Romet Kalpus, Hieu Vo

Smart sauna IOT device

Metropolia University of Applied Sciences

Bachelor of Sciences

ICT

IOT project

12.11.2017

Metropolia
University of Applied Sciences

Author(s) Title Number of Pages Date	Romet Kalpus, Hieu Vo Smart sauna IOT device 23 pages 18.12.2017
Degree	Bachelor of Sciences
Degree Programme	ICT
Specialisation option	Smart Systems
Instructor(s)	Sami Sainio
Keywords	

1 Introduction

This documentation is about connecting STM32F4 development board to the internet with DP83848 as external PHY; working with physical layer, setting the device up in the local network, setting up the TCP/IP stack for the device to be seen in the Internet and finally using the device as an IOT device that one can control sauna remotely over the Internet. The device is accessible over TCP and sends sensor data (temperature and humidity) to the web server over TCP connection through mobile application and web page with the interaction between them.

.

2 IoT integration between devices

As discussing all the part separately and more details, this section will give the overview and tell how all the parts connect and interact to each other.

The main operations will be between Chipboard and Server , Server and Web page, Server and Mobile. First, is the Chipboard and Server interaction. There will be 2 important tasks that keep updating. One is the Send Thermometer Data tasks , which keep pushing the temperature and humidity values to server by send HTTP POST message. The other task is Sauna On Off task, which keep sending HTTP REQUEST to retrieve On/Off values from server and keep checking on it. If the server value is on than, the task will turn the Led on and in reverse. We can not send the value from server to chipboard because the Metropolia doesn't allow to use socket.io, therefore, we need to keep updating On/Off value all the time. In addition, there is a "warning" variable that control On/Off function. If the "warning" variable is true, when temperature larger than 80 degree, then the leds (as the switch in Sauna) will be turned off until temperature below 80.

Secondly, it's the connection between Server and Web page. There will be 3 buttons : On/Off, Clear Data and Export to Excel, table displaying data and line chart showing values. Web page and server communicate back and forth effectively base on the server-side script that we made to push and extract server's data. Importantly, that it has function to write the value OFF to server when temperature reaches to the limit and disable On/Off button for safety reason.

Finally is the interaction between Server and Mobile application. The Android application only send On/Off data to server and display the values as a table. Application also has Alert task, which pop up the notice and send OFF value to server to make sure that the chipboard will get it.

3 Network interface

3.1 Devices and programs

3.1.1 STM32F407 development board

The development board was chosen because it is rich in features, it's effective running at 168 MHz and has 1 MB of flash for programs, has a built in MAC controller which is essential for developing IOT devices, is low powered and relatively low cost. The board doesn't have a built in networking physical interface but has MII (media independent interface) and RMII (reduced media independent interface) interfaces to connect an external PHY to.

3.1.2 DP83848 external PHY

This device acts as a PHY for the project and is connected to the RMII interface of the Ethernet MAC controller of STM32F4 microcontroller. The device supports 25MHz clock input for MII interface and 50 MHZ clock input for the RMII interface. RMII interface is used in this project. The device also supports auto-negotiation to automatically sense the link speed and half-duplex/full-duplex features.

3.1.3 Developing environment

The IDE used in this project is System Workbench for STM32, which is Eclipse based open source IDE developed for STM32 boards. GCC is used as a compiler and linker toolchain. The IDE has built in HAL library derived from STM32 Cube IDE that can be added into the project from the IDE and use the HAL library as drivers for peripherals.

3.1.4 Constructing the application

The application consists of HAL library acting as hardware abstraction layer on the hardware registers, FreeRTOSPlusTCPIP as an operating system and network

interface acting as an interface between the platform specific drivers and the operating system. First thing to do is to build plain FreeRTOS operating system on the platform and ensure it is working. After this, add the FreeRTOSPlusTCPIP application components to the program. To get the hardware communicating with the operating system, a network interface acting as a port needs to be implemented. There is a network interface file for STM32 in FreeRTOS webpage but the interface together with HAL drivers still needs some platform specific adjusting and configuration before it can get built correctly with the operating system. The program is written in C.

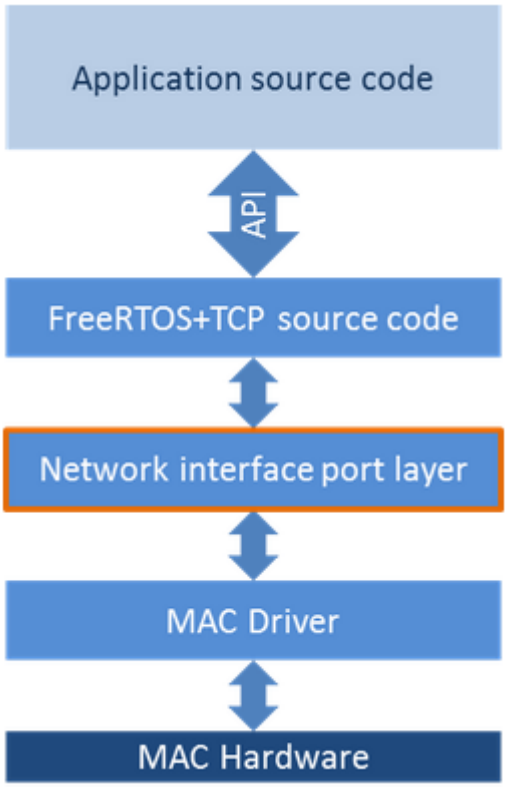

Figure 1 MAC - network IF - OS hierarchy in a nutshell

3.2 Connecting MAC with physical layer device

First thing to do on the hardware layer is to connect the development board and physical layer interface. The project uses RMII interface of MAC. The pinout of STM32F4 and DP83848 connection is as follows:

Interface: RMII

Signal name	DP83848 pin #	STM32 port #	STM32 pin #
MDC	31	PC1	9
MDIO	30	PA2	16
TX_EN	2	PB11	30
TX0	3	PB12	33
TX1	4	PB13	34
RX0	43	PC4	24
RX1	44	PC5	25
CRS	40	PA7	14
OSCIN (X1)	34	PA1	15

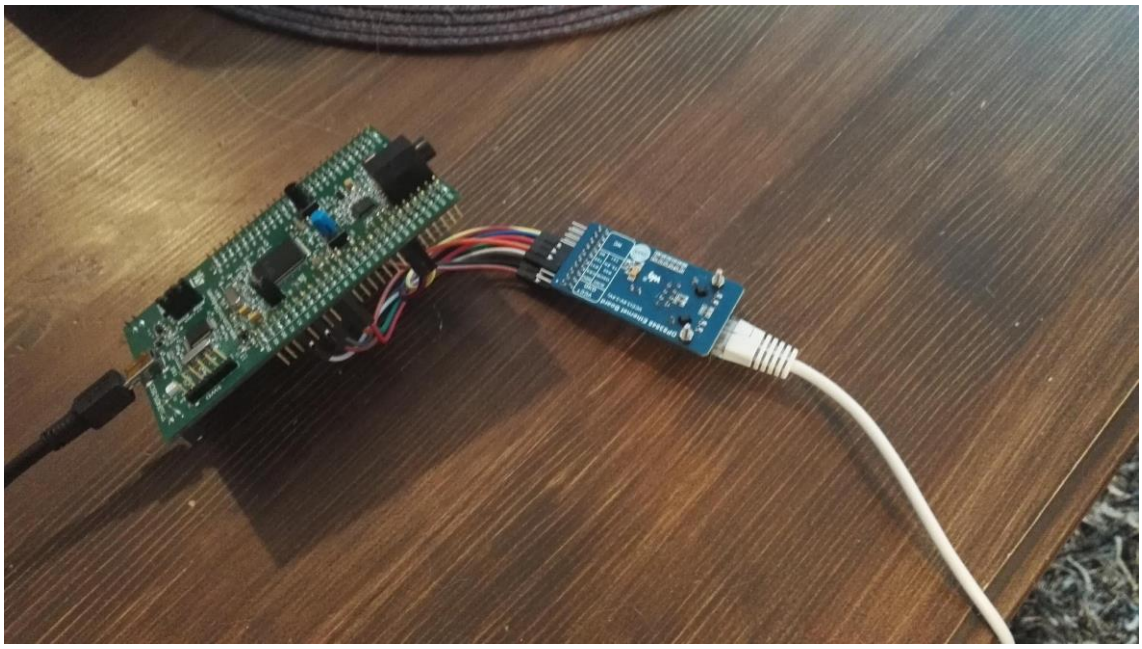

Figure 2 Dev board and PHY connected

When the pins of the devices are connected, the GPIO clocks routed to the specific ports via AHB bus needs to be enabled. STM32 is a low powered device and one of the features is that every aspect needs separate enabling. For example, if a developer wants to use GPIO pins, the clock for the pins need to be enabled separately. Enabling the clock to the GPIO pin enables it for AHB bus to get access to it. After this the Ethernet MAC controller clocks need to be enabled. When the developer wants to enable the GPIO pins with alternate functions routing to Ethernet MAC data receive functions we can configure them like this:

```
/*Setup pins in port C*/
GPIO_InitStruct.Pin = GPIO_PIN_4 | GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF11_ETH;      //Route the pins to MAC peripheral
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

Figure 3 Initialize GPIO pins for Ethernet functions example

STM32 has the alternate function concept where the user can route the signals of a peripheral to the specific GPIO pins. Here the pins PC4 and PC5 have a number of alternate functions the developer can route to these pins from different peripherals. MAC peripheral can route Rx0 and Rx1 signals to these pins and setting 'GPIO_AF11_ETH' value to the 'Alternate' register opens the port for these signals to be routed to these specific pins.

In order for the device to be more energy wise efficient, it is sensible to use MAC interrupts instead of polling the ethernet frames in the loop. To use MAC with interrupts, ETH_IRQn interrupt needs to be enabled. That way when MAC receives the frame, it sends the interrupt request to NVIC and NVIC notifies the CPU which in turn calls the Ethernet interrupt handler. The interrupt handler assigns the current flag value to the variable that the processing task in network interface file uses, clears the interrupt flag and exits the IRQ. The processing task in network interface file sees through the variable that the frame receive flag was set and starts processing the frame.

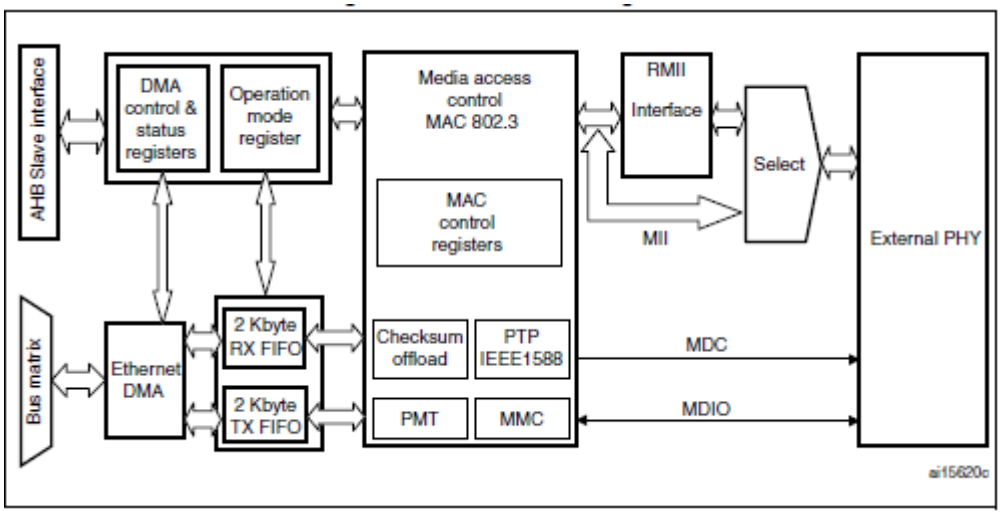

Figure 4 MAC diagram taken from STM32 reference manual

The diagram shows the working principle of MAC in a nutshell. Our DP83848 is acting as an external PHY which is connected to the MAC via RMII interface. MAC has a dedicated DMA which only notifies the CPU when transactions are started, releasing CPU from the calculations related to receiving and transferring frames from memory locations. DMA has the separate descriptors for receive FIFO buffers and transmit FIFO buffers. The descriptors are pointers to the memory locations where the Rx FIFO and Tx FIFO buffers are located. MDC and MDIO pins are for accessing the configuration and status registers of external physical interface.

3.3 Connecting the device into local network

All the packet analysing is done with Wireshark. In the beginning of testing the local area connectivity of the device, it sent out gratuitous ARP requests, advertising that it exists on the local network. The other devices in the same local area network saved the devices MAC address and corresponding IP address to their ARP cache. Nevertheless, the device was not seen on the network. The device could send out ICMP (ping protocol) packets, but it never received the response of the ARP request, meaning where the other device is located. It turned out that the device never received the frames because it got stuck in the interrupt service routine. When tried to receive frames with polling in the loop it worked fine. It took some digging into the manuals to discover that the reason it got stuck in the ISR and never cleared the error flag was that the abnormal interrupt summary (AIS) error flag in direct memory access status register (DMASR) was a sticky bit, meaning that there were multiple other bits connected to this specific bit that needed to be cleared before AIS error flag could be cleared. Turned out that in the start of the program, for some reason, the early transmit status (ETS) flag and receive buffer unavailable status (RBUS) flag are set, and because they are connected to the AIS flag, they need to be cleared first. After the beginning of the program and clearing those bits of the DMASR register in interrupt service routine, they don't seem to be set again. The device was now connected and seen in the local area network.

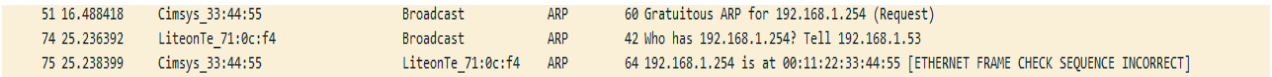

Figure 5 ARP packets from pinging to the STM32

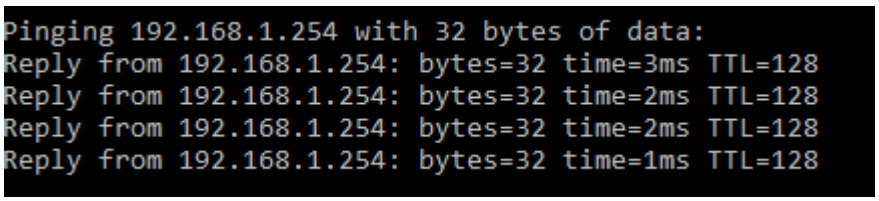

Figure 6 Pinging from desktop to STM32

4 TCP/IP stack

4.1 Devices server socket

The device has a socket that is binded to a certain port that is constantly listening. This socket can be used to access the device remotely. The TCP connection into the listening server socket can be tested with the function that turns the LEDs on and off. The device can be connected remotely with, for example, PuTTY using raw connection. So, when the user types to the terminal the command ‘LED1 ON’ the green LED turns on. When the user types ‘LED1 OFF’ the green LED turns off. This way it can be verified that the TCP connection to the listening server socket of the device works. The server socket is created in the task function ‘vCreateTCPSocket’ of the main program. Because up to 20 devices can be connected into this device in this program, when remote computer connects to the microcontroller, the task creates another task which is handling the specific connection instance to a specific remote computer.

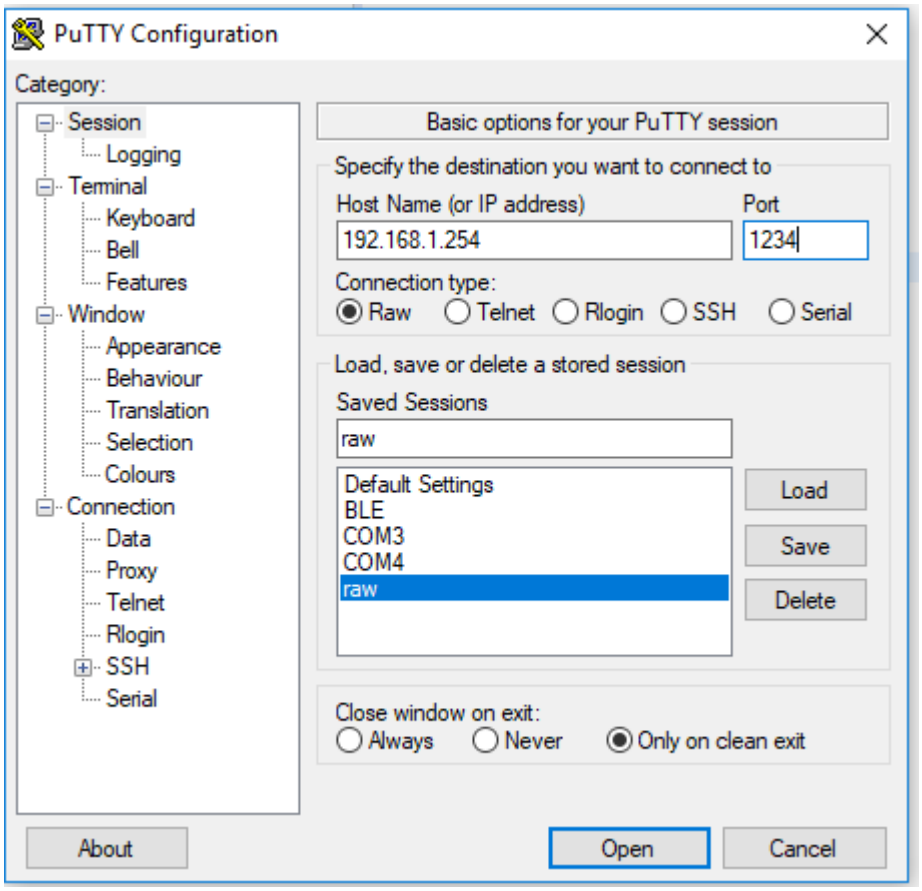

Figure 7 Connection interface with PuTTY

Source	Destination	Protocol	Length	Info
192.168.1.53	192.168.1.254	TCP	66	11294 → 1234 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
192.168.1.254	192.168.1.53	TCP	68	1234 → 11294 [SYN, ACK] Seq=0 Ack=1 Win=1460 Len=0 MSS=1460 WS=1 SACK_PERM=1
192.168.1.53	192.168.1.254	TCP	54	11294 → 1234 [ACK] Seq=1 Ack=1 Win=65536 Len=0

Figure 8 3-way handshake to the listening socket

4.2 Devices client socket

The device sends out temperature and humidity data to the web server from the client socket. The client socket isn't bound to any specific port number but when connecting to the remote web server, the client sockets port number is generated using the devices own random number generator. The STM32 random number generator uses continuous analog noise to generate a random 32-bit value. It generates port numbers between 1024 and 49000. Once the socket is created, the application can start sending temperature and humidity data to the specified remote IP address and the specified remote port. For example, because in this project we use users.metropolia.fi as a web server to upload our data, we need to specify the IP to 195.148.105.101 and remote port to 80. The project doesn't have any application protocol running on top of TCP/IP stack, so the data sent to the web server needs to be in exact HTTP format written by hand (keyboard actually). The client also checks if the sauna is on or off

using the HTTP GET request. If the GET request gets the data that sauna is on, all the LEDs turn on.

Client socket is created in task function ‘sendThermometerData’.

5 Humidity/Temperature sensor

5.1 The sensor

The sensor that is used in the project is HIH8120 humidity/temperature sensor that uses SIP 4-pin housing style. It has the pins VCC, GND, SDA, SCL. The 0.22µF condensator is installed between VCC and GND for noise reduction and 2,2 kOhm pullup resistors are installed to pull up SDA and SCL pins for I2C communication.

5.2 I2C communication

To get data from the sensor, it uses a simple self-made polling I2C driver. The driver first sends a measurement request for the sensor which wakes the sensor up and fills the sensors data buffer with humidity and temperature data. After the measurement request, the driver sends a data fetch request to fetch the data from sensors data buffer.

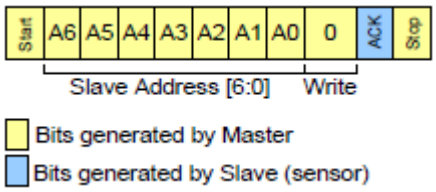

Figure 9 Measurement request protocol

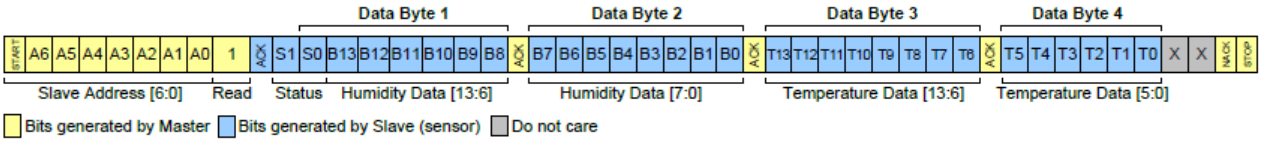

Figure 10 Data fetch protocol

The fetched humidity and temperature data are used to calculate the humidity percent and temperature degrees in Celsius using ready-given formulas found in the sensors’ manual and then sent to the web server using a client socket task function.

6 Web server & Web page

In web server part, we use school server to set up database and use files php or html to interact with it. Chipboard will send the sensor data to server, after that the web and mobile application will retrieve the data and display it.

We use Metropolia web server to store database in this project. The main server-side scripting language, which is used to communicate with server is PHP. Beside that there are some other languages, which are used to support for PHP file: HTML, CSS, and Javascript.

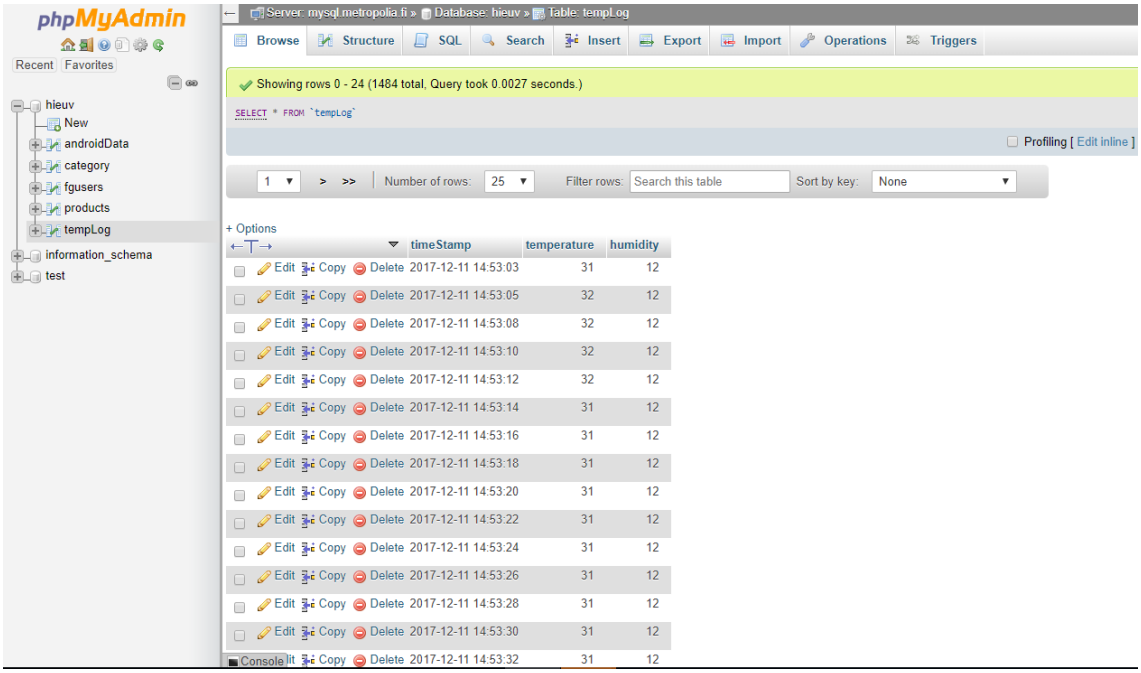

Figure 11 MySQL

The main PHP tempLog file will be displayed at below, first part is the start session and query data from MySQL server:

```
<?php

     session_start();
     require_once("includes/connection.php");
    // Read android data
      $sql="SELECT * FROM androidData ORDER BY id DESC LIMIT
1";
    $resultAndroid=mysqli_query($con, $sql);
    $row=mysqli_fetch_array($resultAndroid);
    $OnOff = $row["name"];
    mysqli_free_result($resultAndroid);
```

```
    //Read newest Temperature
    $sql="SELECT temperature FROM tempLog ORDER BY timeStamp DESC
LIMIT 1";
    $resultTemp=mysqli_query($con, $sql);
    $rowTemp=mysqli_fetch_array($resultTemp);
    $newTemp = $rowTemp["temperature"];
    mysqli_free_result($resultTemp);

    // Read tempLog with formated timeStamp for 30s
    $sql="SELECT temperature, humidity, UNIX_TIMESTAMP(timeStamp) AS
datetime FROM tempLog ORDER BY timeStamp DESC LIMIT 1800";

    $result=mysqli_query($con, $sql);
    $rows = array();
    $table = array();

    $table["cols"] = array(
    array(
      "label" => "Date Time",
      "type" => "datetime" //"string" //
     ),
    array(
     "label" => "Humidity (%)",
     "type" => "number"
     ),
    array(
     "label" => "Temperature (C)",
     "type" => "number"
     )
    );
    while($row = mysqli_fetch_array($result))
    {
    $sub_array = array();
    $datetime = explode(" -:", $row["datetime"]);
    $sub_array[] =  array(
         "v" => 'Date(' . $datetime[0] . '000)'
        );
    $sub_array[] =  array(
         "v" => $row["humidity"]
        );
    $sub_array[] =  array(
         "v" => $row["temperature"]
        );
    $rows[] =  array(
        "c" => $sub_array
       );
    }
    $table['rows'] = $rows;
    //echo json_encode($rows);
    $jsonTable = json_encode($table);
    //echo $jsonTable;
    mysqli_free_result($result);
```

```
    //close the db connection
    //mysqli_close($con);
?>
```

Figure 12 PHP query code

Secondly, It's the html part, which displays the table of temperature and humidity data with the Google line chart format like below:

```
<script type="text/javascript">
      google.charts.load('current', {'packages':['corechart']});
      google.charts.setOnLoadCallback(drawChart);
      function drawChart()
      {
            var data = new google.visualization.DataTable(<?php echo
$jsonTable; ?>);

            var options = {
                 title:'Sensors Data',
                 legend:{position:'bottom'},
                 chartArea:{width:'90%', height:'65%'}
                };

            var chart = new
google.visualization.LineChart(document.getElementById('line_chart')
);

            chart.draw(data, options);
      }
     </script>
```

Figure 13 Line Chart code

The format On/Off button and function of it normally and disable function when it reaches to the limit 80 degree:

```
<body>
  <h3 align="center" >Sauna data readings</h3>
  <p/>
  <div class="row">
    <div class="col-md-6" align="right">
        <h4>Current Status: </h4>
    </div>
    <div class="col-md-1 <?php if ($OnOff == 0) echo 'btn-danger';
else echo 'btn-success'; ?>" align="left">
      <h4>
```

```
            <?php
          if($OnOff == 0)
          {
           echo("Off");
          }else
          {
           echo("On");
          }
            ?>
      </h4>
    </div>
  </div >
  <p/>
  <div style="text-align:center">
  <form action="webButton.php" method="post">
  <?php
    if($OnOff == 1){
       if($newTemp > 90){
        // write 0 (off) to database
          $query2 = "INSERT INTO androidData (name) VALUES ('0')";
          mysqli_query($con,$query2);
       }
  ?>
      <div>
       <input type="submit" name="0" class="btn btn-danger"
value="OFF" style="width: 6em;height: 4em"  />
      </div>
  <?php
    }
    else {
      if($newTemp > 80){
  ?>
      <div>
       <input type="button" disable name="1" class="btn" value="ON"
style="width: 6em;height: 4em"  />
      </div>
  <?php
       }
       else {
  ?>
      <div>
       <input type="submit" name="1" class="btn btn-success"
value="ON" style="width: 6em;height: 4em"  />
      </div>
  <?php
       }
    }
  ?>
  </form>
  </div >
```

Figure 14 On/Off function code

The most important thing is that it has the Alarm function. When the temperature is larger than 80 degrees Celsius, the alert message will pop up to notify the customer.

```
<?php
    if($newTemp > 80){
        $message = "*** HIGH TEMPERATURE ALERT !!!";
        echo "<script
type='text/javascript'>alert('$message');</script>";
    }
  ?>
```

Figure 15 Alert code

There are a table with temperature and humidity data and buttons, which Clear all data and Export to Excel with the code below:

```
<table border="1" cellspacing="1" cellpadding="1" >
    <td width="30%" align="center">
        <h4>Temperature / moisture sensor</h4>
        <!-- this table use data from getTempData.php to display in
a paging grid -->
        <table id="tt" class="easyui-datagrid"
style="width:350px;height:380px"
            url="getTempData.php"
            title="Load Data" iconCls="icon-save"
            rownumbers="true" pagination="true">
           <thead>
            <tr>
                <th field="timeStamp" width="45%">Time</th>
                <th field="temperature" width="30%"
align="right">Temperature</th>
                <th field="humidity" width="25%"
align="right">Humidity</th>
            </tr>
           </thead>
        </table>
      <p/>
      <div class="row">
        <div class="col-md-6">
        <input type="button" value="Clear data" class="btn btn-
danger" onclick="window.location.href='./deleteTempLog.php'" />
        </div>
      <div class="col-md-6">
        <input type="button" value="Export to Excel" class="btn btn-
success" onclick="window.location.href='./excel.php'" />
      </div>
      </div>
```

Figure 16 Data table

There are also many other small server-side scripts but it is used to support for this main file such as convert to JSON file or read Android data and more.

Finally, The final product of web page layout is displayed like below:

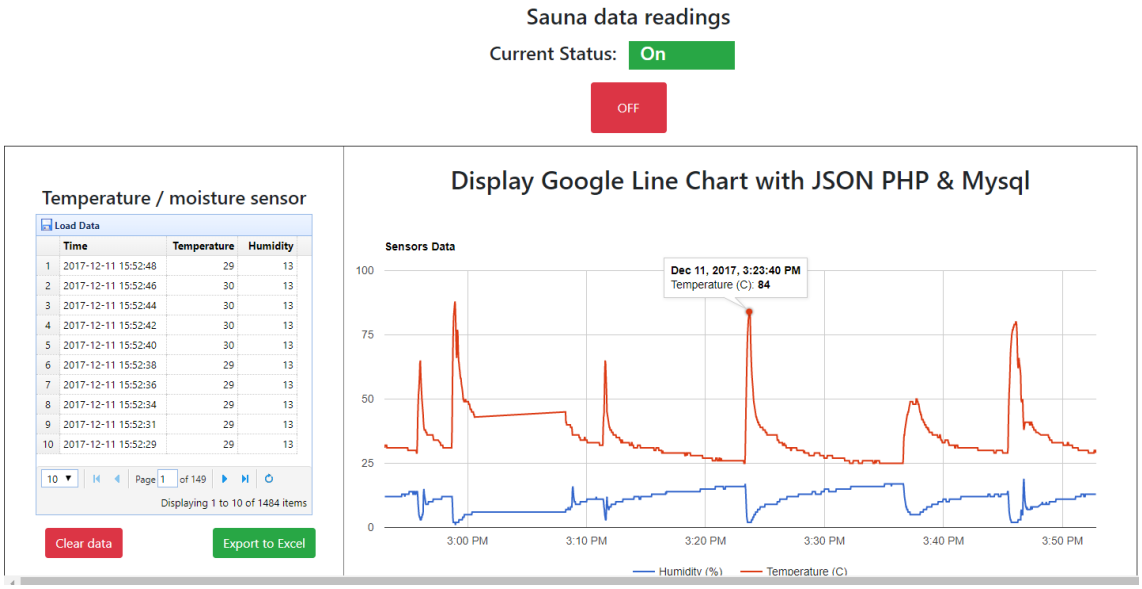

Figure 17 Normal status web pagel

Link of web page :

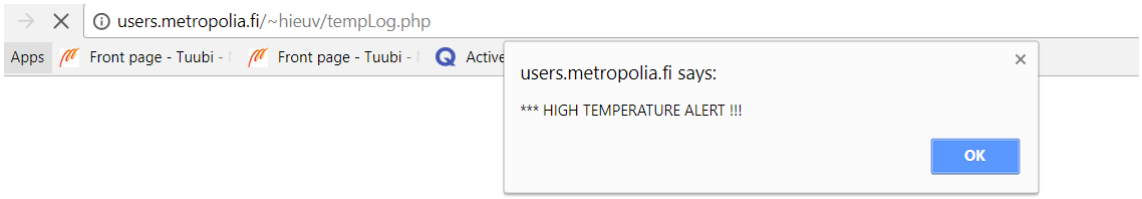

Figure 17.1 Alert message mode

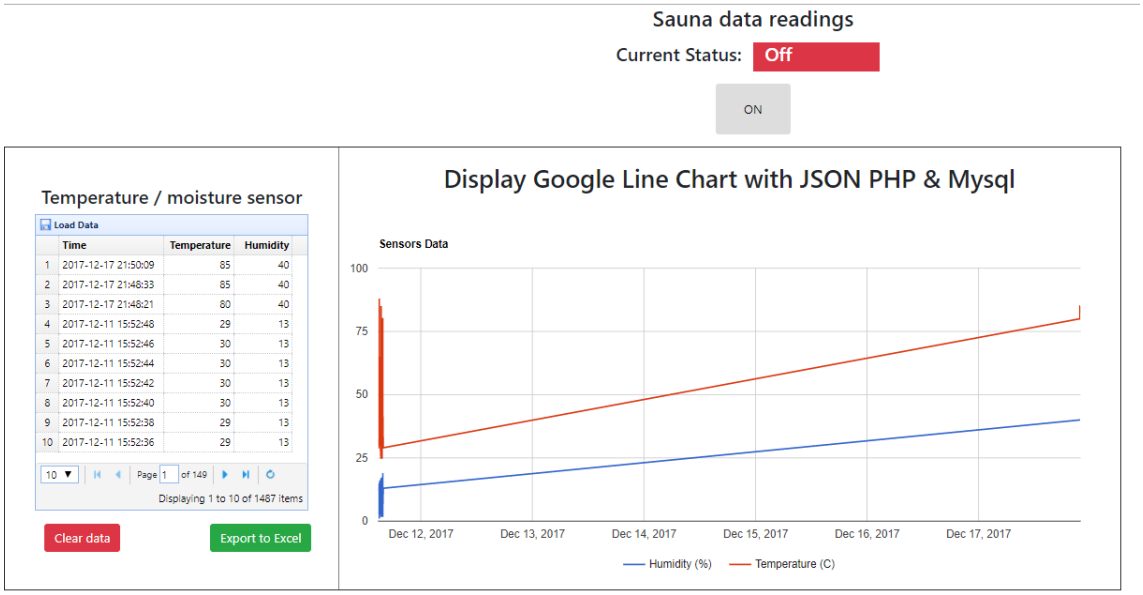

Figure 17.2 On/Off disable mode

7 Mobile Application

In the Mobile part, this device has the jobs that monitor the temperature and humidity data of sauna and On/Off status via web server. The value On/Off not only can be seen but also can be sent.

Since we didn't have much time, therefore, we use framework and just do the best we can to display an app with all necessary features.

First is the code of the displaying the TimeStamp, Temperature, Humidity and Status. 0 newest values. Below is the Temperature displaying task's code, which will display 19. The code will be similar for TimeStamp, Humidity and Status displaying:

```
public class DisplayTemp extends AsyncTask<String, Void,
ArrayList<String>> {
    @Override
    protected ArrayList<String> doInBackground(String...
params) {
        ArrayList<String> respList = new ArrayList<>();

        String stringHTTP =
deleteBlanks("http://users.metropolia.fi/~hieuv/ConvertPHPtoJS
ON.php");

        String contentAsString = "";
        HttpURLConnection urlConnection = null;
        try {
```

```
            // create connection
            URL urlToRequest = new URL(stringHTTP);
            urlConnection = (HttpURLConnection)
urlToRequest.openConnection();
            urlConnection.setReadTimeout(10000); //miliseconds
            urlConnection.setConnectTimeout(15000);
//miliseconds
            // handle issues
            int statusCode = urlConnection.getResponseCode();
            Log.d("MainActivity", "The response is: " +
statusCode);
            if (statusCode ==
HttpURLConnection.HTTP_UNAUTHORIZED) {
                // handle unauthorized (if service requires
user login)
            } else if (statusCode !=
HttpURLConnection.HTTP_OK) {
                // handle any other errors, like 404, 500,..
            }
            // create JSON object from content
            InputStream in = new
BufferedInputStream(urlConnection.getInputStream());
            contentAsString = new
Scanner(in).useDelimiter("\\A").next();

            System.out.println(contentAsString);

            JSONArray entryArray = new
JSONArray(contentAsString);
            String resp;

            for (int i = 0; (i < entryArray.length() -1) && i
< 10; i++) {
                resp =
entryArray.getJSONObject(i).getString("temperature");
                respList.add(resp);
                if(i == 0){
                    int temp = Integer.parseInt(resp);
                    if(temp > 80){
                        warning = true;
                    }
                }
            }

        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return respList;
    }
    protected void onPostExecute(ArrayList<String> list)
    {
        TextView t = findViewById(R.id.result2);
        String result2 = "Temperature\n";
        for (String s: list) {
            Log.d("Main", s);
```

```
            result2 += s + "\n";
        }
        t.setText(result2);
        if(warning){Alert(); new button2task().execute();}
    }
}
```

Figure 18 Display Temperarure

The last few code lines are the one which recognize that the temperature goes above 80 degrees and start to turn on the “warning” global variables and start to send OFF value to server as well as pop up Alarm Notice until the temperature is low down like below:

```
public void Alert(){
    //if(warning){
    AlertDialog.Builder builder = new
AlertDialog.Builder(MainActivity.this);

    builder.setCancelable(true);
    builder.setTitle("High Temperature Alert Dialog");
    builder.setMessage("Fire Risk Alert !");

    builder.setNegativeButton("Cancel", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface,
int i) {
            dialogInterface.cancel();
        }
    });

    builder.setPositiveButton("OK", new
DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface,
int i) {
            alertTextView.setVisibility(View.VISIBLE);
        }
    });
    builder.show();
    //}
}
```

Figure 19 Alert function

When it calls OFF value, it also calls the OFF function from the OFF button function like below and the ON button function will be similar:

```
private class button2task extends AsyncTask<String, Void,
Void> {

    private final HttpClient Client = new DefaultHttpClient();
    private String Content;
    private String Error = null;
    private final String TAG = null;

    @Override
    protected void onPreExecute() {
        Dialog.setMessage("Wait..");
        Dialog.show();
        Log.e(TAG, "-----------------------------------
Here");
    }

    protected Void doInBackground(String... urls) {

        try {
            HttpPost httppost = new HttpPost(URL);

            // Field
            JSONObject jObject = new JSONObject();
            jObject.put("name", "0");

            MultipartEntityBuilder se =
MultipartEntityBuilder.create();
            se.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);
            se.addPart("request", new
StringBody(jObject.toString(), ContentType.TEXT_PLAIN));
            HttpEntity entity = se.build();

            ByteArrayOutputStream baos = new
ByteArrayOutputStream();
            entity.writeTo(baos);
            Log.d("seMain", baos.toString());
            httppost.setEntity(entity);

            HttpResponse resp = Client.execute(httppost);

            Content = EntityUtils.toString(resp.getEntity());

            Log.e("Response", "-------------------------------
-" + Content);

        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ClientProtocolException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
```

```
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Void result) {
        // TODO Auto-generated method stub
        Dialog.dismiss();
        // Log.e("ERROR", "--------------------------------" +
Content);
        Toast.makeText(getBaseContext(), Content,
Toast.LENGTH_LONG).show();
    }
}
```

Figure 20 Off function

At last, there is a function to keep refreshing the application every 15 seconds or you can use the Reload button to refresh the page with the same function startActivity(intent) :

```
Thread t = new Thread() {
    @Override
    public void run() {
        try {
            while (!isInterrupted()) {
                Thread.sleep(30000);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        // update TextView here!
                        Intent intent = getIntent();
                        finish();
                        startActivity(intent);
                    }
                });
            }
        } catch (InterruptedException e) {
        }
    }
};
t.start();
```

Figure 21 Auto reload code

Finally, the completed simple Sauna monitor application looks like below:

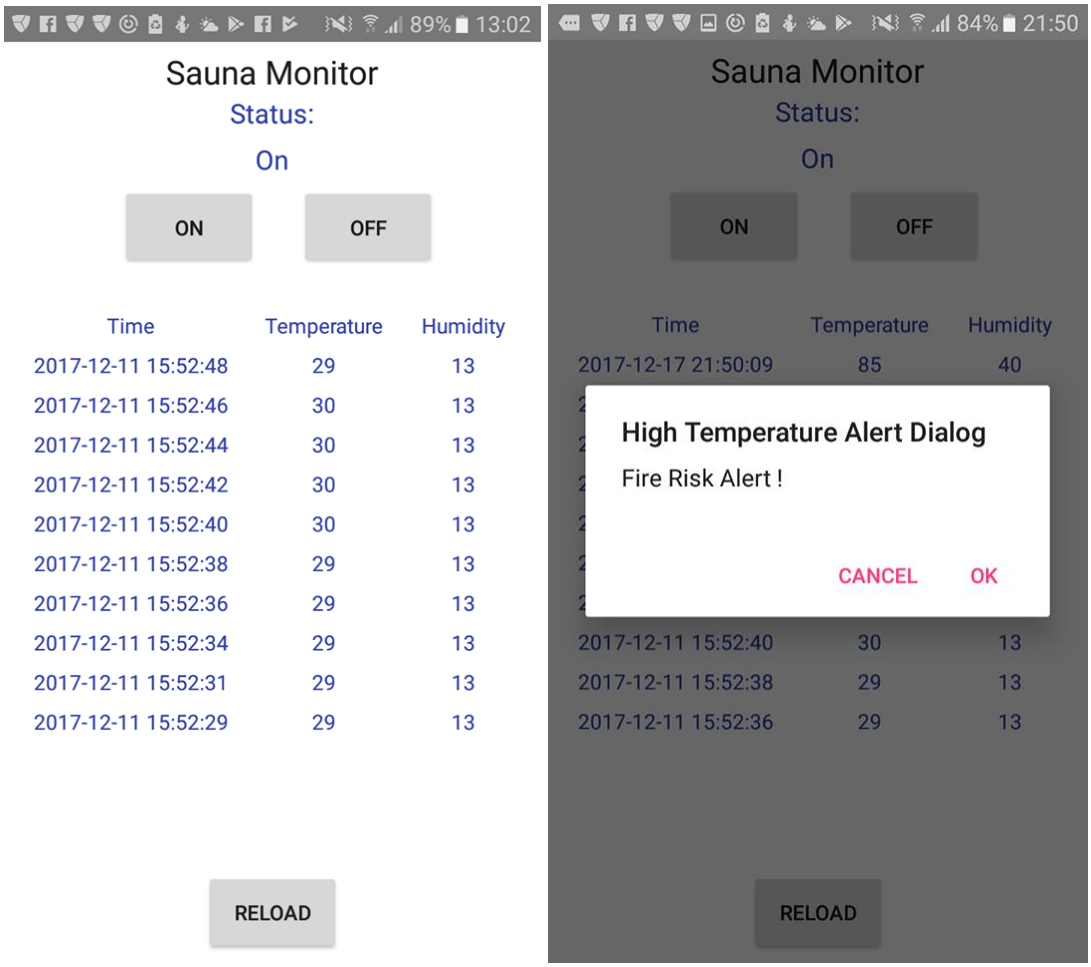

Figure 22 Android application normally and pop-up notice mode

8 Conclusion

To do this project, we faced with many issues from finding and configuring hardware to solving software problems. There are many things that we haven't confronted before such as working with physical layer, configuring network interface with TCP/IP to establishing web server and creating a mobile application. However, there is finally some completely and functionally results for this project as we discussed all of it above. There are still many things that we want to improve for this project such as installing real button to monitor it or real time updating, however, with only a limited time, we did the best we could, learned and improved as much as we were possible.

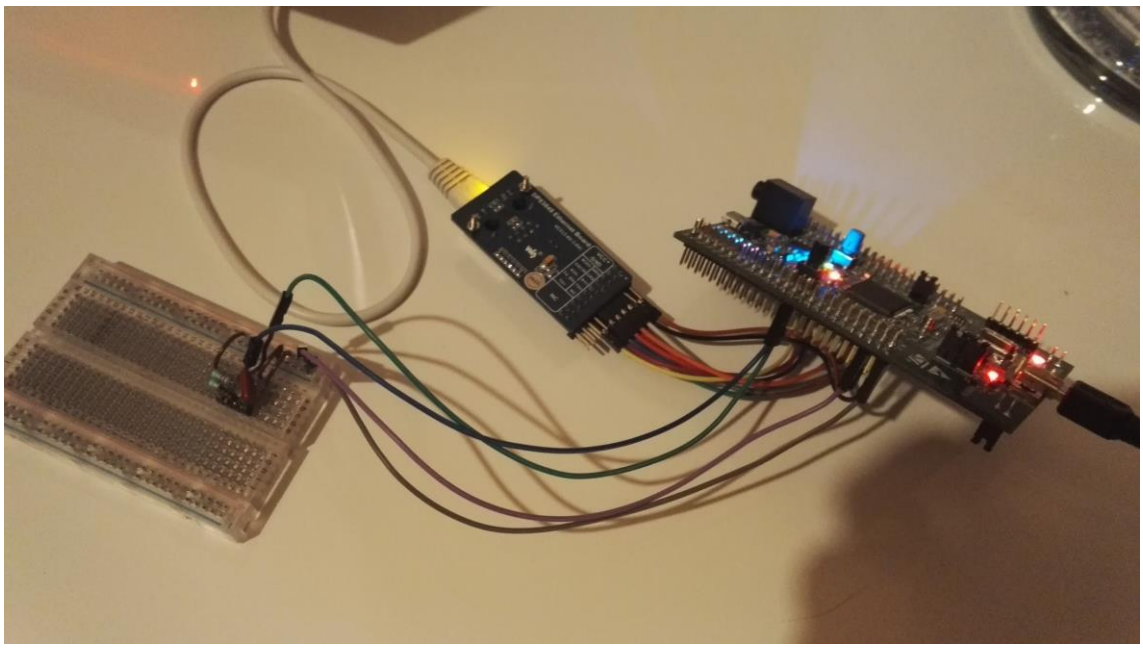

Figure 23 End product

9 Reference

Android, n.d, Dialogs,
https://developer.android.com/guide/topics/ui/dialogs.html

Aneh Thakur, 2014, Send Data to Server from Android application,
https://trinitytuts.com/send-data-to-server-android-application/
EasyUI, n.d, Add a pagination to DataGrid,
https://www.jeasyui.com/tutorial/datagrid/datagrid2.php

NTU, n.d, HTTP (HyperText Transfer Protocol) Basics,
https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

W3schools, n.d, Js_json_php, https://www.w3schools.com/js/js_json_php.asp

Webslesson, 2017, How to make Google line chart by using php json data,
http://www.webslesson.info/2017/08/how-to-make-google-line-chart-by-using-php-json-data.html

Low level programming references:

Manuals:

- reference manual for STM32
- user manual for STM32
- datasheet for STM32

- reference manual for STM32 HAL and LL drivers
- reference manual for DP83848 PHY device

- reference manual for HIH8120 humidity/temperature sensor
- manual for I2C communication interface for HIH8120

Books:

- IBM's TCP/IP Tutorial and Technical Overview
- Mastering the FreeRTOS

Web sites:

- FreeRTOSPlusTCP/IP: https://www.freertos.org/FreeRTOS-Plus/FreeRTOS_Plus_TCP/index.html
- Padding and packing in C: http://www.catb.org/esr/structure-packing/