



Project Report

Smart Control

EECE440 - Embedded System Design w/lab

Fall 2013

Group Members:

Joseph Bejjani Sharbel Dahlan
1002018289 1004018456

Course instructor: **Dr. Adnan El Nasan**
Lab instructor: **Eng. Milad Abbasi**
Due date: **December 12, 2013**

Contents

List of Figures	ii
List of Tables	iii
1 Objectives	1
2 Introduction	3
2.1 About Raspberry Pi	3
2.1.1 About Broadcom BCM2835	5
2.1.2 About Raspbian OS (“Raspberrian”)	6
2.2 About WebIOPi	7
3 Hardware Setup	8
3.1 Components Used	8
3.2 Relay Board vs. Relay	9
3.3 Schematics	10
4 Code	13
4.1 Configuring the WebIOPi	13
4.2 The Web Interface	13
4.3 Implementation (GPIO Control)	16
5 Analysis	18
5.1 The Web Interface	18
5.2 Implementation (GPIO Control)	19
5.2.1 Blink	19
6 Outputs & Results	20
7 Problems Faced	23
8 Future Improvements	24
9 Conclusion	25
A WebIOPi Configuration	26
A.1	26

List of Figures

2.1	Raspberry Pi model B components	3
2.2	Raspberry Pi Model A and B specifications	4
3.1	Normal relay	9
3.2	4 Channel Relay Board	9
3.3	4 channel relay board, schematic	10
3.4	Hardware setup	11
3.5	inout.py script schematic	12
6.1	Final Circuit	20
6.2	The whole system, when all appliances are off	20
6.3	Christmas lights are on, when the Buttons are pressed on the Web page	21
6.4	All LEDs are on, when all buttons are pressed, setting GPIOs to True	22

List of Tables

3.1 Components used for the project	8
---	---

Chapter 1

Objectives

The purpose of this project is to integrate all the topics and concepts, whether learned in class or practiced in the lab, in order to design an embedded system that serves a certain useful purpose. In the past four months, a broad range of topics were studied, including software, hardware, and a combination of both. Concepts such as input/output, interrupts, timers, and many more were all once covered separately, and will now be collectively used in the project. Moreover, the different practices and experiences that were done in the labs will be done again in this project, including:

- Programming the software.
- Creating/configuring the hardware.
- Learning about the microcontroller used and its functionality by going through manuals and datasheets.
- Combining the hardware and the software to make the designed system work.

Particularly, this project targets the following objectives:

- Learn about the Raspberry Pi microcontroller
- Learn about the mechanism of using the web as part of the input to the microcontroller (through the network card).
- Configure and customize the WebIOPi application to serve for the purpose of the project.
- Write different codes: one that takes care of the web interface and one that implements the functionality of the smart control system.

Since the driving force of designing and creating systems is the continuous existence of problems, in addition to the need for improvement in functionality or performance of existing systems, this project was done for an indifferent purpose. What has initiated the idea of the project, is the need to have an easy, effective, and instantaneous control over the switches of a household appliances. The solution was therefore to have a smart control system which to control the GPIOs (General Purpose Input/Output) of the microcontroller through the web. Those GPIOs can then be connected to the switches of the appliances.

Chapter 2

Introduction

Apart from connecting to the outside world, the web acts as host of a great deal of applications, many of which provide easy control over systems. With the rapid increase in the use of the internet, the feasible option that came to mind when building a smart control system was to make the smart-controlling tool possible through the web. Based on this idea, the choice of the microcontroller fell on the *Raspberry Pi*, since, among its alternatives, it has the feature of connecting to the web through a network card. With the Raspberry Pi, which can run different linux systems, a few applications were previously made, including the WebIOPi on the *Rasperrian* operating system.

2.1 About Raspberry Pi

For this project we used the Raspberry pi, the reason we chose the raspberry pi over other devices will be made clear as we go through the introduction of the device.

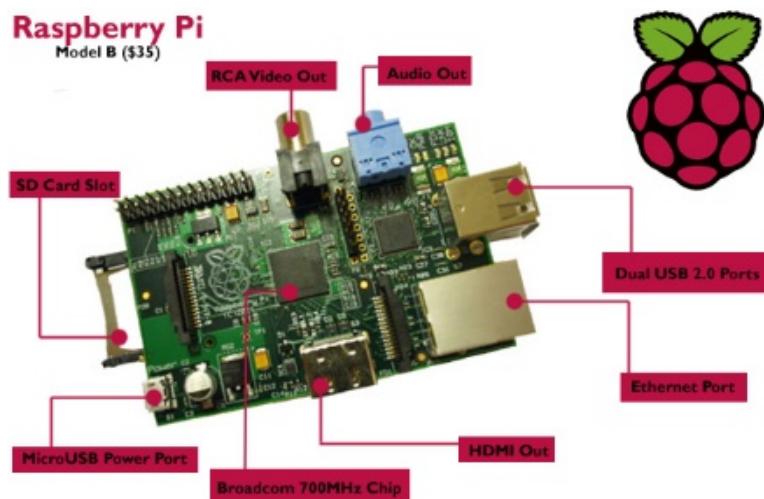


FIGURE 2.1: Raspberry Pi model B components

At the beginning we had the choice between two models of the raspberry pi, the model A or B. the following table shows the differences between the two models.

	Model A	Model B
Price	US\$ 25	US\$ 35
SoC	Broadcom BCM2835	
CPU:	700 MHz ARM1176JZF-S core (ARM11 family, ARMv6 instruction set)	
GPU:	Broadcom VideoCore IV @ 250 MHz OpenGL ES 2.0	
Memory (SDRAM):	256 MB (shared with GPU)	512 MB (shared with GPU)
USB 2.0 ports:	1 (direct from BCM2835 chip)	2 (via the built in integrated 3-port USB hub)
Video input:	A CSI input connector allows for the connection of a RPF designed camera module ^[80]	
Video outputs:	Composite RCA , HDMI	
Audio outputs:	3.5 mm jack, HDMI, and, as of revision 2 boards, I ² S audio	
Onboard storage:	SD / MMC /	
Onboard network:	None	10/100 Ethernet (8P8C)
Low-level peripherals:	8 × GPIO, UART, I ² C bus, SPI bus with two chip selects, I ² S audio +3.3 V, +5 V, ground	
Power ratings:	300 mA (1.5 W)	700 mA (3.5 W)
Power source:	5 volt via Micro USB or GPIO header	
Size:	85.60 mm × 53.98 mm (3.370 in × 2.125 in)	
Weight:	45 g	
Operating systems:	Arch Linux ARM, Debian GNU/Linux, Gentoo, Fedora, FreeBSD, NetBSD, Plan 9, Raspbian OS, RISC OS, Slackware Linux	

FIGURE 2.2: Raspberry Pi Model A and B specifications

As you can see from the specifications: Raspberry Pi Model A and B specifications above, we can see that our choice of using model B was due to one main aspect, the onboard network adapter, however we could have used a Model A device and then used a Wi-Fi dongle to connect to the internet, instead of troubling ourselves we just chose the faster model B raspberry pi that comes with a built in network adapter. The reason we chose raspberry pi over other devices like the Arduino or the lpc2388 that we use in class is simply that it is easier the implement the web interface over the Raspberry compared to any other device.

2.1.1 About Broadcom BCM2835

BCM2835 contains the following peripherals which may safely be accessed by the ARM:

- Timers
- Interrupt controller
- GPIO
- USB
- PCM / I2S
- DMA controller
- I2C master
- I2C / SPI slave
- SPI0, SPI1, SPI2
- PWM
- UART0, UART1

Navigating through the BCM2835 was a very familiar action, and we noticed that most if not all of the peripherals operate in a manner similar to the ones we studied throughout the course. However, we did not have access to each pin in the BCM2835 like we did on the MCB2300 board, so it was almost impossible to connect directly to specific registers. On the other hand, the Raspberry Pi comes with predefined libraries that allow the use of every peripheral in a simple and straight forward method. Due to short time frame and the scope of our project, the peripherals we focused on are:

- Timers
- Interrupt controller
- GPIO
- USB

2.1.2 About Rasbian OS (“Raspberrian”)

Raspbian OS is a free operating system that is optimized for the Raspberry Pi. It holds over 35,000 packages, which come precompiled and are optimized for easy access. Most of the work done, was done through the terminal. We had to first update/download some packages that were not included by default, these packages include:

- RPi.GPIO-0.5.3a
- WebIOPi-0.6.0

Using the terminal we were able to:

- Create directories using `mkdir`
- Create files: `*.html`, `*.py` using `cp`
- Navigate through the files using `cd`
- Download packages: using `wget`
- Editing these files using `nano`
- Run them using python: `sudo python`
- Run/Stop, and check status of the server
 - To start: `sudo etc/init.d/webiopi start`
 - To stop: `sudo etc/init.d/webiopi stop`
 - To check the status [on or off]: `sudo etc/init.d/webiopi status`
- Configure the server properties, we configured the server by opening it in an editor using the following code: `sudo nano /etc/webiopi/config`

2.2 About WebIOPi

WebIOPi is a package that can be installed on the Raspbian that would aid the user in configuring and setting up the server, and web interface. The user can then edit the user interface to implement his specific project. Some important facts to consider about WebIOPi:

- Server written in Python
- Supports more than 30 devices, including GPIO, timers, interrupts...
- Full Python library for the Server and GPIO
- Extensible and highly customizable
- Login/Password protection
- Mobile devices compatible
- JavaScript client library

Chapter 3

Hardware Setup

3.1 Components Used

Table 3.1 contains the components used in this project.

TABLE 3.1: Components used for the project

Component	Amount
Breadboard	× 1
12V DC power supply	× 1
4 channel Relay Board	× 1
Relay	× 4
Switch	× 1
10K resistor	× 1
Male to Female jumper cables	× 4
Male to Male jumper cable	× 4
Raspberry Pi, Model B	× 1
Micro-USB Cable	× 1
Network cable	× 1
Wireless keyboard and mouse	× 1
20 x 20 CM Styrofoam	× 2

Some components will be discussed in detail in the following section.

3.2 Relay Board vs. Relay



FIGURE 3.1: Normal relay

Operates on AC 120V / 0.5, DC 24V / 1.0A. Needs to be manually connect the transistors and other elements in order to drive the current and to prevent the voltage from entering the Raspberry Pi

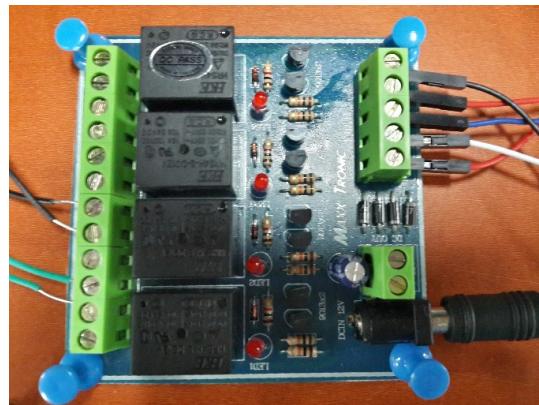


FIGURE 3.2: 4 Channel Relay Board

- Power supply: 9-12 VDC/max. 160 mA.
- 4 relay outputs, each with Normally Open and Normally Closed outputs
- LED indicators to show status of each relay.
- 500 watts maximum output load per channel.

- 2 or more units can be used together to give more inputs and output relays.
- Can be used with Basic Stamps and PIC controllers
- Fully Assembled and tested module

3.3 Schematics

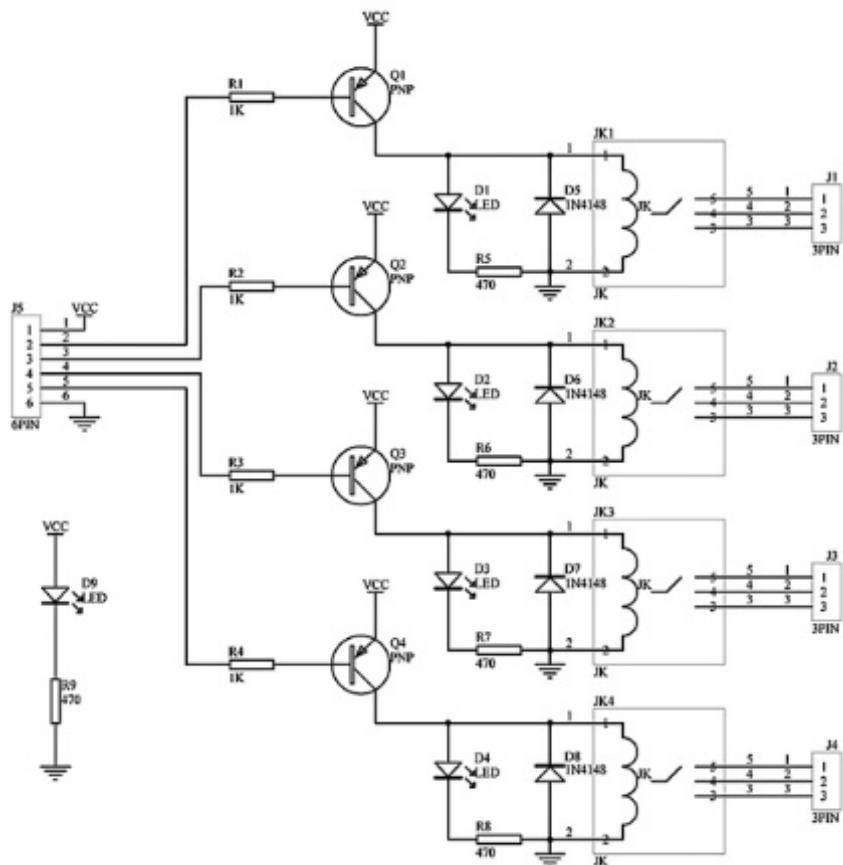


FIGURE 3.3: 4 channel relay board, schematic

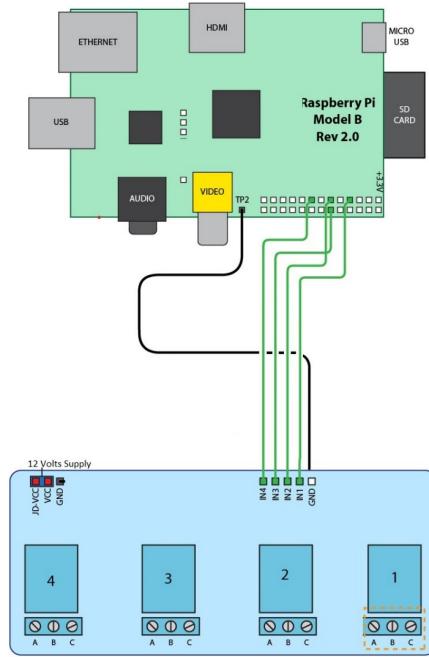


FIGURE 3.4: Hardware setup

During normal operation, when relay is active, its terminals A and B are connected. When de-activated, B and C are connected. Ideally when the GPIO connected to one of the relays is set to one (true) B and C are connected, which in turn connects the appliance to its source voltage, thus turning on the appliance.

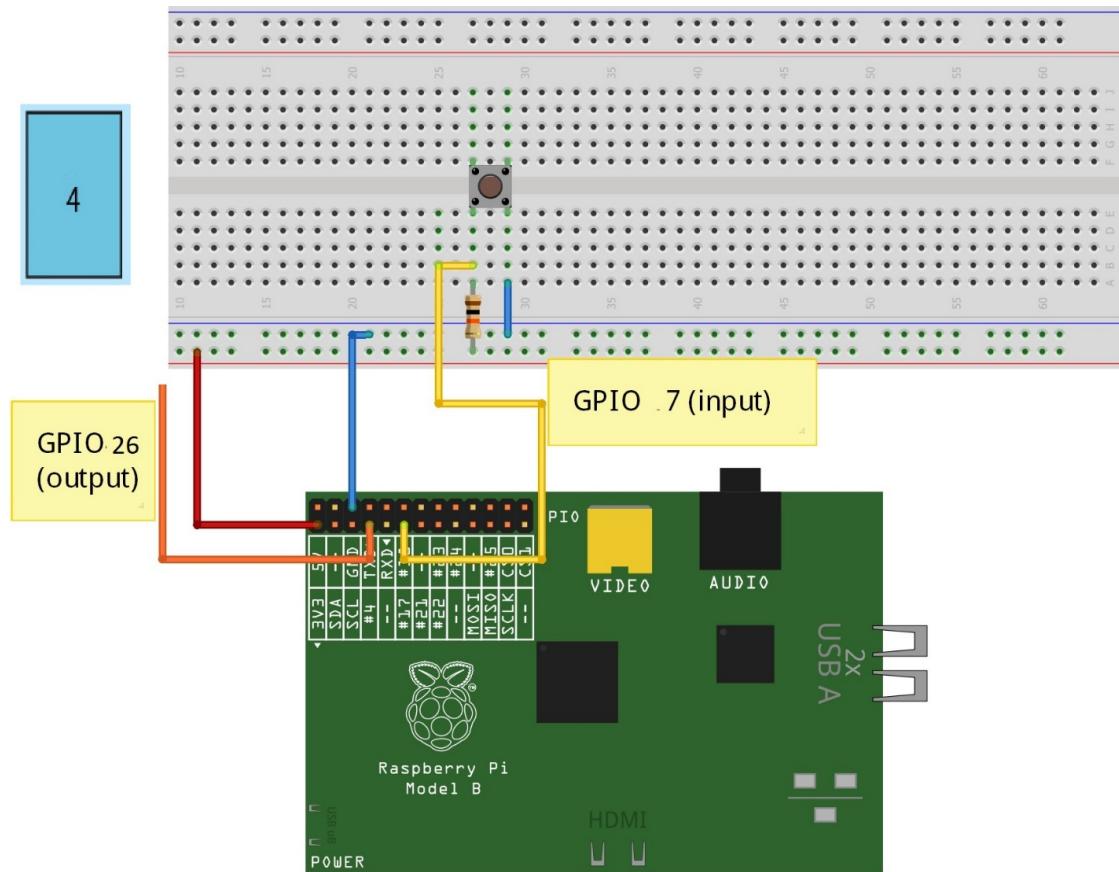


FIGURE 3.5: inout.py script schematic

The previous figure shows the exact connection for the inout.py script, which is discussed in the code and Analysis section.

Chapter 4

Code

Different parts of this project have required different codes to be written in the appropriate language. For the parts that deal with configuration of the WebIOPi server, linux commands were written to the terminal of the Raspberrian system. For the parts that require designing the user interface, which is a web-page in this case, Javascript code was used. However, for the actual code that involves implementing the functionality of the microcontroller or this project (i.e. controlling the GPIOs), python script was written.

4.1 Configuring the WebIOPi

In the process of configuring the WebIOPi, different parameters were set. Parameters include different properties that involve accessing the server of the WebIOPi application. For instance, the port number of the server and the password can be configured and customized according to the needs. Note that the login and password details are stored in a file that is protected (encrypted and hashed). Appendix A contains more details about configuring the GPIOs.

4.2 The Web Interface

The user interface for this system is in the form of a web application that can be opened from any device that can access the internet, whether a desktop computer, a tablet, or even a mobile device. The code that was used to construct the web interface was written in JavaScript, which was found to be easier and faster for us to learn and implement than the other alternatives (HTML or PHP, which need much more work such as setting up a SQL database).

Listing 4.1 contains the JavaScript code that was used for the web interface.

LISTING 4.1: The main JavaScript/HTML code

```
1 <html>
2   <head>
3     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4     <meta name="viewport" content = "height = device-height, width =
5       420, user-scalable = no" />
6     <title>WebIOPi | Demo</title>
7     <script type="text/javascript" src="/webiopi.js"></script>
8     <script type="text/javascript">
9       webiopi().ready(function() {
10
11         webiopi().setFunction(7, "out");
12         webiopi().setFunction(8, "out");
13         webiopi().setFunction(9, "out");
14         webiopi().setFunction(10, "out");
15
16
17         var content, button;
18         content = $("#content");
19
20         button = webiopi().createSequenceButton("On After 5sec", "Turn Coffee
21           Machine After 5sec", 8,1000,"000001");
22         content.append(button);
23
24         button = webiopi().createGPIOButton(9, "Christmas Light 1");
25         content.append(button);
26
27         button = webiopi().createGPIOButton(7, "Christmas Light 2");
28         content.append(button);
29
30         button = webiopi().createGPIOButton(8, "Coffee Machine");
31         content.append(button);
32
33       });
34
35 
```

```
37     </script>
38
39     <style type="text/css">
40         button {
41             display: block;
42             margin: 50px 50px 5px 5px;
43             width: 560px;
44             height: 45px;
45             font-size: 18pt;
46             font-weight: bold;
47             color: black;
48         }
49
50         input[type="range"] {
51             display: block;
52             width: 160px;
53             height: 45px;
54         }
55
56         #gpio7.LOW {
57             background-color: White;
58         }
59
60         #gpio7.HIGH {
61             background-color: Green;
62         }
63
64         #gpio8.LOW {
65             background-color: White;
66         }
67
68         #gpio8.HIGH {
69             background-color: Green;
70         }
71
72         #gpio9.LOW {
73             background-color: White;
74         }
75
76         #gpio9.HIGH {
77             background-color: Green;
78         }
79
80         #gpio10.LOW {
```

```
                                background-color: White;
77                            }

79                        #gpio10.HIGH {
80                            background-color: Green;
81                        }
82                    </style>
83    </head>
84    <body>
85        <div id="content" align="center"></div>
86    </body>
87</html>
```

4.3 Implementation (GPIO Control)

For the implementation part in this project four GPIOs of the Raspberry Pi were used and controlled by code that is written in python. While other alternatives were possible, such as programming in C, the choice fell on python because there were pre-defined libraries that have simplified the implementation, such as libraries for timers.

These codes that were written, however, were only implemented for learning purposes and were not used as part of the project.

Listing 4.3 contains the python code written to control the GPIOs through the web.

LISTING 4.2: The python script code for blinking one GPIOs on and off

```
1 from time import sleep
2 import RPi.GPIO as GPIO
3 GPIO.setmode(GPIO.BOARD)
4 GPIO.setup(24, GPIO.OUT)
5 GPIO.setup(26, GPIO.OUT)
6 GPIO.setup(21, GPIO.OUT)
7 while 1:
8     GPIO.output(21, False)
9     sleep(1)
10    GPIO.output(21, True)
11    sleep(1)
```

```
13     GPIO.output(24, False)
14     sleep(1)
15     GPIO.output(24, True)
16     sleep(1)
17
18     GPIO.output(26, False)
19     sleep(1)
20     GPIO.output(26, True)
21     sleep(1)
```

LISTING 4.3: The python script code for controlling the GPIOs

```
1 import RPi.GPIO as GPIO
2 GPIO.setmode(GPIO.BCM)
3 GPIO.setmode(GPIO.BOARD)
4 GPIO.setup(7, GPIO.IN)
5 GPIO.setup(26, GPIO.OUT)
6 while 1:
7
8     input_value = GPIO.input(7)
9     if input_value == False:
10         GPIO.output(26, True)
11     if input_value == True:
12         GPIO.output(26, False)
```

Chapter 5

Analysis

5.1 The Web Interface

The first part in Listing 4.1 (from line 3 to 8) contain the standard header file for the HTML web interface. This standard file is provided by the WebIOPi application. Lines 10 to 13 set the specific function each GPIO does. GPIOs 7 through 10 are set as outputs.

Next, two variables, `content` and `button` are declared. The `content` is used to append the button on the web interface (i.e to show the button on the web). The `button` would store different functions explained later.

Two functions were used: the first is the timed assertion (turn on after a certain time), and the second is the standard GPIO toggle (on/off). The timed assertion is implemented using the `createSequenceButton()` function (line 19) in which it asserts GPIO 8 following the sequence provided, which is “0,0,0,0,0, and then 1” with a 1000 ms (1 second) interval between each bit, thus creating timer of 5 seconds (time was chosen for the purpose of testing). The GPIO toggle is implemented using the `createGPIOButton()` function (used in lines 22, 25, 28, and 31), which takes the number of the GPIO and the name to be displayed on the button.

In order to create the button, a css code starting from line 36 to 52 has been used to create a button with the given parameters (width, length, margin, etc...).

Finally, in order to simulate the on/off effect on the browser, lines 54 to 81 have been used to make the button white when it is off and green when it is on.

5.2 Implementation (GPIO Control)

5.2.1 Blink

In Listing 4.2, we first have to import the GPIO and timer functionalities using the import function, we then activate the GPIOs to act as output using the GPIO.setmode (GPIO.BOARD). We then set pins 21, 24, and 26 to act as output GPIOs. We enter into an infinite loop were each GPIO would begin outputting False, wait one second (using the Sleep(1) function) then outputting True and waiting for another second before moving on to then next GPIO. This would simulate a blinking effect at every GPIO, one at a time.

In Listing 4.3, we first have to import the GPIO functionalities, it is just like including libraries in a normal c code. Instead of the '#' in python we use import. We then have to set how we are going to be using the GPIOs as inputs or outputs, and in order to that we use: GPIO.setmode(GPIO.BCM), allows the use of GPIOs as inputs GPIO.setmode(GPIO.BOARD) allows the use of GPIOs as outputs We then set GPIO at pin 7 as input and GPIO at pin 26 as output. We then enter an infinite while loop that would check whether the switch at input GPIO '7' is pressed then it would trigger the voltage at output GPIO pin 26. If the switch is off then the output GPIO would not receive any voltage. If we notice the hardware setup of the switch, we can see that the Input GPIO would always have a voltage on it. The output GPIO is set to high only when the switch is pressed and the voltage is ceased.

Chapter 6

Outputs & Results

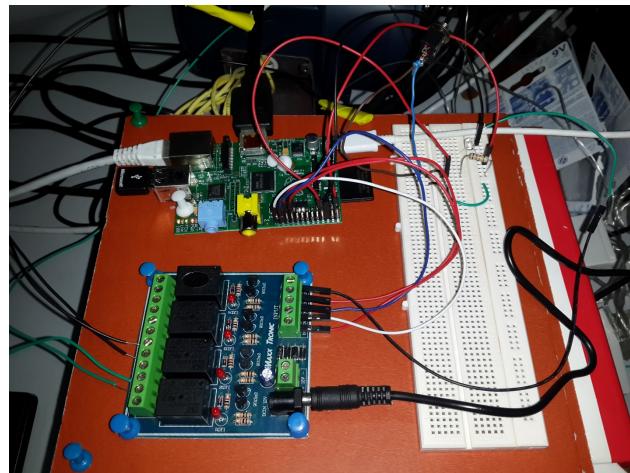


FIGURE 6.1: Final Circuit

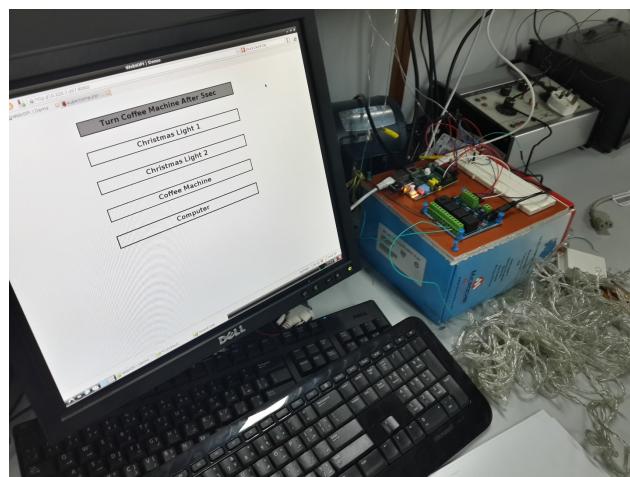


FIGURE 6.2: The whole system, when all appliances are off

Complete system in one picture:

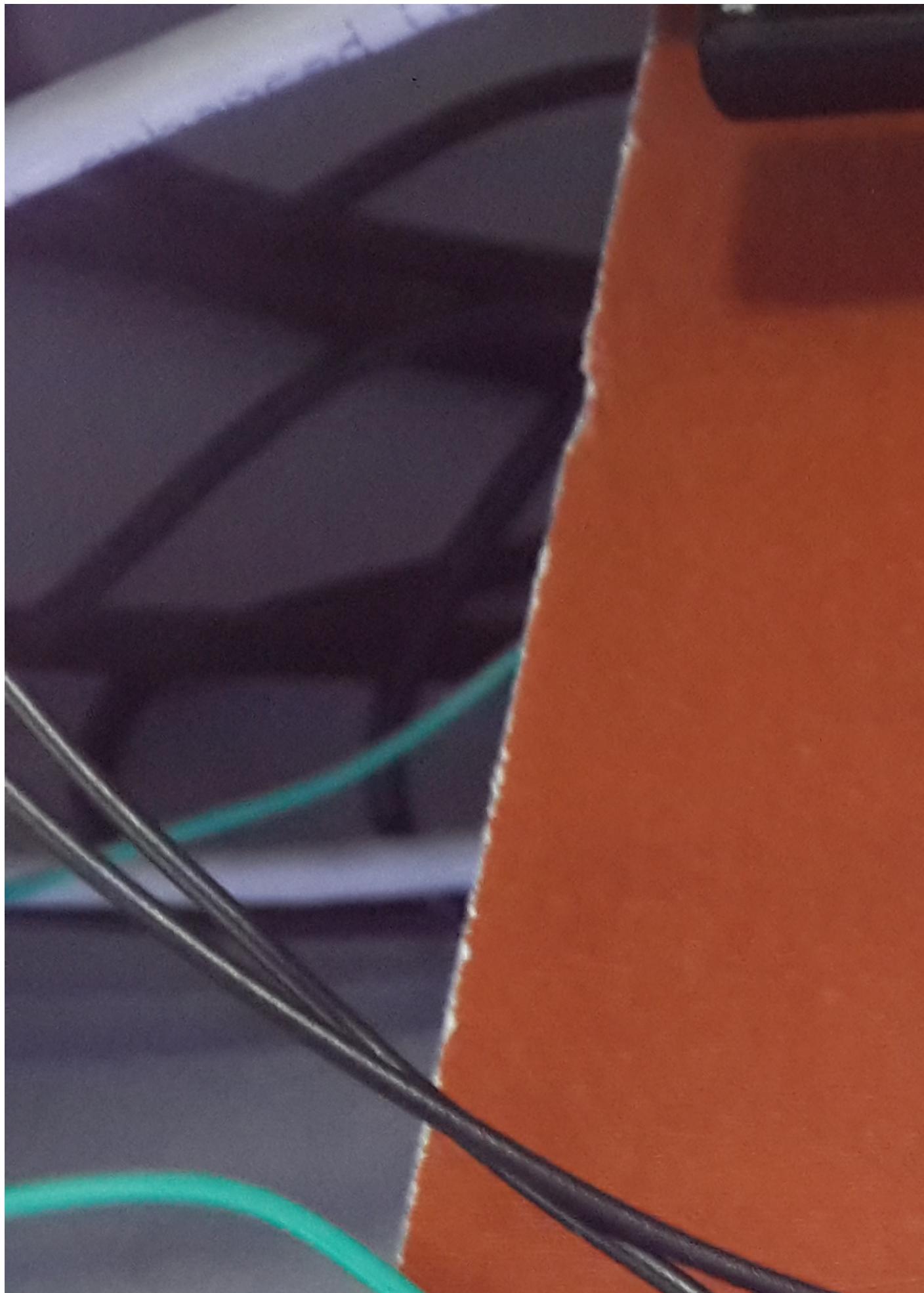
- Web interface

- Circuit
- Appliances



FIGURE 6.3: Christmas lights are on, when the Buttons are pressed on the Web page

Christmas lights are turned on when, the button is pressed. This is possible due to the schematic discussed previously.



Chapter 7

Problems Faced

- Given that Raspberry Pi is new to us, we first faced a couple of problems setting up the OS, the first SD card we got was not being read by the Raspberry pi, we later realized that there is a specific class of SD cards that are recognized by the Raspberry Pi
- Getting accustomed to python scripts.
- In the beginning, we faced some problems setting up the server. However, after some more research we were able to understand the steps needed to set it up.

Chapter 8

Future Improvements

- Set up a fully customizable user interface, in which the user can choose different operations for different appliances.
- Implement an LCD/Touch screen display, so that the user can have an easier access to the customization screen.
- Give the user the choice to select between web control, or manual control.
- Implement the interface through a mobile application

Chapter 9

Conclusion

This project has enhanced both our learning and experience in embedded systems, which is surely beneficial for this course and the upcoming senior design course. The purpose of this project, which is to connect all the topics (that were once taken separately in the course) into one functional project was successfully met. Not only were different concepts, such as interrupts and timers, implemented, but also the “good practices” which involve looking through manuals, creating software systematically, and joining it to the hardware, were also experienced. Ultimately, the process that has started with coming up with the idea until successfully implementing the project was all based on the determination to come up with a useful engineering solution, keeping in mind the need that has excited it. Hence, the result was a system that enables easy (through the web), effective (capable of switching any appliance), and instantaneous control over the household appliances (with a click of a button).

Appendix A

WebIOPi Configuration

In this appendix, the WebIOPi configuration will be included.

A.1

LISTING A.1: The configuration of WebIOPi

```
[GPIO]
# Initialize following GPIOs with given function and optional value
# This is used during WebIOPi start process
#21 = IN
#23 = OUT 0
#24 = OUT 0
#25 = OUT 1

#-----#
[~GPIO]
# Reset following GPIOs with given function and optional value
# This is used at the end of WebIOPi stop process
#21 = IN
#23 = IN
#24 = IN
#25 = OUT 0

#-----#
[SCRIPTS]
# Load custom scripts syntax :
# name = sourcefile
#   each sourcefile may have setup, loop and destroy functions and macros
#myscript = /home/pi/webiopi/examples/scripts/macros/script.py
```

```
26 #myscript = /home/pi/programming/index.py
27
28 #-----#
29
30 [HTTP]
31 # HTTP Server configuration
32 enabled = true
33 port = 8080
34
35 # File containing sha256(base64("user:password"))
36 # Use webiopi-passwd command to generate it
37 passwd-file = /etc/webiopi/passwd
38
39 # Use doc-root to change default HTML and resource files location
40 doc-root = /home/pi/programming
41
42 # Use welcome-file to change the default "Welcome" file
43 welcome-file = index.html
44
45 #-----#
46
47 [COAP]
48 # CoAP Server configuration
49 enabled = true
50 port = 5683
51 # Enable CoAP multicast
52 multicast = true
53
54 #-----#
55 [DEVICES]
56 # Device configuration syntax:
57 # name = device [args...]
58 #   name    : used in the URL mapping
59 #   device  : device name
60 #   args    : (optional) see device driver doc
61 # If enabled, devices configured here are mapped on REST API /device/name
62 # Devices are also accessible in custom scripts using deviceInstance(name)
63 # See device driver doc for methods and URI scheme available
64
65 # Raspberry native UART on GPIO, uncomment to enable
```

```
66 # Don't forget to remove console on ttyAMA0 in /boot/cmdline.txt
  # And also disable getty on ttyAMA0 in /etc/inittab
68 #serial0 = Serial device:ttyAMA0 baudrate:9600

70 # USB serial adapters
#usb0 = Serial device:ttyUSB0 baudrate:9600
72 #usb1 = Serial device:ttyACM0 baudrate:9600

74 #temp0 = TMP102
#temp1 = TMP102 slave:0x49
76 #temp2 = DS18B20
#temp3 = DS18B20 slave:28-0000049bc218
78
#bmp = BMP085

80
#gpio0 = PCF8574
82 #gpio1 = PCF8574 slave:0x21

84 #light0 = TSL2561T
#light1 = TSL2561T slave:0b0101001
86
#gpio0 = MCP23017
88 #gpio1 = MCP23017 slave:0x21
#gpio2 = MCP23017 slave:0x22
90
#pwm0 = PCA9685
92 #pwm1 = PCA9685 slave:0x41

94 #adc = MCP3008
#dac = MCP4922 chip:1
96
#-----#
98
[REST]
100 # By default, REST API allows to GET/POST on all GPIOs
  # Use gpio-export to limit GPIO available through REST API
102 #gpio-export = 21, 23, 24, 25

104 # Uncomment to forbid changing GPIO values
#gpio-post-value = false
```

```
106
107 # Uncomment to forbid changing GPIO functions
108 #gpio-post-function = false
109
110 # Uncomment to disable automatic device mapping
111 #device-mapping = false
112
113 #-----#
114 [ROUTES]
115 # Custom REST API route syntax :
116 # source = destination
117 #   source      : URL to route
118 #   destination : Resulting URL
119 # Adding routes allows to simplify access with Human comprehensive URLs
120
121 # In the next example we have the bedroom light connected to GPIO 25
122 # and a temperature sensor named temp2, defined in [DEVICES] section
123 # - GET  /bedroom/light      => GET  /GPIO/25/value, returns the light
124 #   state
125 # - POST /bedroom/light/0    => POST /GPIO/25/value/0, turn off the light
126 # - POST /bedroom/light/1    => POST /GPIO/25/value/1, turn on the
127 #   light
128 # - GET  /bedroom/temperature => GET  /devices/temp2/temperature/c,
129 #   returns the temperature in celsius
130 #/bedroom/light = /GPIO/25/value
131 #/bedroom/temperature = /devices/temp2/temperature/c
132
133 #/livingroom/light = /devices/expander0/0
134 #/livingroom/brightness = /devices/adc/0/float
135 #/livingroom/temperature = /devices/temp0/temperature/c
136
137 #/weather/temperature = /devices/bmp/temperature/c
138 #/weather/pressure = /devices/bmp/pressure/hpa
```