Internet of Things - 089073

Home Automation and Monitoring System

Angelo Claudio Re
Giovanni Scotti

Professor: *Matteo Cesana*

Academic Year 2017/2018

Document version: 1.0

Contents

Section 1

Introduction

1.1 Purpose

This document is intended to describe a *Home Automation and Monitoring* system, its electronic components as well as its constraints and the interaction with the real world and the end user. A useful example of work environment will be provided to clarify the scope of the system itself and how it can be deployed.

The documentation is full of diagrams, pictures and schematics to let the final user easily implement the infrastructure and take advantage of its countless features.

This document is mainly addressed to computer scientists, IoT enthusiasts, makers and anyone, with some electronics and programming skills, who wants to keep an eye on his/her home and to control devices remotely.

1.2 Scope

The *Home Automation and Monitoring* system aims to offer a smart solution to home automation and monitoring needs. It is intended for those kind of users who want to visualize information about their home, such as temperature or humidity, and control devices remotely.

The system consists mainly of:

- a *back-end*, which is in charge of managing MQTT messaging protocol and runs Node-RED. Data coming from boards provided with sensors and actuators are gathered and sent to a remote service provider to store information into the cloud and to open up further analysis.

- a *front end*, which is offered by a user-friendly and web-accessible Node-RED dashboard that can be easily reached from any device either connected to the same network or to the Internet by means of port forwarding.

The system must be secure. This is why authentication is required in order to access the dashboard. Moreover, MQTT must be secured too and clients provide credentials to the broker to join the network. Last but not least, power consumption has been taken into account: battery-powered boards can enter a deep sleep state after publishing data.

1.3 Definitions, Acronyms, Abbreviations

- **AP:** wireless Access Point, it allows Wi-Fi devices to connect a to wired network. Usually it is an integral component of routers for home or office use.
- **Back-end:** any device or computer program that remains in the background and offers application logic and communication interfaces to work with the front-end counterpart. It can provide a data access layer.
- **Front-end:** any part of a system the users directly interact with. It provides the so called presentation layer.
- **IoT:** Internet Of Things.
- **LAN:** Local Area Network.
- **MQTT:** Message Queuing Telemetry Transport, it is a publish-subscribe messaging protocol.
- **System:** the entire infrastructure and all the devices involved in the *Home Automation and Monitoring* project.
- **SoC:** System-On-Chip.

Section 2

System Implementation

2.1 Architecture Overview

The overall system architecture is depicted by Figure 2.1.

To begin with, the whole system mainly operates within the user Local Area Network. A wireless router is the central point of the entire infrastructure because all the involved devices are connected to it via wireless communication.

The Raspberry Pi board is the backbone of the back-end processing activities. It is in charge of running Node-RED and acts as MQTT broker. It collects information that are sent out by several WiFi-capable modules taking advantage of the publish-subscribe MQTT connectivity protocol. Each module is provided with sensors that gather information about the surrounding environment.

ESP-12E and ESP-32 NodeMCU are the IoT WiFi boards that have been chosen to design the infrastructure. There exists a great variety of sensors that can be hooked up to these ESP modules and ease of programming via Arduino IDE makes them an appropriate choice too. Moreover, they offer full TCP/IP stack support. Each board does not only publish data, but can also subscribe to a specific topic and receive remote commands to control actuators, such as relays.

The end user is able to visualize a user-friendly, web-accessible dashboard directly from his/her own devices, which must be in the same network of the other system components, in case port forwarding is disabled. The interface is provided by Node-RED development tool. Furthermore, data are forwarded to ThingSpeak by the Raspberry board in order to log them in a remote database and to enable further analysis: ThingSpeak has integrated support from the numerical computing software MATLAB.

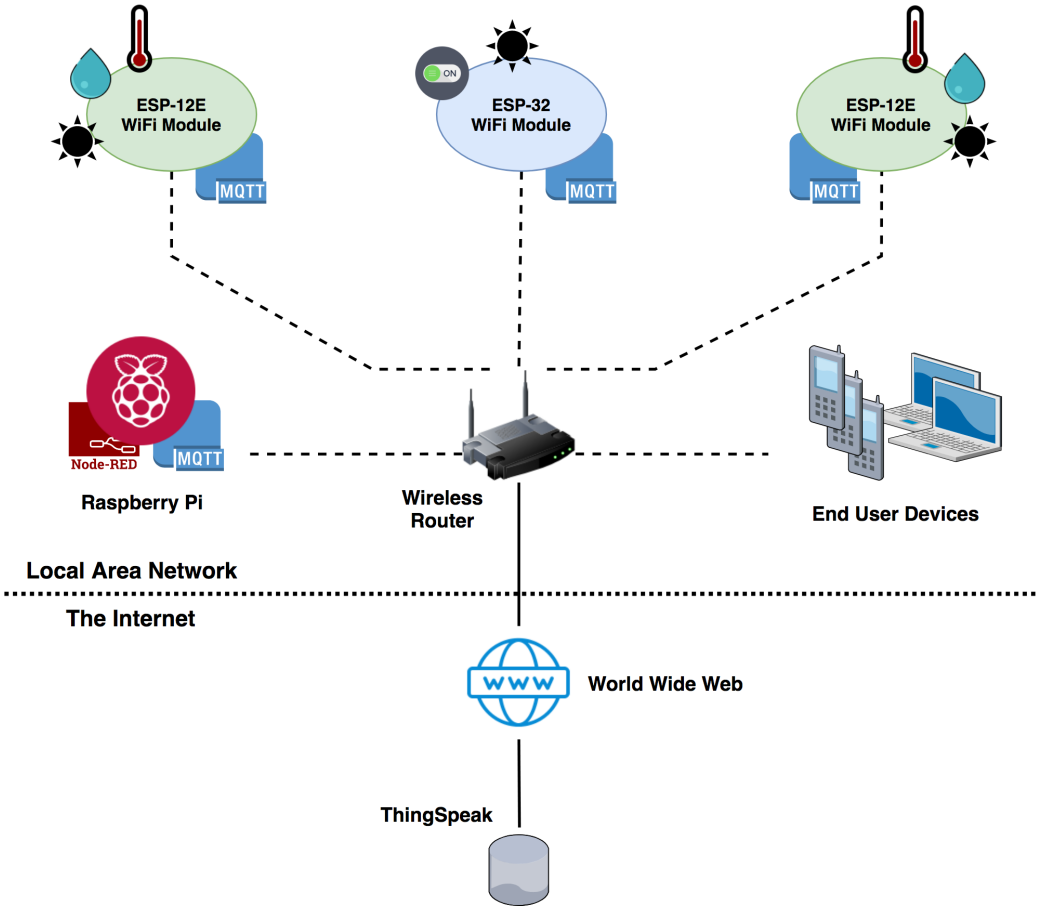

Figure 2.1: System architecture overview. Dashed lines represent wireless communication, solid lines stand for wired connections.

2.2 Configuring the RPi

The Raspberry Pi is a tiny single-board computer featuring a Broadcom System-On-Chip with an integrated ARM compatible processor. There exist several different generations and models and the role in the system will suit them all. Anyway, using a Pi 3 Model B is highly recommended: it has a quad-core processor and on-board WiFi, Bluetooth and USB boot capabilities which do not require any extra module.

The Raspberry board needs a fresh installation of Raspbian Stretch, the official Debian-based Linux operating system of the Raspberry Pi Foundation. Other third-party operating systems are not taken into account by this document.

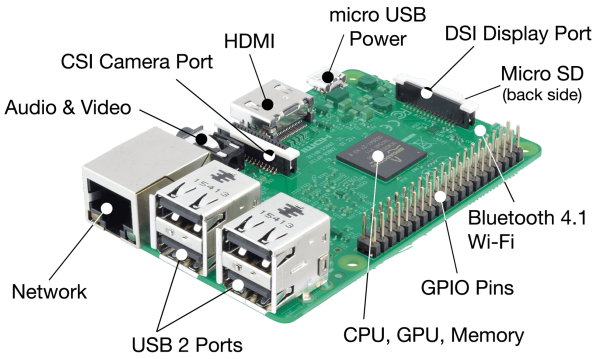

Figure 2.2: Location of connectors and main Integrated Circuits on the Raspberry Pi 3 Model B.

The board is used as headless computer, a device that has been configured to operate without a monitor, keyboard and mouse. The command line is accessed remotely from another computer on the same network using SSH. Since Raspbian has the SSH server disabled by default, it needs to be manually enabled from the desktop or in the terminal via `raspi-config`.

Setting up a static IP address for the Raspberry is strongly recommended. It can be configured in the DHCP context menu of the wireless router. Since the procedure is different on any device, the handbook should be referred to.

Before proceeding with the following sections, all the software must be up-to-date. This straightforward operation is carried out by executing the commands below in a terminal window:

```
sudo apt-get update
sudo apt-get upgrade
```

2.2.1 MQTT

MQTT is a lightweight messaging protocol that works on top of TCP/IP. It provides resource-constrained network clients with a simple way to distribute and exchange information. MQTT is based on the publish-subscribe messaging pattern that requires a message broker to work properly: in this case the role perfectly fits the Raspberry Pi board.

Information about a given topic are sent, or published, to a server that behaves as MQTT message broker. Then, the latter pushes the information out to those clients that have previously subscribed to the above-mentioned topics.

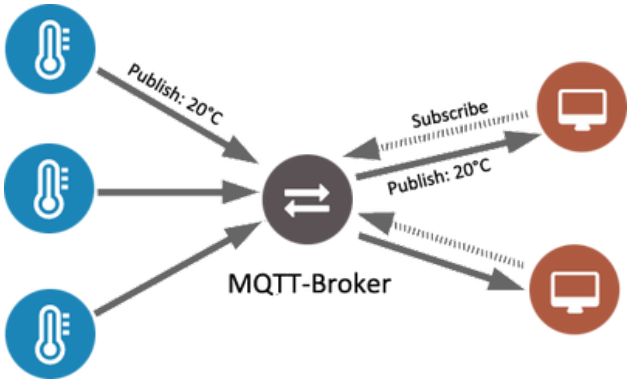

Figure 2.3: The message broker is the central point of the MQTT infrastructure. It receives published information and forward it to subscribed clients.

MQTT offers three quality of service (QoS) levels to guarantee a certain degree of performance to a data flow:

- **0.** At most once delivery: this is the minimal level and guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored by the sender.
- **1.** At least once delivery: all the messages will be delivered at least once to the receiver. This is the QoS level used in the *Home and Automation Monitoring* system because the overhead is lower than QoS 2 and it guarantees the message arrives at least once. Duplicates are not a problem at all because the user interface will be updated one or more time with the same piece of information and this will be completely unnoticeable by the end user.
- **2.** Exactly once delivery. It is the safest and the slowest quality of service level.

2.2.2 Mosquitto

Mosquitto is an open source message broker that implements the MQTT protocol. It is suitable for both low-power single board computers and powerful servers.

To install the broker, the following command must be run from the Raspberry Pi terminal:

```
sudo apt-get install mosquitto
```

The broker will automatically start whenever the Raspberry is powered up. The next step is adding users and passwords to secure the MQTT connection. The current directory must be `/etc/mosquitto` while typing successive commands:

```
// Add the first user
sudo mosquitto_passwd -c passwordfile user

// Add another user to the existing password file
sudo mosquitto_passwd -b passwordfile other_user password
```

The following two lines must be added to the `mosquitto.conf` file.

```
allow_anonymous false
password_file /etc/mosquitto/passwordfile
```

In this case, just a single user has been created to manage the MQTT connection: username is `admin` and password is `hamrpi`. These credentials are used to set up MQTT by Node-RED and any ESP node.

2.2.3 Node-RED

2.3 Getting Started with ESP Boards

The *Home Automation and Monitoring* system makes use of ESP modules to collect data from sensors and transmit them to the Raspberry Pi taking advantage of MQTT protocol.

Two kinds of boards have been selected to play the role of MQTT client; namely NodeMCU ESP-12E and ESP-32 development kits. The former is based on the widely explored ESP8266 System-On-Chip, which combines WiFi and microcontroller capabilities. The latter utilizes an ESP32 SoC that is very similar, but far more powerful. It has been designed to provide robustness, versatility and reliability in a wide variety of applications.

Both the board can be easily programmed using the Arduino IDE after installing the appropriate cores via Arduino Boards Manager. The core installation will not be covered by this document.

Please notice that there is not a particular reason to use one specific board or the other: they both can fulfill the requirements of the project. The usage of different boards creates heterogeneity and variety, which are always present in the real world.

2.3.1 ESP-12E: Sensors, Wiring and Code

An accurate picture of the board pinout is shown in Figure 2.4. The ESP8266 module requires 3.3V power supply. Luckily the board provides a voltage regulator and can be connected to 5V using microUSB connector or *Vin* pin.

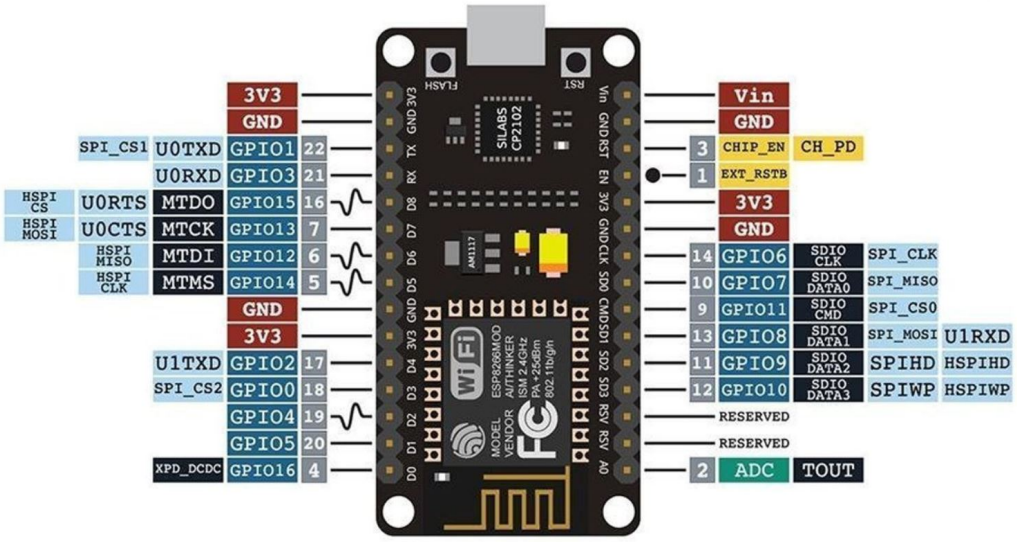

Figure 2.4: ESP-12E pinout.

There is a number of two ESP-12E boards involved in the system. They both have the same configuration, sensors and wiring, but can be deployed in two different areas.

Status LEDs

Three LEDs give information about the status of the board:

- **Red**: the board is trying to get WiFi access.
- **Yellow**: the board is setting up a MQTT connection. The MQTT broker must be up and running to succeed.
- **Green**: the board is active and it is sending sensor data to the MQTT broker or listening to messages related to its subscriptions.

Temperature and humidity sensor

DHT11 and DTH22 are low cost temperature and humidity sensors. They are made of two parts: a thermistor and a capacitive humidity sensor. A small chip inside performs analog to digital conversion to let any microcontroller read the digital output signal.

Even though the two DHT sensors are interchangeable, DHT22 is more accurate and covers a larger range of temperature.

Since the system involves two ESP-12E boards, both the sensors are used for the sake of completeness. Moreover, the fact that it is fairly easy to connect up to them, as shown in Figure 2.5, the low cost and ease of interfacing make them an appropriate choice for the system.

Light sensor

A light sensor can be used to detect the current ambient light level, in particular the presence or absence of light. There exist several types of light sensors: photoresistors, phototransistors, photodiodes...

The chosen approach was making use of a photoresistor, taking advantage of the ADC provided by ESP boards.

A photoresistor changes its resistance based on the amount of light that falls upon it: the more the light, the less the resistance. A simple voltage divider can be used to give an input to the ADC, as illustrated by Figure 2.5.

Deep sleep

ESP8266, the core of ESP-12E boards, supports sleeping. While sleeping, the device draws much less power than while awake: this could be a good approach to save power when batteries are the only source of electricity.

Deep sleep is one out of four supported types of sleep mode: everything is off but the Real Time Clock. This is the most power efficient option.

Pin D0 (GPIO 16) needs to be connected to RST to wake up the device when deep sleep is over: whenever RST receives a LOW signal, it restarts the microcontroller. Once the device is in deep sleep mode, a LOW signal will be sent to GPIO 16 as soon as the timer is up.

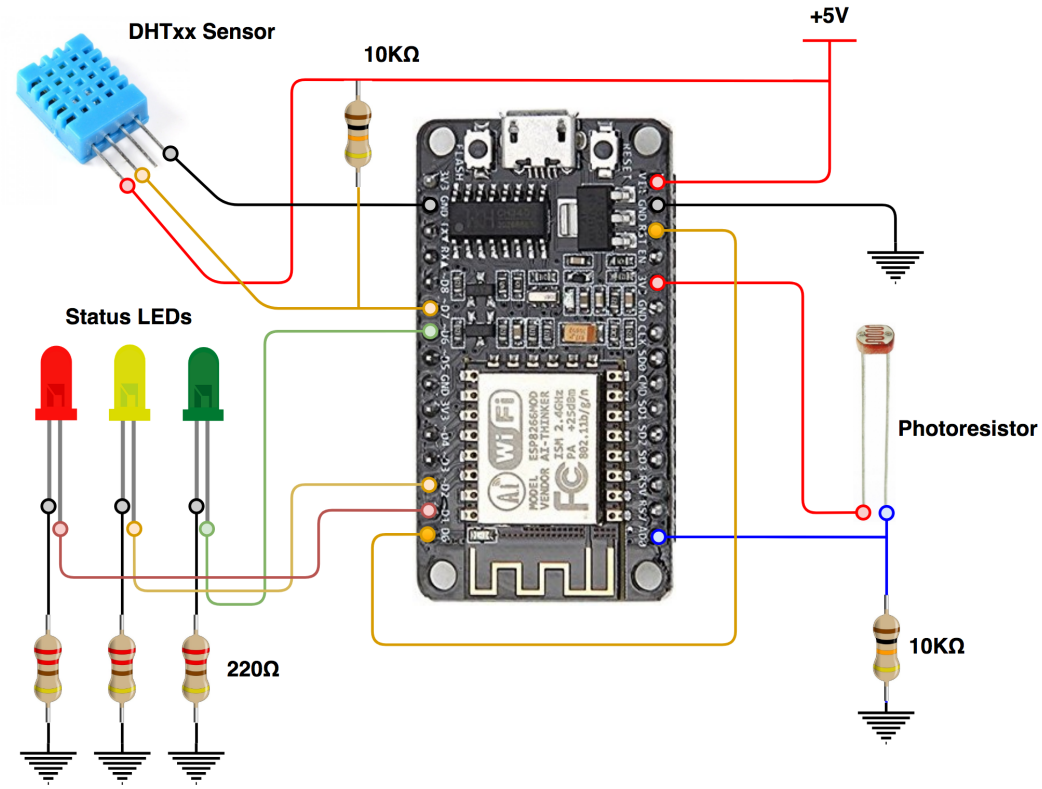

Figure 2.5: ESP-12E schematic with sensors and status LEDs. Notice the wire to tie D0 to RST pin to enable deep sleep.

Code

TODO: code description.

2.3.2 ESP-32: Sensors, Wiring and Code

2.4 Connecting to ThingSpeak

Section 3

System Deployment

3.1 Work Environment Example

3.2 Future Extensions