VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Petrik Loukola

REMOTE CONTROLLED CAR HEATER USING A RASPBERRY PI

Technology & Communication
2015

FOREWORD

This thesis was made in 2015 at Vaasa University of Applied Sciences.

I thank my supervisor Mr. Smail Menani for support and advice. Thanks for Mr. Jukka Matila and Mr. Ari Urpiola who helped in the technical field and also thanks to my school mates Joakim Mattson and Sam Harry Szavar by giving great advice. The schedule to make this project was very tight and without help from the mentioned people this project would have been hard to finish in time.

Petrik Loukola
24 April 2015

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Petrik Loukola
Opinnäytetyön nimi	Auton lämmittimien etäojhaus Raspberry Pi:llä
Vuosi	2015
Kieli	Englanti
Sivumäärä	51 + 10 liitettä
Ohjaaja	Smail Menani

Tämän opinnäytetyön tavoite oli suunnitella ja toteuttaa tegnologia, jolla voidaan etäohjata auton moottori- ja sisätilanlämmittiä ja valvoa auton lämpötilaa verkkoselaimen avulla Wi-Fi verkossa.

Idea tähän työhön tuli Suomen kylmästä ilmastosta. Suomessa talvet voivat olla kylmiä ja, jos autoa aikoo käyttää, on suositeltavaa, että auton moottori ja sisätila esilämmitetään ennen ajoa. Esilämmittäminen tekee ajamisesta turvallisempaa, auto tuottaa vähemmän päästöjä, auton moottorin kesto pitenee ja auto myös kuluttaa vähemnmän.

Tämän työn on tarkoitus helpottaa käyttäjiä auton esilämmittämisessä ilman, että käyttäjän tarvitsee mennä joka kerta ulos kun auto esilämmitystä tarvitsee.

Tässä työssä käytetään Raspberry Pi tietokonetta, itse valmistamaa piirikorttia, Defa PlugIn relettä ja DS18B20 lämpötila-anturia.

Tämän työn tavoitteet saavutettiin. Auton lämmittimien ohjaus ja lämpötila tarkkailu etänä toimi kuten oli tarkoituskin.

Keywords Raspberry Pi, rele, Wi-Fi, etäohjaus, lämpötila

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikan koulutusohjelma

ABSTRACT

Author	Petrik Loukola
Title	Remote controlled car heater using a Raspberry Pi
Year	2015
Language	English
Pages	51 + 10 Appendices
Name of Supervisor	Smail Menani

In this thesis, the goal was to design and implement a technology with which the user can control the car heaters and monitor the car temperature via Wi-Fi network and web browser.

The idea for this project came from the Finnish cold climate. Winters in Finland can be very cold and if the car owners want to drive their cars, it is recommended to preheat the car engine and interior before driving the car. This will make driving safer, the car will make less pollution, better for the engines lifetime, and better for the gas consumption.

The benefit of this project is to help the user to set the car heaters on and off easier from home without going out every time when the car needs heating.

This system was made by using Raspberry Pi computer, self-made PCB, Defa PlugIn relay and DS18B20 thermometer.

The goals of this project were overall achieved. Controlling the car heaters remotely and monitoring the temperature with web browser was working as expected.

Keywords Raspberry Pi, relay, Wi-Fi, remote control, temperature

CONTENTS

LIST OF ABBREVIATIONS

ADC	Analog-to-digital converter
dBm	Decibel-milliwatt
DC	Direct connect
GPIO	General purpose input/output
GPU	Graphics Processing Unit
GUI	Graphical user interface
HTTPd	Hypertext Transfer Protocol Daemon
LED	Light-Emitting Diode
Mbps	Megabit per second
MHz	Megahertz
NCSA	National Center for Supercomputing Applications
OS	Operating system
PCB	Printed circuit board
SD	Secure Digital
SDHC	Secure Digital High Capacity
SoC	System on a chip
UV	Ultraviolet
Wi-Fi	Wireless local area network

1 INTRODUCTION

People who live in colder climates are using car heaters to set the heating on so the car would be warm before driving it. The mains reason to heat the car engine and the interior space is that the driving would be safer, the engine would last longer, there would be less pollution, less gas consumption and also it makes the driving more comfortable. Because of the high cost of the heating systems, most commonly the car owners have in the car the basic heating system which contains the engine and the interior heater. This means that the user needs to go outside and set the heating on/off by attaching the power cable to the car's heating system.

The objective of this thesis is to help car users to set the car heating on/off from home with a little bit less effort. With this system the user can remotely set the car heating on/off with any given device that have access to the home Wi-Fi network and have web browser installed on, considering that the user is keeping the power cable connected to the power grid and to the car heating system. The user just needs to open a web browser and type the right website address. On there the user can see a button which will set the car heating on/off. Also, the website shows the temperature from the interior of car.

There are two major car heating system providers in the Nordic area which are called Defa and Calix. This thesis project is designed to work with the Defa heating system, because the Defa products are used in the most vehicles.

1.1 Project Background

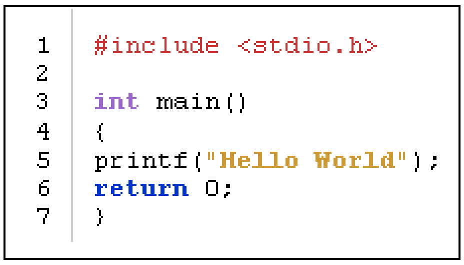
The idea for this project came from Finland's cold climate. In Finland and in the Nordic areas the winter can be very cold and a lot of the people own a car here and it is recommended that cars will have heating systems fitted into the car. The idea came one morning after waking up. The weather was very freezing and that meant that it was better to go outside and set the car heating on before driving to school. At that moment many people might not want to go immediately after waking up into this freezing weather and set the heating on. At that moment an idea came that how this could be done in an easier way from home without going outside to the cold and the answer to that was to make a system which will set the heating on/off remotely.

2 USED TECHNOLOGIES

2.1 C Programming Language

The C is general-purpose, procedural, imperative computer programming language. The C was developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is an outgrowth of two earlier languages, called BCPL and B, they were also developed at Bell Laboratories.

Computer professionals impressed with C's various features and began to promote the use of C. The popularity of C became widespread in the mid-1980s. Many C interpreters and compilers were written for computers of all sizes and many commercial application programs were developed. Numerous software products that were originally written in other languages were rewritten in C, because it was more advanced. Figure 1 shows a simple C code example what prints out text 'Hello World'. /2/



```c
#include <stdio.h>

int main()
{
printf("Hello World");
return 0;
}
```

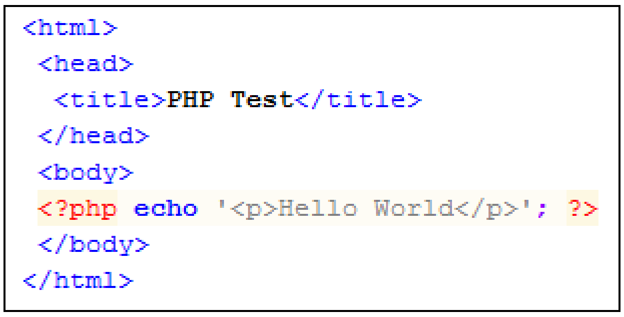
Figure 1. Example of a C code

2.2 PHP

PHP is a server-side scripting language and it is designed for web development. It was created by Rasmus Lerdorf in 1994. PHP was originally to help coders to maintain personal home pages, but later the PHP grew from its purpose.

Nowadays PHP is known as HyperText Preprocessor and usually written in an HTML context. PHP script is not sent to a user by the server like an ordinary HTML page. Elements in HTML the script are left alone, but the PHP code is rendered and performed. The PHP code in a script can produce images, write and read files, talk to remote servers, query databases and many other different functions what is capable to do. The output from PHP code is combined with the HTML and then delivered to the user.

In Figure 2 a very simple PHP program is shown which display *Hello World* text using the PHP echo statement. /13/

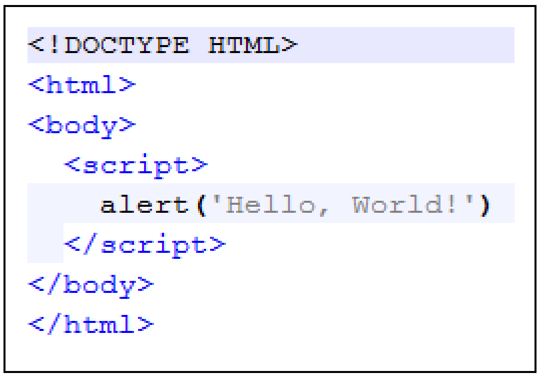


```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

Figure 2. Simple example of a PHP code

2.3 JavaScript

JavaScript is a programming language that lets the user supercharge HTML with interactivity, dynamic visual effects, and animation. Invented in 1995 by Netscape it is nearly as old as the web itself. /8/

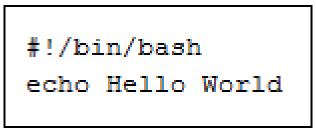


```
<!DOCTYPE HTML>
<html>
<body>
  <script>
    alert('Hello, World!')
  </script>
</body>
</html>
```

Figure 3. Example of a JavaScript

2.4 Bash script

Bash is a command-line interface for interacting with the OS. Bash is a default shell on many GNU/Linux distributions and on Mac OS X. It was created by a programmer named Brian Fox in the late 1980s. Bash like many other shells has ability to run entire script of commands. It is known as Bash script or Bash shell script. Bash script might contain just simple list of commands or might contain loops, conditional constructs, functions, and more. /10/



```
#!/bin/bash
echo Hello World
```

Figure 4. Example of a Bash script

2.5 CGI

The CGI is Common Gateway Interface. It allows Web server to run applications that communicate with Web pages. CGI is run on the server which makes it very useful for applications that wants to have access to files and facilities available on the server. The server is responsible for data transfer, managing connection and network issues related to the client request, where the CGI script handles the application issues, such as data access and document processing. /9/

2.6 Apache Web Server

Apache Web Server is a web server application. It is commonly referred to as Apache. Apache helps to deliver web content to be accessed through the Internet. It is originally based on the NCSA HTTPd server. The development of Apache started in early 1995 after work on NCSA code stalled. Apache overtook NCSA HTTPd. It became the first web server software, to serve over 100 million websites in 2009. Apache is the most popular HTTP server in use and it is developed and maintained by an open community developers. In 2013, Apache has served 54.2% of all websites. Apache Web Server's original author is Robert McCool and it is written in C language. /1/

3 USED DEVIDES AND COMPONENTS

3.1 Raspberry Pi

The Raspberry Pi is a series of low cost credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools.

The Raspberry Pi is a computer that plugs into a computer monitor or TV, and uses standard keyboard and mouse. It is a small device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch, Python and C. It is capable of doing everything a desktop computer is expected to do, from browsing the Internet and playing HD video, to making spreadsheets, word-processing and playing games.

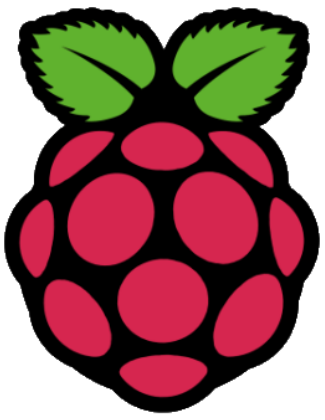


Figure 5. Raspberry Pi logo /11/

3.1.1 Raspberry Pi Model B revision 2.0

For this thesis the Raspberry Pi Model B revision 2.0 (figure 3.) was used. It is the higher-spec variant of the revision 1.0. It has 512-MB RAM. RPi have 2 * USB 2.0 ports, 1 * HDMI, 1 * RCA, 1 * 3.5 mm jack, 1 * 10/100 Ethernet LAN and 1 * SD card slot. The SoC (system on a chip) is Broadcom BCM2835. This contains an ARM ARM1176JZFS, floating point, running at 700 MHz, and VideoCore 4 GPU. The GPU is capable of BluRay quality, using H.263 at 40 Mbps. It has fast 3D core which can be accessed using the supplied OpenGL ES2.0 and OpenVG libraries. For programming C/C++, Perl, Java, PHP/MySQL, Scratch can be used and many other that can run under Linux. The Raspberry Pi’s power consumption is 700mA and 3.5W. In Figure 6 there is a 2 € coin to refer how small this Raspberry Pi really is. /11/

Figure 6. Raspberry Pi Model B revision 2.0

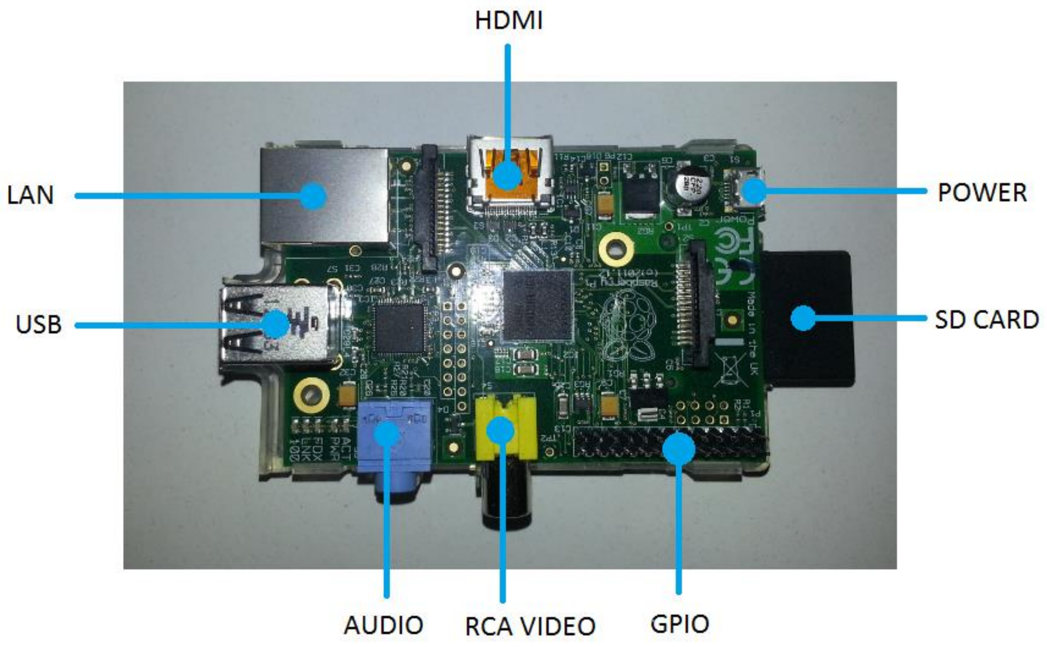


Figure 7. Raspberry Pi Model B revision 2.0 device connections

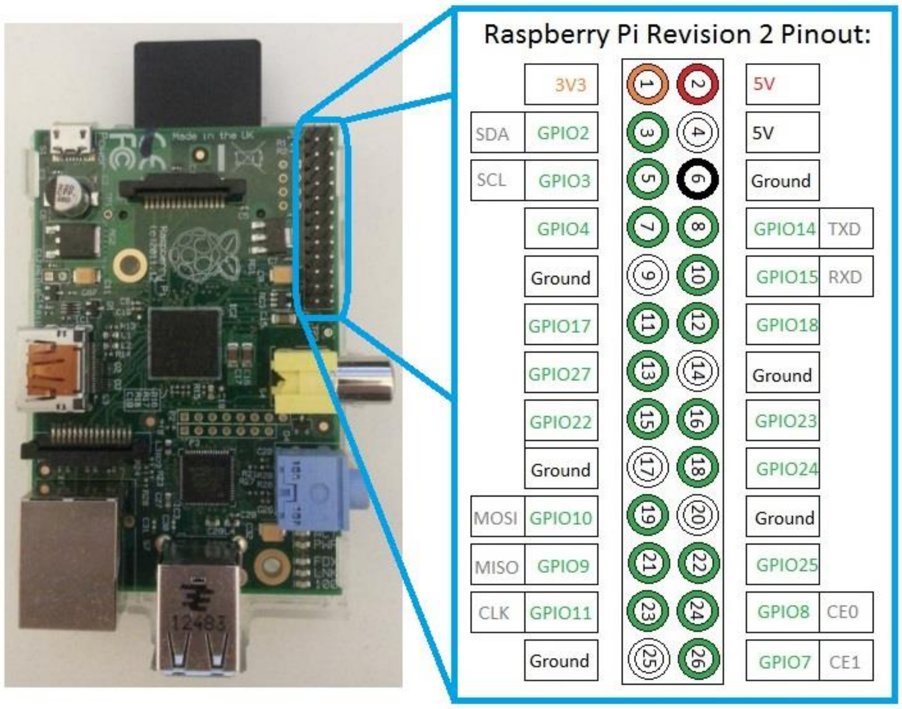


Figure 8. Raspberry Pi Model B revision 2.0 pinout

3.2 Defa PlugIn Relay

DEFA PlugIn relay (12V) (Figure 9) is used for controlling the interior and engine heater on and off. It has one input for the power source. The two outputs are for the interior and engine heaters. The relay is controlled on/off from the two +/- flat connectors. /3/

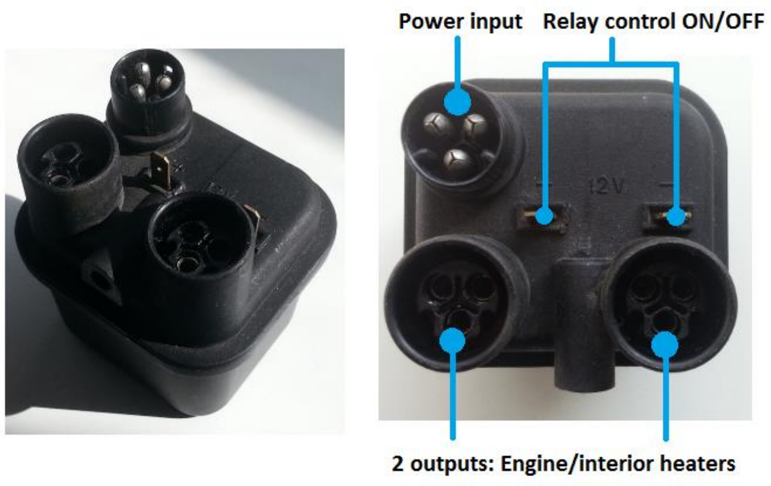


Figure 9. Defa PlugIn relay

3.3 Waterproof DS18B20 Digital Thermometer

A pre-wired and waterproofed version of the DS18B20 sensor (Figure 10) was sued in the thesis. The DS18B20 is 1-Wire digital thermometer and it is capable to provide 9-bit to 12-bit Celsius temperature measurements. DS18B20 requires only one data line to communicate with the microprocessor. The DS18B20 can use also power directly from the data line. The power supply range is 3.0V to 5.5V. The operating range for temperature measuring is -55°C to +125°C. DS18B20 is accurate to ±0,5°C over the range of -10°C to +85°C. Every DS18B20 has a 64-bit unique serial code and it allows multiple DS18B20 to work on the same 1-Wire bus. The cable jacketed in PVC, so it is suggested to keep it under 100°C. /5/

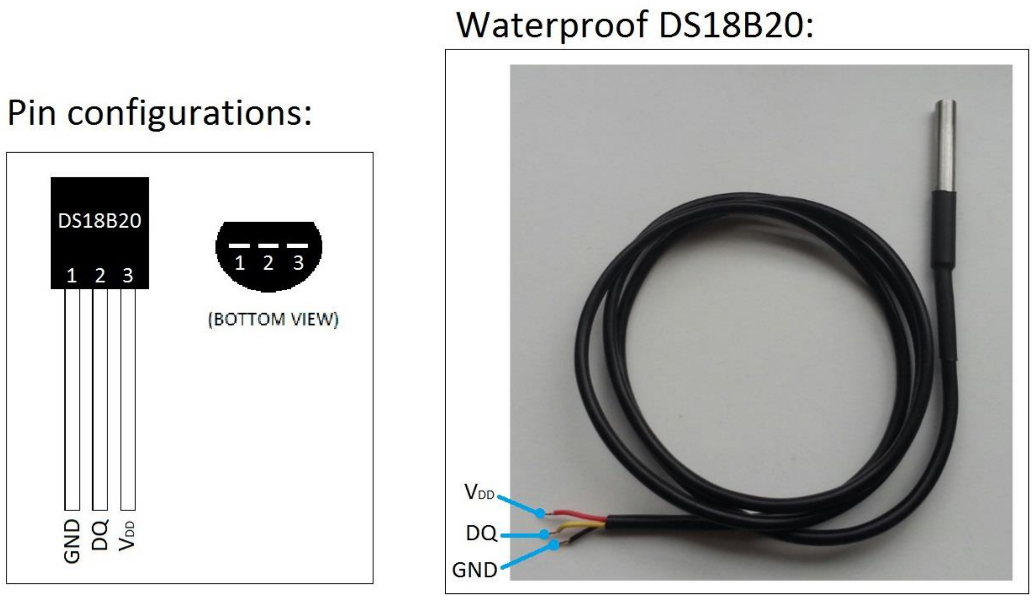


Figure 10. Waterproof DS18B20 and pin configurations

3.4 EDUP EP-N8531

EDUP EP-N8531 is a wireless USB adapter. It allows the user to connect it to a desktop, notebook or Raspberry Pi computer to a wireless network and access high-speed Internet connection. The frequency range is 2.4-2.4835 GHz. EDUP EP-N8531's compliance Standard is IEEE 802.11b/g/n, so it uses the latest Wireless Technology. It provides speed of up to 150Mbps. It can be connected to the computer through USB and has a weight of 125g. It supports the network protocol CSMA/CA with ACK. As security features there are 64/128/152 bit WEB data encryption, WPA, IEEE 802.1X, TKIP and AES.

Figure 11. EDUP EP-N8531 /6/

4 REQUIREMENTS

The requirements for this project are that the user will be able to set the car engine and interior heaters on/off remotely from a website. The website also needs to display temperature so the user will know if the car needs heating or not. The system needs to be made that it will work with the Defa interior and engine heater products. An important thing what needs to be born in mind is that this system should not get too expensive.

4.1 Analyzing the Requirements

First of all to control the Defa engine and interior heaters from a website is to have a control unit inside the car which is capable of receiving commands from the website and setting heating on/off. Also this unit will need to measure the temperature which will be displayed on the website. This control unit will be fitted inside the car so it is better to keep it as small as possible. To control the car engine and interior heaters on/off, the control unit will need a switch that can turn on/off higher currents, because car heaters takes power from the grid (AC 230V). This can be done by using a relay. The Defa company already have a relay for this type of use and it is called Defa PlugIn relay (12V). The control unit will also need a temperature sensor to measure the temperature. One issue to solve was where the control unit will get power, if it could be the car battery or use power from the grid.

As the idea was to make this system work in home environment, the Wi-Fi network was good enough and nowadays the work range of the Wi-Fi routers are quite good. A drawback in using Wi-Fi network is if the user wants to use this system outside of the home network, it can be hard, because not all places have Wi-Fi and if even there is a Wi-Fi available near the car, the user needs to access it first. Another way that this can be done is that the control unit would have its own mobile network, but this would mean that the user needs to make a contract with the network operator and pay monthly fees and eventually this system would get quite expensive.

The obvious choice that the control unit will be was to use Raspberry Pi computer, because they are very small, powerful and it uses 5V to operate. The best thing of those is that the Raspberry Pi computers are very cheap. The Raspberry Pi computers cost from 35 € to 50 €. The Raspberry Pi can have a web server which means it can provide the user interface which will be a website. The Raspberry Pi can receive temperature data and switching heating on/off by using its GPIO pins. The GPIO pins are physical interface between the Pi and the outside world. With these GPIO pins the user can make the Raspberry Pi for example to receive data, send data or control switches, LEDs and relays. The GPIO pins can be controlled with many different programming languages, such as C, Python, Scratch and many more. Raspberry Pi model B revision 2 is a good choice for this project. The Raspberry Pi model B does not take much power so this power could be taken from the car battery. The power consumption of Raspberry Pi model B is in normal use 3.5W and 700mA, a normal car battery is around 50Ah and Defa relay takes 70mA when it is turned on. Altogether this makes 50Ah / 0.77mA = 64.9h, so in theory the Raspberry Pi with the Defa relay could run 2 days 17hours without charging the battery. The system could be fitted with a maintenance charger, but this would increase the cost of the system a lot, but to get the right power consumption value, the system needs to be first operational, before thinking to add a maintenance charger.

The next step was to decide which temperature sensor it was possible to use. The Raspberry Pi does not have ADC (Analog-to-digital converter) so it is better to use a digital temperature sensor. It is possible to use an analog temperature sensor, but it will require more electrical components to work than the digital sensor would need. The sensor which is widely used by people in Raspberry Pi computers is DS18B20 thermometer. It is a particularly popular sensor, because it is inexpensive and easy to use, providing calibrated digital temperature readings directly, so the DS18B20 was suitable for this project.

To make the Raspberry Pi computer work with the Defa PlugIn relay and the DS18B20 it was necessary to have a custom circuit board between them, because to control the Defa PlugIn relay from the Raspberry Pi’s GPIO pin it will need a transistor and few other components to work. To read the DS18B20 temperature sensor it will need one resistor between the sensor and GPIO pin. Also it is good to add on this circuit a 12V to 5V DC converter, because it was planned to take power from the car battery and between the car battery and Raspberry Pi it is necessary to use DC converter, because the car battery gives 12 volts and the Raspberry Pi operates on 5 Volts.

Now, the Raspberry Pi, Defa PlugIn relay, DS18B20 temperature sensor and custom circuit board are chosen for this project, it is time to look how the message sequence can be done. Between the user interface and the control unit, there are many different ways to do the message sequence, because the Raspberry Pi is capable of working with many different programming languages.

At start the purpose was to do the message sequence by using Python and PHP and it basically works, when the user presses a button on the website to set the relay on/off, the PHP script saves a value ‘1’ or ‘0’ into the text file . Then the Python script is checking every two seconds this text file and if the file have ‘1’ it will set the relay ON and if the value is ‘0’ it will set the relay OFF. The problem with this type of message sequence is that when the PHP script is re-writing the value into the text file every time the button is pressed, it will break the SD card eventually, because the SD cards has limited write cycles. SD cards write cycles are usually around 100 000, which is a lot, and more likely it will never run out in this system, but this is not very professional way to do it.

While doing research how to do the message sequence more appropriately without breaking the SD card, an idea came that this could be done by using PHP, JavaScript, CGI, Bash, and C. A short explanation of this more complex method how this can be done is that when the user gives a command by pressing a button on/off on a webpage made in PHP, it will make AJAX request to the Bash script with CGI and the Bash script calls the client program made in C. This client program will write ON or OFF into a named pipe. Another C program will read this named pipe and if the value in the named pipe is ON, the program will set GPIO pin on active state ‘1’ which will set the heating ON and if the value is OFF, the GPIO is on state ‘0’ by setting heating OFF. A good thing with using

AJAX is that it updates only part of the web page, without reloading the whole page. The temperature reading can be done by only using PHP.

5 SETTING UP THE RASPBERRY PI

The first thing to do was to install the operating system into the Raspberry Pi and do a few main set ups.

5.1 Operating System

There were few operating systems where to choose from: Raspbian, Snappy Ubuntu Core, OpenELEC, Raspbmc, Pidora and Risc OS. For this project the Raspbian OS was chosen, because it is easy to use and it is quite fast and light. Raspbian OS gets much more attention and development support from Raspberry Pi Foundation than the other distributors for the Raspberry Pi. A big advantage is that it has good selection of teaching and educational materials. It supports recommend overclocking limits and future official hardware add-ons. Raspbian is also fairly easy and fast to set up. /1/ /14/

5.2 Wi-Fi

No model of the Raspberry Pi is fitted with Wi-Fi, because the SoC does not support native Wi-Fi, and if they would add an additional built in Wi-Fi chip the RPi price would be highly increased. To get Wi-Fi working on Raspberry Pi, it is necessary to use USB Wi-Fi dongle. The Wi-Fi dongle which was used in this project is EDUP EP-N8531. To connect the Raspberry Pi with the specific Wi-Fi network can be done by using terminal or Wi-Fi Config program.

5.3 Hostname

Stock Rasbian hostname *raspberrypi* was changed to *carpcserver*. It is better to change Raspberry Pi's hostname, so users could recognize right Raspberry Pi from the Wi-Fi network and also to avoid name conflicts with the other Raspberry Pi devices. To change the hostname is fairly simple, one only needs to modify files *hosts* and *hostname* in the /etc/ directory.

5.4 Web Server

For this project it was essential to install the web server for the Raspberry Pi, because the web server allows it to serve web pages and the controlling heating and monitoring temperature is done from the web page. The web server which was used for this project was Apache.

6 POWER AND MEASUREMENT UNIT

Power and measurement unit is an additional circuit board for the Raspberry Pi and this chapter explains how this unit was made. The project started by designing and making the circuit board for powering the Raspberry Pi, to control the Defa PlugIn relay and temperature measurement by using the DS18B20 thermometer. The PCB also contains 12V to 5V DC converter, because the Raspberry Pi will take the power from car battery and the car battery is 12 volts and the Raspberry Pi operates on 5 volts.

The control of the Defa PlugIn relay from the Raspberry Pi’s GPIO pin is done by using a transistor to energize the coils of the relay. The relay also needs a 12V power supply. This power is also taken from the car’s 12V battery, but it is better to put a diode and capacitors between the power supply and the circuit to protect the circuit.

To connect the waterproof DS18B20 temperature sensor with the Raspberry Pi, the circuit board needs three position connector and one pull-up resistor.

To power the Raspberry Pi from the car battery it is better to use a buck converter to reduce the voltage of a DC power, because it is very efficient. Originally the purpose was to create a separate circuit board for the 12V to 5V DC converter part, but it turned out that there is more than enough space to put everything on the same circuit board.

The schematic and layout of the circuit board was made with PADS Logic and Pads Layout. The actual circuit board was made in the school laboratory. There were two different ways to make the circuit board by using etching method or CNC mill, but the CNC mill in the school laboratory was broken at the time so the etching method had to be used.

6.1 PCB schematic

Figure 12 shows the complete schematic of PCB and it was made with Pads Logic software.

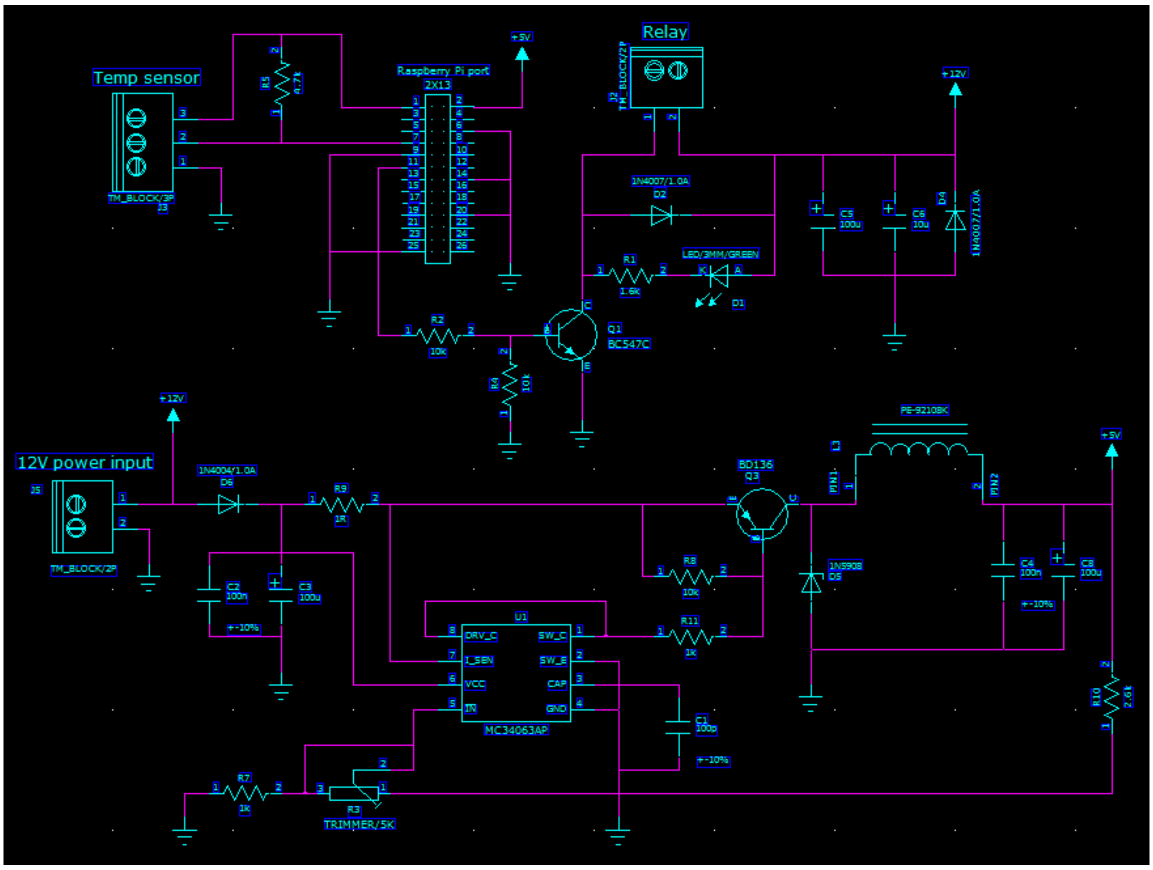


Figure 12. PCB schematic

Figure 13 shows temperature sensor connector for the waterproof DS18B20, the resistor and Raspberry Pi 2x13 female connector. The temperature sensor is (GPIO4) for data and pin 3 to Raspberry Pi pin 1 (3V3/ 3.3V) for power. The resistor was used as a pull-up for the data-line and is required to keep the data transfer stable.

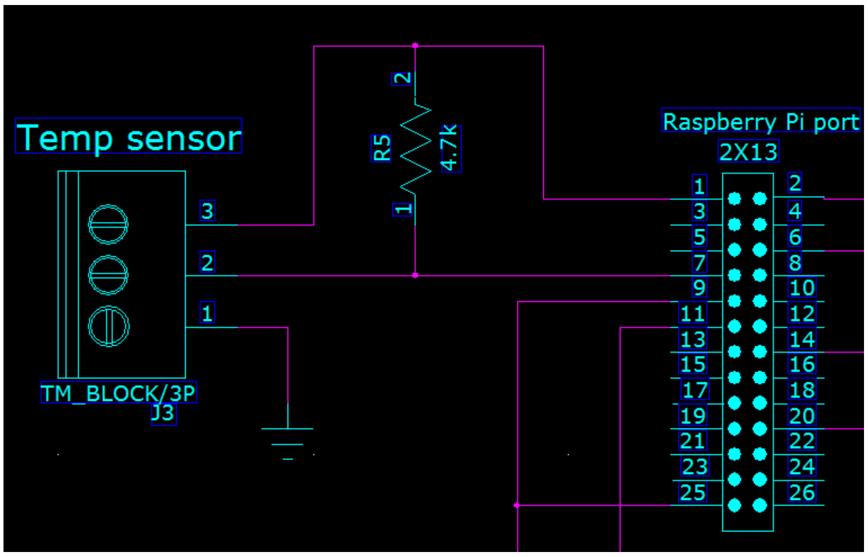


Figure 13. Temp sensor part

In the relay control part of the circuit (Figure 14) is to control the Defa PlugIn relay. The purpose of the transistor BC547C is to work as a switch to energize the relay coil. When the user sets Raspberry Pi’s GPIO17 (pin 11) on active state (1), the current flowing from the transistor’s Base to Emitter (B to E) and then the relay coil current flows through the transistor from the Collector to Emitter (C to E) by setting the relay ON. The LED D1 was added into the circuit to indicate when the relay is on/off. As the Defa PlugIn relay is 12V, it will need a 12V power source which is taken from the car battery. When the power comes from the car battery to the relay, the capacitors are there to protect the circuit from voltage spikes and the diode D2 protects against the reverse current.

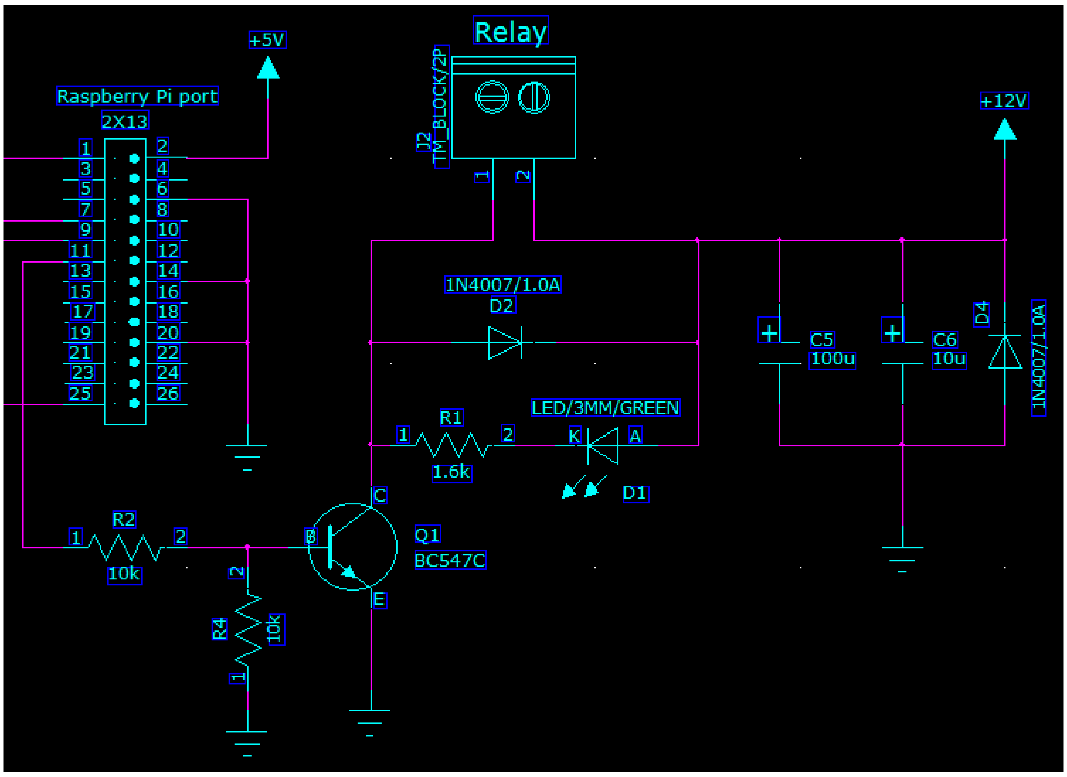


Figure 14. Relay control part

The DC converter part (Figure 15) is to convert car battery 12V DC voltage to 5 volts. The easiest method to reduce the voltage of a DC power supply is to use a linear regulator like the 7805. A problem with the linear regulator is that it wastes energy as they operate by dissipating excess power as heat which is not very efficient. A better way for this is to use a step down buck converter which is much more efficient. The DC converter circuit is based on MC34063AP switching regulator, which is a SMPS driver with ON/OFF control loop an it is designed to work on the continues mode. /4/

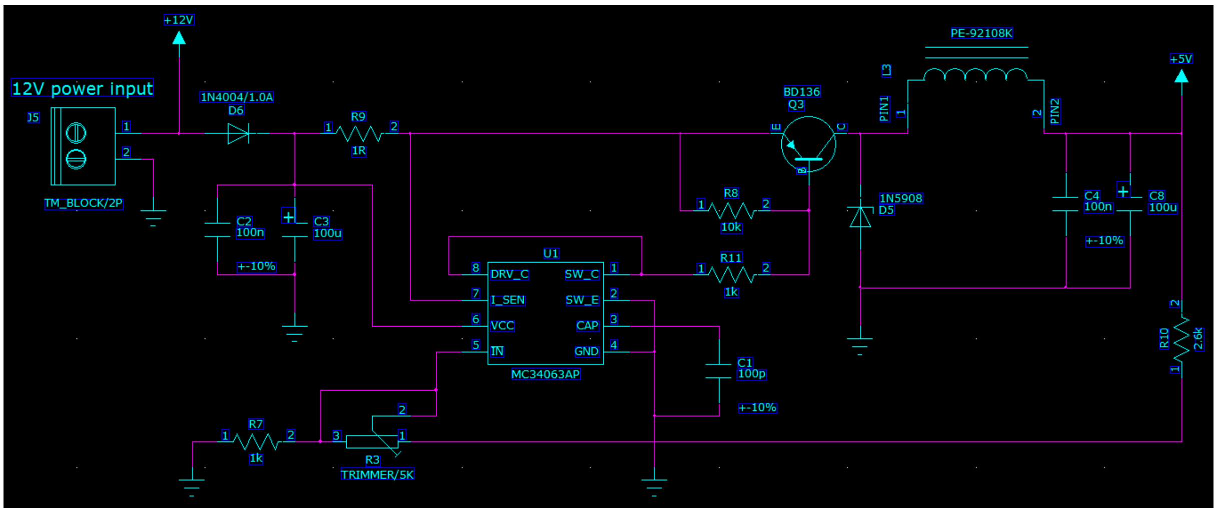


Figure 15. 12V to 5V DC converter part

6.2 Testing with Breadboard

Before the actually circuit board was made, the circuit was tested by using a breadboard. A breadboard is a construction base for prototyping of electronics and is very useful for testing circuits, because to connect the components and wires it does not need any soldering.

First the components and wires were attached on the breadboard, and then first thing was to test the DC converter part of the circuit that it will give the right voltage so it will not break the Raspberry Pi. The DC converter have a 5K trimmer resistor (R3) to get the right voltage. By adjusting the R3 with a screwdriver and using digital multimeter to measure voltage, the 5V could be found. When the right voltage was found, then the Raspberry Pi could be added to the circuit and the Defa PlugIn relay tested and the DS18B20 in action by controlling them with GPIO pins. This testing can be done without coding. The only thing requires is to power up the Raspberry Pi and start the terminal from the Raspbian OS.

6.2.1 DS18B20 Test

First the DS18B20 temperature sensor was tested by writing the command into the terminal to load the 1-wire communication device kernel modules:

```
sudo modprobe w1-gpio && sudo modprobe w1_therm
```

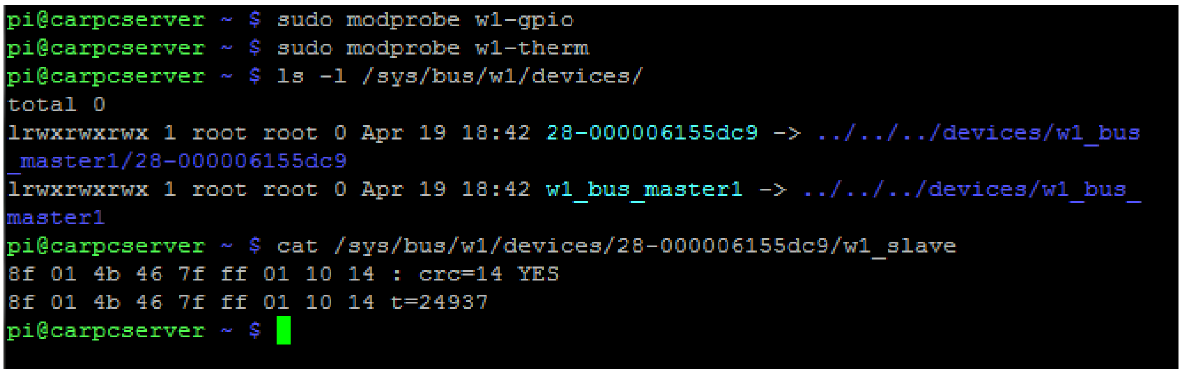
Then to see the list of the connected DS18B20 sensors, the following command line needed to be written:

```
ls –l /sys/bus/w1/devices/
```

The DS18B20 temperature sensor used in this project appeared with an address in the format 28-00000xxxxxx. All DS18B20 sensors have their own unique address. The address of this specific DS18B20 sensor which was used in this project is 28-000006155dc9. To display the temperature of this DS18B20 this command needed to be written:

```
cat /sys/bus/w1/devices/28-000006155dc9/w1_slave
```

The temperate then was displayed on to the terminal screen. The DS18B20 gave the temperature in milli-degree 24937 Celsius, which means that the temperature is 24.937 ^{o}C. The whole procedure is shown in Figure 16.



```
pi@carpcserver ~ $ sudo modprobe w1-gpio
pi@carpcserver ~ $ sudo modprobe w1-therm
pi@carpcserver ~ $ ls -l /sys/bus/w1/devices/
total 0
lrwxrwxrwx 1 root root 0 Apr 19 18:42 28-000006155dc9 -> ../../../devices/w1_bus
_master1/28-000006155dc9
lrwxrwxrwx 1 root root 0 Apr 19 18:42 w1_bus_master1 -> ../../../devices/w1_bus_
master1
pi@carpcserver ~ $ cat /sys/bus/w1/devices/28-000006155dc9/w1_slave
8f 01 4b 46 7f ff 01 10 14 : crc=14 YES
8f 01 4b 46 7f ff 01 10 14 t=24937
pi@carpcserver ~ $
```

Figure 16. Testing the DS18B20 sensor with terminal

6.2.2 Defa PlugIn Relay Test

The next step was to test the Defa PlugIn relay with the Linux terminal and WiringPi library. At first GIT needed to be installed with the following command:

```
sudo apt-get install git-core
```

Then with the following command to obtain WiringPi using GIT:

git clone git://git.drogon.net/WiringPi

Now changing directory:

cd WiringPi

Command to install the WiringPi:

./build

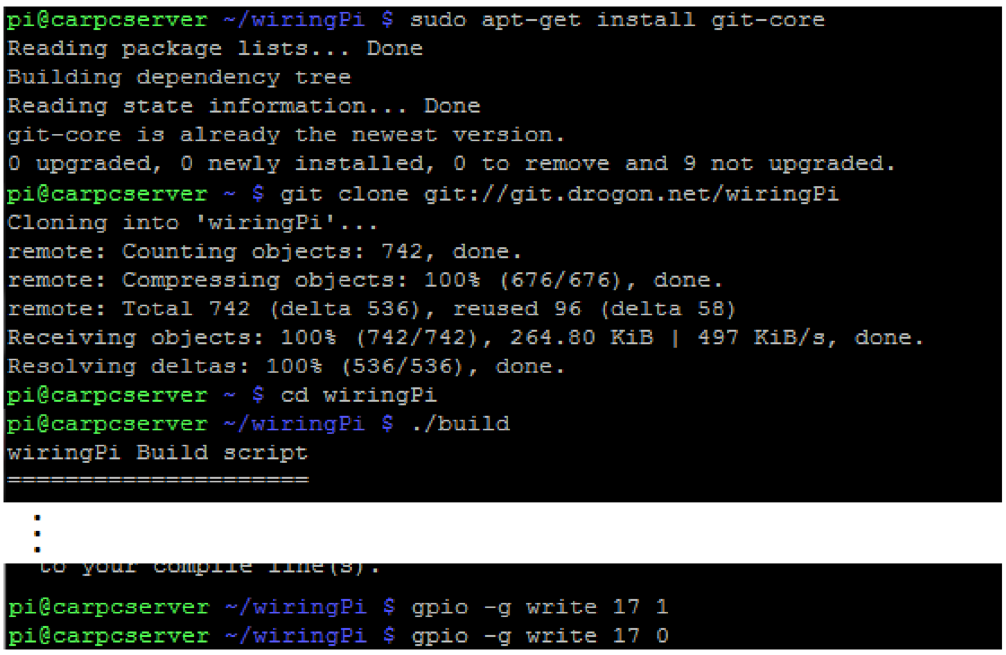
Now the WiringPi library was installed. The next step was to give the following command to set the GPIO-17 pin on active state (1) which will set relay on:

gpio –g write 17 1

The Defa PlugIn relay worked as expected and also the indicate LED from the circuit went on. The following command set the GPIO-17 into deactive state (0) which will set the relay off:

gpio –g write 17 0

The whole procedure is shown in Figure 17.



```
pi@carpcserver ~/wiringPi $ sudo apt-get install git-core
Reading package lists... Done
Building dependency tree
Reading state information... Done
git-core is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
pi@carpcserver ~ $ git clone git://git.drogon.net/wiringPi
Cloning into 'wiringPi'...
remote: Counting objects: 742, done.
remote: Compressing objects: 100% (676/676), done.
remote: Total 742 (delta 536), reused 96 (delta 58)
Receiving objects: 100% (742/742), 264.80 KiB | 497 KiB/s, done.
Resolving deltas: 100% (536/536), done.
pi@carpcserver ~ $ cd wiringPi
pi@carpcserver ~/wiringPi $ ./build
wiringPi Build script
=====================
  :
  to your compile line(s).

pi@carpcserver ~/wiringPi $ gpio -g write 17 1
pi@carpcserver ~/wiringPi $ gpio -g write 17 0
```

Figure 17. Testing the Defa PlugIn relay with terminal

Now when all of the functions were working, it was time to make the actual circuit board.

6.3 PCB Layout

The circuit board layout was done with the Logic Layout software. The circuit board will be attached on top of the Raspberry Pi, so the circuit board has to be of a certain size so it will fit and most importantly the components of the circuit board must not touch the Raspberry Pi's components, otherwise the short circuit might happen and in the worst case both devices will break down. When placing the components on the layout, it should be done in as rationally as possible.

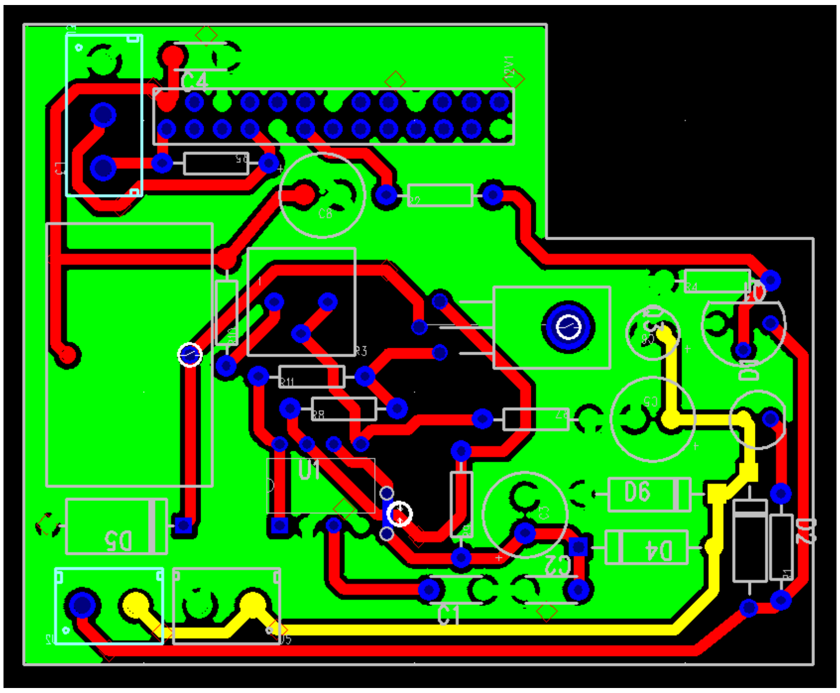


Figure 18. PCB layout

After the circuit layout was ready then the layout needed to be printed on the overhead projector film for the etching. It was also a good idea to print the layout only with the components, because this was useful when doing the drilling and soldering.

6.4 PCB UV Exposure and Etching

For the UV exposure and etching, a new PCB board, UV exposure unit and etching machine were required. Before the etching, first the film with the layout and the photosensitive PCB were put inside the UV exposure unit and then wait that the photo process for the PCB is done.

When the UV exposure was done then it was necessary to get rid of unwanted copper. This was done by putting the UV exposed PCB inside the acid tank of the etching machine. The first circuit board failed, because the PCB was too long time exposed to the UV light and this was fixed by keeping the second PCB inside the UV exposure unit for a shorter time.

6.5 Drilling and Soldering

Now holes needed to be drilled for the electronic components. Drilling was done by using a bench drill. When drilling the holes, it is better not to use too big drill bits so that the soldering will be easier.

After the drilling, it was critical to cut the PCB into the right size, because if the cutting is done after the soldering, the components might get damaged. The soldering was done by using a soldering iron and tin wire.

After soldering, the PCB was ready for testing and the test was done with the same method as with the breadboard by using a terminal earlier. The PCB was working as it was supposed to. Figure 19 shows the PCB attached to the Raspberry Pi.

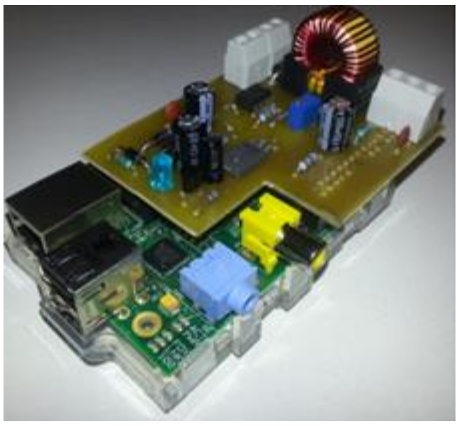


Figure 19. The Raspberry Pi and the power and measurement unit

Figure 20 shows all the connectors of the power and measurement unit. The power input (12V) is to power the Raspberry Pi from the car 12V battery and also power the Defa PlugIn relay.

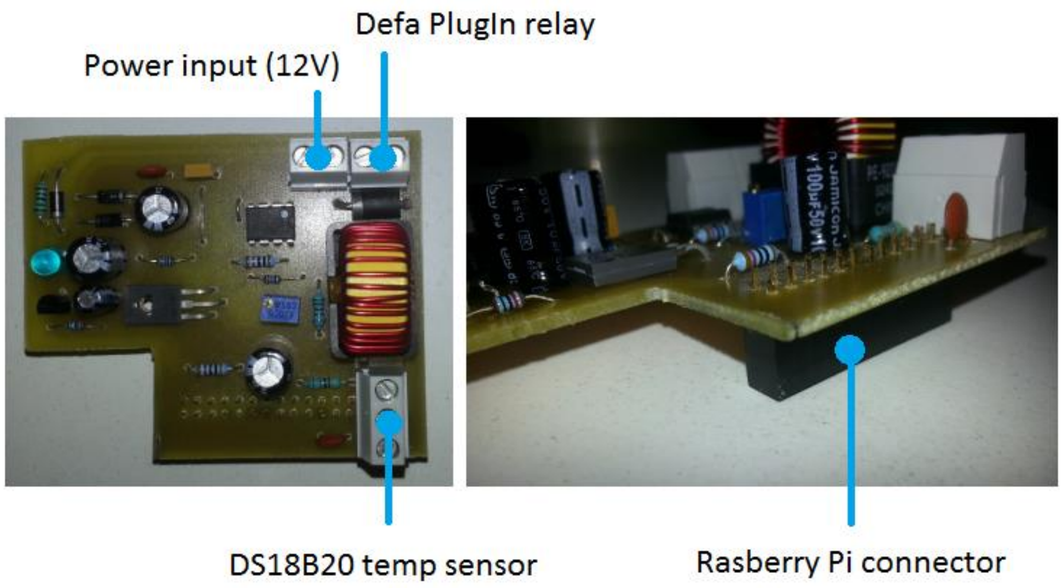


Figure 20. The PCB’s connectors

7 POWER CONSUMPTION TEST

Now that the Raspberry Pi had been tested with the power measurement unit, the power consumption was tested next. The system requirements analysis was made when this project started, the power consumption was around 770mA with the Raspberry Pi and Defa PlugIn relay and the basic car battery is around 50Ah so it was calculated that in theory the car battery would go flat in 2 days and 17 hours if the Raspberry Pi and the Defa PlugIn relay are all the time on. Now when the power consumption test was done with all the necessary components which included the Raspberry Pi, the power measurement unit and Defa PlugIn relay, the average consumption was 300mA. When the relay was in off mode, the consumption was around 240mA. The Raspberry Pi's power consumption is on normal use 700mA, so the power consumption of the system is more than half less. This is, because this system will not have to do a lot of tasks and it also does not need the Ethernet and HDMI ports to operate which are consuming more power.

When calculating again how long the average 50Ah car battery would last in this system (50Ah / 0.3A = 166,7h), in theory it would last almost 7 days. Also the relay will not be all the time on, so the battery will last even longer.

The test was done with 12V 62Ah car battery. The Raspberry Pi and relay stayed on around 6 days in a row. The used car battery was already quite old, so it is not so efficient anymore as the new ones are. If this system is in normal everyday use, this would mean that it is not necessary to fit the car with a maintenance charger which would increase the cost of this system a lot.

8 MESSAGE SEQUENCE

This chapter explains how the message sequence works between the user interface and the Raspberry Pi. The structure of the message sequence is shown in Figure 21.

The message sequence was made in PHP, JavaScript, Bash script and C. The user interface was made in PHP (index.php) which contains also JavaScript and it is on the web server. It can be accessed by using a web browser. The user interface has a button to turn the heating on/off and also it displays the temperature from the car.

The user presses the button to set the heating on/off from the user interface on the web page. It will make an AJAX request which reads the state (ON or OFF) of the button just pressed and sends it to Bash script using CGI. The Bash script (relay.sh) calls C-program (client.c) and the client.c will write ON or OFF inside the named pipe. The named pipe is created by another C-program the server.c and this program is all the time checking inside the named pipe and if it contains "ON" or "OFF", it will then set the GPIO17 state to "1" or "0" to control the relay. To make things more automated, when the Raspberry Pi starts, the server program needs to start also so the user can control the heating immediately.

Every time the webpage is refreshed, the index.php is executing read_GPIO17.c program which returns the state of the GPIO17 by giving the update status of the relay button on the webpage. Without checking the state of the GPIO17 it will always show the relay button in the "OFF" mode after refreshing the page.

To get the temperature displayed on the webpage, the DS18B20 returns 9 bytes of data which is placed into the w1_slave file. The index.php reads this file and displays the temperature value on the webpage. The webpage is refreshed every 15 min to give updated temperature value. /7/

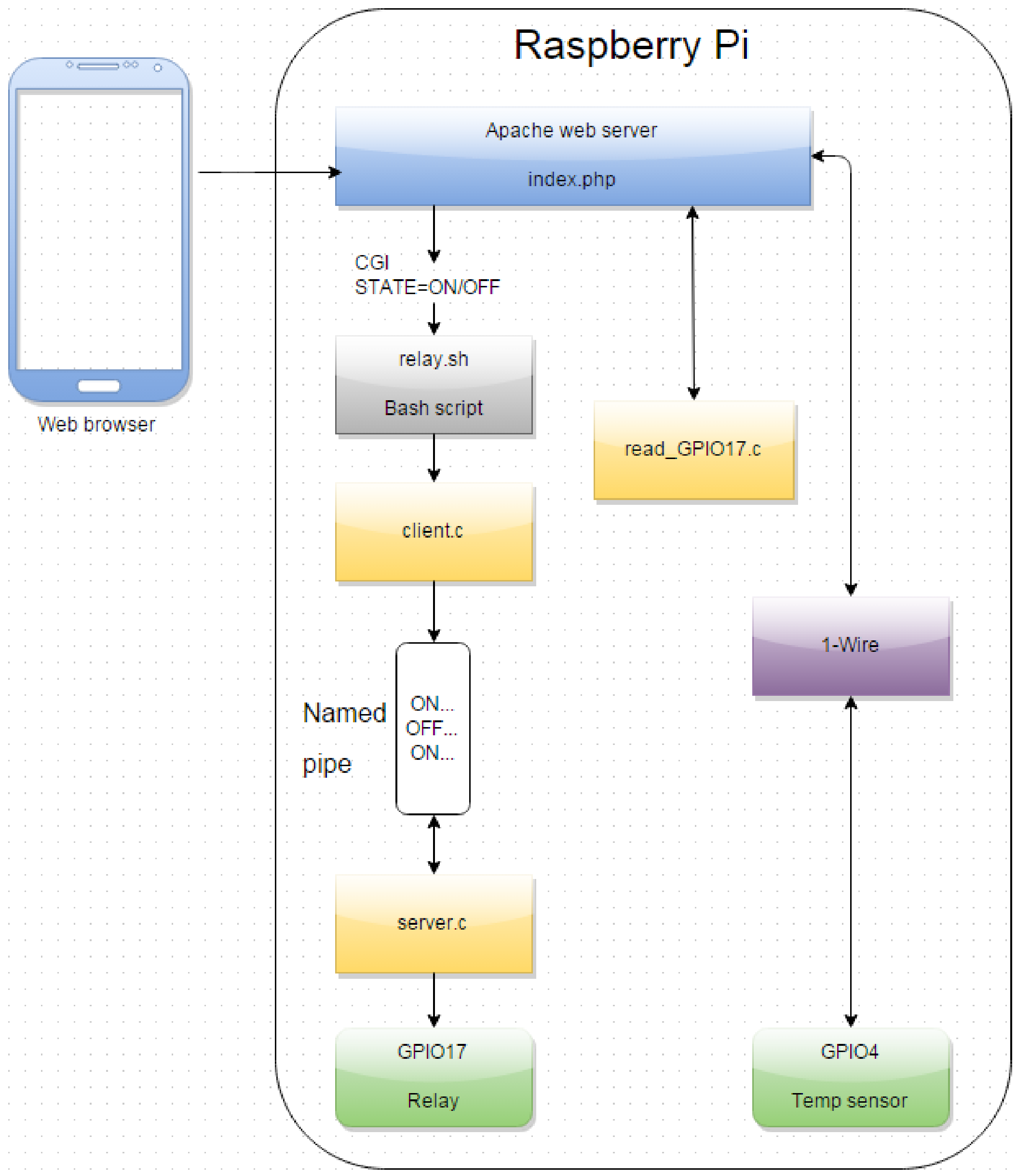
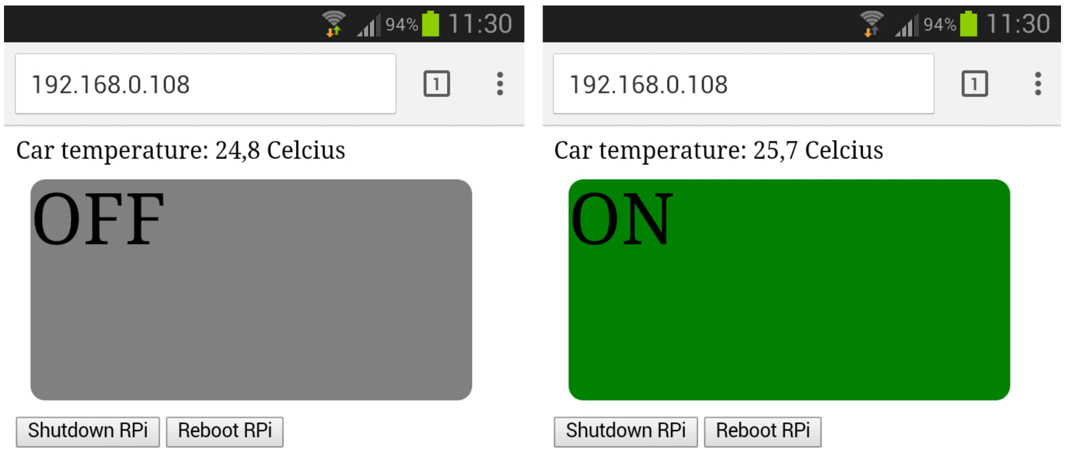


Figure 21. Structure of the message sequence

9 THE CODE

This chapter shows the main commands for the message sequence. The user interface (Figure 22) shows the temperate from the car and the button to set the relay on. The shutdown and reboot buttons were added there to shut down or reboot Pi safely. /12/



192.168.0.108

Car temperature: 24,8 Celcius

192.168.0.108

Car temperature: 25,7 Celcius

Figure 22. The user interface

9.1 Relay control

The relay control was made in PHP, JavaScript, CGI, Bash script and C. When user presses the button to set heating on the user interface, the AJAX request is made:

```
$.ajax({ type: "GET", url: "/cgi-bin/relay_control.sh", data: "state="+state});
```

It will call the relay.sh file and post value “ON” or “OFF” value. The Bash script “cleans” the posted message for the client (client.c) and posts it to the client with the command:

```
STATE=`echo "$QUERY_STRING" | sed -n 's/^.*state=\([^&]*\).*$/\1/p' | sed "s/%20/ /g"`./client $STATE
```

The client program writes into the named pipe “ON” or “OFF”:

```
    if((fp = fopen(FIFO_FILE, "w")) == NULL) {
        perror("fopen");
        exit(1);
     }
fputs(argv[1], fp);
fclose(fp);
```

The server program opens the named pipe for reading and reads the “ON” or “OFF” value into readbuf:

```
      while(1){
            fp = fopen(FIFO_FILE, "r");
            fgets(readbuf, 8, fp);
```

If the readbuf contains the “ON” value, it will set the GPIO17 on active state “1”, which will set the heaters on:

```
            if(strcmp(readbuf, "ON")==0){
                  gpio_file=fopen("/sys/class/gpio/gpio17/value", "w");
                  if(gpio_file!=NULL){
                        fprintf(gpio_file, "1"); // GPIO17 1
                  }
                  fclose(gpio_file);
            }
```

If the readbuf contains the “OFF” value it will set the GPIO17 on “0” state, which will set the heaters OFF:

```
else if(strcmp(readbuf, "OFF")==0){
    gpio_file=fopen("/sys/class/gpio/gpio17/value", "w");
    if(gpio_file!=NULL){
        fprintf(gpio_file, "0"); // GPIO17 0
    }
    fclose(gpio_file);
}
```

There was a problem with the relay control button, because when the relay button was in the “ON” mode and when the user refreshed the webpage, the button went back to the “OFF” mode. It did not set the relay to “OFF”, but the problem was that the user did not know if the heater was on or not. This was fixed by creating a C-program which is reading the GPIO17 state. Every time the webpage is refreshed the index.php executes read_GPIO17.c program and the read_GPIO17.c returns the state of the GPIO17 with following command:

```
exec('sudo ./read_GPIO17', $retArr, $retVal);
```

The read_GPIO17.c program opens the value file from the /sys/class/gpio/gpio17 directory where the GPIO17 state is located and returns the state:

```
gpio_file=fopen("/sys/class/gpio/gpio17/value", "r");
if(gpio_file!=NULL){
    fprintf(gpio_file, "17");
}
```

```
  ch = fgetc(gpio_file);

  printf("%c", ch);

  fclose(gpio_file);

  return ch-'0';
```

When the GPIO17 state is returned, the index.php checks if the returned value is ‘1’ it will set the relay button in “ON” mode:

```
if($retArr[1] == 1 )

  echo '<div class="button" id="switch" style="background-color: green"

  onclick="relay_button(\'OFF\')"> <FONT SIZE=9>ON</FONT></div>';
```

If the GPIO17 state is not “1”, it will set the relay button “OFF”.

```
else

  echo '<div class="button" id="switch" style="background-color: gray"

  onclick="relay_button(\'ON\')"> <FONT SIZE=9>OFF</FONT></div>';
```

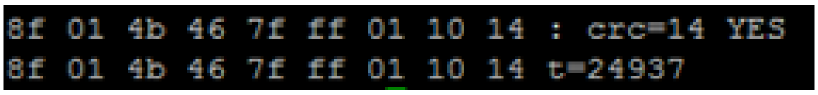
9.2 Displaying Temperature

The temperature reading and displaying was done in the PHP language and the code is located in index.php. First the file needed to be found to read with command:

```
$file = '/sys/devices/w1_bus_master1/28-000006155dc9/w1_slave';
```

All DS18B20 thermometers have their own unique addresses. The used DS18B20’s address is 28-000006155dc9. Reading the file line by line:

```
$lines = file($file);
```



```
8f 01 4b 46 7f ff 01 10 14 : crc=14 YES
8f 01 4b 46 7f ff 01 10 14 t=24937
```

Figure 23. W1_slave file's data

The temperature value is on second line (Figure 23) of the file and reading from the second line is done with command:

```
$temp = explode('=', $lines[1]);
```

Now the following line gives it a better format that will be easier to read, because the temperature is written into the file in milli-degrees.

```
$temp = number_format($temp[1] / 1000, 1, ',', '');
```

Then display the temperature value on the user interface:

```
echo $temp . " Celcius";
```

To give the updated temperature from the car the page refreshes itself in every 15 min (900 s). The refresh commands are located in index.php:

```
$url1=$_SERVER['REQUEST_URI'];

header("Refresh: 900; URL=$url1");
```

9.3 Reboot and Shutdown

The webpage also have reboot and shutdown buttons. These buttons were added there to shut down or reboot the Raspberry Pi safely.

```
"<form action='' method='post'>

<input type='submit' name='shutdown' value='Shutdown RPi' />

<input type='submit' name='reboot' value='Reboot RPi' />

</form>";
```

```
if (isset($_POST['reboot'])) {

shell_exec("sudo reboot");

}

if (isset($_POST['shutdown'])) {

exec("sudo shutdown -h now");

}
```

10 SYSTEM STRUCTURE AND IMPEMENTATION

This chapter explains the system structure and implementation. Figure 24 shows how the system is fitted into the car. The power comes from the power grid (230V) which powers the motor heater and the interior heater. The grid cable is attached to the 12V Defa PlugIn relay’s input and the engine and the interior heater cables are attached to the relay’s outputs. The control unit represents the Raspberry Pi and the power and measurement unit. For the safety, it is best to put the control unit inside the car so it will be kept as dry as possible. The relay is controlled on/off by the control unit. When the relay is in the active state, the power from the grid is let trough to by setting the heating on. The control unit gets power from the car battery and it is essential to add a fuse to protect the control unit from overload. 1A fuse is enough for this. The waterproof DS18B20 temperature sensor is attached to the control unit. The switch is there to turn control unit on.

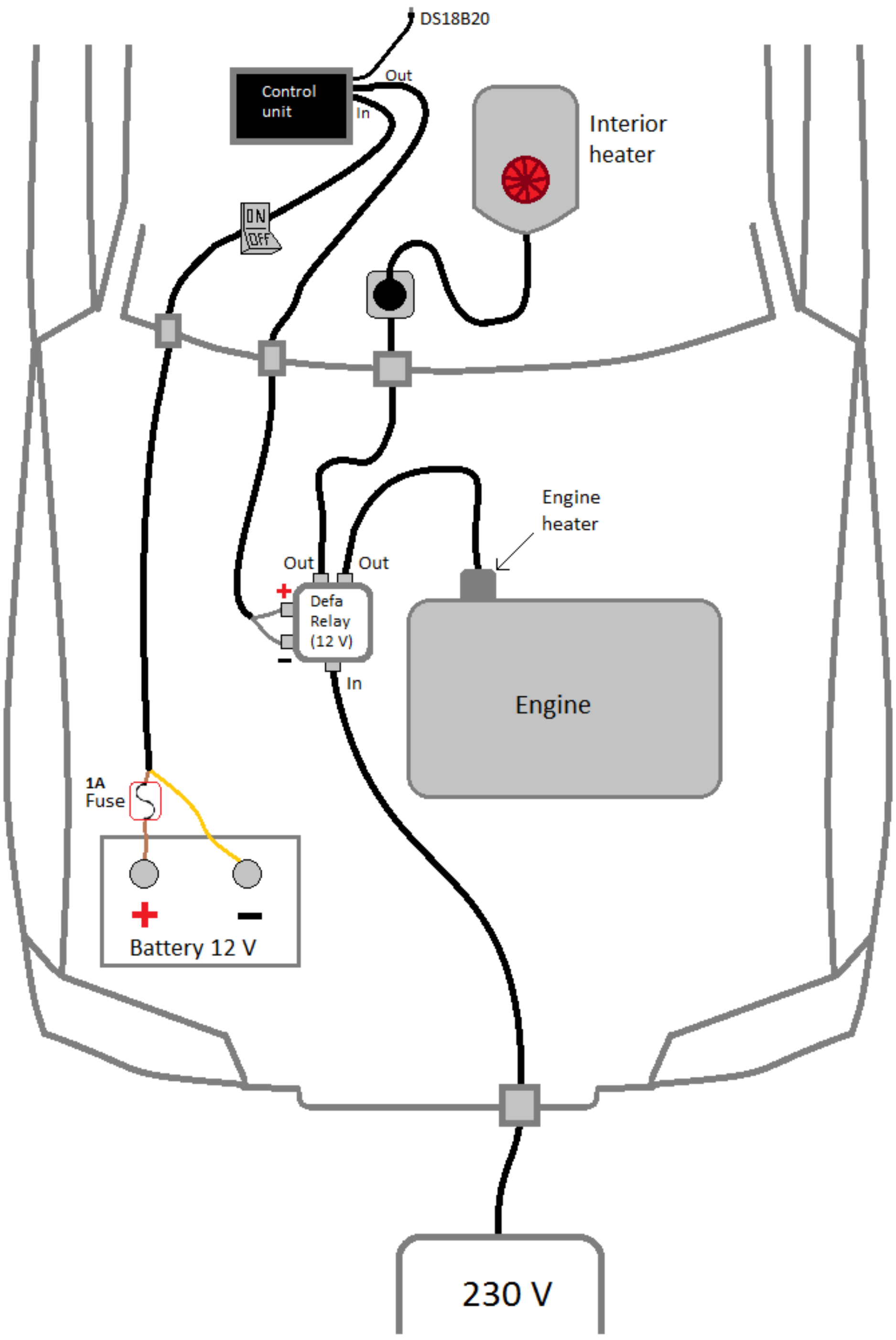


Figure 24. Device diagram

11 TESTING AND VALIDATION

During the development of this system, the testing was made several times in the Wi-Fi network of the laboratory by using the Raspberry Pi + power and measurement unit, Defa PlugIn relay, web browser and a 12V car battery. The tests were overall successful, but the testing described in this chapter was done in the car environment. Before doing the actual testing some preparations were done for the Raspberry Pi and the Power and measurement unit.

11.1 Preparations

Before testing the system in the car environment it was better to protect the Raspberry Pi and the power and measurement unit from dust and moisture by fitting them into a protection case and also to use more appropriate wires to connect the devices with each other. The case that was used to protect the Raspberry Pi and the power and measurement unit is made by TEKO and the case is built to protect electronic devices. The wires used to connect the devices with each other were particularly made for car environment. The protections could be done in a better way, but as this is on a prototype level this was enough. Figure 25 shows the protected Raspberry Pi + Power and measurement unit with the more appropriate wires.

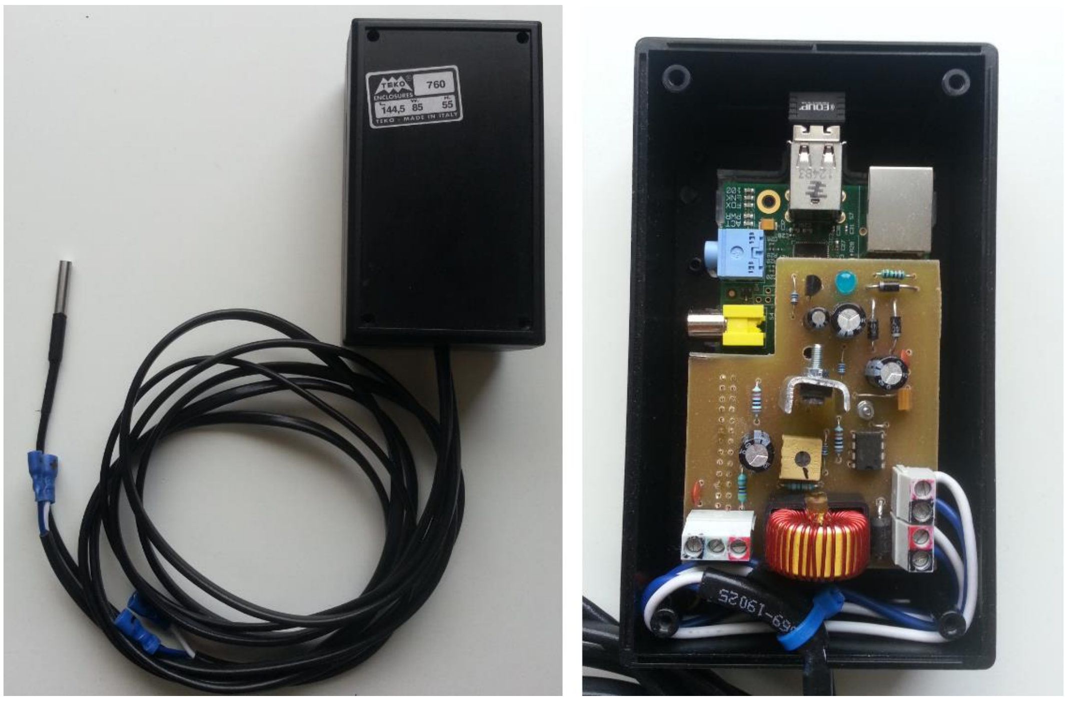


Figure 25. Control unit

11.2 Test Equipment

The following equipment was used in the testing:

Raspberry Pi model B revision 2.0

Power and measurement unit

Defa PlugIn relay (12V)

Car battery (12V)

Car with Defa engine and interior heater system

Android smartphone

Power switch

Fuse case + 1A fuse

11.3 Test

First, the devices were connected into the car. When the devices were connected the test was done by starting the Raspberry Pi by pressing the power switch on. The Raspberry Pi automatically started the web server and the server.c program and after the server was up, it was time to test the heaters. The IP address of the Raspberry Pi had to be typed from the smartphones web browser and both devices have to use the same Wi-Fi network. The webpage displayed the temperature and when the heating button was pressed, the engine and the interior heaters started. After the test, the Raspberry Pi was turned off safely from the webpage.

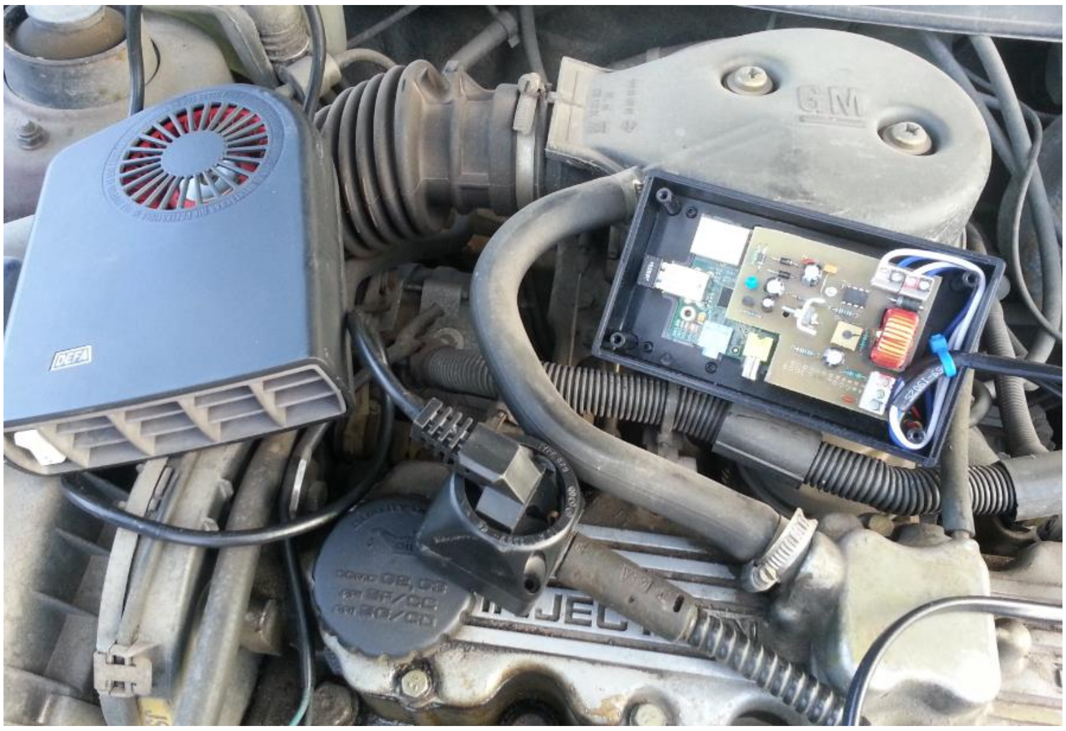


Figure 26. Testing with the car

11.4 Analysis

The heating was remotely set on from the user interface and the interface displayed the temperature without problems. The functions were tested several times without any issues except that one when the Raspberry Pi did not find the network that was used on this test. The test was overall successful.

12 CONCLUSION AND FUTURE WORK

All goals were achieved and all functions were working as expected. As this system is on the prototype level, there are many things to improve such as adding a timer function to the heating.

The security of this system must be improved, because the webpage does not have a password and anybody who has access to the same Wi-Fi network can control the heating and also the code part could have been written in a more efficient way, such as making the temperature reading in real-time.

The system also could be more user friendly, because now when the user wants to make this system up and running, first he needs to press the button to start the Raspberry Pi and then put the power cable to the heaters. Also when the user wants to shut down the Raspberry Pi, the user needs to open the webpage and press Raspberry Pi shutdown button. This could be done by only putting the power cable for the heaters which would make a wake up signal to the Raspberry Pi to start up and when the user wants to turn the Raspberry Pi off, the user just takes the power cable from the car heaters and the Raspberry Pi would notice it and then shut itself down. If the Raspberry Pi is turned off without proper shutdown, the SD card can eventually be damaged.

Also the user might want to use the heaters without this remote system, so this system should have a switch to bypass the relay.

When there is enough cold weather to start the heating, the control unit could send a notification email to the user.

Considering the tight schedule of making this project all worked very well. Nevertheless, there are a lot of things what needs more developing, but that is for the future work.

REFERENCES

/1/ Apache homepage. Accessed 15.4.2015. http://httpd.apache.org/

/2/ Byron Gottfried. 1996. Programming with C. McGraw-Hill.

/3/ Defa homepage. Accessed 20.3.2015. http://www.defa.com/

/4/ Denkovi homepage. Accessed 2015.4.22. http://denkovi.com

/5/ DS18B20 datasheet

/6/ EDUP. Accessed 21.5.2015. http://www.edupdriver.com/edup-ep-n8531-mini-150mbps-802-11ngb-usb-wireless-nano-adapter/

/7/ KMtronic. Accessed 19.4.2015. http://kmtronic.com/raspberry-pi-ds18b20-connect-to-gpio.html

/8/ McFarland, David Sawyer. 2011. JavaScript & jQuery: The Missing Manual, 2nd Edition. O'Reilly Media.

/9/ NCSA HTTPd Development Team. Accessed 19.4.2015. http://www.diam.unige.it/informatica/documentazione/httpd_docs/cgi/

/10/ Newham, Cameron. 2005. Learning the bash Shell, 3rd Edition. O'Reilly Media.

/11/ Raspberry Pi foundation’s homepage. Accessed 28.3.2015. http://www.raspberrypi.org/

/12/ Santoro, Rosario. Accessed 15.3.2015. http://rosariosantoro1.altervista.org/wp2013/remote-controlling-room-lights-through-raspberry-pi/?doing_wp_cron=1427967473.9817600250244140625000

/13/ Welling, Luke & Thompson, Laura. 2003. PHP and MySQL Web Development, Second edition. Sams.

/14/ Zwetsloot, Rob. Accessed 21.5.2015. http://www.linuxuser.co.uk/reviews/top-4-raspberry-pi-os

APPENDICES

12.1 index.php

```
<html>

<head>

<title>Remote heater</title>

</head>



<?php

/* Read and dislay DS18B20 temperature */

$file = '/sys/devices/w1_bus_master1/28-000006155dc9/w1_slave';

$lines = file($file);

$temp = explode('=', $lines[1]);

print "Car interior temperature: ";

$temp = number_format($temp[1] / 1000, 1, ',', '');

echo $temp . " Celcius";



/* Read the relay state */

exec('sudo ./read_GPIO17', $retArr, $retVal);

echo '<body>';

if($retArr[1] == 1 )
```

```
  echo '<div class="button" id="switch" style="background-color: green"
  onclick="relay_button(\'OFF\')"> <FONT SIZE=9>ON</FONT></div>';
else
  echo '<div class="button" id="switch" style="background-color: gray"
  onclick="relay_button(\'ON\')"> <FONT SIZE=9>OFF</FONT></div>';

/* Shutdown and reboot Buttons */
echo
"<form action='' method='post'>
<input type='submit' name='shutdown' value='Shutdown RPi' />
<input type='submit' name='reboot' value='Reboot RPi' />
</form>";

if (isset($_POST['reboot'])) {
shell_exec("sudo reboot");
}

if (isset($_POST['shutdown'])) {
exec("sudo shutdown -h now");
}
echo '</body>';
echo '</html>';


/* Auto Refresh webpage every 900 second (15 min) */

$url1=$_SERVER['REQUEST_URI'];

header("Refresh: 900; URL=$url1");

print '<br>';

?>


<style>

div.button{

 height: 150px;

 width: 300px;

 margin: 10px;

 border-radius: 10px; }

</style>

<script src="jquery-1.11.2.js"></script>


<script>

function relay_button(state){

 if(state=="ON"){
```

```
 $("#switch").html("<FONT SIZE=9>ON</FONT>");

 $("#switch").css("background-color", "green");

 $("#switch").attr("onclick", "relay_button('OFF')");

 }

 else{

 $("#switch").html("<FONT SIZE=9>OFF</FONT>");

 $("#switch").css("background-color", "gray");

 $("#switch").attr("onclick", "relay_button('ON')");

 }

<!-- AJAX request -->

 $.ajax({ type: "GET", url: "/cgi-bin/relay_control.sh", data: "state="+state});

}

</script>
```

12.2 relay.sh

```
#!/bin/bash

echo "Content-type: text/html"

echo '<html>'

echo '<head>'

echo '<title>Relay manager</title>'

echo '</head>'

echo '<body>'

STATE=`echo "$QUERY_STRING" | sed -n 's/^.*state=\([^&]*\).*$/\1/p' | sed
"s/%20/ /g"`

./client $STATE

echo '</body>'

echo '</html>'
```

12.3 client.c

```
#include <stdio.h>

#include <stdlib.h>

#define FIFO_FILE       "/tmp/RELAY"



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

    FILE *fp;

    if ( argc != 2 ) {

        printf("USAGE: client [string]\n");

        exit(1);

     }

    if((fp = fopen(FIFO_FILE, "w")) == NULL) {

        perror("fopen");

        exit(1);

     }

fputs(argv[1], fp);

fclose(fp);

return(0);

}
```

12.4 server.c

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/stat.h>

#include <unistd.h>

#include <linux/stat.h>

#define FIFO_FILE "/tmp/RELAY"



int main(void)

{

    FILE *fp;

    char readbuf[8];

    printf("Run relay control C... ");

    FILE* gpio_file = NULL;

    gpio_file=fopen("/sys/class/gpio/export", "w");

    if(gpio_file!=NULL){

        fprintf(gpio_file, "17");

    }

    fclose(gpio_file);

    gpio_file=fopen("/sys/class/gpio/gpio17/direction", "w");
```

```
if(gpio_file!=NULL){
        fprintf(gpio_file, "out");
}
fclose(gpio_file);
printf("OK\n");
     umask(0);

 mknod(FIFO_FILE, S_IFIFO|0666, 0);
while(1){
        fp = fopen(FIFO_FILE, "r");
        fgets(readbuf, 8, fp);
        if(strcmp(readbuf, "ON")==0){
                gpio_file=fopen("/sys/class/gpio/gpio17/value", "w");
                if(gpio_file!=NULL){
                        fprintf(gpio_file, "1"); // GPIO17 1
                }
                fclose(gpio_file);
        }
        else if(strcmp(readbuf, "OFF")==0){
                gpio_file=fopen("/sys/class/gpio/gpio17/value", "w");
                if(gpio_file!=NULL){
                        fprintf(gpio_file, "0"); // GPIO17 0

                    }

                     fclose(gpio_file);

                }



                else{

                     printf("Unknown command\n");

                }

                fclose(fp);

            }

            return(0);

        }
```

12.5 read_GPIO17.c

```
#include <stdio.h>

/* read GPIO 17 state */

int main()

{

  unsigned char ch;

      printf("GPIO init...\n");

      FILE* gpio_file = NULL;

      gpio_file=fopen("/sys/class/gpio/gpio17/value", "r");

      if(gpio_file!=NULL){

              fprintf(gpio_file, "17");

      }

  ch = fgetc(gpio_file);

  printf("%c", ch);

  fclose(gpio_file);

  return ch-'0';

}
```