gui_ext.py" method="POST">
        Show the temperature logs for
        <select name="timeinterval">""

    if option is not None:

        if option == "6":
            print "<option value=\"6\" selected=\"selected\">the last 6 hours</option>"
        else:
            print "<option value=\"6\">the last 6 hours</option>"

        if option == "12":
            print "<option value=\"12\" selected=\"selected\">the last 12 hours</option>"
        else:
            print "<option value=\"12\">the last 12 hours</option>"

        if option == "24":
            print "<option value=\"24\" selected=\"selected\">the last 24 hours</option>"
        else:
            print "<option value=\"24\">the last 24 hours</option>"

    else:
        print ""<option value="6">the last 6 hours</option>
            <option value="12">the last 12 hours</option>
            <option value="24" selected="selected">the last 24 hours</option>""

    print ""            </select>
            <input type="submit" value="Display">
    </form>""

# check that the option is valid
# and not an SQL injection
def validate_input(option_str):
    # check that the option string represents a number
    if option_str.isalnum():
        # check that the option is within a specific range
        if int(option_str) > 0 and int(option_str) <= 24:
            return option_str

```

```
        else:
            return None
    else:
        return None

#return the option passed to the script
def get_option():
    form=cgi.FieldStorage()
    if "timeinterval" in form:
        option = form["timeinterval"].value
        return validate_input (option)
    else:
        return None

# main function
# This is where the program starts
def main():

    cgitb.enable()

    # get options that may have been passed to this script
    option=get_option()

    if option is None:
        option = str(24)

    # get data from the database
    records=get_data(option)

    # print the HTTP header
    printHTTPHeader()

    if len(records) != 0:
        # convert the data into a table
        table=create_table(records)
    else:
        print "No data found"
        return

    # start printing the page
    print "<html>"
    # print the head section including the table
    # used by the javascript for the chart
    printHTMLHead("Chamber Temperature", table)

    # print the page body
    print "<body>"
    print "<h1>Chamber Temperature</h1>"
    print "<hr>"
    print_time_selector(option)
    show_graph()
    show_stats(option)
    print "</body>"
```

D:\AIIProject\gui\_ext.py

11. toukokuuta 2016 12:33

```
print "</html>"

sys.stdout.flush()

if __name__ == "__main__":
    main()
```

## 10 REFERENCES

/ 1 / Raspberry Pi Debian 8 download <URL:

<https://www.raspberrypi.org/downloads/raspbian/> >

Accessed on 2016/03/06

/ 2 / Raspberry Pi Foundation <URL:

<https://www.raspberrypi.org> > Accessed on 2015/05/15

/ 3 / GPIO input and output

<http://www.modmypi.com/blog/ds18b20-one-wire-digital-temperature-sensor-and-the-raspberry-pi> > Accessed on 2015/05/15

/ 4 / Connection of External sensor <URL:

<http://www.reuk.co.uk/DS18B20-Temperature-Sensor-with-Raspberry-Pi.html> > Accessed on 2015/07/21

/ 5 / Serial Communication with RS-232 by Christian Blum <URL:

<http://www.z80.info/1656.html> > Accessed on 2015/05/15

/ 6 / The connection of sensor and in to the pi with the circuit <URL:

<http://kmtronic.com/raspberry-pi-ds18b20-connect-to-gpio.html>>Accessed on 2015/07/21

/ 7 / The Crontab Function <URL:

<http://www.computerhope.com/unix/ucrontab.htm> >

Accessed on 2015/07/28

/ 8 / Raspberry pi Model B+ 500 Mb Specifications <URL:

<http://www.linuxuser.co.uk/reviews/raspberry-pi-model-b-review-a-new-evolution> >Accessed on 2015/05/15

/ 9 / RS-232 DB 9 Cable <URL:

[http://www.zytrax.com/tech/layer\\_1/cables/tech\\_rs232.htm](http://www.zytrax.com/tech/layer_1/cables/tech_rs232.htm) > Accessed on 2015/06/28

/ 10 / Real Term Serial Terminal <URL:

<http://realterm.sourceforge.net/> >Accessed on 2015/04/16

/ 11 / Raspberry Pi SQLite Temperature Logger <URL:

<http://raspberrypiwebserver.com/cgi scripting/rpi-temperature-logger/> > Accessed on 2015/07/28

/ 12 / Software and Hardware Specifications of RPi model B+ <URL:

[http://www.geeetech.com/wiki/index.php/Raspberry\\_Pi\\_Model\\_B%2B](http://www.geeetech.com/wiki/index.php/Raspberry_Pi_Model_B%2B) >

Accessed on 2016/05/08