Final Project
Embedded Real-time Systems
Dr. Gholampour

Mohammad Hossein Shafizadegan
99104781

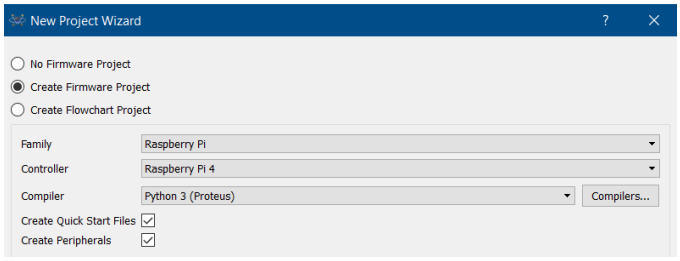
July 6, 2023

Contents

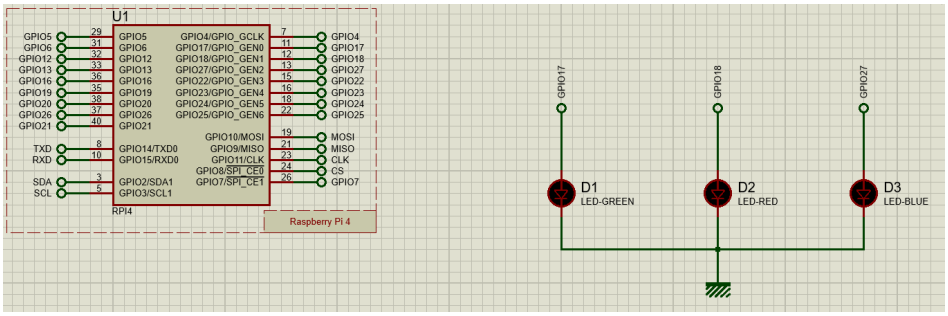
1 Restful Server Simulation

1.1 Creating Proteus project

Using Proteus we simply create a firmware project with following settings.



In the schematic tab, we add three LEDs with different colors and connect the 17, 18 and 27 GPIO pins of the Raspberry Pi to the LEDs correspondingly. The schematic of our circuit can be seen below:



1.2 Developing the Http server

In order to create the Http server we have developed a HTML code which will be used in the python code developed to create and run the server using HTTP.Server library. First we explain the HTML code and then the python code in the following.

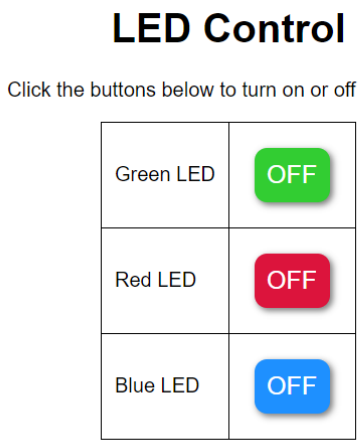
1.2.1 Creating HTML code

This HTML code contains a form which is actually a table. There is a row for each LED in the table with a button for each one. The button will show the status of the LED and by clicking the button the state of the corresponding LED will change and toggle.By clicking the button, it will send a POST request to the server. This can be achieved by setting the method=”POST” for the form. The value for each button will be set using the python variables (%s) to change the text of the buttons to ”On” or ”OFF” relatively whenever they clicked.

```
table1 = '''
    <h1>LED Control</h1>
    <p>Click the buttons below to turn on or off the LEDs.</p>
    <form action="/handle_form" method="post">
        <table>
            <tr>
                <td> Green LED </td>
                <td><input id="green" class="green" name="g_btn" type="submit" value=%s></td>
            </tr>
            <tr>
                <td> Red LED </td>
                <td><input id="red" class="red" name="r_btn" type="submit" value=%s></td>
            </tr>
            <tr>
                <td> Blue LED </td>
                <td><input id="blue" class="blue" name="b_btn" type="submit" value=%s></td>
            </tr>
        </table>
    </form>
'''
```

In order to make our webpage more sytlish and elegant we have used CSS codes and scripts which is stored the ”style” variable. Finally we have created a variable called ”RES1” which is the main section of the HTML code containing the body tag. In this variable we will use the ”table” and ”style” variables we defined before.

Our webpage will look like this :



Now in order to implement the webpage using bootstrap, we have also created two more variables called ”RES2” and ”table2”. In ”RES2” we simply add and import the required bootstrap library which actually includes some stylesheet and javascript files. Since we didn’t want to download and host Bootstrap ourselves, we include it from a CDN (Content Delivery Network). Now for making our HTML code more stylish and elegant, we will use bootstrap predefined class and ID e.g. container, row, col-sm-4, btn and others.

```
RES2 = '''
    <html>
    <head>
        <title>LED Control</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-9ndCyUaIbzAi2F
        <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js" integrity="sha384-geWF76RCwLtnZ8qwWowPQNguL
    </head>
    <body>
        <div class="container">
        <h1 class="text-center">LED Control</h1>
        <p class="text-center">Click the buttons below to turn on or off the LEDs.</p>
        %s
        </div>
        <script src="path/to/bootstrap.min.js"></script>
    </body>
    </html>
```

1.2.2 Python code

Now we explain how we developed the python code which is actually the back-end of the server.

First we add required libraries:

```
import RPi.GPIO as GPIO
 import sys
 import time
 from urllib.parse import parse_qs, urlparse
 from http.server import BaseHTTPRequestHandler, HTTPServer
```

Then we configure the Raspberry Pi GPIO pins as follows:

```
LED_1 = 17
 LED_2 = 18
 LED_3 = 27

 GPIO.setmode(GPIO.BCM)
 GPIO.setup(LED_1, GPIO.OUT)
 GPIO.setup(LED_2, GPIO.OUT)
 GPIO.setup(LED_3, GPIO.OUT)
```

We will add the parts of the code and variables related to HTML part of the server. Also we have created a variable called "port" and set it's value to 9000.

Now as the main part of the code. We have created a class called "LED_Handler" from BaseHTTPRequestHandler. For this class we will overwrite the "do_GET()" and "do_POST()" methods to handle GET and POST requests.

The "do_GET()" function can be seen below. By this whenever the server receives a request for root "/" the HTML part of the code will be sent. In this section as it can be seen, we can choose whether using simple HTML and CSS for our webpage or using the webpage powered by bootstrap by commenting and uncommenting the related lines.

```
def do_GET(self):
 try:
     #body_content = table1 % (LED_g, LED_r, LED_b,)
     body_content = table2 % (LED_g, LED_r, LED_b,)
     self.send_response(200)
     self.send_header("Content-type", "text/html")
     self.end_headers()
     #self.wfile.write(bytes(RES1 % (style, body_content,), "utf-8"))
     self.wfile.write(bytes(RES2 % (body_content,), "utf-8"))
 except:
     self.send_error(404, f"{sys.exc_info()[0]}")
```

Now we have to handle POST requests sent from the buttons to change the LED's state. The "do_POTS()" function will be as follows:

```
def do_POST(self):
 global LED_b, LED_r, LED_g
 try:
     content_length = int(self.headers['Content-Length'])
     form_data = self.rfile.read(content_length)

     # parse the form data into a dictionary
```

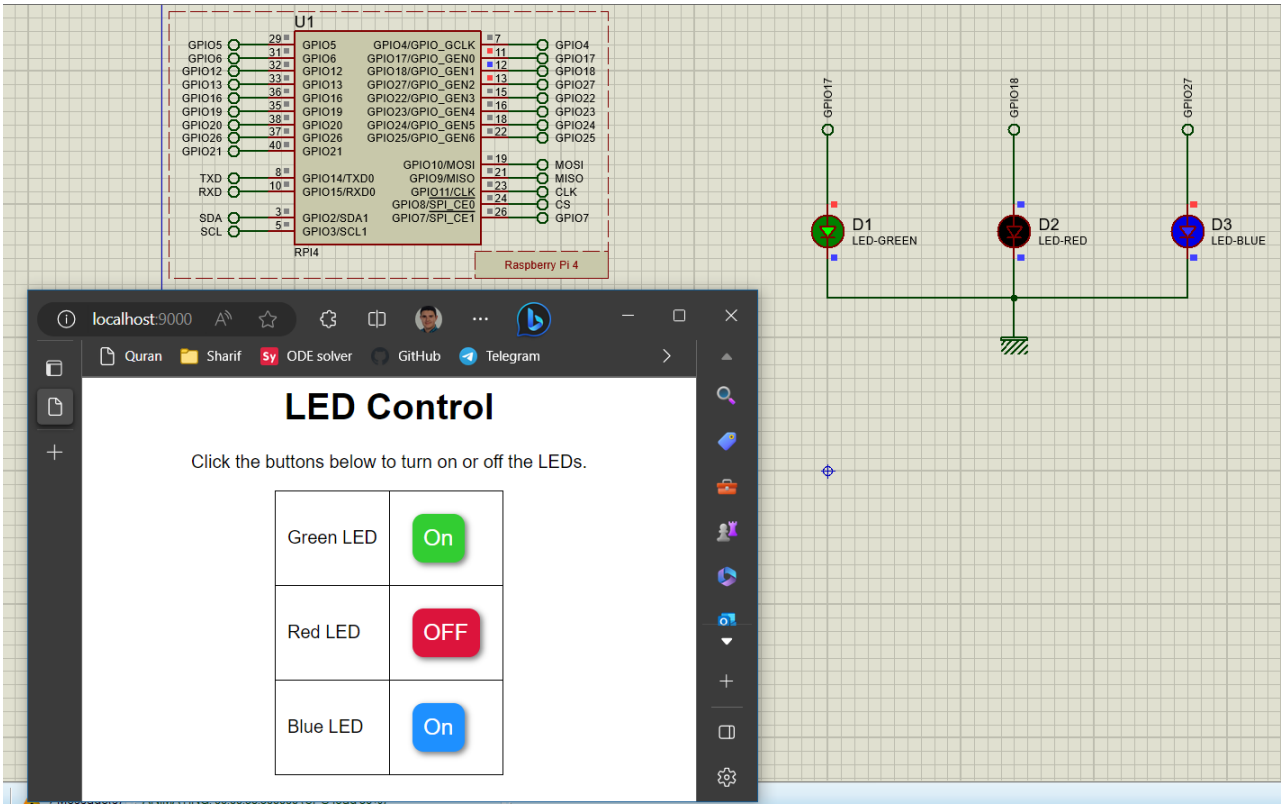
```
            form_data = parse_qs(form_data.decode())

            if 'g_btn' in form_data:
                LED_g = 'On' if LED_g == 'OFF' else 'OFF'
                state = GPIO.HIGH if LED_g == 'On' else GPIO.LOW
                GPIO.output(LED_1, state)
                time.sleep(0.1)
            elif 'r_btn' in form_data:
                LED_r = 'On' if LED_r == 'OFF' else 'OFF'
                state = GPIO.HIGH if LED_r == 'On' else GPIO.LOW
                GPIO.output(LED_2, state)
                time.sleep(0.1)
            elif 'b_btn' in form_data:
                LED_b = 'On' if LED_b == 'OFF' else 'OFF'
                state = GPIO.HIGH if LED_b == 'On' else GPIO.LOW
                GPIO.output(LED_3, state)
                time.sleep(0.1)

            self.send_response(303)
            self.send_header("Location", "/")
            self.end_headers()

        except:
            self.send_error(404, f"{sys.exc_info()[0]}")
```

Using this function, we first read and parse the POST request using "self.rfile.read(content_length)" and "parse_qs(form_data.decode())" functions to store the request in a python dictionary. Then using some continuous else if, we check which button is clicked. Then for the corresponding button, we simply toggle the text of the button from "On" to "OFF" or vice versa and also toggle the GPIO pin using "GPIO.output()" function.
Finally we redirect the page to root "/". This is the main part of our HTTP server.

Finally we create an object from our HTTPServer class and run the server as follows:

```
    myServer = HTTPServer(("localhost", Port), LED_Handler)
    try:
        myServer.serve_forever()
    except KeyboardInterrupt:
        myServer.server_close()
        GPIO.cleanup()
```

We run the Proteus project and in a browser tab, we visit our webpage. It can be seen that we can successfully turn the LEDs On or OFF.



Here we have used bootstrap for our webpage interface: