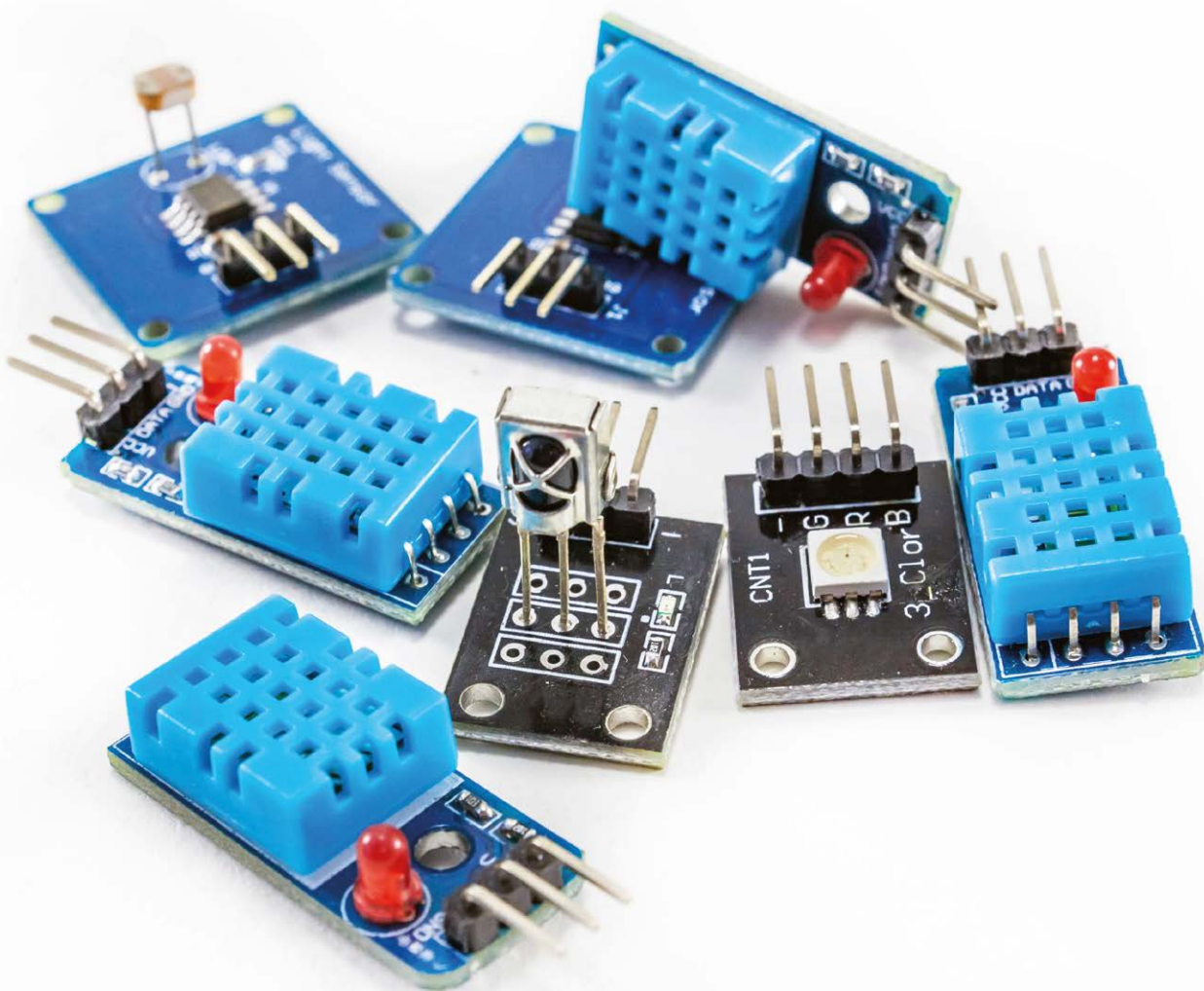


# The Ultimate Compendium of Sensor Projects





---

# **The Ultimate Compendium of Sensor Projects**

**40+ Projects using Arduino,  
Raspberry Pi and ESP32**



**Dogan Ibrahim**



**elektor**

**LEARN > DESIGN > SHARE**

● This is an Elektor Publication. Elektor is the media brand of  
Elektor International Media B.V.  
78 York Street, London W1H 1DP, UK  
Phone: (+44) (0)20 7692 8344

● All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licensing Agency Ltd., 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's permission to reproduce any part of the publication should be addressed to the publishers.

● Declaration

The author and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause.

● Acknowledgements

The author would like to express his thanks to Ferdinand te Walvaart of Elektor for the valuable suggestions he made throughout the duration of the preparation of this book. The author would like to thank also to his wife Nadire for her encouragement, motivation, and for being patient with me while working on this book.

● British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

● **ISBN 978-1-907920-78-3**

© Copyright 2019: Elektor International Media b.v.

Prepress Production: D-Vision, Julian van den Berg

First published in the United Kingdom 2019

Printed in the Netherlands by Wilco



Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (including magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics. [www.elektormagazine.com](http://www.elektormagazine.com)

**LEARN > DESIGN > SHARE**

<b>Preface</b> .....	<b>11</b>
<b>Chapter 1 • Arduino Uno hardware interface and project development</b> .....	<b>13</b>
1.1 Overview .....	13
1.2 Arduino Uno board .....	13
1.3 Arduino Uno program development .....	14
1.4 Project – Flashing two colour LEDs .....	15
1.5 Summary .....	19
<b>Chapter 2 • Raspberry Pi hardware interface and project development</b> .....	<b>20</b>
2.1 Overview .....	20
2.2 The Raspberry Pi 3 board .....	20
2.3 Raspberry Pi 3 GPIO pin definitions .....	21
2.4 Setting up the Wi-Fi and remote access .....	22
2.5 Shutting down or rebooting in GUI mod .....	27
2.6 Remote access of the desktop .....	27
2.7 Creating and running a Python program .....	29
2.8 The GPIO library .....	32
2.8.1 Pin numbering .....	32
2.8.2 Channel (I/O port pin) configuration .....	32
2.9 Raspberry Pi project development cycle .....	35
2.10 Project – Flashing two colour LEDs .....	36
2.11 Summary .....	39
<b>Chapter 3 • ESP32 hardware interface and project development</b> .....	<b>40</b>
3.1 Overview .....	40
3.2 ESP32 DevKitC hardware .....	40
3.3 Arduino IDE for the ESP32 DevKitC .....	42
3.3.1 Installing the Arduino IDE for the ESP32 DevKitC .....	43
3.4 Project – Flashing two colour LEDs .....	47
3.5 Summary .....	50
<b>Chapter 4 • Basic sensor projects: Arduino - Raspberry Pi - ESP32</b> .....	<b>51</b>
4.1 Overview .....	51

4.2 Light Projects . . . . .	51
4.2.1 Project 1 – Changing the LED brightness . . . . .	51
4.2.2 Project 2 – Using an RGB LED – Rainbow colours . . . . .	62
4.2.3 Project 3 – Magic wand . . . . .	67
4.2.4 Project 4 – Silent door alarm . . . . .	75
4.2.5 Project 5 – Dark sensor with timed relay – Arduino Uno project . . . . .	81
4.2.6 Project 6 – Dark sensor with timed relay – Raspberry Pi project . . . . .	85
4.2.7 Project 7 – Dark sensor with timed relay – ESP32 DevKitC project . . . . .	93
4.2.8 Project 8 – Turn ON lights when it is dark and the door is opened – Arduino Uno project . . . . .	96
4.2.9 Project 9 – Secret Key using the photo interrupter (light barrier) module . . . . .	100
4.2.10 Project 10 – Using the magic light cup module . . . . .	108
4.3 Summary . . . . .	113
<b>Chapter 5 • Infrared receiver-transmitter projects . . . . .</b>	<b>114</b>
5.1 Overview . . . . .	114
5.2 Project 1 – Receiving and decoding the codes of commercial IR handsets. . . . .	114
5.3 Project 2 – Controlling 2 - Colour LEDs with a commercial IR handset . . . . .	131
5.4 Project 3 – Infrared transmitter - Scanning the TV channels using a commercial IR handset. . . . .	136
5.5 Project 4 – Two communicating Arduino Uno's using IR . . . . .	141
5.6 Summary . . . . .	145
<b>Chapter 6 • Vibration and shock projects . . . . .</b>	<b>146</b>
6.1 Overview . . . . .	146
6.2 Project 1 – Target shooting detector . . . . .	146
6.3 Project 2 – Vibration based toggle switch . . . . .	150
6.4 Project 3 – No shock time duration measurement . . . . .	153
6.5 Summary . . . . .	155
<b>Chapter 7 • Ultrasonic sensor projects . . . . .</b>	<b>156</b>
7.1 Overview . . . . .	156
7.2 Project 1 – Ultrasonic reverse parking with buzzer. . . . .	156
7.3 Summary . . . . .	167
<b>Chapter 8 • Sound sensor projects . . . . .</b>	<b>168</b>

---

8.1 Overview . . . . .	168
8.2 Project 1 – Toggle lights by clapping hands. . . . .	168
8.3 Summary . . . . .	171
<b>Chapter 9 • Passive piezo buzzer sensor projects . . . . .</b>	<b>172</b>
9.1 Overview . . . . .	172
9.2 Project 1 – Playing melody . . . . .	172
9.3 Summary . . . . .	179
<b>Chapter 10 • Magnetic sensor projects . . . . .</b>	<b>180</b>
10.1 Overview . . . . .	180
10.2 Project 1 – Measuring magnetic field strength . . . . .	180
10.3 Project 2 – Magnetic door alarm . . . . .	183
10.4 Project 3 – Magnetic musical instrument. . . . .	185
10.5 Summary . . . . .	187
<b>Chapter 11 • Flame sensor projects. . . . .</b>	<b>188</b>
11.1 Overview . . . . .	188
11.2 Project 1 – Flame sensor with buzzer output. . . . .	188
11.3 Summary . . . . .	191
<b>Chapter 12 • Joystick module projects . . . . .</b>	<b>192</b>
12.1 Overview . . . . .	192
12.2 Project 1 – Using the joystick . . . . .	192
12.3 Project 2 – Joystick based musical instrument. . . . .	198
12.4 Summary . . . . .	200
<b>Chapter 13 • Obstacle sensor projects . . . . .</b>	<b>201</b>
13.1 Overview . . . . .	201
13.2 Project 1 – Aid with car parking. . . . .	201
13.3 Project 2 – Metal touch sensor . . . . .	204
13.4 Summary . . . . .	206
<b>Chapter 14 • Tracking sensor module projects. . . . .</b>	<b>207</b>
14.1 Overview . . . . .	207
14.2 Project 1 – Line tracking. . . . .	207
14.3 Project 2 – Secret code lock . . . . .	211
14.4 Summary . . . . .	216

<b>Chapter 15 • Rotary encoder module projects . . . . .</b>	<b>217</b>
15.1 Overview . . . . .	217
15.2 Project 1 – Rotary encoder evaluation . . . . .	217
15.3 Project 2 – Rotary encoder direction and position . . . . .	220
15.4 Summary . . . . .	224
<b>Chapter 16 • Heartbeat sensor module projects . . . . .</b>	<b>225</b>
16.1 Overview . . . . .	225
16.2 Project 1 – Displaying heartbeat . . . . .	225
16.3 Summary . . . . .	227
<b>Chapter 17 • Temperature, humidity, and pressure sensor projects . . . . .</b>	<b>228</b>
17.1 Overview . . . . .	228
17.2 Project 1 – Displaying and plotting the ambient temperature on the monitor . . .	228
17.3 Project 2 – Temperature sensor with buzzer . . . . .	236
17.4 Project 3 – Displaying the temperature on LCD – Arduino Uno . . . . .	239
17.5 Project 4 – Saving temperature as CSV file on PC with timestamp – Arduino Uno . . . . .	246
17.6 Project 5 – Displaying the temperature on LCD – ESP32 DevKitC. . . . .	252
17.7 Project 6 – Displaying the temperature on LCD – Raspberry Pi . . . . .	254
17.8 Project 7 – Saving temperature as CSV file on PC with timestamp – Raspberry Pi . . . . .	258
17.9 Project 8 – ON/OFF temperature control – Arduino Uno . . . . .	260
17.10 Project 9 – ON/OFF temperature control – ESP32 DevKitC . . . . .	265
17.11 Project 10 – ON/OFF temperature control – Raspberry Pi . . . . .	266
17.12 Summary . . . . .	269
<b>Chapter 18 • Wi-Fi and Bluetooth based projects using sensors –         ESP32 DevKitC . . . . .</b>	<b>270</b>
18.1 Overview . . . . .	270
18.2 Project 1 – Displaying temperature and humidity on a mobile phone using Wi-Fi . . . . .	270
18.3 Project 2 – Remote control from mobile phone using Wi-Fi . . . . .	279
18.4 Project 3 – Sending temperature and humidity to mobile phone using Bluetooth classic . . . . .	284
<b>Chapter 19 • Wi-Fi and Bluetooth based projects using sensors – Raspberry Pi . .</b>	<b>288</b>
19.1 Overview . . . . .	288



---

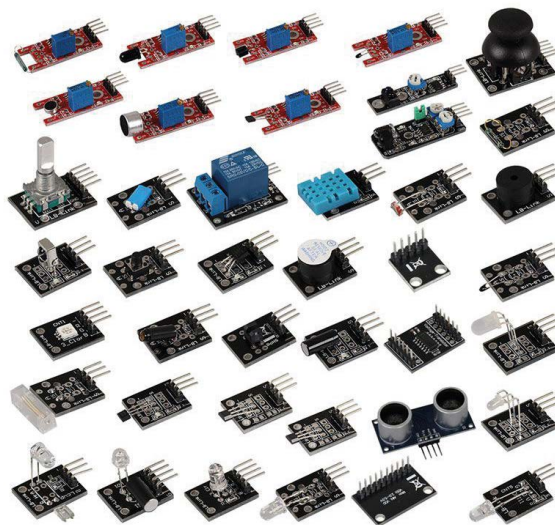
19.2 Project 1 – Displaying temperature and humidity on a mobile phone using Wi-Fi . . .	288
19.3 Project 2 – Sending the temperature and humidity data to the Cloud using Wi-Fi . . . . .	291
19.4 Project 3 – Bluetooth based remote control from mobile phone . . . . .	297
<b>Chapter 20 • Wi-Fi and Bluetooth based projects using sensors – Arduino Uno . .</b>	<b>307</b>
20.1 Overview . . . . .	307
20.2 Project 1 – Controlling a relay from mobile phone using Wi-Fi . . . . .	307
20.3 Project 2 – Displaying temperature and humidity on a mobile phone using Wi-Fi . . . . .	313
<b>Appendix A - Sensor kit contents (JOY-iT Sensor-Kit X40) . . . . .</b>	<b>323</b>
<b>Appendix B – Projects and sensor modules used. . . . .</b>	<b>324</b>
<b>Appendix C – Sensor modules and projects using them . . . . .</b>	<b>326</b>
<b>Index . . . . .</b>	<b>328</b>

## Preface

Sensors are devices or components that detect events or changes in their environments and send information to other electronics, frequently to microcontroller systems. Sensors are used in everyday life, to measure such things as temperature, humidity, pressure, wind and rain, touch, light levels, liquid levels, altitude, force, and many more.

Although some sensors have digital outputs, most sensors used in everyday life have analog outputs, usually in the form of voltages that are proportional to a measured quantity. This output voltage is normally fed to the input port of a microcontroller for processing. For example, the output of an analog temperature sensor is connected to an analog input port (an analog-to-digital converter) of a microcontroller. The microcontroller reads the temperature as a digital value and converts it into real physical temperature, which is then displayed or used to control the temperature of a machine or room.

This book is about using the sensors found in the sensor kit. Altogether, there are 40 sensors distributed with the kit. Some sensors provide analog outputs while others have digital outputs, and some have both analog and digital outputs. The kit includes sensors to measure temperature, humidity, atmospheric pressure, light intensity, and sound. There are also 2 and 3 colour LEDs, tilt switches, magnetic switches, relay, reed switches, piezo buzzer, button, joystick, obstacle detector, heartbeat sensor, analog-digital converter, voltage translator, vibrations switch, etc.



*40 Sensors All-in-1 Kit*

The book is intended to teach how to use sensors with the popular microcontroller development systems: Arduino Uno, ESP32 DevKitC, and Raspberry Pi. Example projects are given to show how to use the sensors with these microcontrollers. The programs can be modified for other microcontrollers, such as PIC, STM32, Banana Pi, CubieBoard, Beaglebone,

etc. All the projects given in the book were built using a standard size breadboard, and they were fully tested and were all working. The projects are described with the following sub-headings:

- Project description
- Aim of the project
- Block diagram
- Circuit diagram
- Program listing

The operation of each sensor and each program listing are described in detail so that the readers will have no difficulty in either constructing or expanding a given project. Some projects make use of more than one sensor from the kit. It is recommended that readers follow the projects in the given order since some of the software tools used in some projects depend on the installation of these tools in an earlier project.

Full program listings of the projects with many comments are available at the Elektor website of the book and readers should be able to copy and use these programs without having to make modifications.

Arduino Uno and ESP32 DevKitC programs are based on using the Arduino IDE with the C language. Raspberry Pi projects use the Python programming language. Although the Arduino based projects specify Arduino Uno as the development board, these projects should also work with other Arduino development boards, such as Arduino Mega, Arduino Nano, etc.

I hope the readers find the book useful and enjoy experimenting with the various sensors.

*Dogan Ibrahim*  
*June 2019*

## Chapter 1 • Arduino Uno hardware interface and project development

### 1.1 Overview

In this book, we will use the Arduino Uno board in the Arduino based sensor projects. This Chapter shows the location of the various components on the Arduino Uno board and also describes the hardware interface to the external world. A simple project is given in this Chapter to make the reader familiar with developing programs with the Arduino Uno. The first program flashes a two colour LED alternately every second.

### 1.2 Arduino Uno board

Figure 1.1 shows the Arduino Uno board in close-up with the major components marked. The pin definitions are as follows (see Figure 1.2).

A0 – A5:	Analog input ports
0 – 13:	Digital input/output ports
~3,~5,~6,~9,~10,~11:	PWM output ports
0,1:	UART RX/TX pins. LEDs labelled TX, RX will flash when data is transmitted or received respectively
GND:	Power supply ground pin
5V:	Regulated +5V output
3.3V:	Regulated +3.3V output
Vin:	Voltage input (instead of using Power In or USB). The voltage must be in the range 7-12V. It is regulated to +5V internally. This pin can also be used as a voltage output (if power is supplied using Power In or USB port). The voltage is a copy of the voltage supplied through Power In or the USB port.
I0Ref:	Used by external shield boards to know if they should operate as +3.3V or as +5V devices
Power In:	Power supply Barrel Jack pin (6V to 12V)
USB port:	Power and data port (connect to computer)
User LED:	Onboard LED connected to output port 13 (can be used for testpurposes)

Notice that when the Arduino Uno is powered by the USB port (e.g. from a computer) the maximum current capacity is around 400mA for the +5V pin and 150mA for the +3.3V pin. When powered by an external source, the maximum current for the 5V pin is around 900mA and 150mA for the +3.3V pin. Any current drawn from the +3.3V goes through the +5V pin. Therefore, you have to take this into account when powering external devices.

The absolute maximum current for any I/O pin is specified as 40mA (it is however recommended not to exceed 20mA). The absolute total current from all the I/O pins is 200mA.

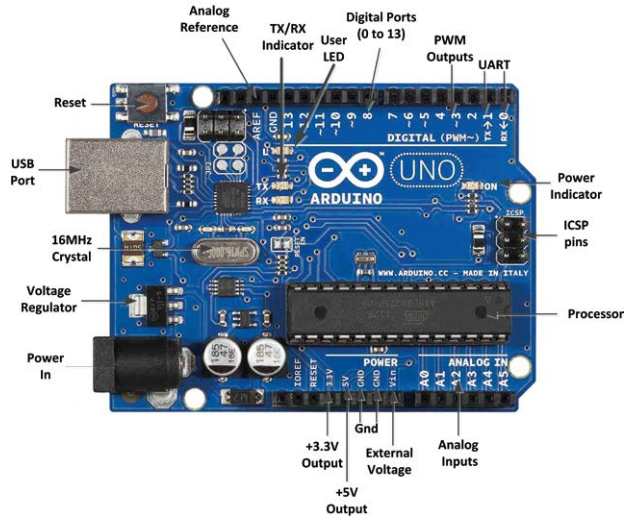
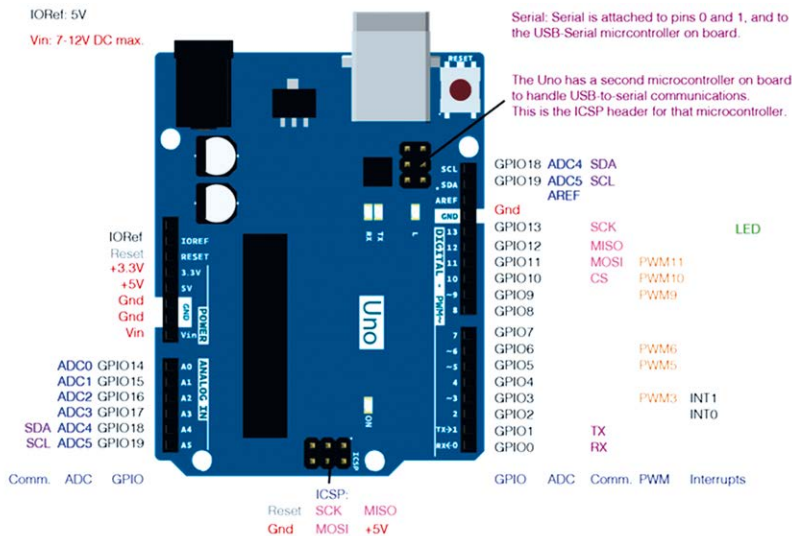


Figure 1.1 Arduino Uno board



The steps to writing and upload a program to your Arduino Uno are as follows:

- Connect your Arduino Uno to the USB port of your computer
- Start the Arduino IDE on your computer
- Click **Tools -> Board** and select board type as **Arduino/Gerduino Uno**
- Click **Tools -> Port** and select the serial port that your Arduino Uno is connected to
- Write and then save your program
- Click **Sketch -> Verify/Compile** to compile your program. Make sure there are no compilation errors
- Click **Sketch->Upload** to upload the executable code to the program memory of your Arduino Uno

Two example projects are given in the following sections to make the reader familiar with the project development cycle using the Arduino Uno. It is assumed that the reader has some background in basic electronics and also some working knowledge of writing a high-level language.

#### **1.4 Project – Flashing two colour LEDs**

**description:** This is perhaps the easiest project you can design using your Arduino Uno. In this project, a two colour LED with common cathode is connected to the Arduino Uno. The LED colours are flashed alternately at a rate of one second.

**Aim:** The aim of this project is to show how to write, compile and upload a program to the Arduino Uno. The project additionally shows how to use some of the I/O and timing functions of the Arduino.

**Sensor Used:** Sensor KY-011 is used in this project. This sensor consists of a red and green colour LED housed in a package with a common cathode terminal. The sensor and its pins are shown in Figure 1.3. The sensor has 3 pins: red LED pin, green LED pin and GND pin. **The green LED pin is marked with letter S on the board for identification.** An LED is turned ON when logic 1 is applied to its pins. The basic specifications of this sensor are as follows:

Operating voltage:	2.0V to 2.5V
Operating current:	10mA (depends on the required brightness)
Wavelength:	green (571nm), red (644nm)
Luminous intensity (MCD):	green (20-40), red (40-80)



Figure 1.3 Sensor KY-011

**Block diagram:** The block diagram of the project is shown in Figure 1.4.

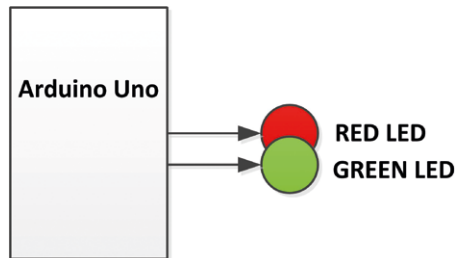


Figure 1.4 Block diagram of the project

**Circuit diagram:** The circuit diagram of the project is shown in Figure 1.5. The green and red LED pins are connected to port pins 2 and 3 of the Arduino Uno respectively through current limiting resistors. The GND pin is connected to Arduino Uno GND pin. The value of the current limiting resistor is calculated as follows:

The high voltage of an output pin is 5V. The forward voltage across an LED is approximately 2.0V (for the red LED this is 1.8V, and for the green LED this is 2.8V). Assuming that the forward current to the LED will be set to 10mA (reasonable brightness), then, the value of the current limiting resistor is:

$$R = (5V - 2V) / 10mA = 300 \text{ ohm, use 330 ohm resistor (you can use smaller value, e.g. 270 ohms for the green LED)}$$

In Figure 1.5 the LED is operated in current sourcing mode where a high output from the I/O pin drives the LED. The LED can also be operated in current sinking mode where the other end of the LED is connected to +5V supply and not to the ground. In current sinking mode, the LED is turned ON when the I/O pin is at logic low.

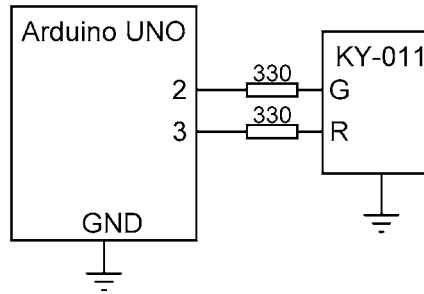


Figure 1.5 Circuit diagram of the project

**Construction:** The project is constructed on a breadboard as shown in Figure 1.6. Jumper wires are used to connect the KY-011 to digital port pins 2 and 3 and GND of the Arduino Uno.

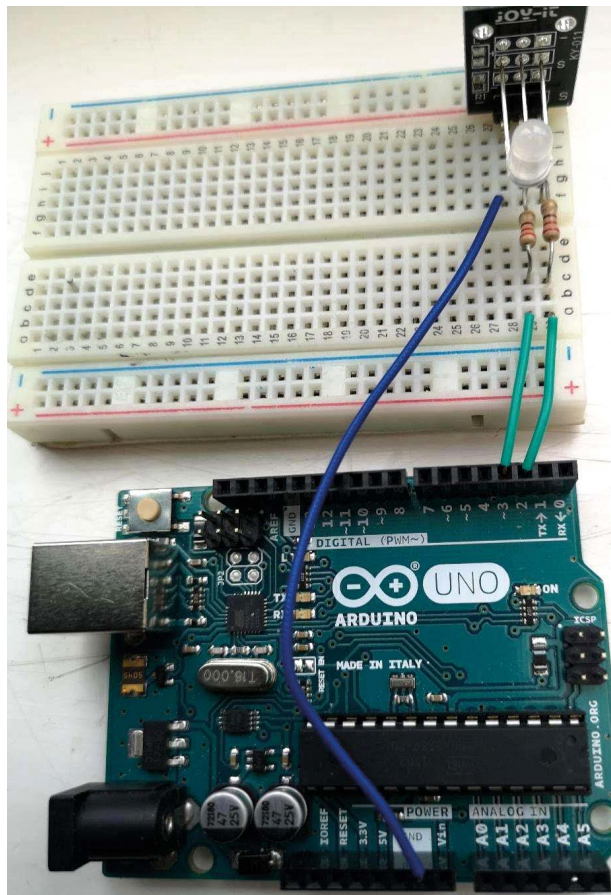


Figure 1.6 The project constructed on a breadboard



**Program listing:** The program is called **TWOCOLOUR** and the listing is shown in Figure 1.7. At the beginning of the program, port pins 2 and 3 are assigned to the green LED and red LED pins respectively. Inside the setup routine, port pins 2 and 3 where the LED pins are connected to are configured as outputs. Inside the main program loop, the two colours are turned ON and OFF alternately with a one-second delay between each output. **Verify/Compile** to make sure that there are no errors and then **Upload** the program code to your Arduino Uno. You should see the two colours flashing at a rate of one second.

It is highly recommended by the author to comment the lines in your program and describe the operation being performed as shown in Figure 1.7. You should also include a heading and describe what the program does briefly. This makes it easier to understand and maintain the program at a later date. It also makes it easier for anyone else to understand the logic of the program when they read it.

```

/*****
 *          FLASHING GREEN AND RED LEDs
 *          =====
 *
 * In this program the KY-011 sensor is used which has a green and
 * a red LED with a common cathode terminal. The green LED and the
 * red LED are connected to I/O pins 2 and 3 of the Arduino Uno
 * respectively. The program flashes the LEDs alternately at a rate
 * of one second.
 *
 * Author: Dogan Ibrahim
 * Date  : April 2019
 * File  : TWOCOLOUR
 *****/
int GreenLED = 2;           // Green LED pin
int RedLED = 3;             // Red LED pin
#define ON HIGH
#define OFF LOW

void setup()
{
    pinMode(GreenLED, OUTPUT);    // Set as output
    pinMode(RedLED, OUTPUT);      // Set as output
}

void loop()
{
    digitalWrite(GreenLED, ON);    // Turn ON green
    digitalWrite(RedLED, OFF);     // Turn OFF red
    delay(1000);                  // Wait 1 sec
    digitalWrite(GreenLED, OFF);   // Turn OFF green
    digitalWrite(RedLED, ON);      // Turn ON red
}

```

```
    delay(1000);                // Wait 1 sec
}
```

*Figure 1.7 Program listing of the project*

**What we have learned:** In this project, we have learned how to use the following Arduino functions:

<b>int:</b>	declares an integer variable
<b>#define:</b>	assigns value or text to a string
<b>pinMode (port pin, mode):</b>	used to configure an I/O port pin as input or output
<b>digitalWrite(port pin, value):</b>	used to output digital value (logic LOW or HIGH) to a port pin
<b>delay(n):</b>	creates a delay of <b>n</b> milliseconds

### 1.5 Summary

In this Chapter, we have seen the various components and the pin definitions of the Arduino Uno microcontroller. Additionally, a simple project is given to demonstrate how to design simple projects and also how to compile upload the program code to the program memory of the Arduino Uno microcontroller.

In the next Chapter, we will be looking at how to use the Raspberry Pi microcontroller in projects.

## Chapter 2 • Raspberry Pi hardware interface and project development

### 2.1 Overview

In this book, we will be using a Raspberry Pi 3 Model B in our Raspberry Pi projects. This Chapter shows the location of the various components on the Raspberry Pi board and also describes the hardware interface to the external world. Setting up Wi-Fi and remote access to your Raspberry Pi computer, and becoming familiar with the Python programming environment are also described briefly. A simple project is given to familiarize the reader with the steps of designing a project. In this book, when we write Raspberry Pi 3 we actually mean Raspberry Pi 3 Model B.

### 2.2 The Raspberry Pi 3 board

Figure 2.1 shows the Raspberry Pi 3 board with the major components marked. Some details on each component are given in this section.

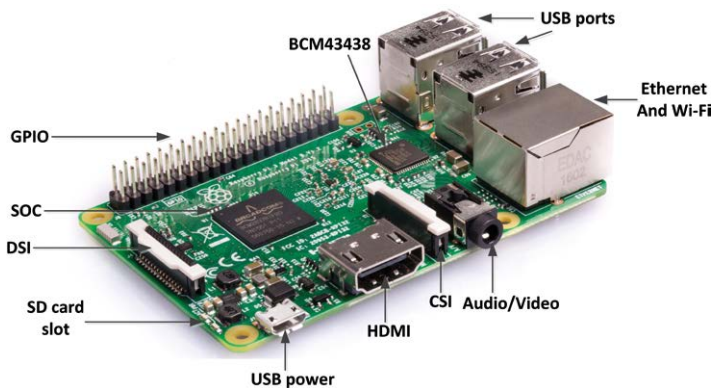


Figure 2.1 Raspberry Pi 3 board

- USB ports:** Raspberry Pi 3 has 4 USB ports to connect a mouse, keyboard, webcam, etc.
- Ethernet and Wi-Fi:** Although the Raspberry Pi 3 has built-in Wi-Fi, it can also directly be connected to a router through an Ethernet cable connected to this socket.
- Audio/Video Jack:** A headphone or a speaker can be connected to this 3.5mm socket. This socket also carries a composite video interface.
- CSI:** This is the Camera Serial Interface where a compatible Raspberry Pi camera can be attached here.
- HDMI:** A suitable monitor can be connected to this port. The port carries both audio and video signals.
- USB power:** A +5V 2A power supply should be connected to this USB socket to provide power to the Raspberry Pi 3.

<b>SD card slot:</b>	A micro SD card carrying the operating system must be attached to this slot.
<b>DSI:</b>	A suitable display can be connected to this Display Interface connector.
<b>SOC:</b>	This is the Broadcom BCM2837 System On Chip (SOC) which contains the 1.2GHz 64-bit quad-core ARM Cortex-A53 processor.
<b>GPIO:</b>	The General Purpose Input-Output port is 40 pins wide.
<b>BCM43438:</b>	This chip provides the Wi-Fi and Bluetooth to the Raspberry Pi 3.

### 2.3 Raspberry Pi 3 GPIO pin definitions

The Raspberry Pi 3 is connected to external digital electronic circuits and devices using its GPIO (General Purpose Input Output) port connector. This is a 2.54mm, 40-pin expansion header, arranged in a 2x20 strip as shown in Figure 2.2.

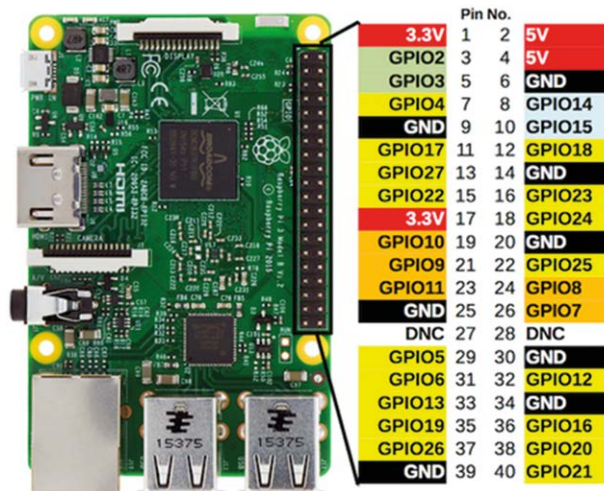


Figure 2.2 Raspberry Pi 3 GPIO pins

When the GPIO connector is at the far side of the board, the pins at the bottom, starting from the left of the connector are numbered as 1, 3, 5, 7, and so on, while the ones at the top are numbered as 2, 4, 6, 8 and so on.

The GPIO provides 26 general purpose bi-directional I/O pins. Some of the pins have multiple functions. For example, pins 3 and 5 are the GPIO2 and GPIO3 input-output pins respectively. These pins can also be used as the I<sup>2</sup>C bus I<sup>2</sup>C SDA and I<sup>2</sup>C SCL pins respectively. Similarly, pins 9,10,11,19 can either be used as general purpose input-output, or as the SPI bus pins. Pins 8 and 10 are reserved for UART serial communication.

Two power outputs are provided: +3.3V and +5.0V. The GPIO pins operate at +3.3V logic levels (not like many other computer circuits that operate with +5V). A pin can either be an input or an output. When configured as an output, the pin voltage is either 0V (logic

0) or +3.3V (logic 1). Raspberry Pi 3 is normally operated using an external power supply (e.g. a mains adapter) with +5V output and minimum 2A current capacity. A 3.3V output pin can supply up to 16mA of current. The total current drawn from all output pins should not exceed the 51mA limit. Care should be taken when connecting external devices to the GPIO pins as drawing excessive currents or short-circuiting a pin can easily damage your Pi. The amount of current that can be supplied by the 5V pin depends on many factors such as the current required by the Pi itself, current taken by the USB peripherals, camera current, HDMI port current, and so on.

When configured as an input, a voltage above +1.7V will be taken as logic 1, and a voltage below +1.7V will be taken as logic 0. Care should be taken not to supply voltages greater than +3.3V to any I/O pin as large voltages can easily damage your Pi. The Raspberry Pi 3, like others in the family, has no over-voltage protection circuitry.

## 2.4 Setting up the Wi-Fi and remote access

It is very likely that you will want to access your Raspberry Pi 3 remotely from your desktop or laptop computer. The easiest option here is to enable Wi-Fi on your Pi computer and then access it from your computer using the SSH client protocol. This protocol requires a server and a client. The server is your Pi computer and the client is your desktop or laptop computer. In this section, we will see how to enable the Wi-Fi on your Pi computer and how to access it remotely.

### Setting up Wi-Fi

To enable the Wi-Fi on your Pi, the steps are as follows:

- Click on the Wi-Fi icon which is a pair of red crosses at the top right-hand side of the screen
- Select your Wi-Fi router from the displayed list (see Figure 2.3)

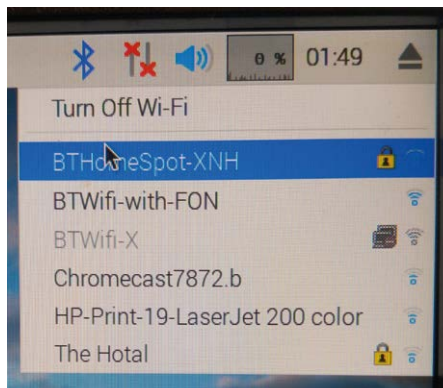


Figure 2.3 Select your Wi-Fi from the list

- Enter the password for your Wi-Fi router

- The WiFi icon should become a typical Wi-Fi image. If you click on the icon now you should see a green tick next to the selected router as shown in Figure 2.4.

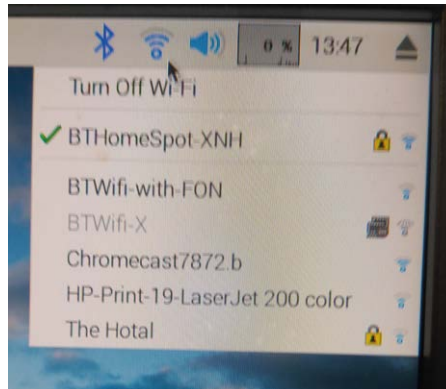


Figure 2.4 Connected to the Wi-Fi successfully

- To see the IP address of your Wi-Fi connection, place the mouse over the Wi-Fi icon as shown in Figure 2.5. In this example, the IP address was 192.168.1.84

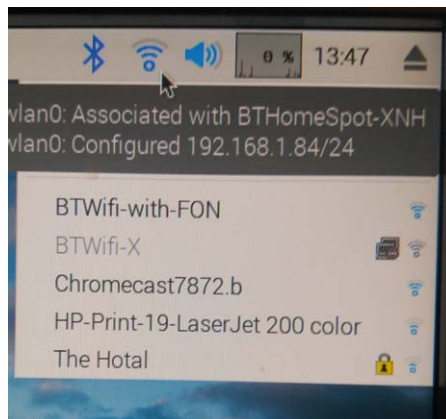


Figure 2.5 IP address of our connection

## Remote Access

The program we will be using to access our Raspberry Pi 3 is called **Putty** with the SSH protocol. The steps to download and use Putty are as follows:

- Download Putty from the following link (or search Google for "Download Putty")

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

- For security reasons, the SSH protocol is disabled by default on a new operating system. To enable it, click on the **Applications** menu at the top left of the screen, click **Accessories**, and then click **Terminal** (see Figure 2.6)

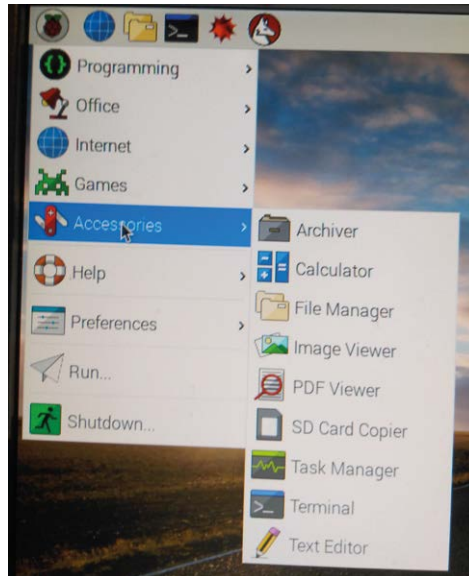


Figure 2.6 Access the Terminal menu

- You should now be in the Raspberry Pi 3 command prompt. Type:

```
sudo raspi-config
```

- to go into the configuration menu and select **Interface Options**. Go down to **P2 SSH** and enable SSH as shown in Figure 2.7

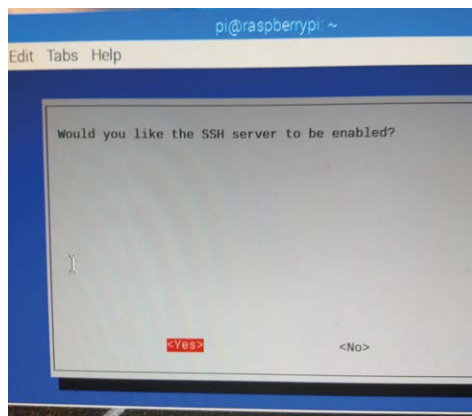


Figure 2.7 Enable the SSH server

- Click **<Finish>** to exit the configuration menu. You should now be back in the command mode, identified by the prompt:

```
pi@raspberrypi:~ $
```

- Putty is a standalone program and there is no need to install it. Simply double click to run it. You should see the Putty startup screen as in Figure 2.8.

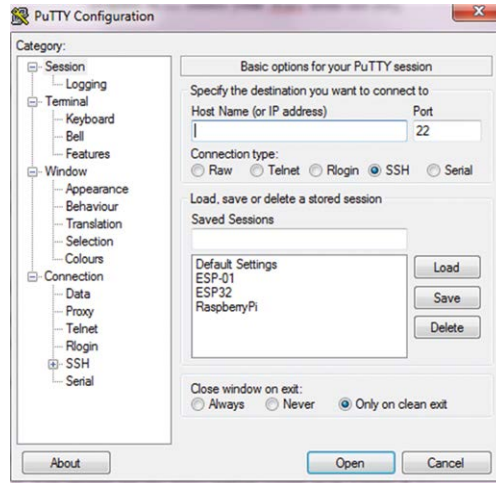


Figure 2.8 Putty startup screen

- Make sure that the Connection type is SSH and enter the IP address of your Raspberry Pi 3. Click Open as shown in Figure 2.9.

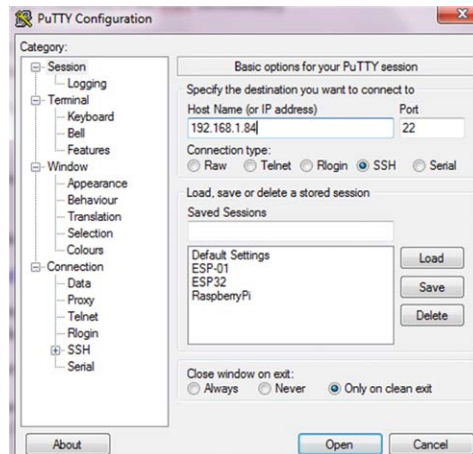


Figure 2.9 Enter the IP address

- The message shown in Figure 2.10 will be displayed on the PC screen the first time you access the Raspberry Pi 3. Click Yes to accept this security alert.



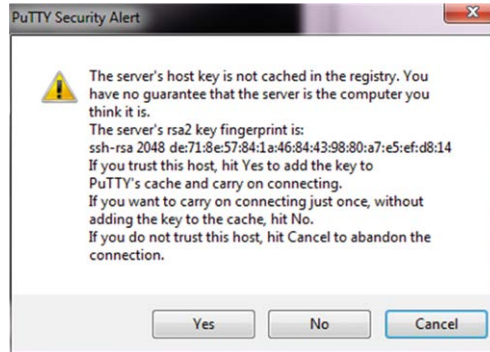


Figure 2.10 Click Yes to accept

- You will then be prompted for the username and password. The default values are:

Username: **pi**

Password: **raspberry**

- After a successful login, you should see the Raspberry Pi command prompt as in Figure 2.11.

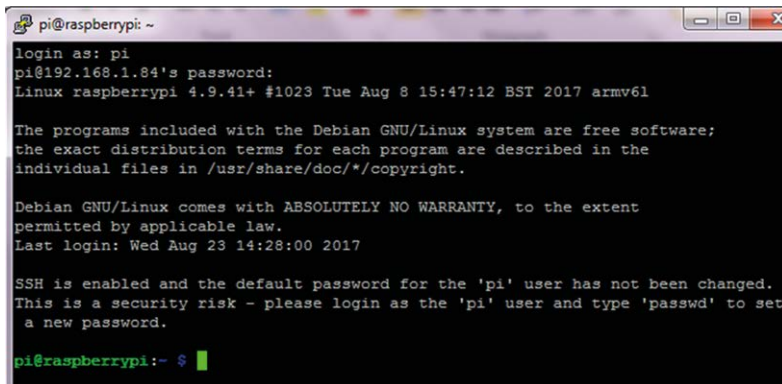


Figure 2.11 Successful login

- To change your password, enter the following command:

`passwd`

- To restart the Raspberry Pi ZW enter the following command:

`sudo reboot`

- To shut down the Raspberry Pi ZW enter the following command. Never shut down by pulling the power cable as this may result in the corruption or loss of files:

```
sudo shutdown -h now
```

## 2.5 Shutting down or rebooting in GUI mod

You must always shut down your Pi computer properly. To shut down while in the GUI mode, follow the steps given below:

- Click Applications menu (top left corner)
- Click Shutdown (see Figure 2.12)
- Click Shutdown (or Reboot as required)

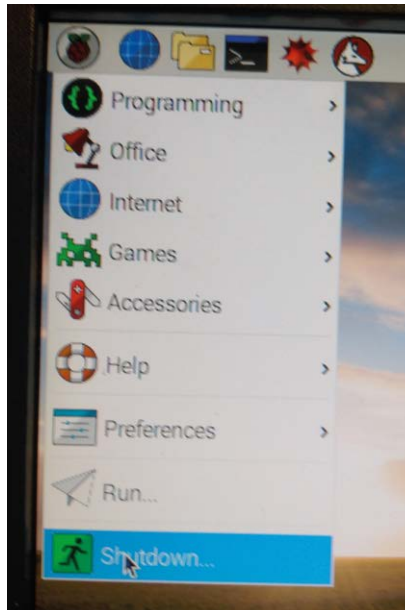


Figure 2.12 Shutdown or reboot in GUI mode

## 2.6 Remote access of the desktop

If you will be using your Raspberry Pi 3 with local keyboard, mouse, and monitor you can skip this section. If on the other hand, you want to access your Desktop remotely over the network, you will find that SSH services cannot be used. The easiest and simplest way to access your Desktop remotely from a computer is by installing the VNC (Virtual Network Connection) client and server. The VNC server runs on your Pi and the VNC client runs on your computer. The steps to install and use the VNC are given below:

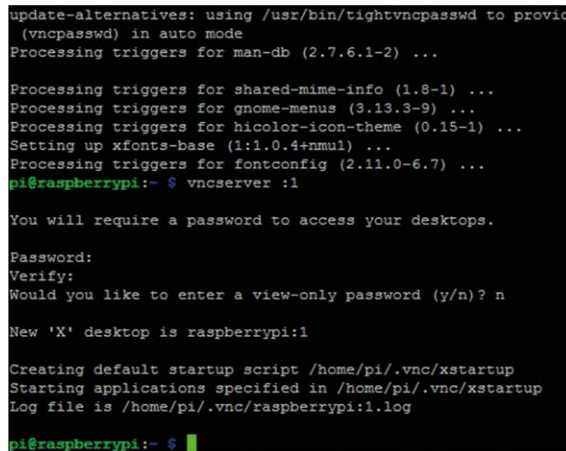
- Connect to your Pi computer using SSH as explained earlier. Then enter the following command to install a program called **TightVNC** server on your Pi computer. You will see many lines of messages. Make sure there are no error messages:

```
sudo apt-get update  
sudo apt-get install tightvncserver
```

- Run the VNC server on your Pi computer by entering the following command:

```
vncserver :1
```

- You will be prompted to enter and verify a password. This will be the password you will be using to access the Desktop remotely (see Figure 2.13).



```
update-alternatives: using /usr/bin/tightvncpasswd to provide
(vncpasswd) in auto mode
Processing triggers for man-db (2.7.6.1-2) ...
Processing triggers for shared-mime-info (1.8-1) ...
Processing triggers for gnome-menus (3.13.3-9) ...
Processing triggers for hicolor-icon-theme (0.15-1) ...
Setting up xfonts-base (1:1.0.4+nmu1) ...
Processing triggers for fontconfig (2.11.0-6.7) ...
pi@raspberrypi:~$ vncserver :1

You will require a password to access your desktops.

Password:
Verify:
Would you like to enter a view-only password (y/n)? n
New 'X' desktop is raspberrypi:1

Creating default startup script /home/pi/.vnc/xstartup
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/raspberrypi:1.log
pi@raspberrypi:~$
```

*Figure 2.13 Enter a password for the VNC server*

- The VNC server is now running on your Pi computer. The only command you need to enter on your Pi computer to start the VNC server is:

```
vncserver :1
```

- We must now set up a VNC client on our laptop (or desktop). There are many VNC clients available, but the recommended one which is compatible with TightVNC is the VNC Viewer, which can be downloaded from the following link. Notice that this program is not free of charge, but a 30-day free trial version is available. You should register to get a trial license and then apply this license to the software to use free of charge for 30 days:

<http://www.realvnc.com>

Download the VNC Viewer program into a suitable directory on your computer.

Double click to install it and enter the required license. Start the **VNC Viewer** program by double-clicking its icon in your desktop. Enter the IP address of your Raspberry Pi 3 followed by **1** as shown in Figure 2.14 and click **Connect**.

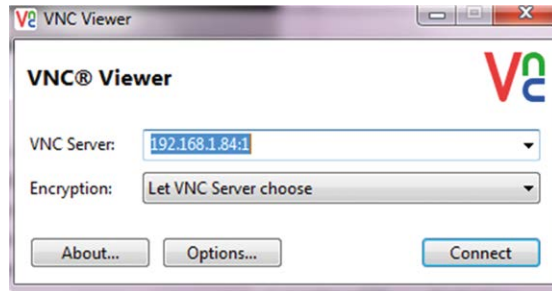


Figure 2.14 Enter the IP address

Enter the password selected previously. You should now see the Raspberry Pi 3 Desktop displayed on your laptop (or desktop) computer as in Figure 2.15 and you can access all of the Desktop applications remotely.

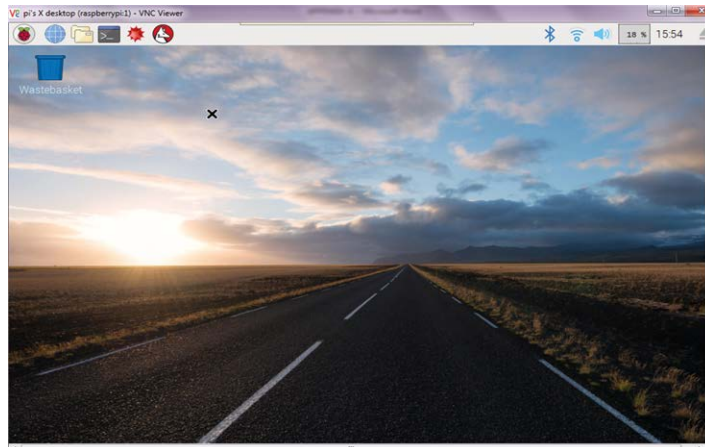


Figure 2.15 Raspberry Pi 3 Desktop displayed on the laptop

## 2.7 Creating and running a Python program

We will be programming our Raspberry Pi 3 using the Python programming language. It is worthwhile to look at the creation and running of a simple Python program on our Pi computer. In this section, we will display the message **Hello From Raspberry Pi 3** on our PC screen.

As described below, there are 3 methods that we can create and run Python programs on our Raspberry Pi 3:

### Method 1 – Interactively from Command Prompt

In this method, we will log in to our Raspberry Pi 3 using the SSH and then create and run our program interactively. This method is excellent for small programs. The steps are as follows:

- Login to the Raspberry Pi 3 using SSH
- At the command prompt enter **python**. You should see the Python command mode which is identified by three characters **>>>**
- Type the program:

```
print ("Hello From Raspberry Pi 3")
```

- The required text will be displayed interactively on the screen as shown in Figure 2.16

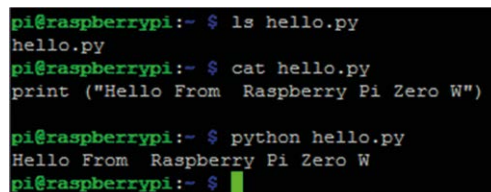
```
pi@raspberrypi:~ $ python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170124] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello From Raspberry Pi 3")
Hello From Raspberry Pi 3
>>> █
```

*Figure 2.16 Running a program interactively*

## Method 2 – Create a Python File in Command Mode

In this method, we will log in to our Raspberry Pi 3 using the SSH as before and then create a Python file. A Python file is simply a text file with the extension **.py**. We can use a text editor, e.g. the **nano** text editor to create our file. In this example, a file called **hello.py** is created using the **nano** text editor. Figure 2.17 shows the contents of file **hello.py**. This figure also shows how to run the file under Python. Notice that the program is run by entering the command:

```
>>> python hello.py
```



```
pi@raspberrypi:~ $ ls hello.py
hello.py
pi@raspberrypi:~ $ cat hello.py
print ("Hello From  Raspberry Pi Zero W")

pi@raspberrypi:~ $ python hello.py
Hello From  Raspberry Pi Zero W
pi@raspberrypi:~ $ █
```

*Figure 2.17 Creating and running a Python file*

## Method 3 – Create a Python File in GUI mode

In this method, we will log in to our Raspberry Pi 3 using the VNC and create and run our program in GUI mode. The steps are given below:

- Click Applications menu
- Click Programming and select Python 2 or Python 3 (see Figure 2.18)

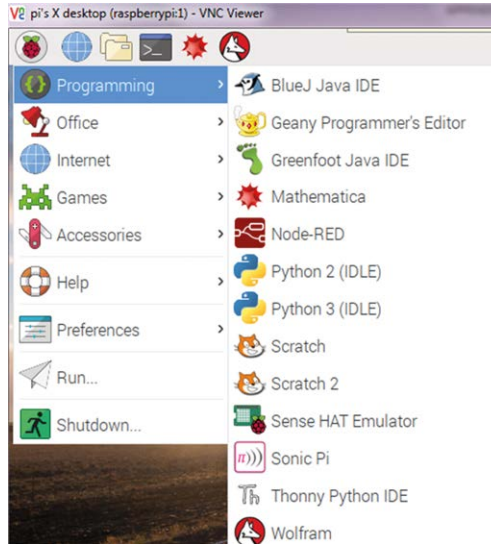


Figure 2.18 Select Python 2 programming

- You should see the Python command mode, identified by characters `>>>`
- Click **File** and then click **New File** and write your program
- Save the file by giving it a name (e.g. hello2)
- Run the program by clicking **Run** and then **Run Module** as shown in Figure 2.19

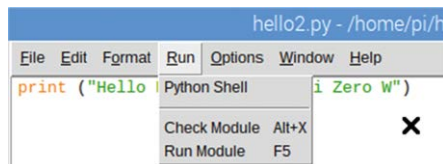


Figure 2.19 Run the program

- A new screen will be shown with the output of the program displayed as in Figure 2.20

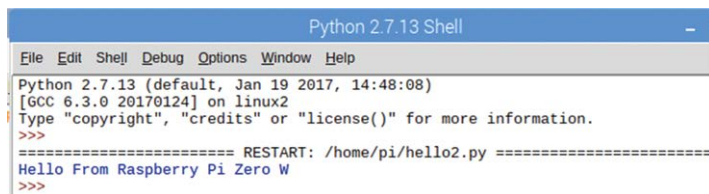


Figure 2.20 Output of the program

### Which Method?

The choice of a method depends upon the size and complexity of a program. Small programs can be run interactively without creating a program file. Larger programs can be created as Python files and then they can run either in the command mode or in the GUI mode. In this book, program files are created for all the Python programs.

## 2.8 The GPIO library

The GPIO library is called `RPi.GPIO` and it should already be installed on your Raspberry Pi 3. This library must be included at the beginning of your Python programs if you will be using the GPIO functions. The statement to include this library is:

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

If you get an error while trying to import the GPIO library then it is possible that the library is not installed. Enter the following commands while in the command mode (identified by the prompt **`pi@raspberrypi:~ $`**) to install the GPIO library (characters that should be entered by you are in bold):

```
pi@raspberrypi: ~ $ sudo apt-get update
pi@raspberrypi: ~ $ sudo apt-get install python-dev
pi@raspberrypi: ~ $ sudo apt-get install python-rpi.gpio
```

The GPIO provides a number of useful functions. The available functions are given in the next sections

### 2.8.1 Pin numbering

There are two ways that we can refer to the GPIO pins. The first is using the BOARD numbering, where the pin numbers on the GPIO connector of the Raspberry Pi 3 are used. Enter the following statement to use the BOARD method:

**`GPIO.setmode(GPIO.BOARD)`**

The second numbering system, also known as the BCM method is the preferred method and it uses the channel numbers allocated to the pins. This method requires that you know which channel number refers to which pin on the board. In this book, we will be using this second method. Enter the following statement to use the BCM method:

**`GPIO.setmode(GPIO.BCM)`**

The GPIO is a 40 pin header, mounted at one side of the board. Appendix A shows the Raspberry Pi 3 GPIO pin configuration.

### 2.8.2 Channel (I/O port pin) configuration

#### Input Configuration

You need to configure the channels (or port pins) you are using whether they are input or output channels. The following statement is used to configure a channel as an input. Here,

channel refers to the channel number based on the **setmode** statement above:

**GPIO.setup(channel, GPIO.IN)**

When there is nothing connected to an input pin, the data at this input is not defined. We can specify additional parameters with the input configuration statement to connect pull-up or pull-down resistors by software to an input pin. The required statements are:

For pull-down:

**GPIO.setup(channel, GPIO.IN, pull\_up\_down=GPIO.PUD\_DOWN)**

For pull-up:

**GPIO.setup(channel, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)**

We can detect an edge change of an input signal at an input pin. Edge change is when the signal changes from LOW to HIGH (rising edge), or from HIGH to LOW (falling edge). For example, pressing a push-button switch can cause an edge change at the input of a pin. The following statements can be used to wait for an edge of the input signal. These are blocking functions. i.e. the program will wait until the specified edge is detected at the input signal. For example, if this is a push-button, the program will wait until the button is pressed:

To wait for a rising edge:

**GPIO.wait\_for\_edge(channel, GPIO.RISING)**

To wait for a falling edge:

**GPIO.wait\_for\_edge(channel, GPIO.FALLING)**

We can also wait until either a rising or a falling edge is detected by using the following statement:

**GPIO.wait\_for\_edge(channel, GPIO.BOTH)**

We can use event detection function with an input pin. This way, we can execute the event detection code whenever an event is detected. Events can be rising edge, falling edge, or change in either edge of the signal. Event detection is usually used in loops where we can check for the event while executing other code.

For example, to add rising event detection to an input pin:

**GPIO.add\_event\_detect(channel, GPIO.RISING)**



We can check whether or not the event occurred by the following statement:

```
If GPIO.event_detected(channel):
```

```
.....  
.....
```

Event detection can be removed by the following statement:

```
GPIO.remove_event_detect(channel)
```

We can also use interrupt facilities (or callbacks) to detect events. Here, the event is handled inside a user function. The main program carries on its usual duties and as soon as the event occurs the program stops whatever it is doing and jumps to the event handling function. For example, the following statement can be used to add interrupt based event handling to our programs on rising edge of an input signal. In this example, the event handling code is the function named **MyHandler**:

```
GPIO.add_event_detect(channel, GPIO.RISING, callback=MyHandler)
```

```
.....  
.....
```

```
def MyHandler(channel):
```

```
.....  
.....
```

We can add more than one interrupt by using the `add_event_callback` function. Here the callback functions are executed sequentially:

```
GPIO.add_event_detect(channel, GPIO.RISING)
```

```
GPIO.add_event_callback(channel, MyHandler1)
```

```
GPIO.add_event_callback(channel, MyHandler2)
```

```
.....  
.....
```

```
def MyHandler1(channel):
```

```
.....  
.....
```

```
def MyHandler2(channel):
```

```
.....  
.....
```

When we use mechanical switches in our projects we get what is known as the switch bouncing problem. This occurs as the contacts of the switch bounce many times until they settle to their final state. Switch bouncing could generate several pulses before it settles down. We can avoid switch bouncing problems in hardware or software. GPIO library pro-

vides a parameter called `bouncetime` that can be used to eliminate the switch bouncing problem. An example use of this parameter is shown below where the switch bounce time is assumed to be 10ms:

```
GPIO.add_event_detect(channel,GPIO=RISING,callback=MyHandler,  
bouncetime=10)
```

We can also use the callback statement to specify the switch bouncing time as@

```
GPIO.add_event_callback(channel, MyHandler, bouncetime=10)
```

To read the state of an input pin we can use the following statement:

```
GPIO.input(channel)
```

### **Output Configuration**

The following statement is used to configure a channel as an output. Here, `channel` refers to the port number based on the **setmode** statement described earlier:

```
GPIO.setup(channel, GPIO.OUT)
```

We can specify a value for an output pin during its setup. For example, we can configure a channel as output and at the same time set its value to logic HIGH (+3.3V):

```
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```

To send data to an output port pin we can use the following statement:

```
GPIO.output(channel, value)
```

Where `value` can be 0 (or `GPIO.LOW`, or `False`), or 1 (or `GPIO.HIGH`, or `True`)

At the end of the program, we should return all the used resources to the operating system. This is done by including the following statement at the end of our program:

```
GPIO.cleanup()
```

## **2.9 Raspberry Pi project development cycle**

The project development cycle, in general, includes both hardware and software development. In simple projects where there is no external hardware used, we can simply use the Raspberry Pi board as it is and only software development is then required. Most projects, however, are more complex and require additional external components, such as LEDs, motors, displays, keypads, etc. The developer then has the tasks of making sure that the hardware is set up correctly and is in full working order before any software development is started. If the hardware is not set up correctly then time will be wasted trying to develop the software. Hardware development may require additional skills such as the familiarity

with interfacing and correctly using various electronic components in microcontroller based systems.

The project development cycle, especially the software development cycle is an iterative process where the programmer may have to go back and keep making changes to the code until the required response is obtained from the system under development. There are various tools that can be helpful during the development of a project. For example, Flow Charts, Program Description Language (PDL), UML, State Machines, and many other tools can be used during the development cycle.

In Raspberry Pi projects in this book, we will be using the Python programming language. Python is a very powerful interactive programming language that is taught as the first programming language in many universities and colleges around the world. Python is available on the Raspberry Pi computers and it supports a large number of libraries that simplify the programming task considerably.

In the remainder of this Chapter, we will design a simple project using the KY-011 two colour LED board as in the previous Chapter.

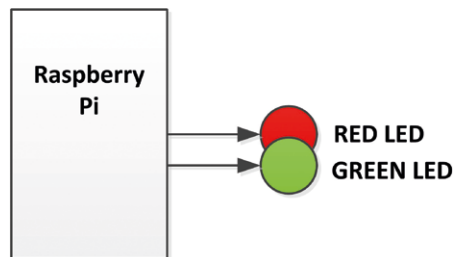
## 2.10 Project – Flashing two colour LEDs

**Description:** This is perhaps the easiest project you can design using your Raspberry Pi. In this project, a two colour LED with common cathode is connected to the Raspberry Pi. The LED colours are flashed alternately at a rate of one second.

**Aim:** The aim of this project is to show how to write and run a program on the Raspberry Pi. The project additionally shows how to use some of the I/O and timing functions of the Raspberry Pi.

**Sensor Used:** Sensor KY-011 is used in this project as in the previous Chapter. The sensor and its pins are shown in Figure 1.3. The sensor has 3 pins: red LED pin, green LED pin, and GND pin, where **the green LED pin is marked with letter S on the board for identification**. An LED is turned ON when logic 1 is applied to its pins.

**Block diagram:** The block diagram of the project is shown in Figure 2.21.



*Figure 2.21 Block diagram of the project*

**Circuit diagram:** The circuit diagram of the project is shown in Figure 2.22. The green and red LED pins are connected to port pin GPIO 2 (pin 3) and GPIO 3 (pin 5) of the Raspberry Pi respectively through current limiting resistors. The GND pin is connected to Raspberry Pi GND pin. The value of the current limiting resistor is calculated as follows:

The output high voltage of a GPIO pin is 3.3V. The voltage across an LED is approximately 2.0V (for red LED this is 1.8V, and for green LED 2.8V). The forward current through the LED depends upon the type of LED used and the amount of required brightness. Assuming that we choose 5mA of forward current, the value of the current limiting resistor is (the LED brightness can be increased by increasing the forward current):

$R = (3.3V - 2V) / 5mA = 260 \text{ ohm}$ . We can choose 220-ohm resistors to give around 6mA forward currents for each LED (for green LED you can use smaller value, e.g. 150 ohms)

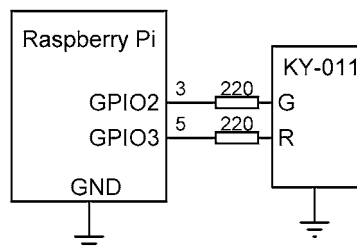


Figure 2.22 Circuit diagram of the project

**Construction:** The project is constructed on a breadboard as shown in Figure 2.23. Jumper wires are used to connect the KY-011 to digital port pins 2 and 3 and GND of Raspberry Pi.

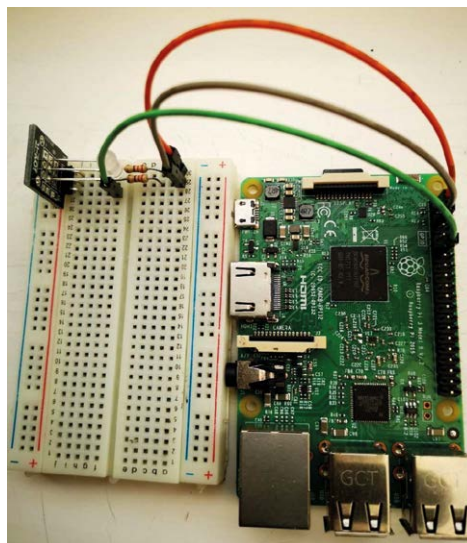


Figure 2.23 Project constructed on a breadboard

**Program listing:** The program is called **rpitwoled.py** and the listing is shown in Figure 2.24. At the beginning of the program, libraries **RPi.GPIO** and **time** are imported into the program. Green and red LED pins are assigned to port GPIO port numbers 2 and 3 respectively. The LEDs are then configured as outputs. The remainder of the program runs in an endless loop. Inside this loop, the green and the red LEDs are flashed alternately with a one-second delay between each output.

You can run the program from the command line by entering the following command (characters entered by the user are in bold for clarity):

```
pi@raspberrypi:~ $ python rpitwoled.py

#-----
#   TWO COLOUR LED FLASHING
#   =====
#
# In this project the two colour LED KY-011 is used. The
# green and red LED pins are connected to Raspberry Pi port
# pins GPIO2 and GPIO3. The LEDs flash alternately with a
# rate of one second
#
# Author: Dogan Ibrahim
# Date  : April 2019
# File  : rpitwoled.py
#-----

import RPi.GPIO as GPIO      # import Rpi library
import time                  # import time library
GPIO.setwarnings(False)     # disable warnings
GPIO.setmode(GPIO.BCM)      # set BCM mode

GreenLED = 2                 # green LED
RedLED = 3                   # red LED
ON = 1
OFF = 0

GPIO.setup(GreenLED, GPIO.OUT) # Configure as output
GPIO.setup(RedLED, GPIO.OUT)  # configure as output

while True:                  # do forever
    GPIO.output(GreenLED, ON) # green LED ON
    GPIO.output(RedLED, OFF)  # red LED OFF
    time.sleep(1)             # wait 1 second
    GPIO.output(GreenLED, OFF) # green LED ON
    GPIO.output(RedLED, ON)   # red LED ON
    time.sleep(1)             # wait 1 seconds
```

*Figure 2.24 Program listing of the project*

**Note:** You may find it easier to create and run your Python programs from the GUI desktop interface (IDLE 2) since the correct indentation is automatically placed in your code as you type the code.

**What we have learned:** In this project, we have learned how to use the following Raspberry Pi functions:

<b>import:</b>	import a library module to the program
<b>GPIO.setmode(mode):</b>	set the GPIO pin usage mode in the program
<b>GPIO.setwarnings(mode):</b>	disable (False) or enable (True) warning messages
<b>GPIO.setup(port pin, mode):</b>	set the pin mode as either input or output
<b>GPIO.output(port pin, value):</b>	send value (0 or 1) to specified port pin
<b>time.sleep(n):</b>	create a delay of <b>n</b> seconds

### 2.11 Summary

In this chapter we have looked at the pin configuration of the GPIO connector and how to enable the Wi-Fi module and access the Raspberry Pi remotely from a desktop computer or a laptop. Additionally, we have looked at the design of a simple KY-011 two colour LED-based project using the Python programming language.

In the next Chapter, we will be looking at how to install the firmware on the ESP32 micro-controller and how to use it in projects.

## Index

### A

ADC	85
ADS1115,	86
API key,	294
Arduino Uno pin layout	14

### B

BC108,	137
BCM43438	21
BiColor library	55
Bi-phase coding	115
Bluetooth	284, 297, 307, 317
Bluetooth controller	320
Bluetooth MAC address	300
Board manager	43
Boot	41, 46

### C

Cadmium selenide	82
Cloud	291
Commercial IR handset	116
Config.txt	232
CSV file	246, 258

### D

Dallas temperature library	230
Dark sensor	85
Data export	296
Decoding IR	114
DHT-11	271
DHT-22	271
DS18B20	229
Duty Cycle	52

### E

Echo	156
ESP-01	308
ESP32 analog channels	93
ESP32 DevKitC	40
ESP32 DevKitC pin layout	42
Excel form	250

### F

Flame sensor	188
--------------	-----

### H

Hall effect	97
Hall sensor	181
HC-06	317
Hciconfig	299
HC-SR04	156
HD44780	241
HDMI	20
Heartbeat sensor	225
Humidity sensor	228

### I

I <sup>2</sup> C	89
IoRef	13
IR filter	201
IR handset	114
IR projects	114
Irsend	139

### J

Joystick	192
Joystick coordinates	194

### K

Key code	102
KY-001	228
KY-002	146, 150
KY-003	96, 180
KY-004	141
KY-005	136, 141
KY-006	172, 186
KY-009	62
KY-010	100
KY-011	16
KY-012	100, 141, 158, 204
KY-013	239, 255
KY-015	270, 288, 291
KY-016	73
KY-017	73
KY-018	81, 94
KY-019	82, 94, 146, 150, 168
KY-021	75, 96, 180
KY-022	114, 131, 141
KY-017	108
KY-023	192
KY-024	180

KY-025	184	<b>P</b>	
KY-026	188	Parallel LCD	240
KY-028	236	Parking	156
KY-029	131, 297	Photo interrupter	101
KY-031	147, 153	Piezzo buzzer	172
KY-032	201	Playing melody	172
KY-033	207, 211	PowerIn	13
KY-034	76	Pressure sensor	228
KY-035	185	Pulse distance code	115
KY-036	204	Pulse length code	115
KY-037	168	Putty	23
KY-039	225	PWM	51
KY-040	217	PyIrc	126
KY-050	156, 158		
KY-051	85, 94	<b>R</b>	
KY-052	260	Random	71
KY-053	85, 255	Raspberry Pi pin layout	21
		Raspi-config	89
<b>L</b>		RC5	116
LCD	239	RealTerm	249
LDR	82	Reed switch	76
Light projects	51	Remote access	23
Line tracking	207	Remote control	279
LM393	236	RGB LED	62, 69
		Rotary encoder	217
<b>M</b>		Rotary encoder direction	220
Magic light cup	108	Rotary encoder position	220
Magic wand	67	ROYGBIV	63
Magnetic sensor	180		
Matplotlib	234	<b>S</b>	
Metal touch sensor	204	Scanning TV channels	136
Millis	154	SCLK	261
MISO	261	Secret code	211
MOSI	261	Secret key	100
Musical instrument	185	Serial Bluetooth	286
		Serial LCD	240
<b>N</b>		Serial monitor	231
NEC code	116	Serial plotter	226, 232
		Shaft encoder	218
<b>O</b>		Sketch	15
Obstacle sensor	201	Sound projects	168
One-wire	230	Speed of sound	157
On-off control	260	SPI bus	261
		SS	261
		SSH	24
		Steinhart-Hart coefficients	239



## **T**

Target shooting	146
Temperature sensor	228
Thermistor	237
Thingspeak	291
TightVNC	27
Tilt switch	68
Timestamp	258
Tone	174
Tracking sensor	207

## **U**

UDP	271, 290
UDP server	312
Ultrasonic projects	156

## **V**

Vibration and shock projects	146
Vncserver	28

## **W**

Wi-Fi	270, 288, 307
Wi-Fi watch	313



**Ibrahim** has a BSc. in Electronic Engineering, an MSc. in Automatic Control Engineering, and a Ph.D. in Digital Signal Processing. He worked in many industrial organisations before returning to academia. Prof. Ibrahim is the author of over 60 technical books and over 200 technical articles on microcontrollers, microprocessors, and related fields. He is a Chartered Electrical Engineer and a Fellow of the Institution of Engineering Technology.



9 781907 920783



- Changing LED brightness
- RGB LEDs
- Creating rainbow colours
- Magic wand
- Silent door alarm
- Dark sensor with relay
- Secret key
- Magic light cup
- Decoding commercial IR handsets
- Controlling TV channels with IR sensors
- Target shooting detector
- Shock time duration measurement
- Ultrasonic reverse parking
- Toggle lights by clapping hands
- Playing melody
- Measuring magnetic field strength
- Joystick musical instrument
- Line tracking
- Displaying temperature
- Temperature ON/OFF control
- Mobile phone-based Wi-Fi projects
- Mobile phone-based Bluetooth projects
- Sending data to the Cloud

The projects have been organized with increasing levels of difficulty. Readers are encouraged to tackle the projects in the order given. A specially prepared sensor kit is available from Elektor. With the help of this hardware, it should be easy and fun to build the projects in this book.