Smart Switch proof of concept using ESP8266

IoT Project - PoliMi - A.Y. 2019/20

Elia Battiston - matr. 939500

Contents

Introduction

This project's goal is to create a simple device with similar functions to a "Smart Switch". This means the device should:

▷ Receive commands from devices such as PCs and smartphones that are connected to the household LAN

▷ Provide or cut mains power to any connected appliance through a regular power plug

The proposed solution is based on the use of a NodeMCU ESP8266 board to manage communication and the control of a relay.

1 Hardware

ESP8266

The ESP8266 is a SoC (System on a Chip) that provides a full TCP/IP stack, microcontroller capabilities and WiFi communication.
In this project, a NodeMCU development board that mounts the ESP8266 was used. These kinds of boards make the process of prototyping and flashing firmwares much easier than using the standalone SoC, providing a way to quickly connect components with jumper cables or breadboards instead of having to solder them to the pins.

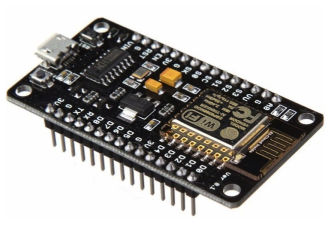
Figure 1: NodeMCU board

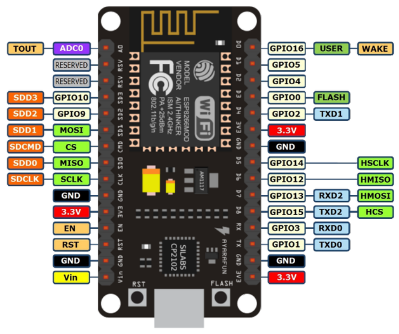

Figure 2: Pinout of the board

Power

The board can be powered in different ways:

- ▷ Using a regular micro USB smartphone charger, through the same port used to flash firmwares on the board when it is connected to a PC
- ▷ Providing between $5V$ and $10V$ to the `Vin` pin, which uses the included voltage regulator to power the board
- ▷ Providing exactly $3.3V$ to one of the pins labeled as `3V`

Either way, the board operates at $3.3V$. This is the voltage provided by pins in the HIGH state and by the different pins labeled as `3V`.
For the project, in order to keep the hardware setup as simple as possible and focus on the software part, I chose to power the NodeMCU using a smartphone charger.
Of course, in case this was a real product, the device should include a rectifier and a voltage regulator to power the microcontroller directly from the wall socket. This way, a single plug could be used to power the circuits and the connected appliance at the same time.

Relay

Since the NodeMCU operates at $3.3V$, a relay board that can be operated with a similar voltage is required.
The one used in this project mounts two relays that can be powered and independently controlled by the microcontroller, even though only one is required for our use.

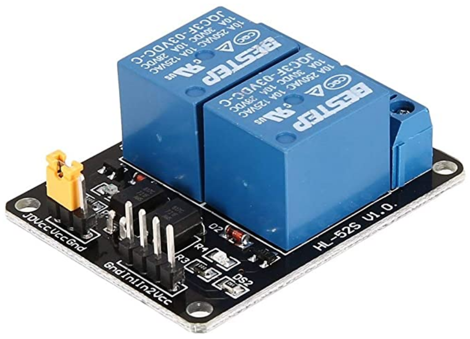
Figure 3: Relay board

Circuit

The circuit is very simple and only consists of 3 jumper wires:

- ▷ 2 wires ($3.3V$ and ground) provide power to the relays board

- ▷ 1 more wire connects the pin used to set the state of a relay to the `GPIO0` pin of the NodeMCU (labeled as D3). The voltage level of the pin will be set with through code to decide if the relay closes the circuit or not.

The relay is connected to one of the three wires of an extension cord. The appliance will connect to the female end of the extension cord, while the male end will be plugged to a wall socket. It will receive mains power only if the relay is active.

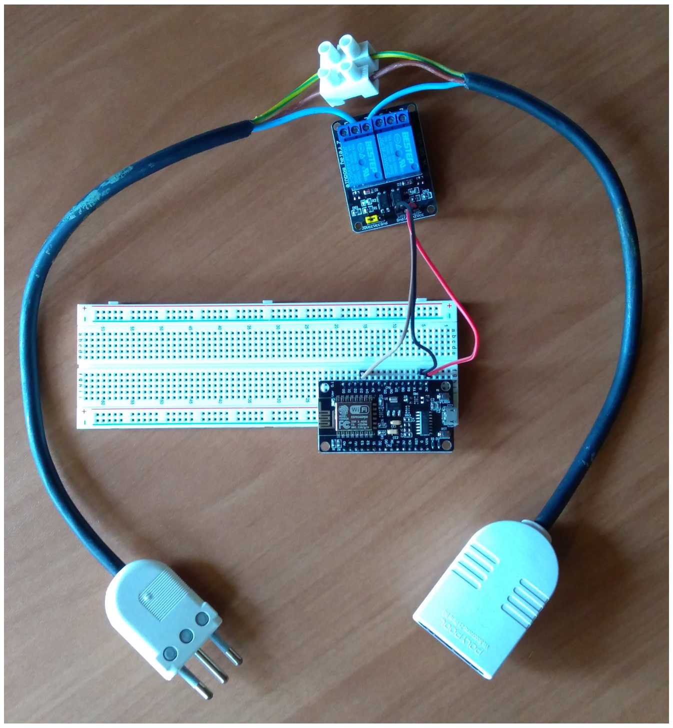
Figure 4: Photo of the complete circuit

It's important to note that Italian plugs can be inserted in two ways in the wall, making it impossible to know if the live wire of the wall plug will be connected to the blue or brown wire of the extension cord. In a real product, both wires should be interrupted with a relay to avoid safety hazards.

2 Software

Arduino code

The NodeMCU board was programmed using Arduino IDE. The source code can be found in the repository (4), in the file named `SmartSwitch.ino`.
This code has the task of making the ESP connect to the designated WiFi access point and listen for commands. When a command is received, the microcontroller should toggle the state of the connected relay.

Communication protocol

Different protocols could have been used to send commands to the device, such as MQTT, COAP, HTTP etc.
For this project, considering that the ESP provides a full fledged TCP/IP stack and that power consumption is not a problem (because the device would always be connected to a power plug), I chose to use HTTP. This also gives the opportunity to make the device accessible with any web browser.
In particular, the device will act as an HTTP web server: once it is connected to WiFi it will start to listen on TCP port 80. If the `/on` page is requested, it will turn on the relay, while if the `/off` page is requested it will turn it off. In either cases (or if the index page is requested) a simple interface consisting of the current relay state and two buttons is returned, as shown in Figure 5.

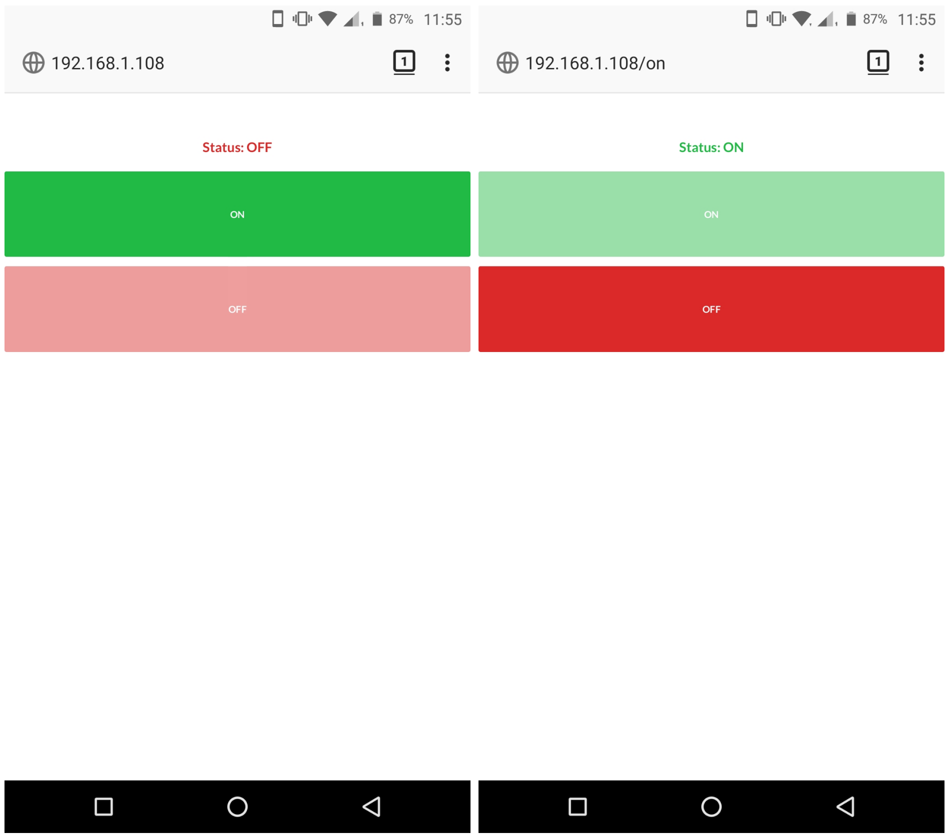

Figure 5: Pages for ON and OFF states

The use of common HTTP requests allows standalone use, but also makes it possible to quickly integrate the device in a more complex workflow (e.g. with Node-RED, as will be shown later).

Finding the address of the device

When the ESP connects to the WiFi network, an IP address is assigned to it through DHCP. In order to communicate with it, the user needs to find out this address or have another way to reliably reach it.
While prototyping I scanned my LAN to find the right address and this problem was not taken into consideration, but in a final version of the device it would be mandatory to solve it.
Different approaches could be used:

- The device can be set up with a static IP address outside of the DHCP pool of the router
- A static association between the MAC address of the ESP and an IP address can be set up on the DHCP server
- If some software (such as a mobile application) is developed to make communication with the device easier, the ESP could periodically send multicast UDP messages to advertise its current IP. These messages could be read by the application that will remember the current IP of the device.
- If the device needs to be available from outside the LAN, a combination of Dynamic DNS (to link the IP address to a domain name) and UPnP (to dynamically set up port forwarding) can make communication possible without further configuration. However, in this case, connections to the device should be properly secured.

In the rest of the document we assume the IP address of the device is correctly set up and known.

Node-RED and ThingSpeak

While the device as described until now works on its own, integrating it with IoT platforms can make it more useful and easy to use.
For this project, I decided to create a Node-RED flow to:

- ▷ Make the switch controllable from a Node-RED dashboard
- ▷ Show the current state of the switch on a ThingSpeak channel

An export of the flow can be found in the repository (4), in the file named `SmartSwitchFlow.json`.

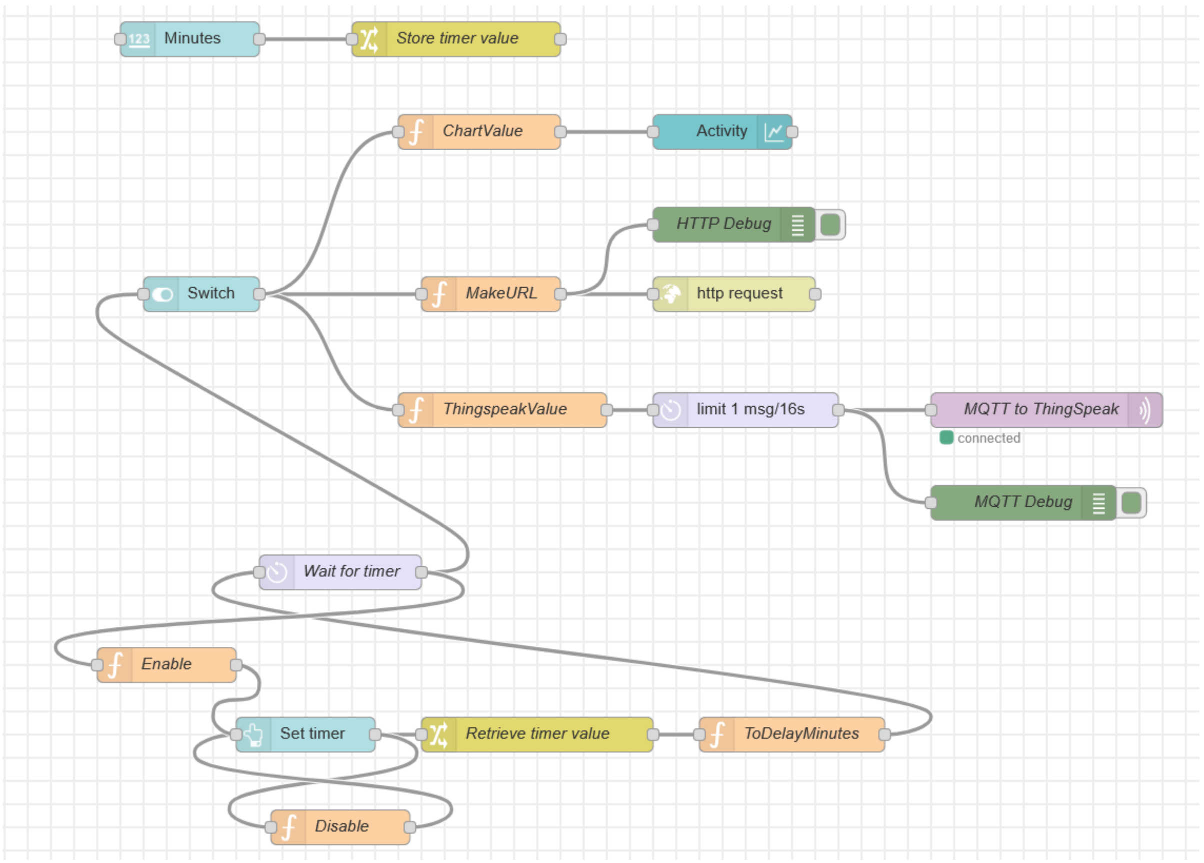

Figure 6: Node-RED flow

Dashboard

The implemented Node-RED dashboard provides two ways of controlling the relay:

- ▷ A Switch to immediately toggle its state
- ▷ A timer that can be set to switch the device on after a configurable number of minutes

On the bottom of the dashboard, a chart also shows the recent activity of the switch.

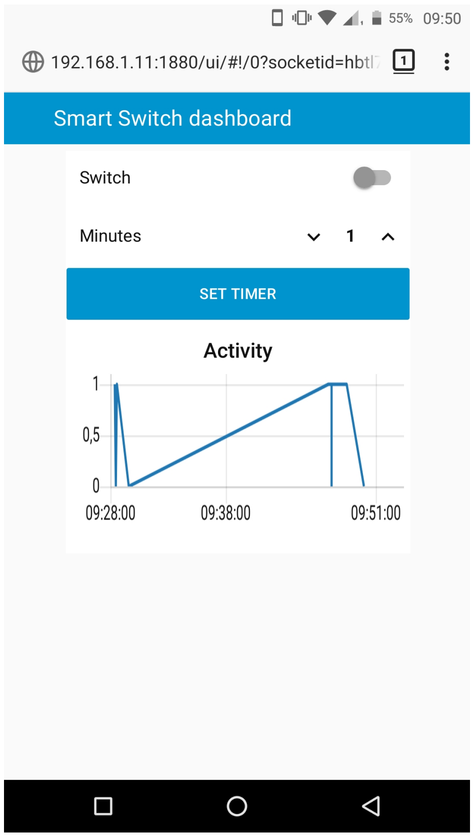

Figure 7: Node-RED dashboard

Timer

The controls used to set the timer are a numeric input and a button.
Every time the value of the numeric input changes, it is stored in the `flow.timerValue` variable using a *change* node.
When the button is pressed, `flow.timerValue` is retrieved with another *change* node and converted to a millisecond value with a *function* node. This value is stored in `msg.delay` and fed to a *delay* node, that waits for the duration set by the user. When the delay is over, a message telling the *Switch* node to turn on is sent forward.
Using two additional *function* nodes the button used to start the timer is disabled while the flow is waiting. This serves a double function: it indicates that the timer is currently active and prevents other timers to be started in the meanwhile.

Request to the device

The message coming from the *Switch* node is passed to a *function* node that computes the correct URL to request in order to obtain the desired behavior. For example, if the IP address of the device is `192.168.1.10` and the incoming `msg.payload` is `true`, the URL will be "`http://192.168.1.10/on`".
The output of the node is logged and sent to an *http request* node to complete the request.

ThingSpeak

One branch of the flow is used to update the status of the switch on the project's ThingSpeak channel (4).
This channel includes a graph of the switch's activity in time, and a lamp indicator that shows its current status.
A *function* node is used to craft the correct string to update the switch's status through MQTT. Additionally, a *delay* node limits updates to 1 every 16 seconds to avoid exceeding the limit that discards messages that are not

spaced apart by 15 seconds at least.
The resulting message is logged and sent to the *MQTT out* node that will perform a PUBLISH to ThingSpeak's broker.

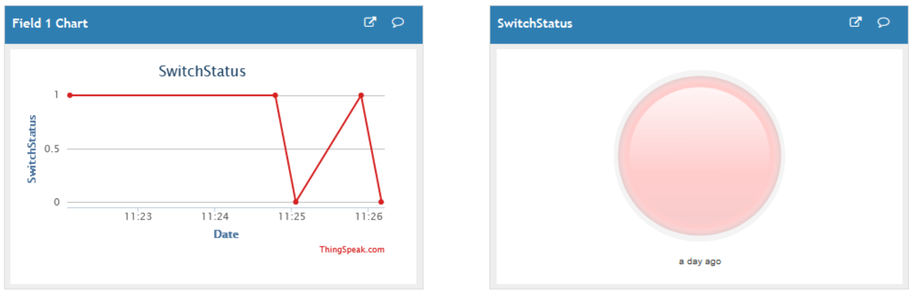

Figure 8: ThingSpeak channel

Chart

When the value of *Switch* changes, the message it sends contains a boolean value that can't be shown in a chart. With a *function* node the value is converted to an integer (0 or 1) and sent to the *chart* node.
This node takes the value of `msg.payload` and plots it in a graph located in the dashboard, showing the progression of the switch's activity.

3 Testing

To prove the device works as intended, the "Links" (4) section includes links to:

▷ A video that shows the device while in use, both in its standalone mode and using Node-RED

▷ Wireshark capture files of the packets sent between a PC and the switch while sending commands

Video demonstration

The video shows, in this order:

▷ The device being switched on and off by directly connecting to the ESP

▷ The device being switched on and off from the Node-RED dashboard

▷ Setting a timer of 1 minute to achieve delayed switching

Wireshark captures

▷ `standalone.pcapng` shows a series of commands sent from a PC with IP 192.168.1.1 to the device with IP 192.168.1.127

▷ `nodered.pcapng` shows the interaction between Node-RED (on 192.168.1.11), the ESP (192.168.1.127) and the ThingSpeak MQTT broker.
In the beginning, the device is switched off using the dashboard. After that, a timer of 1 minute is started in order to turn on the device with a delay. Each time, an MQTT publish is performed.

4 Links

▷ Code repository: https://github.com/EliaBattiston/SmartSwitchPOC

▷ ThingSpeak channel: https://thingspeak.com/channels/1090887

- ▷ Video demonstration: https://tinyurl.com/SmartSwitchVideo
- ▷ Wireshark packet captures: https://tinyurl.com/SmartSwitchPcap