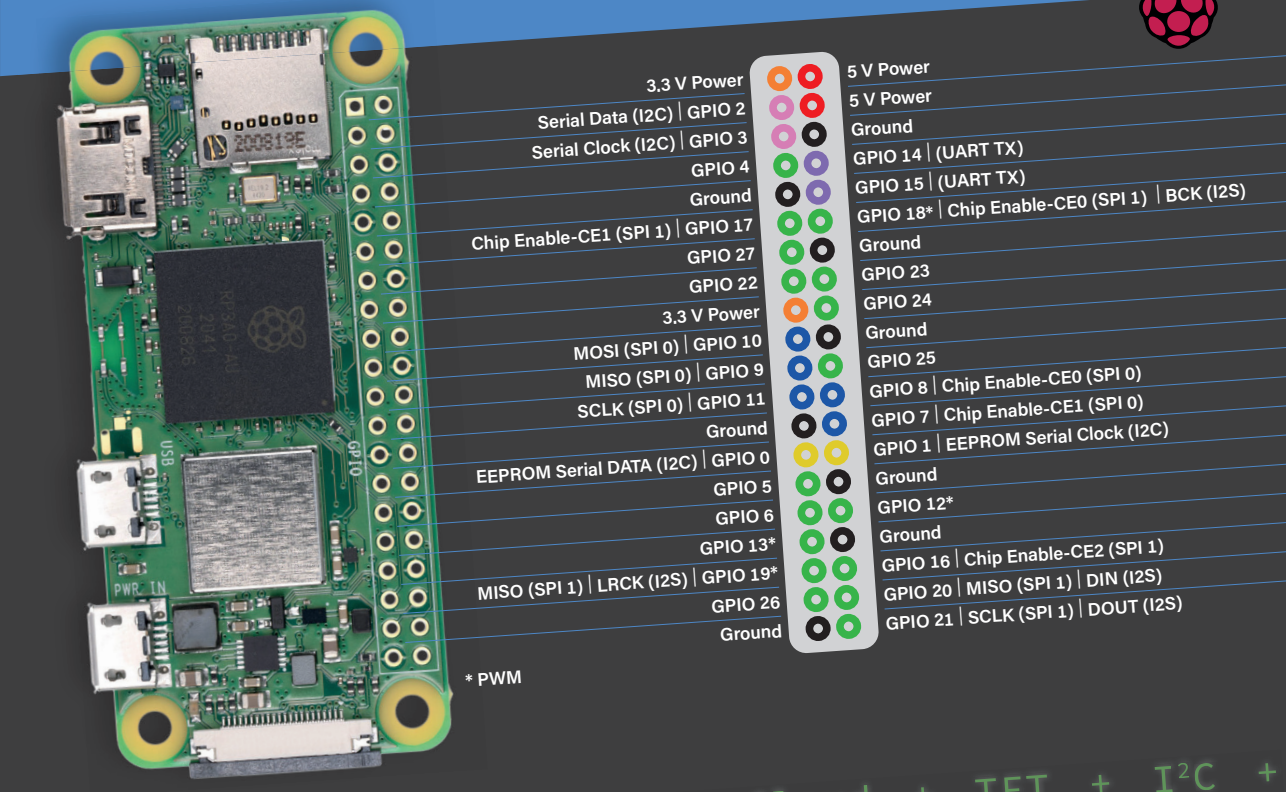
elektor books

The Raspberry Pi Zero 2 W *GO!* Book

A Fast-Lane Ride From Concept to Project



+ UDP/TCP + WiFi + Flask Cloud + TFT + I²C +
LCD + Arduino + Sensor HAT + Node-RED + Alexa
+ IoT + MQTT + Bluetooth + Motors + Python +

Dogan Ibrahim

elektor
design > share > earn

The Raspberry Pi Zero 2 W GO! Book

A Fast-Lane Ride From Concept to Project

Dogan Ibrahim

elektor
design › share › earn

This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
PO Box 11, NL-6114-ZG Susteren, The Netherlands
Phone: +31 46 4389444

Declaration

The author, editor, and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, and 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. All the programs given in the book are Copyright of the Author and Elektor International Media. These programs may only be used for educational purposes. Written permission from the Author or Elektor must be obtained before any of these programs can be used for commercial purposes.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 978-3-89576-549-0 Print
ISBN 978-3-89576-550-6 eBook

Editor: Jan Buiting, MA
Prepress Production: D-Vision, Julian van den Berg

Elektor is 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**

Contents

Preface

The Raspberry Pi Zero 2 W is the latest member of the Raspberry Pi family of credit card sized computers that can be used in many applications including IoT, digital audio and video media centers, as a desktop computer, in industrial controllers, robotics, and in countless domestic, commercial, and industrial applications. In addition to many features found in "bigger" family members, the Raspberry Pi Zero 2 W also offers Wi-Fi and Bluetooth capability which makes it highly desirable in remote and Internet-based control and monitoring applications. The Raspberry Pi Zero 2 W replaces the older Raspberry Pi Zero W, provides 5 times more computing power, and in addition features Bluetooth capability. As a bonus feature, the Raspberry Pi Zero 2 W comes with a colored pinheader connector, making it easy to link hardware to its GPIO pins.

The book kicks off with the installation of the latest Raspberry Pi operating system on an SD card, followed by the configuration and practical use of the GPIOs. Next, the core of the book explains the use of the Raspberry Pi Zero 2 W running the Python programming language, always in simple terms and backed by many tested and working example projects. On part of the reader, familiarity with the Python programming language and some experience with one of the Raspberry Pi computers will prove helpful. Although previous electronics experience is not required, some knowledge of basic electronics is beneficial, especially when venturing out to modify the projects for your own applications.

The book includes many simple, software-only examples to widen your knowledge of Python. Besides these, over 30 tested and working hardware-based projects are given in the book, covering the use of Wi-Fi, communication with smartphones and with a Raspberry Pi Pico W computer. Additionally, there are Bluetooth projects including elementary communication with smartphones and with the popular Arduino Uno.

All the projects given in the book have been fully tested and are working. The following sub-headings are used in the projects where applicable:

- Project title
- Project description
- Aim of the project
- Block diagram
- Circuit diagram
- Program description using PDL
- Program listing with full documentation

All programs discussed in this book are contained in an archive file you can download free of charge from the Elektor website. Head to: www.elektor.com/books and enter the book title in the Search box. You can easily download and use these programs without any modifications. Alternatively, feel free to modify the programs to suit your own applications.

I hope you find the book helpful and enjoy reading it.

Prof Dogan Ibrahim
London, January 2023

Chapter 1 • The Raspberry Pi Zero 2 W

1.1 Overview

The Raspberry Pi Zero 2 W is an immensely popular member of the Raspberry Pi family of development boards for embedded microcontroller applications. In this chapter, you"ll discover the hardware details of this development board and also learn how to install the latest operating system on an SD card for use with the Raspberry Pi Zero 2 W. From now on, this development board will be referred to as the *Zero 2 W* for short.

1.2 The Zero 2 W development board

The Zero 2 W (Figure 1.1) is essentially an upgraded replacement to the popular Raspberry Pi Zero W. It features an RP3A0 system-in-package (SiP), which is a quad-core 64-bit Arm Cortex-A53 CPU clocked at 1 GHz. A Broadcom BCM2710A1 die is incorporated with 512 MB of SDRAM (LPDDR2). The processor provides 40% more single-threaded performance and 5 times more multi-threaded performance than the standard Raspberry Pi Zero.

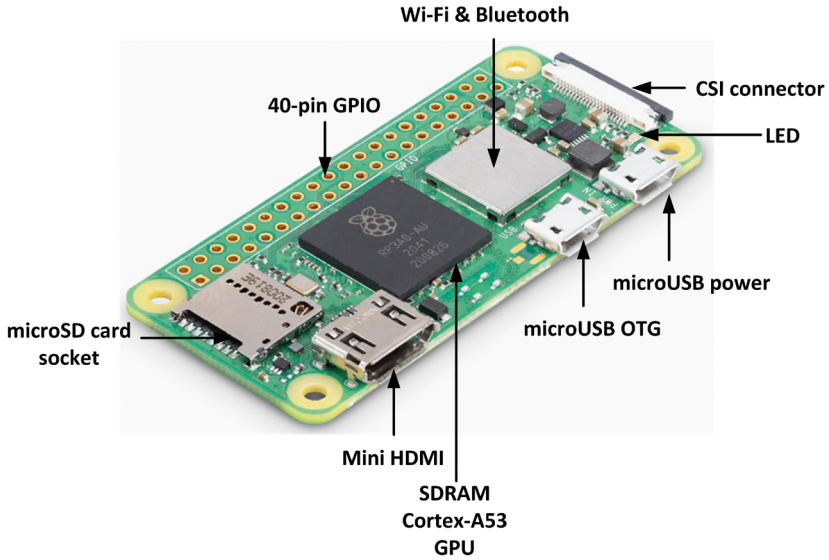


Figure 1.1: Raspberry Pi Zero 2 W.

The basic features of the Zero 2 W may be summarized as follows.

- 64-bit Arm Cortex-A53 processor @ 1 GHz
- Broadcom BCM2710A1 SiP
- 512 MB SDRAM
- 2.4-GHz IEEE 802.11b/g/n wireless LAN
- Bluetooth 4.2 BLE
- On-board antenna
- microSD card slot
- Mini HDMI and Composite video ports

- CSI-2 camera connector
- 1× MicroUSB OTG
- 1× MicroUSB power input
- H.264 MPEG-4 decode (1080p30)
- OpenGL ES 1.1, 2.0 graphics
- HAT compatible 40-pin I/O header
- 5 VDC, 2.5 A input power
- Size 30 mm × 65mm × 13mm
- Weight: 39.97 grams
- Temperature range: –20º to +70º (operation)
- Stated production lifetime: until 2028

The major advantage of the Zero 2 W is that the board incorporates both Wi-Fi and Bluetooth interface modules along with Bluetooth Low Energy (BLE). The board shares the same form factor as the original Raspberry Pi Zero and therefore fits inside existing Raspberry Pi Zero cases. The board is powered externally with 5 V, 2.5 A via a MicroUSB socket.

The Zero 2 W board is available with pre-soldered male color-coded header (**ZERO 2 WHC**) for ease of identifying the pins. This is shown in Figure 1.2. Thick internal copper layers are used to conduct heat away from the processor and consequently achieve higher performance without heating the processor.

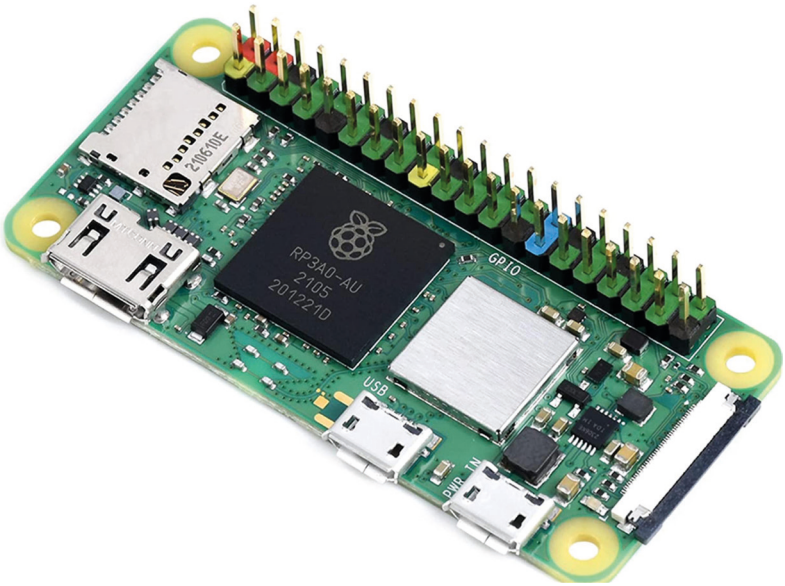


Figure 1.2: Color-coded pinheader.

Table 1.1 shows a comparison of the Zero 2 W with other Raspberry Pi Zero development boards. Notice that the Zero 2 W is a 64-bit device and about 5 times faster than the original Zero W.

	ZERO	ZERO W	ZERO 2 W	ZERO 2WH	ZERO 2 WHC
Processor	BCM2835		BCM2710A1		
CPU	32-bit Single core Arm11		64-bit Quad core Arm Cortex-A53		
Memory	512 MB SDRAM				
GPU	OpenGL ES 1.1, 2.0				
WIFI		2.4 GHz IEEE 802.11b/g/n			
Bluetooth		Bluetooth 4.1 BLE	Bluetooth 4.2 BLE		
USB					
Camera	CSI-2 connector				
GPIO	40-pin				
Pre-soldered header				Black	Colored

Table 1.1: Raspberry Pi Zero W and Zero 2 W comparison.

Figure 1.3 shows the pin configuration of the Zero 2 W board. In this book, the use of a Zero 2 WHC with color-coded header is assumed. The following colors are used on the header (Figure 1.4):

- Red = 5 V
- Orange = 3.3 V
- Black = GND
- Pink = I^2C
- Purple = UART
- BLUE = SPI
- Yellow = DNC (reversed I^2C)
- Green = GPIO

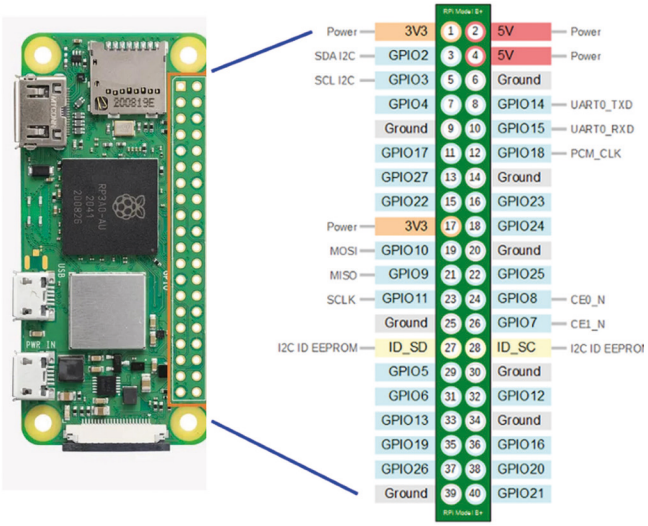


Figure 1.3: Pin configuration.

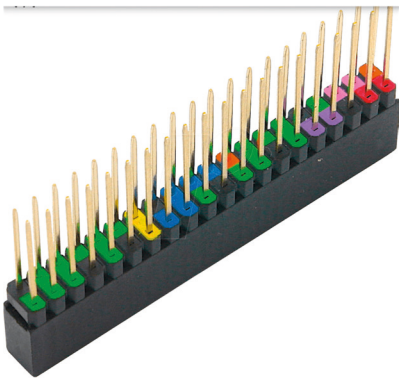


Figure 1.4: Colored header.

1.2.1 Zero 2 W current consumption, and GPIO voltages and currents

The Zero 2 W has 4 cores running at 1 GHz. In idle mode, the Zero 2 W's temperature is about 36.5 degrees Celsius and about 63.4 degrees Celsius when under heavy computing loads. Adding a heatsink helps to reduce the chip temperature. Additional cooling from a heatsink can reduce the temperature considerably.

The Zero 2 W operates at +3.3 V and **the GPIO pins are not +5 V tolerant**. It is therefore important not to connect any peripherals to Zero 2 W that operate at supply voltages exceeding +3.3 V. A voltage between +1.8 V and +3.3 V is read as logic High by the processor. A voltage lower than +1.8 V is read as logic Low.

The Zero 2 W draws about 280 mA when idle and 580 mA when under a heavy computing load. The current limit of a GPIO pin is about 17 mA and more than 51 mA should not be drawn from all GIO pins at any time.

The GPIO pins can be configured as inputs or outputs. By default, they are configured as inputs on power-up, except GPIO 14 and GPIO 15.

1.3 Installing the operating system

The Raspberry Pi OS (called Raspbian previously) is an officially supported operating system for Raspberry Pi systems. It is a 32-bit Linux based OS that has been configured and optimized to run on Raspberry Pi systems.

The Raspberry Pi operating system is normally stored on an SD card although it is also possible to store it on other media such as an SSD drive. The Raspberry Pi operating system is complimentary and can be downloaded to a PC and then transferred to an SD card that can be used to boot the Raspberry Pi development boards.

The steps to download and transfer the operating system to an SD card are given below. Please note that these are the steps that apply if you are using a Windows PC.

- Go to the Raspberry Pi operating system website (Figure 1.5):

  https://www.raspberrypi.com/software/

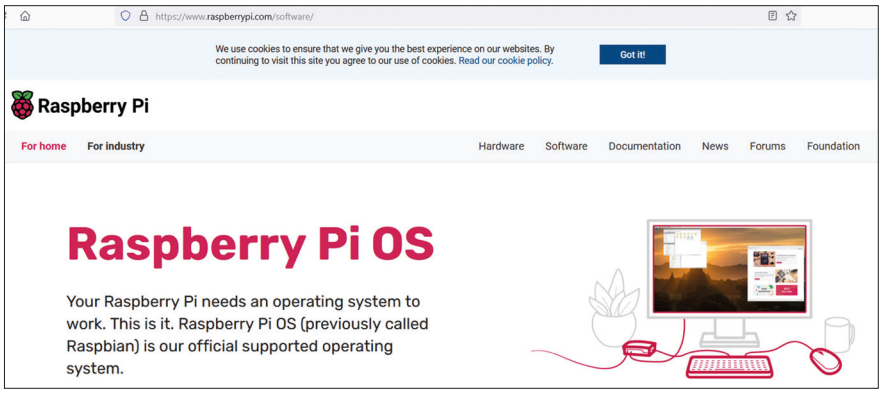


Figure 1.5: Raspberry Pi OS website.

- Scroll down and click **Download for Windows**.

- You should see a file with the name **imager_1.7.3.exe** downloaded to your PC (this was the filename at the time of authoring this book. Your filename may be different).

- Insert the micro SD card into the card holder of your PC (you may have to use an adapter).

- Double-click on the file to install it on your PC.

- Click **Finish** to run the Raspberry Pi Imager (Figure 1.6).

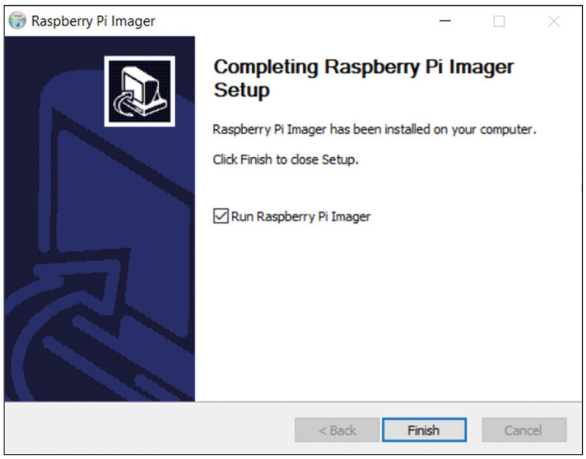


Figure 1.6: Run the Raspberry Pi Imager.

- Click **CHOOSE OS** and click to choose the **Raspberry Pi OS (32-bit)** as shown in Figure 1.7.

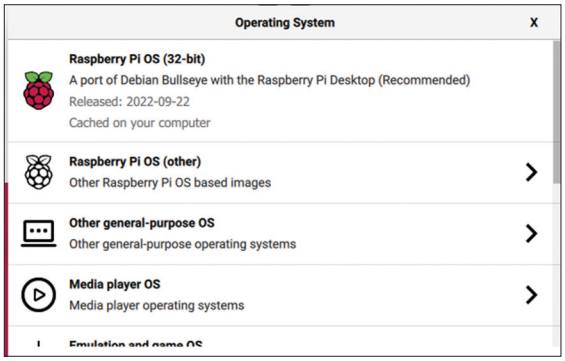


Figure 1.7: Choose the Raspberry Pi OS.

- Click **CHOOSE STORAGE** and select the SD card (Figure 1.8) as the storage medium. A 32-GB micro SD card was used for the examples in this book.

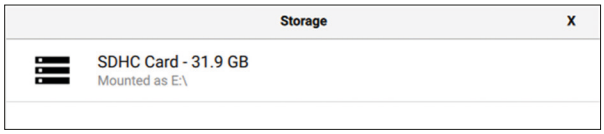


Figure 1.8: Select the SD card.

- Click the Settings icon (gear-shaped) at the bottom right of the screen and set the following (Figure 1.9):

  Set hostname: raspberrypi
  Enable SSH
  Select password authentication
  Set Wi-Fi username and password
  Click configure wireless LAN and set the SSID and password
  Set the wireless LAN country: GB
  Set local settings if desired

- Click **Save** to save the settings.

- Click **Write** to write the operating system on the micro SD card. Wait until the write and verify operations are finished (Figure 1.10).

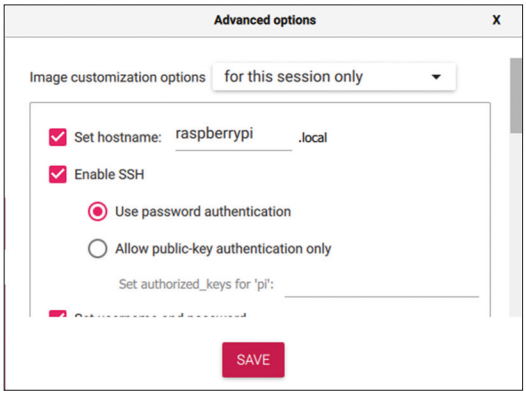


Figure 1.9: Settings options.

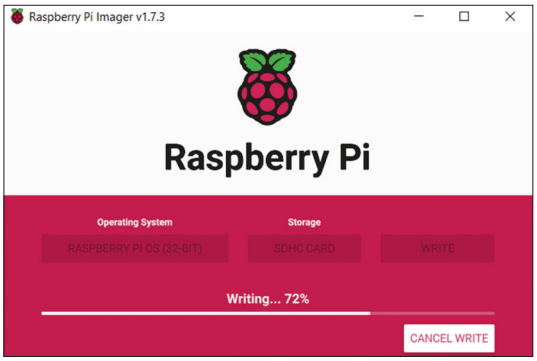


Figure 1.10: Copying the operating system to the micro SD card.

- Remove the micro SD card from the PC.

1.4 Powering-up the Zero 2 W

Insert the SD card into your Zero 2 W. Plug in a monitor and a keyboard if you have these. Plug in the MicroUSB power cable. Your Zero 2 W should boot. If you do not have a monitor and a keyboard you can access your Zero 2 W remotely as described in the next section.

1.5 Remote access

It is much easier to access the Raspberry Pi remotely over the Internet, for example using a PC rather than connecting a keyboard, mouse, and monitor to it. You can use the **Putty** program to access your Zero 2 W remotely from your PC. Before connecting to your Zero 2 W you have to know its IP address. This can be obtained from your Wi-Fi router, or by using an app on your smartphone. There are many freely available apps for both Android and iOS operating systems (e.g., **Who's on my Wi-Fi** for Android) that will scan the Wi-Fi network and display the IP addresses of the devices connected to the Wi-Fi router.

Copy **Putty** to your PC from the following website:

https://www.putty.org/

Putty is a standalone program and there is no need to install it. Simply double-click to run it and the **Putty** startup screen will be displayed. Click **SSH** and enter the Raspberry Pi IP address, then click **Open** (see Figure 1.11). The message shown in Figure 1.12 will be displayed the first time you access the Raspberry Pi. Click **Yes** to accept this security alert.

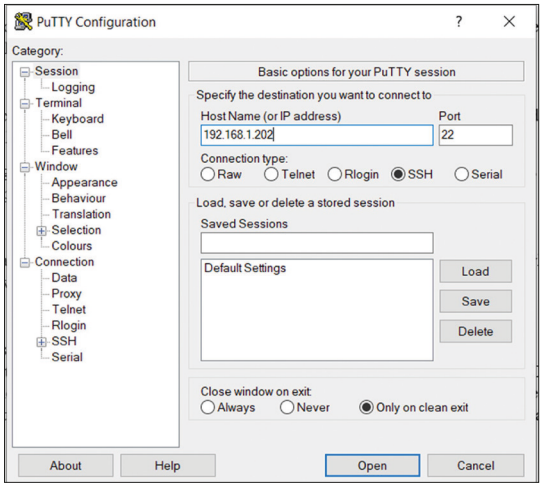


Figure 1.11: Putty startup screen.

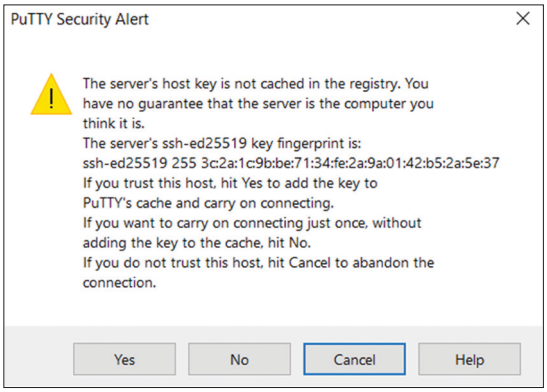


Figure 1.12 Click Yes to accept.

You will be prompted to enter the username and password. After a successful login, you should see the command mode as follows:

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

You should now enable **VNC** so that your Zero 2 W can be accessed graphically over the Internet using your PC. This can be done by entering the following command at the terminal session:

```
pi$raspberrypi:~ $ sudo raspi-config
```

Go to the configuration menu and select **Interface Options**. Go down to **P3 VNC** and enable VNC. Click **<Finish>** to exit the menu.

1.5.1 Configuring Putty

By default, the **Putty** screen background is black with white foreground characters. The author prefers to have white background with black foreground characters, with the character size set to 12 points bold. You should save your settings so that they are available next time you want to use the Putty. The steps to configure the Putty with these settings are given below.

- Restart Putty.
- Select **SSH** and enter the Raspberry Pi IP address.
- Click **Colors** under **Window**.
- Set the **Default Foreground** and **Default Bold Foreground** colors to black (Red:0, Green:0, Blue:0).
- Set the **Default Background** and **Default Bold Background** to white (Red:255, Green:255, Blue:255).
- Set the **Cursor Text** and **Cursor Color** to black (Red:0, Green:0, Blue:0).
- Select **Appearance** under **Window** and click **Change** in **Font settings**. Set the font to **Bold 12**.
- Select **Session** and give a name to the session (e.g., MyZero) and click **Save**.
- Click **Open** to open the **Putty** session with the saved configuration.
- Next time you restart **Putty**, select the saved session, and click **Load** followed by **Open** to start a session with the saved configuration.

1.5.2 Remote access of the Desktop

If you are using your Zero 2 W with a local keyboard, mouse, and display 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. It is recommended to use the **tightvncserver** on your Zero 2 W. The steps are:

- Enter the following command:

```
pi$raspberrypi:~ $ sudo apt-get install tightvncserver
```

- Run the **tightvncserver**:

  pi$raspberrypi:~ $ **tightvncserver**

You will be prompted to create a password for remotely accessing the Raspberry Pi desktop. You can also set up an optional read-only password. The password should be entered every time you want to access the Desktop. Enter a password and remember it.

- Start the VNC server after reboot by the following command:

  pi$raspberrypi:~ $ **vncserver :1**

  You can optionally specify the screen pixel size and color depth in bits as follows:

  pi$raspberrypi:~ $ **vncserver :1 –geometry 1920x1080 –depth 24**

- You should now set up a VNC viewer on your laptop (or desktop) PC. There are many VNC clients available but the recommended one which is compatible with TightVNC is the **TightVNC for the PC** which can be downloaded from the following link:

  https://www.tightvnc.com/download.php

- Download and install the **TightVNC** software for your PC. You will have to choose a password during the installation.

- Start the **TightVNC Viewer** on your PC and enter the Raspberry Pi IP address followed by :1. Click **Connect** to connect to your Raspberry Pi (Figure 1.13).

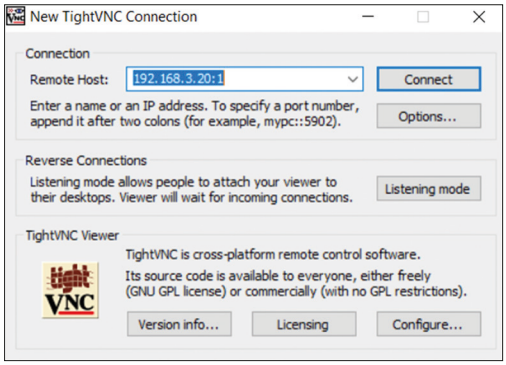


Figure 1.13: Connect to TightVNC Viewer.

- Enter the password you have chosen earlier. You should now see the Raspberry Pi Desktop displayed on your PC screen (Figure 1.14).

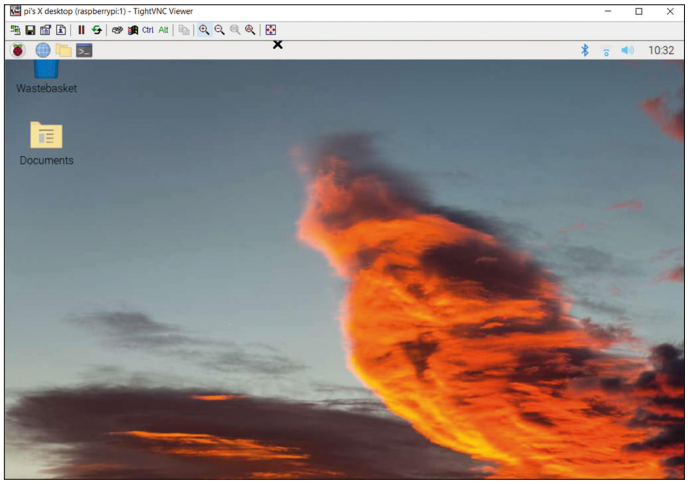


Figure 1.14: Raspberry Pi Desktop.

- The VNC server is now running on your Raspberry Pi Zero 2 W and you have access to the Desktop.

1.6 Assigning static IP address to your Zero 2 W

When you try to access your Raspberry Pi remotely over your local network, it is possible that the IP address given by your Wi-Fi router changes from time to time. This is annoying as you have to find out the new IP address allocated to your Raspberry Pi. Without a knowledge of the IP address, you cannot login using the SSH or the VNC.

In this section, you will learn how to fix your IP address so that it does not change between reboots. The steps are as follows:

- Log in to your Raspberry Pi via SSH.
- Check whether DHCP is active on your Raspberry Pi (it should normally be active):

  ```
  pi@raspberrypi:~ $ sudo service dhcpcd status
  ```

  If DHCP is not active, activate it by entering the following commands:

  ```
  pi@raspberrypi:~ $  sudo service dhcpcd start
  pi@raspberrypi:~ $  sudo systemctl enable dhcpcd
  ```

- Find the IP address currently allocated to you by entering command **ifconfig** or **hostname – I** (Figure 1.15). In this example the IP address was: 192.168.3.20. You can use this IP address as your fixed address since no other device on the network is currently using it.

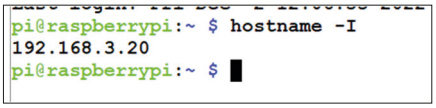


Figure 1.15: Finding the IP address allocated to you.

- Find the IP address of your router by entering the command **ip r** (Figure 1.16). In this example, the IP address was: 192.168.3.1.

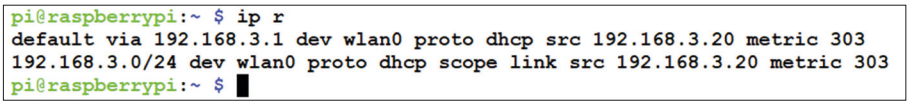


Figure 1.16 Finding the IP address of your router.

- Find the IP address of your DNS by entering the following command (Figure 1.17). This is usually same as your router address:

pi@raspberrypi:~ $ **grep "nameserver" /etc/resolv.conf**

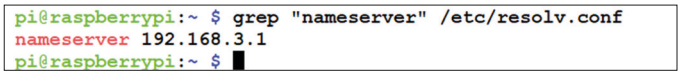


Figure 1.17: Finding the DNS address.

- Edit file **/etc/dhcpcd.conf** by entering the command:

pi@raspberrypi:~ $ **nano /etc/dhcpcd.conf**

- Add the following lines to the bottom of the file (these will be different for your router). If these lines already exist, remove the comment character **#** at the beginning of the lines and change the lines as follows (you may notice that **eth0** for Ethernet is listed):

```
interface wlan0
static_routers=192.168.3.1
static domain_name_servers=192.168.3.1
static ip_address=192.168.3.20/24
```

- Save the file by entering **CNTRL + X** followed by **Y** and reboot your Raspberry Pi.

- In this example, the Raspberry Pi should reboot with the static IP address: 192.168.3.20.

Chapter 2 • Using a Text Editor in Command Mode

A text editor is used to create or modify the contents of a text file. There are many text editors around for the Linux operating system. Some popular ones go by the names of **nano**, **vim**, and **vi**. In this chapter, you will learn to use "**nano**" which is the most commonly used text editor in Linux. In later chapters, you will see that the **nano** text editor can be used to create your Python programs.

2.1 The "nano" text editor

Start the **nano** text editor by entering the word **nano**, followed by the filename you wish to create or modify. An example is given below where a new file called **first.txt** is created:

```
pi@raspberrypi ~ $ nano first.txt
```

You should see the editor screen as in Figure 2.1. The name of the file to be edited is written at the top middle part of the screen. The message **New File** at the bottom of the screen shows that this is a newly created file. The shortcuts at the bottom of the screen are there to perform various editing functions. These shortcuts are accessed by pressing the Ctrl key together with another key. Some of the useful shortcuts are given below.

Ctrl+W: Search for a word

Ctrl+V: Move to next page

Ctrl+Y: Move to previous page

Ctrl+K: Cut the current row of txt

Ctrl+R: Read file

Ctrl+U: Paste the text you previously cut

Ctrl+J: Justify

Ctrl+\: Search and replace text

Ctrl+C: Display current column and row position

Ctrl+G: Get detailed help on using the nano

Ctrl+-: Go to specified line and column position

Ctrl+O: Save (write out) the file currently open

Ctrl+X: Exit nano

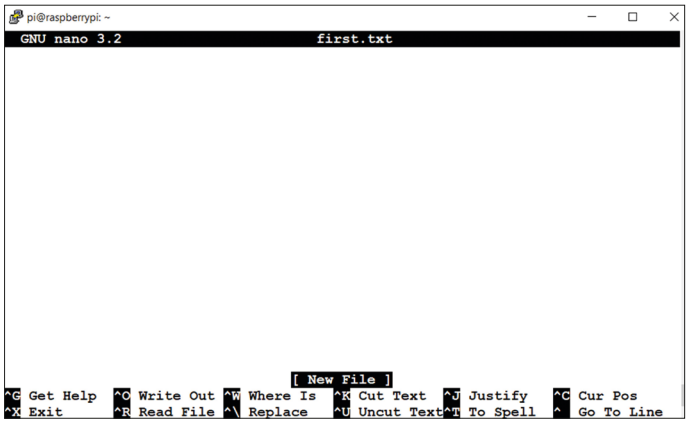


Figure 2.1: The nano text editor startup screen.

Now, type the following text into the file:

> nano is a simple and yet powerful text editor.
> This simple text example demonstrates how to use nano.
> This is the last line of the example.

The use of **nano** is now demonstrated with the following steps:

Step 1: Go the beginning of the file by moving the cursor.

Step 2: Look for word **simple** by pressing **Ctrl+W** and then typing **simple** in the window opened at the bottom left hand corner of the screen. Press the Enter key. The cursor will be positioned on the word **simple** (see Figure 2.2).

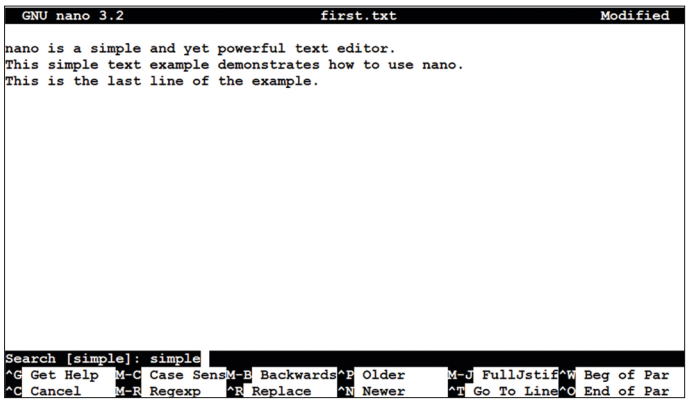


Figure 2.2 Searching for the word "simple".

Step 3: Cut the first line by placing the cursor anywhere on the line and then pressing Ctrl+K. The first line will disappear.

Step 4: Paste the line cut after the first line. Place the cursor on the second line and press **Ctrl+U** (see Figure 2.3).

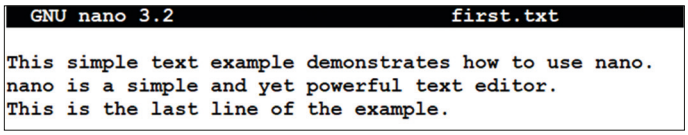


Figure 2.3: Pasting the text line you cut previously.

Step 5: Place cursor at the beginning of the word **simple** on the first row. Enter **Ctrl+C**. The row and column positions of this word will be displayed at the bottom of the screen.

Step 6: Press **Ctrl+G** to display help page as in Figure 2.4. Notice that the display is many pages long and you can jump to the next pages by pressing **Ctrl+Y** or to the previous pages by pressing **Ctrl+V**. Press **Ctrl+X** to exit the help page.

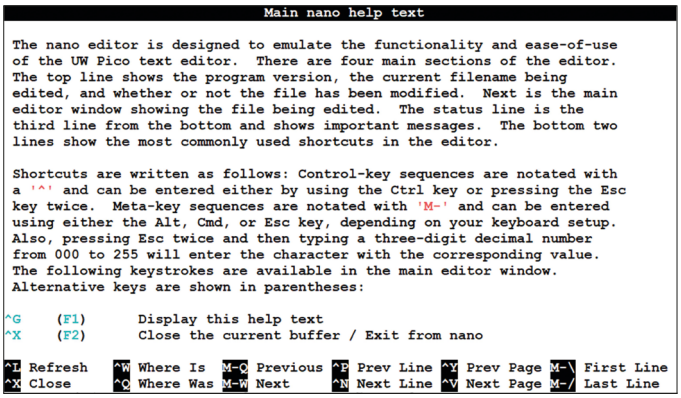


Figure 2.4: Displaying the help page.

Step 7: Press **Ctrl+-** and enter line and column numbers as 2 and 5, followed by the Enter key, to move cursor to line 2, column 5.

Step 8: Replace word **example** with word **file**. Press **Ctrl+** and type the first word as **example** (see Figure 2.5). Press Enter and then type the replacement word as **file**. Press Enter and accept the change by typing y.

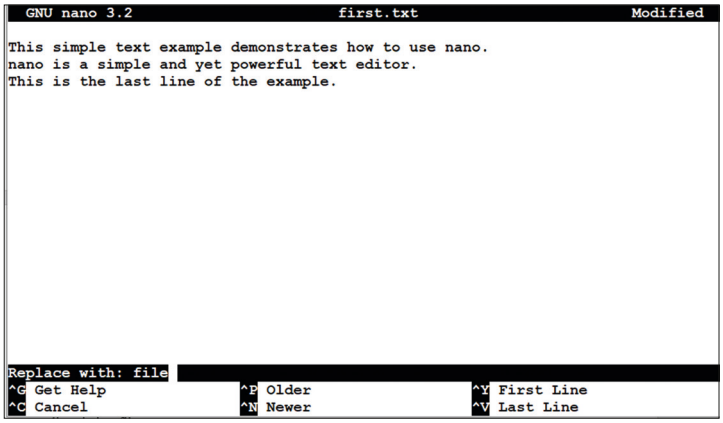


Figure 2.5: Replacing text.

Step 9: Save the changes. Press **Ctrl+X** to exit the file. Type **Y** to accept the saving, then enter the filename to be written to, or simply press Enter to write to the existing file (**first.txt** in this example). The file will be saved in your current working directory.

Step 10: Display the contents of the file:

```
pi@raspberrypi ~ $ cat first.txt

        This simple text file demonstrates how to use nano.
        Nano is a simple and yet powerful text editor.
        This is the last line of the example.

pi@raspberrypi ~ $
```

In summary, **nano** is a simple and yet powerful text editor allowing us to create new text files or edit existing files.

Chapter 3 • Creating and Running a Python Program

3.1 Overview

Read this chapter and learn how to create and then run a very simple Python program. As described here, basically there are three methods that you can use to create and run a Python program. The text message **Hello From Raspberry Pi Zero 2 W** will be displayed on your screen as an example.

Version 3.9.2 of Python was the current version at the time this book was written and this is the version that you will be using in this book. You may find that both version 2.x and 3.x are installed on your Zero 2 W. Enter the following commands to display the versions of Python you have on your Raspberry Pi (Figure 3.1):

pi@raspberrypi:~ $ **python –version**

and pi@raspberrypi:~ $ **python3 --version**

The advantage of using Python 3 is that most of the new libraries are being developed for it. Additionally, version 3.x has better error handling and improved GUI support over earlier releases.

```
pi@raspberrypi:~ $ python --version
Python 2.7.18
pi@raspberrypi:~ $ python3 --version
Python 3.9.2
```

Figure 3.1: Displaying the Python versions.

3.2 Method 1 — Interactively from command prompt

In this method, you will login to your Zero 2 W remotely using the SSH and then create and run your program interactively. This method is excellent for small programs. The steps are as follows:

- Log in to the Zero 2 W using SSH.
- At the command prompt enter **python3**. You should see the Python command mode which is identified by three characters **>>>.**
- Type the program:

  print ("Hello From Raspberry Pi Zero 2 W")

- The required text will be displayed interactively on the screen as shown in Figure 3.2.

```
pi@raspberrypi:~ $ python3
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello From Raspberry Pi Zero 2 W")
Hello From Raspberry Pi Zero 2 W
>>>
```

Figure 3.2: Running a program interactively.

- Enter **Cntrl+Z** to exit from Python.

3.3 Method 2 — Create a Python file in command mode

In this method, you will log in to your Zero 2 W using the SSH as before and then create a Python file. A Python file is simply a text file with the extension **.py**. You can use a text editor like **nano** to create your file. In this example, a file called **hello.py** is created using the **nano** text editor. Exit and save the file by entering **CNTRL + X** followed by **Y**. Figure 3.3 shows the contents of file **hello.py**. This figure also shows how to run the file from the command line. Notice that the program is run by entering the command:

pi@raspberrypi:~ $ **python3 hello.py**

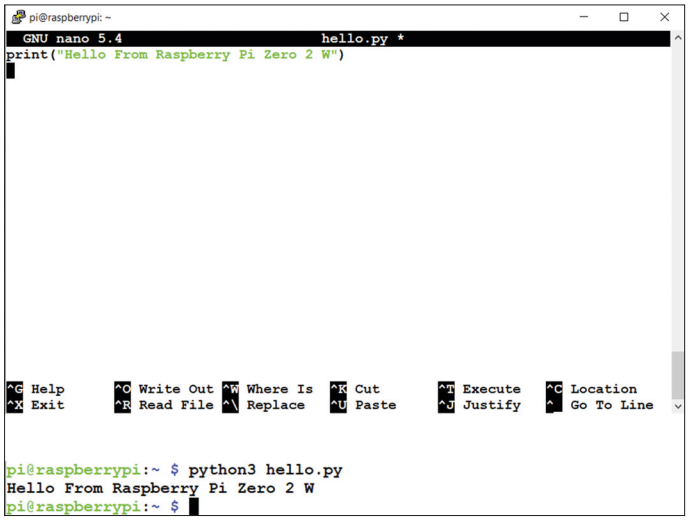


Figure 3.3: Creating and running a Python file.

3.4 Method 3 — Create a Python file in GUI mode

In this method, you will log in to your Zero 2 W using the VNC and create and run your program in GUI mode on the Desktop. The steps are given below.

- Connect to your Zero 2 W using the SSH.
- Start the VNC server on your Pi:

  pi@raspberrypi:~ $ **vncserver :1**

- Start the **VNC Viewer** on your PC by entering the IP address of your Zero 2 W followed by **:1**. e.g.,

  192.168.3.20:1

- Click **Continue**.
- Enter your password and click **OK**.
- You should see the Desktop screen.
- Click on **Applications** menu on Desktop (red Raspberry Pi icon).
- Click **Programming** and then click **Thonny Python IDE** (see Figure 3.4).

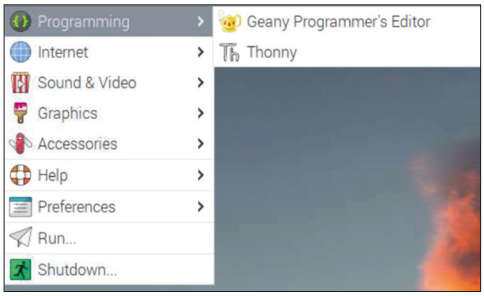


Figure 3.4: Select Thonny.

- Type in your program as shown in Figure 3.5 and save it with a name, for example, **hello2.py**.

Figure 3.5: Entering your program.

- Run the program by clicking **Run**. You should see the program output displayed at the bottom part (Shell) of the screen as shown in Figure 3.6.

Figure 3.6: Output of the program.

Notice that you can also run interactive Python statements in the Shell part of the screen. An example is shown in Figure 3.7.

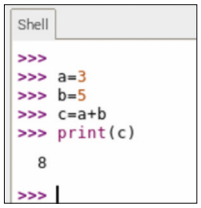


Figure 3.7: Running Python statements in the Shell.

3.5 Which method?

The choice of a method depends upon the size and complexity of a program. Small programs or small code to test an idea 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 method 2 as well as method 3 are used in programming examples and projects depending on the size of the program to be developed.

3.6 The Thonny screen

The Thonny screen has the following menu options:

New: used to create a new file
Load: used to load a file to Thonny
Save: used to save a file
Run: click to run the current file in Thonny
Debug: used to debug the current file in Thonny
Over: debug option
Into: debug option
Out: debug option
Stop: stop debugger
Zoom: used to zoom the screen
Quit: quit Thonny
Support: help (support) on Thonny

3.6.1 Using the debugger

The debugger is very useful during problem development. A simple example of using the debugger is shown in Figure 3.8. Variables **a**, **b**, and **c** are assigned integer values and variable **d** calculates the sum of these variables. After writing the program, click the Debug icon. Then, click the **Into** icon to step through the program. You should see the values of the variables at the right hand side as you single step through the program.

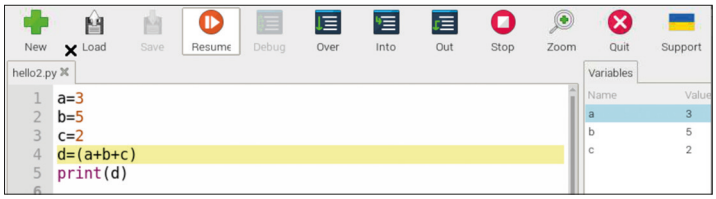


Figure 3.8: Example of using the debugger.

Chapter 4 • Software-Only Python Programs using the Zero 2 W

4.1 Overview

In this Chapter, example Python programs are given to familiarize you with the Python programming language. The plans in this chapter are software-only and they do not access any external or internal hardware of the Zero 2 W development board.

4.2 Example 1 — Average of two numbers read from the keyboard

In this example, two numbers are read from the keyboard and their average is displayed. The aim of this example is to show how data can be read from the keyboard.

Solution 1

The program is named **Average** and the program listing and an example run of the program are shown in Figure 4.1. Function **input** is used to read the numbers in the form of strings from the keyboard. These strings are then converted into floating-point numbers and stored in variables **n1** and **n2**. The average is calculated by adding and then dividing the numbers by two. The result is displayed on the screen.

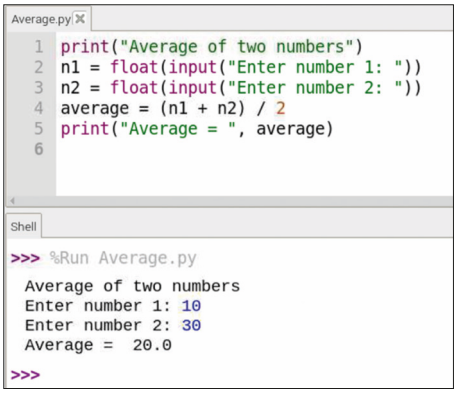


Figure 4.1: Program: **Average** *and a sample run.*

4.3 Example 2 — Average of 10 numbers read from the keyboard

In this example, 10 numbers are read from the keyboard and their average is displayed. The aim of this example is to show how a loop can be constructed in Python.

Solution 2

This demo program is named **Average10** and the program listing and an example run are shown in Figure 4.2. In this program, a loop is constructed which runs from 0 to 9 (i.e., 10 times). Inside this loop the numbers are read from the keyboard, added, and stored in variable **sum**. The average is then calculated and displayed by dividing **sum** by 10. Notice that a new-line is not printed after the print statements since the option **end = " "** is used inside the print statement.

```
Average10.py
1 print("Average of 10 numbers")
2 sum = 0
3 for n in range(10):
4     print("Enter number", n+1, ":",end='')
5     m = float(input())
6     sum = sum + m
7 average = sum / 10
8 print("Average = ", average)

Shell
>>> %Run Average10.py
  Average of 10 numbers
  Enter number 1 :2
  Enter number 2 :4
  Enter number 3 :6
  Enter number 4 :8
  Enter number 5 :10
  Enter number 6 :1
  Enter number 7 :3
  Enter number 8 :5
  Enter number 9 :6
  Enter number 10 :7
  Average =  5.2
>>>
```

Figure 4.2: Program: **Average10** *and a sample run.*

4.4 Example 3 — Surface area of a cylinder

In this example the radius and height of a cylinder are read from the keyboard and its surface area is displayed on the screen.

Solution 3

The program is named **CylArea** and the program listing and an example run of the program are shown in Figure 4.3. The surface area of a cylinder is given by:

$$\text{Surface area} = 2\,\pi\,r\,h$$

Where r and h are the radius and height of the cylinder respectively. In this program the **math** library is imported so that function **Pi** can be used in the program. The surface area of the cylinder is displayed after reading its radius and height.

```
CylArea.py
1 import math
2 print("Surface area of a cylinder")
3 r = float(input("Enter radius: "))
4 h = float(input("Enter height: "))
5 area = 2 * math.pi * r * h
6 print("Surface area = %.2f" % area)

Shell
>>> %Run CylArea.py
  Surface area of a cylinder
  Enter radius: 12
  Enter height: 4
  Surface area = 301.59
>>>
```

Figure 4.3: Program: **CylArea** *and a sample run.*

4.5 Example 4 — ºC to ºF conversion

In this example, the program reads degrees Celsius from the keyboard and converts and displays the equivalent in degrees Fahrenheit.

Solution 4

The program is named **CtoF** and the program listing and an example run of the program are shown in Figure 4.4. The formula to convert ºC to ºF is:

$$F = 1.8 \times C + 32$$

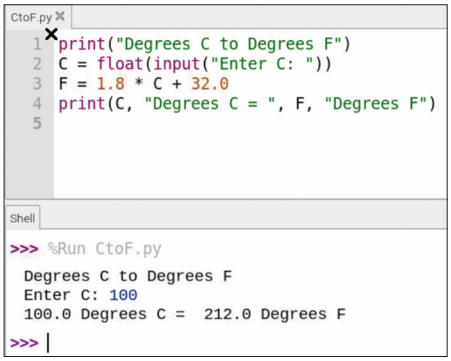


Figure 4.4: Program: **CtoF** *and a sample run.*

4.6 Example 5 — Surface area and volume of a cylinder; user function

In this example the surface area and volume of a cylinder are calculated whose radius and height are given. The program uses a function to calculate and return the surface area and the volume.

Solution 5

The program is named **CylAreaSurf** and the program listing and an example run of the program are shown in Figure 4.5. The surface area and the volume of a cylinder are given by:

$$\text{Surface area} = 2\,\pi\, r\, h$$
$$\text{Volume} = \pi\, r^2\, h$$

Where r and h are the radius and height of the cylinder respectively. The function **Calc** is used to get the radius and height of the cylinder. The function returns the surface area and volume to the main program which puts both values on the screen.

```
CylAreaSurf.py
1  import math
2  print("Surface area and volume of a cylinder")
3
4  def Calc(r, h):
5      area = 2 * math.pi * r * h
6      volume = math.pi * r * r * h
7      return(area, volume)
8
9  radius = float(input("Enter radius: "))
10 height = float(input("Enter height: "))
11 area, vol = Calc(radius, height)
12 print("Surface area = %.2f  Volume = %.2f" %(area,vol))

Shell
>>> %Run CylAreaSurf.py
 Surface area and volume of a cylinder
 Enter radius: 10
 Enter height: 4
```

Figure 4.5 Program: **CylAreaSurf** *and a sample run.*

4.7 Example 6 — Table of squares of numbers

In this example, the squares of numbers from 1 to 11 are calculated and tabulated.

Solution 6

The program is named **Squares** and the program listing and an example run of the program are shown in Figure 4.6. Notice that **\t** prints a tab so that the data can be tabulated nicely.

```
Squares.py
1  print("TABLE OF SQAURES")
2  print("================")
3  print("N      Square")
4  for i in range(11):
5      n = i + 1
6      print(n, "\t", n*n)

Shell
>>> %Run Squares.py
 TABLE OF SQAURES
 ================
 N      Square
 1        1
 2        4
 3        9
 4        16
 5        25
 6        36
 7        49
 8        64
 9        81
 10       100
 11       121
>>>
```

Figure 4.6 Program: **Squares** *and a sample run.*

4.8 Example 7 — Table of trigonometric sine

In this example, the trigonometric sine is tabulated from 0 to 45 degrees in steps of 5 degrees.

Solution 7

The program is named **Sines** and the program listing and an example run of the program are shown in Figure 4.7. It is important to notice that the arguments of the trigonometric functions must be in radians and not in degrees.

```
Sines.py
1  import math
2  print("TABLE OF TRIGONOMETRIC SIN")
3  print("==========================")
4  print("N          Sin")
5  for i in range(0, 50, 5):
6      d = math.radians(i)
7      print(i, "\t", math.sin(d))

Shell
>>> %Run Sines.py
  TABLE OF TRIGONOMETRIC SIN
  ==========================
  N          Sin
  0        0.0
  5        0.08715574274765817
  10       0.17364817766693033
  15       0.25881904510252074
  20       0.3420201433256687
  25       0.42261826174069944
  30       0.49999999999999994
  35       0.573576436351046
  40       0.6427876096865393
  45       0.7071067811865475
>>>
```

Figure 4.7 Program: **Sines** *and sample output.*

4.9 Example 8 — Table of trigonometric sine, cosine, and tangent

In this example, the trigonometric sine, cosine, and tangent are tabulated from 0 to 30 degrees in steps of 5 degrees.

Solution 8

The program is named **Trig** and the program listing and an example run of the program are shown in Figure 4.8.

```
Trig.py
1  import math
2  print("         TABLE OF TRIGONOMETRIC FUNCTIONS")
3  print("         ================================")
4  print("N\t   Sin\t\t  Cos\t\t   Tan")
5  for i in range(0, 35, 5):
6      d = math.radians(i)
7      s = math.sin(d)
8      c = math.cos(d)
9      t = math.tan(d)
10     print(i, "\t%9.7f\t%9.7f\t%9.7f" %(s,c,t))

Shell
>>> %Run Trig.py
          TABLE OF TRIGONOMETRIC FUNCTIONS
          ================================
N          Sin             Cos             Tan
0       0.0000000       1.0000000       0.0000000
5       0.0871557       0.9961947       0.0874887
10      0.1736482       0.9848078       0.1763270
15      0.2588190       0.9659258       0.2679492
20      0.3420201       0.9396926       0.3639702
25      0.4226183       0.9063078       0.4663077
30      0.5000000       0.8660254       0.5773503
```

Figure 4.8: Program: **Trig** *and sample output.*

4.10 Example 9 — Trigonometric function of a required angle

In this example, an angle is read from the keyboard. Also, the user specifies whether the sine (s), cosine (c), or the tangent (t) of the angle is required.

Solution 9

The program is named **TrigUser** and the program listing and an example run of the program are shown in Figure 4.9.

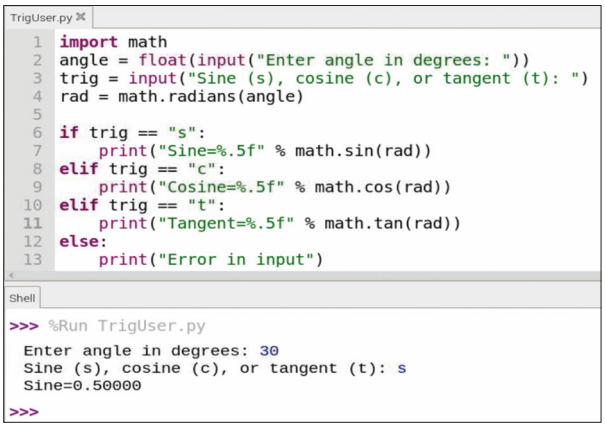


Figure 4.9: Program: **TrigUser** *and sample output.*

4.11 Example 10 — Series and parallel resistors

This program calculates the total resistance of a number of series or parallel-connected resistors. The user specifies whether the connection is in series or in parallel. Additionally, the number of resistors used is also specified at the beginning of the program.

Solution 10

When a number of resistors are in series, the resultant resistance is the sum of the resistances of each individual resistor. When the resistors are in parallel, the reciprocal of the resultant resistance is equal to the sum of the reciprocal resistances of each resistor.

Figure 4.10 shows the program listing (program: **Serpal**). At the beginning of the program a heading is displayed and the program enters into a **while** loop. Inside his loop, the user is prompted to enter the number of resistors in the circuit and whether they are connected in series or in parallel. The function **str** converts a number into its equivalent string. For example, number 5 is converted into string "5". If the connection is serial (mode equals **"s"**) then the value of each resistor is accepted from the keyboard and the result is calculated and displayed on the screen. If, on the other hand, the connection is parallel (mode equals **"p"**), then again the value of each resistor is accepted from the keyboard and the reciprocal of the number is added to the total. When all resistor values are entered, the resultant resistance is displayed on the screen.

```
print("RESISTORS IN SERIES OR PARALLEL")
print("==============================")
yn = "y"

while yn == "y":
  N = int(input("\nHow many resistors are there?: "))
```

```
  mode = input("Are the resistors series (s) or parallel (p)?: ")
  mode = mode.lower()
#
# Read the resistor values and calculate the total
#
  resistor = 0.0

  if mode == "s":
    for n in range(0,N):
       s = "Enter resistor " + str(n+1) + " value in Ohms: "
       r = int(input(s))
       resistor = resistor + r
    print("Total resistance = %d Ohms" %(resistor))

  elif mode == "p":
    for n in range(0,N):
      s = "Enter resistor " + str(n+1) + " value in Ohms: "
      r = float(input(s))
      resistor = resistor + 1 / r
    print("Total resistance = %.2f Ohms" %(1 / resistor))
#
# Check if the user wants to exit
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Figure 4.10: Program: **Serpal**.

Figure 4.11 shows a typical run of the program.

```
Shell
RESISTORS IN SERIES OR PARALLEL
==============================

How many resistors are there?: 3
Are the resistors series (s) or parallel (p)?: s
Enter resistor 1 value in Ohms: 100
Enter resistor 2 value in Ohms: 200
Enter resistor 3 value in Ohms: 300
Total resistance = 600 Ohms

Do you want to continue?: n
>>>
```

Figure 4.11: Typical run of **Serpal**.

4.12 Example 11 — Words in reverse order

Write a program to read a word from the keyboard and then display the letters of this word in reverse order on the screen.

Solution 11

The required program listing is shown in Figure 4.12 (program: **Letters**). A word is read from the keyboard and stored in string variable **word**. Then the letters of this word are displayed in reverse order. An example run of the program is shown in Figure 4.12.

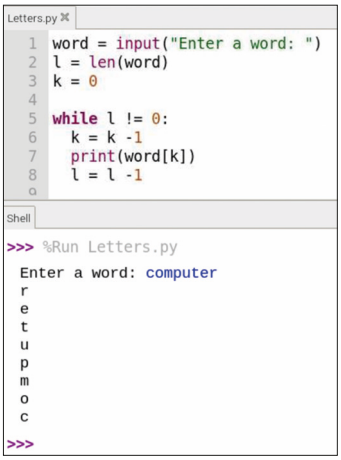


Figure 4.12: Program: **Letters** *and a sample output.*

4.13 Example 12 — Calculator

Write a calculator program to carry out the four simple mathematical operations of addition, subtraction, multiplication, and division on two numbers received from the keyboard.

Solution 12

The required program listing is shown in Figure 4.13 (program: **Calc**). Two numbers are received from the keyboard and stored in variables **n1** and **n2**. Then, the required mathematical operation is received and it is performed. The result, stored in variable **result**, is displayed on the screen. The user is given the option of terminating the program.

```
any = "y"
while any == "y":
   print("\nCalculator Program")
   print("==================")

   n1 = float(input("Enter first number: "))
   n2 = float(input("Enter second number: "))
   op = input("Enter operation (+-*/): ")

   if op =="+":
      result = n1 + n2
   elif op == "-":
      result = n1 - n2
   elif op == "*":
      result = n1 * n2
```

```
elif op == "/":
   result = n1 / n2
print("Result = %f" %(result))
any = input("\nAny more (yn): ")
```

Figure 4.13: Program: **Calc**.

An example run of the program is shown in Figure 4.14.

```
Calculator Program
==================
Enter first number: 25
Enter second number: 4
Enter operation (+-*/): *
Result = 100.000000

Any more (yn): n
>>>
```

Figure 4.14 Example run of **Calc**.

4.14 Example 13 — File processing: writing

In this example a text file called **MyFile.txt** will be created and text **Hello from Raspberry Pi Pico!** will be written to this file

Solution 13

The program is named **Filew** and its listing and an example run are shown in Figure 4.15. The file is opened in write (**w**) mode and the text is written in it using function **write**. Notice here that **fp** is the file handle.

Filew.py

```
1 print("Open the file and write the text")
2 fp = open("MyFile.txt", "w")
3 fp.write("Hello from Raspberry Pi Zero 2 W!\n")
4 fp.close()
5 print("End of file operation")
6
7
```

Shell

```
>>> %Run Filew.py
  Open the file and write the text
  End of file operation
>>>
```

Figure 4.15: Program: **Filew**.

4.15 Example 14 — File processing: reading

In this example the text file **MyFile.txt** created in the previous example is opened and its contents is displayed on the screen

Solution 14

The program is named **Filer** and its listing and an example run are shown in Figure 4.16. The fire is opened in read (**r**) mode and its contents is displayed.

```
Filer.py
1 print("Open the file and read its contents")
2 fp = open("MyFile.txt", "r")
3 str = fp.read(80)
4 fp.close()
5 print(str)
6
7

Shell
>>> %Run Filer.py
  Open the file and read its contents
  Hello from Raspberry Pi Zero 2 W!

>>>
```

Figure 4.16 Program: **Filer**.

4.16 Example 15 — Squares and cubes of numbers

Assignment: Write a program to tabulate the squares and cubes of numbers from 1 to 10.

Solution 15

The program is named **Cubes** and its listing and an example run are shown in Figure 4.17.

```
Cubes.py
1 print("Squares and cubes of numbers")
2 print(' N  N*N  N*N*N')
3 for i in range(1,11):
4     print('{0:2d} {1:3d} {2:4d}'.format(i, i*i, i*i*i))
5

Shell
>>> %Run Cubes.py
  Squares and cubes of numbers
   N  N*N  N*N*N
   1   1    1
   2   4    8
   3   9   27
   4  16   64
   5  25  125
   6  36  216
   7  49  343
   8  64  512
   9  81  729
  10 100 1000
>>>
```

Figure 4.17: Program: **Cubes** *and example output.*

4.17 Example 16 — Multiplication timetable

Assignment: Write a program to read a number from the keyboard and then display the timetable for this number from 1 to 12.

Solution 16

The program is named **Times** and its listing and an example run are shown in Figure 4.18.

```
Times.py
1  num = int(input("Enter a number: "))
2  print("Timetable of number ", num)
3  for i in range(1, 13):
4     print(num, 'x', i, '=', num*i)

Shell
>>> %Run Times.py
  Enter a number: 5
  Timetable of number  5
  5 x 1 = 5
  5 x 2 = 10
  5 x 3 = 15
  5 x 4 = 20
  5 x 5 = 25
  5 x 6 = 30
  5 x 7 = 35
  5 x 8 = 40
  5 x 9 = 45
  5 x 10 = 50
  5 x 11 = 55
  5 x 12 = 60
>>>
```

Figure 4.18: Program: **Times** *and example output.*

4.18 Example 17 — Odd or even

Assignment: Write a program to read a number from the keyboard and check and display if this number is odd or even

Solution 17

The program is named **OddEven** and its listing and an example run are shown in Figure 4.19.

```
OddEven.py
1  num = int(input("Enter a number: "))
2  if (num % 2) == 0:
3     print("Number {0} is Even".format(num))
4  else:
5     print("Number {0} is Odd".format(num))

Shell
>>> %Run OddEven.py
  Enter a number: 15
  Number 15 is Odd
>>>
```

Figure 4.19 Program: **OddEven** *and example output.*

4.19 Example 18 — Binary, octal, and hexadecimal

Assignment: Write a program to read a decimal number from the keyboard. Convert this number into binary, octal, and hexadecimal and display it on the screen.

Solution 18

The program is named **Conv** and its listing and an example run are shown in Figure 4.20.

```
Conv.py
1  dec = int(input("Enter a number: "))
2  print("The decimal value of", dec, "is:")
3  print("in binary: ",bin(dec))
4  print("in octal: ",oct(dec))
5  print("in hexadecimal: ",hex(dec))

Shell
>>> %Run Conv.py
  Enter a number: 17
  The decimal value of 17 is:
  in binary:  0b10001
  in octal:  0o21
  in hexadecimal:  0x11

>>>
```

Figure 4.20: Program: **Conv** *and example output.*

4.20 Example 19 — Add two matrices

Assignment: Write a program to add two given matrices and display the elements of the new matrix.

Solution 19

The program is named **AddMatrix** and its listing and an example run are shown in Figure 4.21.

```
AddMatrix.py
1  A = [[5,4,2],
2      [6 ,3,1],
3      [2 ,8,3]]
4
5  B = [[2,4,2],
6      [0,4,10],
7      [8,2,8]]
8
9  res = [[0,0,0],
10        [0,0,0],
11        [0,0,0]]
12
13 for i in range(len(A)):
14    for j in range(len(A[0])):
15        res[i][j] = A[i][j] + B[i][j]
16
17 for i in res:
18    print(i)

Shell
>>> %Run AddMatrix.py
  [7, 8, 4]
  [6, 7, 11]
  [10, 10, 11]
```

Figure 4.21: Program: **AddMatrix** *and example output.*

4.21 Example 20 — Shapes

Assignment: Write a program to use functions to calculate and display the areas of these shapes: square, rectangle, triangle, circle, and cylinder. The sizes of the required sides should be received from the keyboard.

Solution 20

The areas of the shapes to be used in the program are as follows:

Square: side = *a*	area = *a*2
Rectangle: sides *a*, *b*	area = *a b*
Circle: radius *r*	area = π *r*2
Triangle: base *b*, height *h*	area = *b h* / 2
Cylinder: radius *r*, height *h*	area = 2 π *r h*

The required program listing is shown in Figure 4.22 (program: **areas.py**). A different function is used for each shape and the sizes of the sides are received inside the functions. The main program displays the calculated area for the chosen shape.

```
import math

def Square(a):                      # square
  return a * a

def Rectangle(a, b):                # rectangle
  return(a * b)

def Triangle(b, h):                 # triangle
  return(b * h / 2)

def Circle(r):                      # circle
  return(math.pi * r * r)

def Cylinder(r, h):                 # cylinder
  return(2 * math.pi * r * h)

print("AREAS OF SHAPES")
print("===============\n")
print("What is the shape?: ")

shape = input("Square (s)\nRectangle(r)\nCircle(c)\n\
Triangle(t)\nCylinder(y): ")

shape = shape.lower()
if shape == 's':
  a = float(input("Enter a side of the square: "))
  area = Square(a)
  s = "Square"
elif shape == 'r':
  a = float(input("Enter one side of the rectangle: "))
  b = float(input("Enter other side of the rectangle: "))
  area = Rectangle(a, b)
  s = "Rectangle"
elif shape == 'c':
  radius = float(input("Enter radius of the circle: "))
```

```
  area = Circle(radius)
  s = "Circle"
elif shape == 't':
  base = float(input("Enter base of the triangle: "))
  height = float(input("Enter height of the triangle: "))
  area = Triangle(base, height)
  s = "Triangle"
elif shape == 'y':
  radius = float(input("Enter radius of cylinder: "))
  height = float(input("Enter height of cylinder: "))
  area = Cylinder(radius, height)
  s = "Cylinder"

print("Area of %s is %f" %(s, area))
```

Figure 4.22: Program: **areas**.

An example run of the program is shown in Figure 4.23.

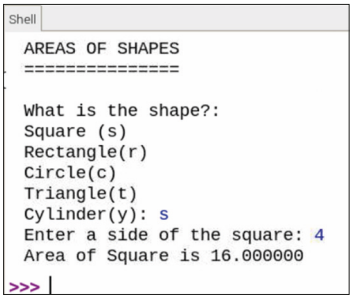


Figure 4.23: Example run of the program.

4.22 Example 21 — Solution of a quadratic equation

Assignment: Write a program to find the roots of a quadratic equation of the form:

$$a\,x^{**}2 + b\,x + c = 0$$

Solution 21

The required program listing is shown in Figure 4.24 (program: **quadratic**). The coefficients *a*, *b*, and *c* are specified at the beginning of the program. The two roots are calculated and displayed on the screen.

quadratic.py

```
1  import cmath
2
3  a = 1
4  b = -5
5  c = 6
6
7  d = (b**2) - (4*a*c)            # the discriminant
8
9  root1 = (-b-cmath.sqrt(d))/(2*a)
10 root2 = (-b+cmath.sqrt(d))/(2*a)
11
12 print('The roots are {0} and {1}'.format(root1,root2))
```

Shell

```
>>> %Run quadratic.py
  The roots are (2+0j) and (3+0j)
>>>
```

Figure 4.24: Program: **quadratic** *and example output.*

4.23 Example 22 — Matrix multiplication

Write a program to multiply two matrices.

Solution 22

The elements of the two matrices are specified at the beginning of the program. Figure 4.25 shows the program listing (program: **multmatrix**).

quadratic.py

```
1  import cmath
2
3  a = 1
4  b = -5
5  c = 6
6
7  d = (b**2) - (4*a*c)            # the discriminant
8
9  root1 = (-b-cmath.sqrt(d))/(2*a)
10 root2 = (-b+cmath.sqrt(d))/(2*a)
11
12 print('The roots are {0} and {1}'.format(root1,root2))
```

Shell

```
>>> %Run quadratic.py
  The roots are (2+0j) and (3+0j)
>>>
```

Figure 4.25: Program: **multmatrix** *and example output.*

4.24 Example 23 — Factorial of a number

Assignment: Write a program to calculate the factorial of a number entered from the keyboard.

Solution 23

The program listing is shown in Figure 4.26 (program: **factorial**).

```
factorial.py
 1 numbr = int(input("Enter a number: "))
 2 factorial = 1
 3
 4 # check if the number is negative, zero, or positive
 5 if numbr < 0:
 6    print("Negative numbers do not have factorials")
 7 elif numbr == 0:
 8    print("The factorial of 0 is 1")
 9 else:
10    for i in range(1, numbr + 1):
11        factorial = factorial*i
12    print("The factorial of",numbr,"is",factorial)
13
Shell
>>> %Run factorial.py
 Enter a number: 4
 The factorial of 4 is 24
>>>
```

Figure 4.26: Program: **factorial** *and example output.*

4.25 Example 24 — Compound interest

Assignment: Write a program to calculate the compound interest given the initial value, interest rate, and the number of years.

Solution 24

Compound interest is calculated using the formula:

$$FV = IV(1 + i / 100) ** yr$$
$$\text{Compound interest} = FV - IV$$

Where *FV* and *IV* are the future and initial values, *i* is the interest rate and *yr* is the number of years.

Figure 4.27 gives the program listing (program: **compound**).

```
compound.py
 1 def compound_interest(initial, rate, years):
 2
 3     FV = initial * (pow((1 + rate / 100), years))
 4     interest = FV - initial
 5     print("Compound interest is: %.5f" % interest)
 6
 7 IV = float(input("Enter initial value: "))
 8 i = float(input("Enter interest rate :"))
 9 yr = float(input("Enter number of years: "))
10
11 compound_interest(IV, i, yr)
12
Shell
>>> %Run compound.py
 Enter initial value: 100
 Enter interest rate :3
 Enter number of years: 2
 Compound interest is: 6.09000
>>>
```

Figure 4.27 Program: **compound** *and example output.*

4.26 Example 25 — Guess the number

Write a program to generate a secret number between 1 and 50 and let the user guess this number in 5 attempts.

Solution 25

The program listing is shown in Figure 4.28 (program: **guess**). An example run of the program is shown in Figure 4.29.

```
import random
total_guesses = 0

number = random.randint(1, 50)
print ("The secret number is between 1 and 50. You have 5 attempts")

while total_guesses < 5:
    guess = int(input("Your guess: "))
    total_guesses = total_guesses + 1

    if guess < number:
        print ("Too low...")
    if guess > number:
        print ("Too high...")
    if guess == number:
        break

if guess == number:
    print ("You guessed in {0} attempts".format(total_guesses))
else:
    print ("Sorry... The secret number was {0}".format(number))
```

Figure 4.28: Program: **guess**.

```
The secret number is between 1 and 50. You have 5 attempts
Your guess: 25
Too low...
Your guess: 30
Too high...
Your guess: 27
Too low...
Your guess: 28
Too low...
Your guess: 29
You guessed in 5 attempts
```

Figure 4.29: Example run of the program.

4.27 Example 26 — Numerical integration

Assignment: Write a program to read a function from the keyboard and calculate and display the integral of this function between two given points.

Solution 26

The program listing is shown in Figure 4.30 (program: **integrate**). The integration is performed inside function **integrate**. This function is divided into equal small segments between the required lower and upper limits. The area of each segment is calculated. The total area is equal to the required integral of the function. The arguments of this function are the function to be integrated, low and high limits, and the number of points to consider.

```
def integrate(func, xlow, xhigh, n):
    length = xhigh - xlow
    sect = length / n
    area = 0
    for j in range(n):
        x = xlow + sect * j
        y = eval(func.replace("x", str(x)))
        secarea = y * sect
        area = area + abs(secarea)
    return (area)

function = input("Enter function: ")
lowl = float(input("Low limit:" ))
highl = float(input("High limit:"))
N = int(input("No of points: "))
r = integrate(function, lowl, highl, N)
print("integral of %s between %5.2f and %5.2f = %6.2f" %(function,lowl,highl,r))
```

Figure 4.30: Program: **integrate***.*

Figure 4.31 shows an example run of the program where the following integral is calculated by using 100 segments and the result displayed on the Thonny screen:

$$\int_0^1 \left(x^2+2\right)dx$$

```
Enter function: x**2 + 2
Low limit:0
High limit:1
No of points: 100
integral of x**2 + 2 between  0.00 and  1.00 =   2.33
```

Figure 4.31: Example run of the program.

4.28 Example 27 — Practise arithmetic

Assignment: Write a program to display the following menu:

1. Addition
2. Subtraction
3. Multiplication
4. Division

Choice:
Generate two random integer numbers between 1 and 1000 and ask the user to carry out the required operation. Check the user answer and display CORRECT or INCORRECT.

Solution 27

Figure 4.32 shows the required program (Program: **arithmetic**). Functions are used for the addition, subtraction, multiplication, and division. The number entered by the user is checked and appropriate message is displayed.

```
import random

def addition():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, «+», n2, «= «)
    user_answer = int(input(«Enter answer: «))
    correct_answer = n1 + n2
    return (user_answer, correct_answer)

def subtraction():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, «-», n2, «= «)
    user_answer = int(input(«Enter answer: «))
    correct_answer = n1 - n2
    return (user_answer, correct_answer)

def multiplication():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
    print(n1, «X», n2, «= «)
    user_answer = int(input(«Enter answer: «))
    correct_answer = n1 * n2
    return (user_answer, correct_answer)

def division():
    n1 = random.randint(1, 1000)
    n2 = random.randint(1, 1000)
```

```
    print(n1, «/», n2, «= «)
    user_answer = int(input(«Enter answer: «))
    correct_answer = int(n1 / n2)
    return (user_answer, correct_answer)

def chk(user, correct):
    if user ==correct:
        print(«CORRECT»)
    else:
        print(«INCORRECT»)

print(«   1. Addition»)
print(«   2. Subtraction»)
print(«   3. Multiplication»)
print(«   4. Integer Division»)
ch = int(input(«Choice: «))
if ch == 1:
    user,correct = addition()
elif ch == 2:
    user,correct = subtraction()
elif ch == 3:
    user,correct = multiplication()
else:
    user,correct = division()
chk(user,correct)
```

Figure 4.32: Program: **arithmetic**.

Figure 4.33 shows an example run of the program.

```
    1. Addition
    2. Subtraction
    3. Multiplication
    4. Integer Division
Choice: 1
825 + 201 =
Enter answer: 1026
CORRECT
```

Figure 4.33: Example run.

4.29 Plotting in Python

Plotting graphs in Python is very easy with the **matplotlib** module. This module enables you to plot offline as well as real-time graphs. This section explains how to draw graphs to make the user familiar with the functions of the **matplotlib** module.

The **matplotlib** library module must be installed in Python before it can be used. This is done in command mode by entering the following command:

```
pi@raspberrypi~ $ sudo apt-get install python-matplotlib
```

4.29.1 Graph of a quadratic function

As an example, in this section, the graph of the quadratic function $y = x^2$ is drawn using Python. At the beginning of the program, **matplotlib** module and the **numpy** modules are imported into Python. **numpy** is a scientific package including many mathematical functions that can be used in Python programs.

The graphics can only be drawn in GUI Desktop mode. You should therefore use the VNC Viewer to get into the GUI mode and then create and run your program from there by selecting **Python 2 (IDLE).** The program listing is as follows:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.plot(x, x**2)
plt.show()
```

Figure 4.34 shows the graph plotted. Function **linspace(0,4,100)** creates a list 100 integer numbers in **x**, starting from 0 and terminating at 4. Function **plot** draws the graph where the **y** value is equal to **x2** . Function **show()** physically displays the graph. Notice that there are some buttons at the bottom of the window. These buttons are (from left to right as shown in Figure 4.35):

Home: Clicking this button displays the default figure as in Figure 4.34
Back: Brings back the plot after zooming
Next: This button is opposite of Back button
Pan: This button moves the window coordinates
Zoom: This button selects a zoom window
Adjust: this button adjusts the plot parameters
Save: click to save the plot

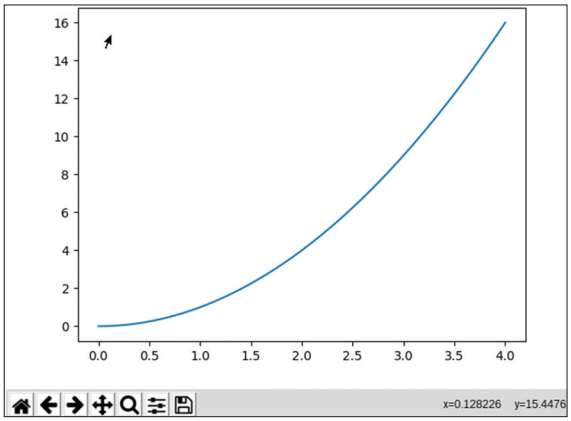


Figure 4.34: Graph of a quadratic function.

Figure 4.35: Graph buttons.

The graph in Figure 4.34 can be made more user friendly by labeling the axes and giving a title to the graph. The modified program is given below:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.plot(x, x**2, label="Quadratic")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Graph of y = x**2")
plt.legend()
plt.show()
```

The new graph is shown in Figure 4.36.

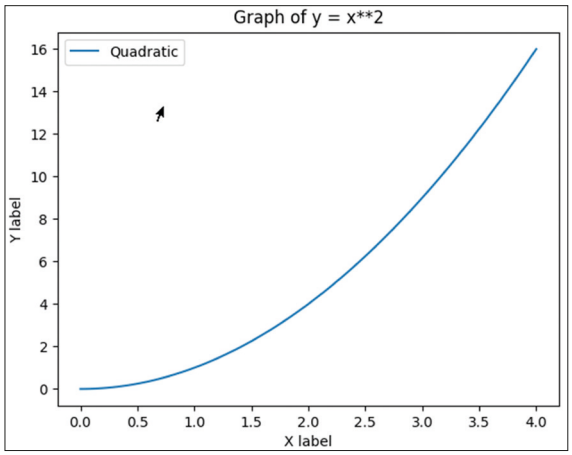


Figure 4.36: Graph with the axes labeled, and a title.

4.29.2 Drawing multiple graphs

You can easily plot more than one function on the same graph. In the example code given below, three graphs are drawn on the same axes: graph of $y = x+5$, $y = x2$, and $y = x***3$:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.plot(x, x+5,label="x+5")
plt.plot(x, x**2, label="Quadratic")
```

```
plt.plot(x, x**3,label="Cubic")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Graphs of linear, quadratic, and cubic functions")
plt.legend()
plt.show()
```

Figure 4.37 shows the graph with three functions.

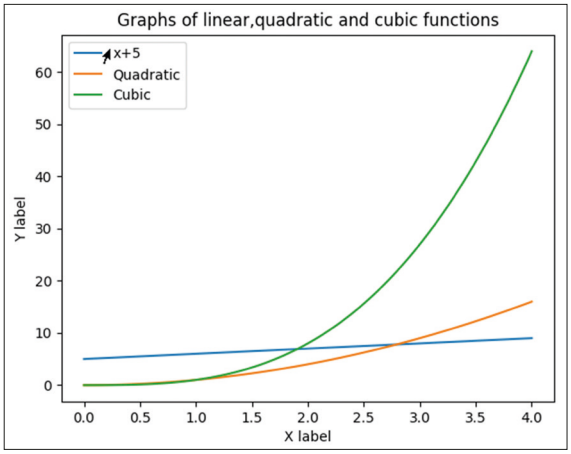


Figure 4.37: Graph with three functions.

Function **plot** draws a smooth graph by joining the *x,y* values. You can also draw different types of graphs. For example, function **scatter** draws a scatter graph as shown in Figure 4.38. The program for this graph is as follows:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.scatter(x, x+5,label="x+5")
plt.scatter(x, x**2, label="Quadratic")
plt.scatter(x, x**3,label="Cubic")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Graphs of linear,quadratic,and cubic functions")
plt.legend()
plt.show()
```

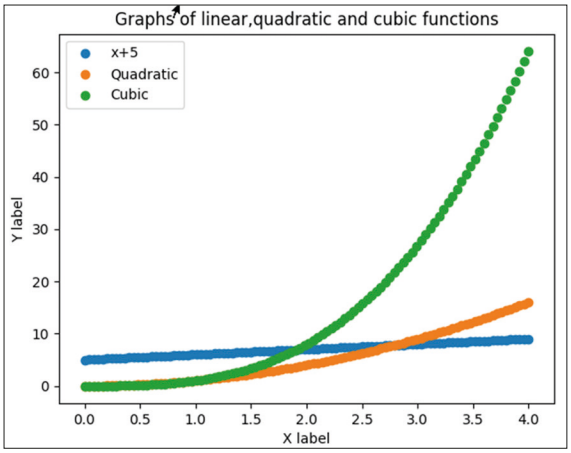


Figure 4.38: Drawing a "scatter" graph.

Function **bar** draws a bar chart as shown in Figure 4.39. The program for this graph is as follows.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,4,100)
plt.bar(x, x+5,label="x+5")
plt.bar(x, x**2, label="Quadratic")
plt.bar(x, x**3,label="Cubic")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Graphs of linear, quadratic, and cubic functions")
plt.legend()
plt.show()
```

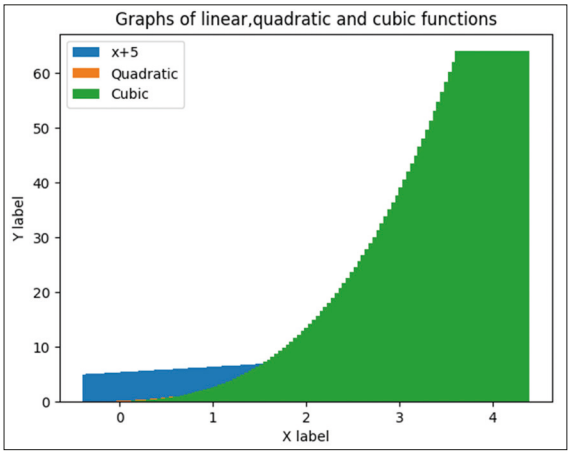


Figure 4.39: Drawing bar chart graph.

matplotlib is a large graphics library with many functions and the details of these functions are beyond the scope of this book. Interested readers can find books, tutorials, and applications notes on matplotlib on the Internet, for example,

https://matplotlib.org/faq/usage_faq.html

Chapter 5 • Simple Projects for the Raspberry Pi Zero 2 W

5.1 Overview

In this chapter you will be developing simple hardware projects with the Raspberry Pi Zero 2 W, using the **nano** text editor. The following sub-headings will be given for each project where applicable:

- Title
- Description
- Aim
- Block diagram
- Circuit diagram
- PDL (where necessary)
- Program listing
- Suggestions for future work (where necessary)

All the programs in this chapter have been developed using the **nano** text editor, although you can also use the Thonny IDE program if you prefer. A breadboard was used to construct and test the projects where necessary.

5.2 Project 1: External flashing LED

Description: In this project, an external LED is connected to Zero 2 W and the LED is flashed every second. The aim of this project is to show how an external LED can be connected to the Zero 2 W as well as explore how an external device can be accessed using a Python program.

Block diagram: Figure 5.1 shows the block diagram of the project. The colored header connector is not shown in this diagram.

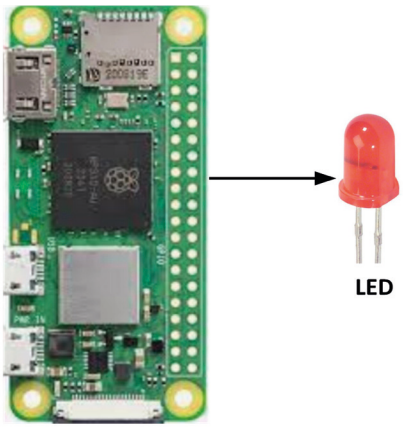


Figure 5.1: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.2, where the LED is connected to port pin GPIO2 (pin 3) of Zero 2 W through a 470-ohm current limiting resistor.

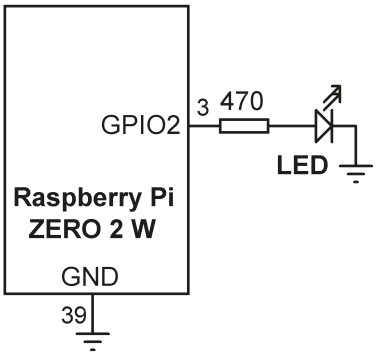


Figure 5.2: Circuit diagram of the project.

The LED can be connected in either current-sourcing or in current-sinking modes. In current-sourcing mode (Figure 5.3), the LED is turned ON when a logic High level is applied to the port pin. In current-sinking mode (Figure 5.4) the LED is turned ON when logic Low level is applied to the port pin.

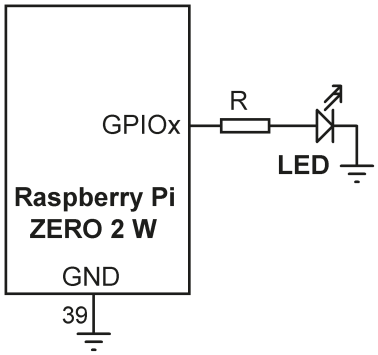


Figure 5.3: LED in current-sourcing mode.

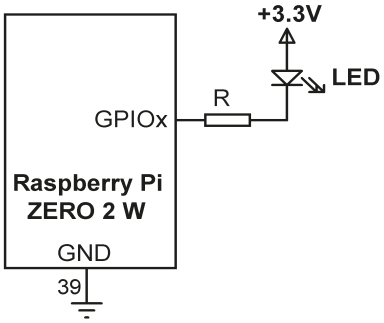


Figure 5.4: LED in current-sinking mode.

The required value of the current limiting resistor can be calculated as follows. In current-sourcing mode, assuming the output High voltage is +3.3 V, the voltage drop across the LED is 2 V, and the current through the LED is 3 mA, the required value of the current limiting resistor R is:

R = (3.3 – 2) / 3 = 433 ohms

So you choose 470 ohms as the nearest practical value.

Program listing: Figure 5.5 shows the program listing (Program: **ExtFlash.py**). At the beginning of the program, the GPIO and time libraries are imported to the program and warning messages are disabled. Then, BCM pin numbering is used so that, for example, port pin GPIO2 is referenced as 2 and not as the physical pin number 3. The LED at GPIO2 is configured as an output. A **while** loop is then formed which runs until stopped by the user. Inside this loop, the LED is turned ON and OFF with one second delay between each output.

```
#-----------------------------------------------------------
#              FLASHING AN EXTERNAL LED
#              ========================
#
# In this program an external LED is connected to port pin
# GPIO2 (pin 3). The LED is flashed every second
#
# Author: Dogan Ibrahim
# File  : ExtFlash.py
# Date  : December, 2022
#-----------------------------------------------------------
import RPi.GPIO as GPIO                # import GPIO library
import time                            # import time library
GPIO.setwarnings(False)                # disable warning messages
GPIO.setmode(GPIO.BCM)                 # set BCM pin numbering

LED = 2                                # LED at port GPIO2
ON = 1                                 # ON = 1
OFF = 0                                # OFF = 0
GPIO.setup(LED, GPIO.OUT)              # configure LED as output

while True:
  GPIO.output(LED, ON)                 # turn ON LED
  time.sleep(1)                        # wait 1 second
  GPIO.output(LED, OFF)                # turn OFF LED
  time.sleep(1)                        # wait 1 second
```

Figure 5.5: Program: **ExtFlash.py**.

Assuming that you have created the program using the **nano** text editor, enter the following statement in command mode to run the program:

pi@raspberry~ $ **python3 ExtFlash.py**

The program is halted by entering **CNTRL+C** at the terminal. Although the program in Figure 5.5 works perfectly well, terminating the program this way is not recommended. If the warnings are not disabled, then a second run of the program will give error messages since the GPIO buffers are not cleared.

Figure 5.6 shows the modified program (Program: **ExtFlash2.py**) where pressing CNTRL+C is detected and the program clears the GPIO buffers and terminates orderly. Notice that it is not necessary to disable warnings in this version of the program.

```
#-------------------------------------------------------------
#              FLASHING AN EXTERNAL LED
#              ========================
#
# In this program an external LED is connected to port pin GPIO2
# (pin 3). The LED is flashed every second.
#
# This program detects CNTRL+C and terminates orderly
#
# Author: Dogan Ibrahim
# File  : ExtFlash2.py
# Date  : December, 2022
#-------------------------------------------------------------
import RPi.GPIO as GPIO                # import GPIO library
import time                            # import time library
GPIO.setmode(GPIO.BCM)                 # set BCM pin numbering

LED = 2                                # LED at port GPIO2
ON = 1                                 # ON = 1
OFF = 0                                        # OFF = 0
GPIO.setup(LED, GPIO.OUT)              # configure LED as output

try:

   while True:                         # DO FOREVER
        GPIO.output(LED, ON)           # turn ON LED
        time.sleep(1)                  # wait 1 second
        GPIO.output(LED, OFF)          # turn OFF LED
        time.sleep(1)                  # wait 1 second

except KeyboardInterrupt:              # CNTRL+C detected
```

```
        GPIO.output(LED, 0)            # set LED OFF
        GPIO.cleanup()                 # clear GPIO buffers
```

Figure 5.6: Program: **ExtFlash2.py**.

Figure 5.7 shows the project built with a resistor and an LED.

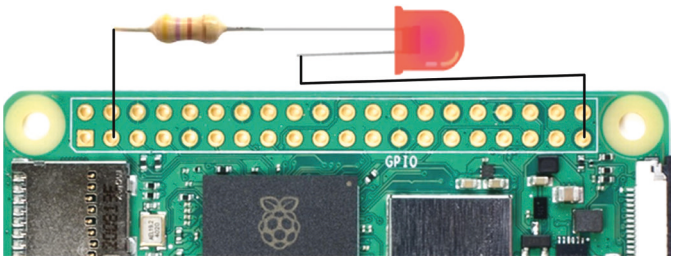


Figure 5.7: The project.

5.3 Project 2: Flashing the SOS signal

Description: In this project an external LED flashes the SOS signal (three dots, followed by three dashes, followed by three dots) continuously. A dot is represented with the LED being ON for 0.25 seconds (Dot time) and a dash is represented with the LED being ON for 1 second (Dash time). The delay between the dots and dashes is set to 0.2 second (GAP time). This process is repeated continuously after 2 seconds of delay.

The block diagram and circuit diagram of this project are same as in Figure 5.1 and Figure 5.2 respectively.

Program listing. Figure 5.8 shows the program listing (Program: **SOS.py**). At the beginning of the program, the dot, dash, and gap times are defined. Then a loop is formed using a **while** statement. Inside this loop, three **for** loops are formed, each iterating three times. The first loop displays three dots, the second loop, three dashes, and finally the last loop displays three dots. This process is repeated after two seconds of delay.

```
#-------------------------------------------------------------
#                      LED FLASHING SOS
#                      ================
#
# In this program the external LED at port GPIO2 flashes the
# SOS signal (... --- ...)
#
# Author: Dogan Ibrahim
# File  : SOS.py
# Date  : December, 2022
#-------------------------------------------------------------
import RPi.GPIO as GPIO                # import GPIO library
import time                            # import time library
```

```
GPIO.setwarnings(False)                # disable warning messages
GPIO.setmode(GPIO.BCM)                 # set BCM pin numbering

LED = 2                                # LED at port GPIO2
ON = 1                                 # ON = 1
OFF = 0                                # OFF = 0
GPIO.setup(LED, GPIO.OUT)              # configure LED as output

Dot = 0.25                             # Dot time
Dash = 1.0                             # Dash time
Gap = 0.2                              # Gap time

while True:                            # DO FOREVER
    for i in range(0, 3):
        GPIO.output(LED, ON)
        time.sleep(Dot)                # Wait Dot time
        GPIO.output(LED, OFF)          # LED OFF
        time.sleep(Gap)                # Wait Gap time

    time.sleep(0.5)                    # 0.5 second delay

    for i in range(0, 3):
        GPIO.output(LED, ON)           # LED ON
        time.sleep(Dash)               # Wait Dash time
        GPIO.output(LED, OFF)          # LED OFF
        time.sleep(Gap)                # Wait Gap time

    time.sleep(0.5)                    # 0.5 second delay

    for i in range(0, 3):
        GPIO.output(LED, ON)           # LED ON
        time.sleep(Dot)                # Wait Dot time
        GPIO.output(LED, OFF)          # LED OFF
        time.sleep(Gap)                # Wait Gap time

    time.sleep(2)                      # Wait 2 seconds
```

Figure 5.8: Program: **SOS.py**.

Suggestions: You could easily replace the LED with a buzzer to make the SOS signal audible. There are two types of buzzers: active and passive. Passive buzzers require an audio signal to be sent to them and the frequency of the output signal depends on the frequency of the supplied signal. Active buzzers are ON/OFF type devices and they produce audible sound when activated. In his project you can use an active buzzer with a transistor switch (any NPN type transistor can be used) as shown in Figure 5.9.

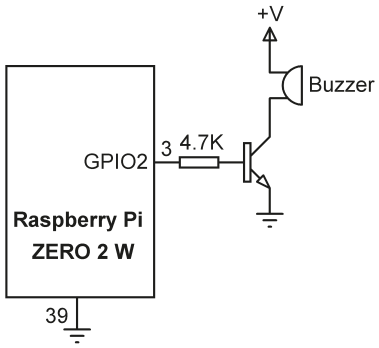


Figure 5.9: Using an active buzzer.

Note: You may find it easier to create and run your Python programs in Desktop mode using Thonny since the correct Python indentation is automatically placed in your code as you type the code.

5.4 Project 3: Binary counting with 8 LEDs

Description: In this project 8 LEDs are connected to the Zero 2 W GPIO pins. The LEDs count up in binary every second.

Aim: The aim of this project is to show how 8 LEDs can be connected to the Zero 2 W GPIO pins. In addition, the project shows how to group the LEDs as an 8-bit port and control them as a single port.

Block diagram: The block diagram of the project is shown in Figure 5.10.

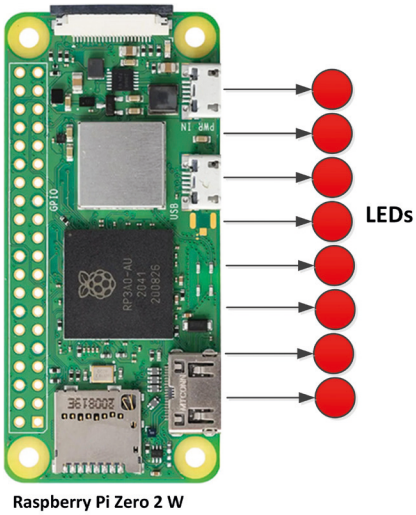


Figure 5.10: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.11. The LEDs are connected to 8 GPIO pins through 470-ohm current limiting resistors. The following 8 GPIO pins are grouped as an 8-bit port, where GPIO 2 is configured as the LSB and GPIO 9 is configured as the MSB:

	MSB							LSB
GPIO:	9	10	22	27	17	4	3	2
Pin no:	21	19	15	13	11	7	5	3

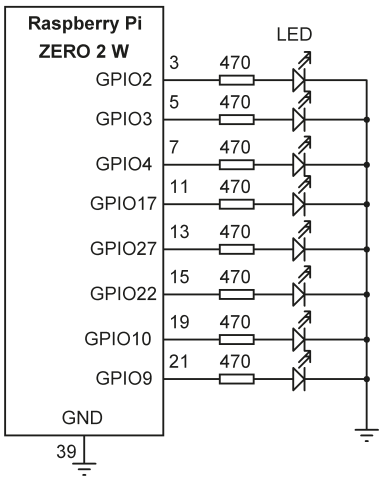


Figure 5.11: Circuit diagram of the project.

Construction: The project is constructed on a breadboard as shown in Figure 5.12. Female-to-male jumper cables are used to connect the LEDs to the GPIO ports. Notice that the short side of the LEDs must be connected to ground.

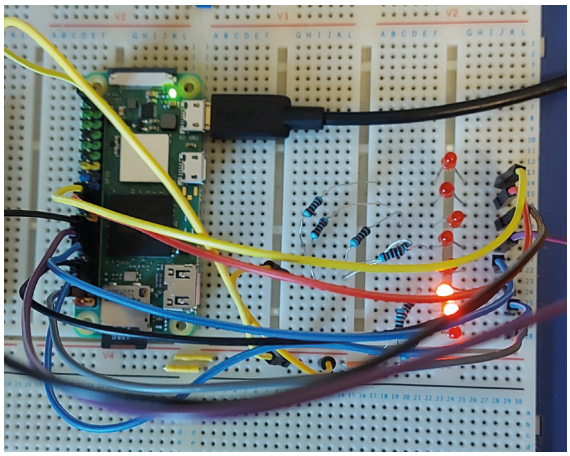


Figure 5.12: Constructing the project on a breadboard.

Project PDL: PDL (**P**rogram **D**escription **L**anguage) describes the operation of a program in simple English-like statements. The project PDL is shown in Figure 5.13.

```
BEGIN
       Import GPIO library
       Import time library
       CALL Configure Port to configure the port as output
       Set cnt = 0
       DO FOREVER
               CALL Port_Output with cnt
               Wait 1 second
               Increment cnt
       ENDDO
END

BEGIN/Configure_Port
       IF port is output THEN
               CALL GPIO.setup to configure the port as output
       ELSE
               CALL GPIO.setup to configure the port as input
       ENDIF
END/Configure_Port


BEGIN/Port_Ouput
       CALL GPIO.output to send the byte to the port
END/Port_Output
```

Figure 5.13: The Project PDL.

Program listing: The program is called **LEDCNT.py** and the listing is shown in Figure 5.14. The program was written using the **nano** text editor. At the beginning of the program the **RPi.GPIO** and the **time** modules are imported to the project. Then the pin numbering is configured to use the BCM notation. All the 8 GPIO channels used in the project are configured as outputs using function **Configure_Port**. Notice that the **Configure_Port** function is general and list **DIR** sets the directions of the GPIO pins. An "O" sets as an output and an "I" sets as an input. Then, a loop is formed to execute forever, and inside this loop the LEDs count up by one in binary. Variable **cnt** is used as the counter. Function **Port_Output** is used to control the LEDs. This function can take integer numbers from 0 to 255 and it converts the input number (x) into binary using the built-in function **bin**. Next, the leading "0b" characters are removed from the output string **b** (**bin** function inserts characters "0b" to the beginning of the converted string). Then, the converted string **b** is made up of 8 characters by inserting leading 0s. The string is then sent to the PORT bit by bit, starting from the most-significant bit (GPIO 9) position.

```
#--------------------------------------------------------------
#
#               BINARY UP COUNTING LEDs
#               =======================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# the LEDs count up in binary with one second delay between
# each output.
#
# Program: LEDCNT.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#--------------------------------------------------------------
import RPi.GPIO as GPIO                         # import GPIO library
import time                                     # import time library
GPIO.setwarnings(False)                         # disable warning messages
GPIO.setmode(GPIO.BCM)                          # set BCM pin numbering
PORT = [9, 10, 22, 27, 17,  4,  3,  2]          # port connections
DIR = ["0","0","0","0","0","0","0","0"]         # port directons

#
# This function configures the port pins as outputs ("0") or
# as inputs ("I")
#
def Configure_Port():
   for i in range(0, 8):
      if DIR[i] == "0":
         GPIO.setup(PORT[i], GPIO.OUT)
      else:
         GPIO.setup(PORT[i], GPIO.IN)
   return

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
```

```
      b = "0" + b                    # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         GPIO.output(PORT[i], 1)
      else:
         GPIO.output(PORT[i], 0)
   return
#
# Configure PORT to all outputs
#
Configure_Port()

#
# Main program loop. Count up in binary every second
#
cnt = 0
while True:
  Port_Output(cnt)                   # send cnt to port
  time.sleep(1)                      # wait 1 second
  cnt = cnt + 1                      # increment cnt
```

Figure 5.14: Program: **LEDCNT.py**.

Recommended modifications: Modify the program such that the LEDs count down every two seconds.

5.5 Project 4: Christmas Lights (randomly flashing 8 LEDs)

Description: In this project, 8 LEDs are connected to Zero 2 W GPIO pins. The LEDs flash randomly every 0.5 seconds just like fancy Christmas Lights. The aim of this project is to show how 8 LEDs can be connected to Zero 2 W GPIO pins. In addition, the project shows how to generate random numbers between 1 and 255 and then shows how to use these numbers to turn the individual LEDs ON and OFF randomly.

The block diagram and circuit diagram of this project are as in Figure 5.10 and Figure 5.11 respectively.

Project PDL: The project PDL is shown in Figure 5.15.

```
BEGIN
       Import GPIO library
       Import time library
       Import random number library
       CALL Configure_Port to configure the port as output
       DO FOREVER
```

```
                Get a random number between 1 and 255
                CALL Port_Output to send the number to the LEDs
                Wait 0.5 second
        ENDDO
END

BEGIN/Configure_Port
        IF port is output THEN
                CALL GPIO.setup to configure the port as output
        ELSE
                CALL GPIO.setup to configure the port as input
        ENDIF
END/Configure_Port


BEGIN/Port_Ouput
        CALL GPIO.output to send the byte to the port
END/Port_Output
```

Figure 5.15: Project PDL.

Program listing: The program is called **XMAS.py** and the listing is shown in Figure 5.16. The program was written using the **nano** text editor. At the beginning of the program, the **RPi.GPIO**, **time,** and **random** modules are imported to the project. Then, the pin numbering is configured to use the BCM notation. All the 8 GPIO channels used in the project are configured as outputs using function **Configure_Port** as in the previous project. Then, a loop is formed to execute forever, and inside this loop a random number is generated between 1 and 255. This number is used as an argument conveyed to function **Port_Output.** The binary pattern corresponding to the generated number is sent to the port which turns the LEDs ON or OFF in a random manner.

```
#-------------------------------------------------------------
#
#                      CHRISTMAS LIGHTS
#                      ================
#
# In this project 8 LEDs are connected to the Raspberry Pi 3
# and these LEDs flash randomly at 0.5 second intervals. The
# connections of the LEDs are to the following GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# generates random numbers between 1 and 255 and turns the
# LEDs ON and OFF depending on the generated number.
```

```
#
# Program: XMAS.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import RPi.GPIO as GPIO                         # import GPIO library
import time                                     # import time library
import random                                   # import random library
GPIO.setwarnings(False)                         # disable warning messages
GPIO.setmode(GPIO.BCM)                          # set BCM pin numbering
PORT = [9, 10, 22, 27, 17,  4,  3,  2]          # port connections
DIR = ["0","0","0","0","0","0","0","0"]         # port directons

#
# This function configures the port pins as outputs ("0") or
# as inputs ("I")
#
def Configure_Port():
   for i in range(0, 8):
      if DIR[i] == "0":
         GPIO.setup(PORT[i], GPIO.OUT)
      else:
         GPIO.setup(PORT[i], GPIO.IN)
   return

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
      b = "0" + b                       # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         GPIO.output(PORT[i], 1)
      else:
         GPIO.output(PORT[i], 0)
   return
#
# Configure PORT to all outputs
#
Configure_Port()
```

```
#
# Main program loop. Count up in binary every second
#
while True:
  numbr = random.randint(1, 255)       # generate a random number
  Port_Output(numbr)                   # send cnt to port
  time.sleep(0.5)                      # wait 0.5 second
```

Figure 5.16: Program: **XMAS.py**.

Recommended modifications: Modify the program such that 10 LEDs can be connected to the Zero 2 W and flashed randomly.

5.6 Project 5: Rotating LEDs with pushbutton switch control

Description: In this project 8 LEDs are connected to the Zero 2 W GPIO pins as in the previous project. In addition, a pushbutton switch is connected to one of the GPIO ports. The LEDs rotate in one direction when the button is not pressed, and in the opposite direction when the button is pressed. Only one LED is ON at any time. One second delay is inserted between each output. The aim of this project is to show how a pushbutton switch can be software-connected to a GPIO pin.

Block diagram: The block diagram of the project is shown in Figure 5.17.

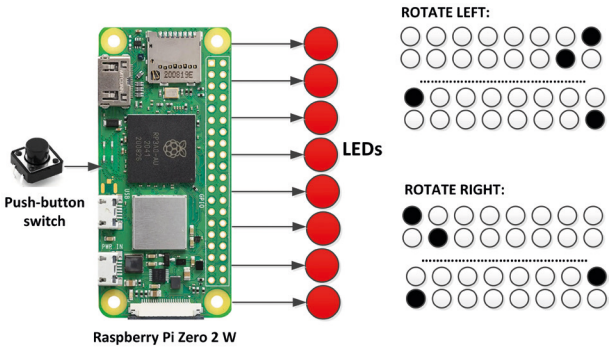


Figure 5.17: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.18. The LEDs are connected to 8 GPIO pins through 470-ohm current limiting resistors as in the previous project. The pushbutton switch is connected to GPIO 11 (pin 23) of the Zero 2 W. The pushbutton switch is connected through a 10 k-ohm resistor. When the switch is not pressed, the input is at logic 1. When the switch is pressed the input changes to logic 0.

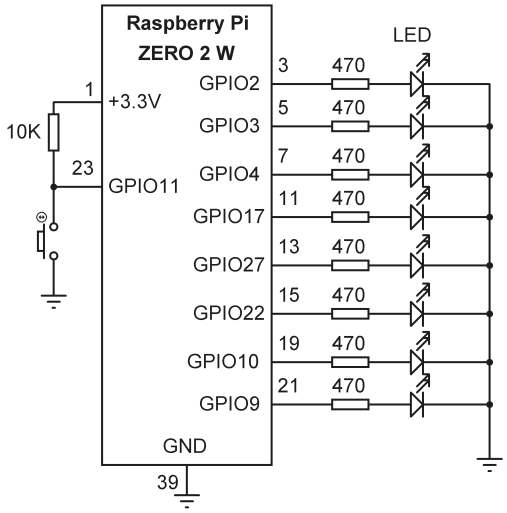


Figure 5.18: Circuit diagram of the project.

Project PDL: The project PDL is shown in Figure 5.19.

```
BEGIN
       Import GPIO library
       Import time library
       CALL Configure_Port to configure the port as output
       Configure GPIO 11 as input
       Set rot = 1
       DO FOREVER
              IF button is pressed THEN
                     Shift rot left
              ELSE
                     Shift rot right
              ENDIF
              CALL Port_Output to send rot to the LEDs
              Wait one second
       ENDDO
END

BEGIN/Configure_Port
       IF port is output THEN
              CALL GPIO.setup to configure the port as output
       ELSE
              CALL GPIO.setup to configure the port as input
       ENDIF
END/Configure_Port
```

```
BEGIN/Port_Ouput
       CALL GPIO.output to send the byte to the port
END/Port_Output
```

Figure 5.19: Project PDL.

Program listing: The program is called **rotate.py** and the listing is shown in Figure 5.20. The program was written using the **nano** text editor. At the beginning of the program, the **RPi.GPIO**, and **time** modules are imported to the project. Then, the pin numbering is configured to use the BCM notation. All the 8 GPIO channels used in the project are configured as outputs using function **Configure_Port** as in the previous project. Then, a loop is formed to execute forever and inside this loop variable **rot** is used as an argument to the **Port_Output** function. If the button is not pressed, then **rot** is shifted right and the LED ON sequence is from left to right (from MSB to LSB). If, on the other hand, the button is pressed, then the LED On sequence is from right to left (from LSB to MSB). A 1-second delay is inserted between each output.

```
#--------------------------------------------------------------------
#
#               ROTATING LEDs WITH PUSH-BUTTON
#               ==============================
#
# In this project 8 LEDs are connected to the Raspberry Pi 3.
# In addition, a push-button switch is connected to GPIO 11
# (pin 23) through resistors. Normally the output of the button
# is at logic 1 and goes to logic 0 when the button is pressed.
# The LEds rotate in one direction when the button is not pressed
# and in the opposite direction when the button is pressed.
#
# Connections of the LEDs are to the following GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# On second delay is inserted between each output.
#
# Program: rotate.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#--------------------------------------------------------------------
import RPi.GPIO as GPIO                        # import GPIO library
import time                                    # import time library
GPIO.setwarnings(False)                        # disable warning messages
GPIO.setmode(GPIO.BCM)                         # set BCM pin numbering
PORT = [9, 10, 22, 27, 17,  4,  3,  2]         # port connections
DIR =  ["0","0","0","0","0","0","0","0"]       # port directions
```

```
GPIO.setup(11, GPIO.IN)                        # GPIO 11 is input

#
# This function configures the port pins as outputs ("0") or
# as inputs ("I")
#
def Configure_Port():
   for i in range(0, 8):
      if DIR[i] == "0":
         GPIO.setup(PORT[i], GPIO.OUT)
      else:
         GPIO.setup(PORT[i], GPIO.IN)
   return

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
      b = "0" + b                       # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         GPIO.output(PORT[i], 1)
      else:
         GPIO.output(PORT[i], 0)
   return
#
# Configure PORT
#
Configure_Port()

#
# Main program loop. Rotate the LEDs
#
rot = 1
while True:
  Port_Output(rot)
  time.sleep(1)                         # wait 1 second
  if GPIO.input(11) == 0:               # button pressed?
    rot = rot << 1                      # shift left
    if rot > 128:                       # at the end
```

```
      rot = 1                          # back to beginning
  else:                                # button not pressed
    rot = rot >> 1                     # shift right
    if rot == 0:                       # at the end
      rot = 128                        # back to beginning
```

Figure 5.20: Program: **rotate.py**.

5.7 Project 6 – Morse Code exerciser with buzzer

Description: In this project, a buzzer is connected to GPIO 2 (pin 3) of the Zero 2 W. The user enters a text from the keyboard. The buzzer is then turned ON and OFF to sound the entered letters of the text in Morse code. The aim of this project is to show how a buzzer can be connected to a Zero 2 W, and also how to use various statements in Python programs.

Block diagram: The block diagram of the project is shown in Figure 5.21.

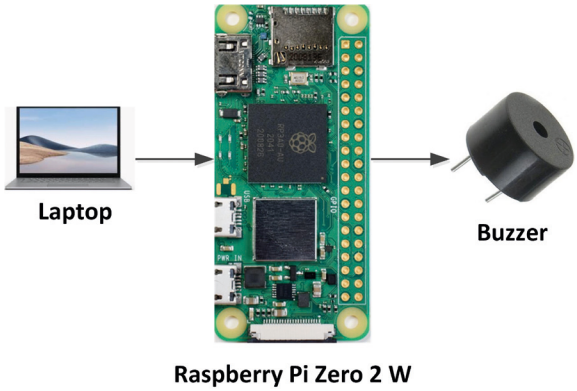


Figure 5.21: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.22 where an active buzzer is connected to GPIO 2 of the Zero 2 W.

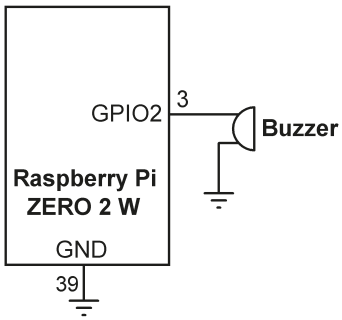


Figure 5.22: Circuit diagram of the project.

Project PDL: In a Morse code, each letter is made up of dots and dashes. Figure 5.23 shows the Morse code of all the letters in the English alphabet (this table can be extended by adding the Morse code of numbers and punctuation marks). The following rules apply to the timing of dots and dashes:

- The duration of a dot is taken as the unit time and this determines the speed of the transmission. Normally the speed of transmission is quoted in words per minute (wpm). The standard required minimum in Morse code based communication is 12 wpm.
- The duration of a dash is 3 unit times
- The time between each dot and dash is a unit time
- The time between the letters is 3 unit times
- The time between the words is 7 unit times

The unit time in milliseconds is calculated using the following formula:

Time (ms) = 1200 / wpm

In this project the Morse code is simulated at 10 wpm. Thus, the unit time is taken to be 1200 / 10 = 120 ms.

```
Letter    Morse code
    A:     .-
    B :   -...
    C :   -.-.
    D :   -..
    E :   .
    F :   ..-.
    G :   --.
    H :   ....
    I :   ..
    J :   .---
    K :   -.-
    L :   .-..
    M :   --
    N :   -.
    O :   ---
    P :   .--.
    Q :   --.-
    R :   .-.
    S :   ...
    T :   -
```

```
U :   ..-
V :   ...-
W :   .--
X :   -..-
Y :   -.--
Z :   --..
```

Figure 5.23: Morse code of English letters.

The project PDL is shown in Figure 5.24.

```
BEGIN
       Import GPIO library
       Import time library
       Configure channel 2 as an output pin
DO until text QUIT is received
       Read text from the keyboard
       IF space detected (inter-word character) THEN
               CALL Do_SPACE
       ELSE
               DO for all letters of the text
                       Find the Morse code
                       IF the code contains a dot THEN
                               CALL DO_DOT
                       ELSE IF code contains a dash THEN
                               CALL DO_DASH
                       ENDIF
               ENDDO
               Wait 2 seconds
ENDDO
Cleanup the I/O resources used
END

BEGIN/DO_DOT
       Send 1 to GPIO pin 2
       Wait 120ms (unit time)
       Send 0 to GPIO pin 2
       Wait 120ms (unit time)
END/DO_DOT


BEGIN/DO_DASH
       Send 1 to GPIO pin 2
       Wait 360ms (3 x unit time)
       Send 0 to GPIO pin 2
```

```
       Wait 120ms (unit time)
END/DO_DASH


BEGIN/DO_SPACE
       Wait 7 unit time
END/DO_SPACE
```

Figure 5.24 Project PDL.

Program listing: The program is called **morse.py** and the listing is shown in Figure 5.25. At the beginning of the program, the **RPi.GPIO** and the **time** modules are imported to the project. Then the pin numbering is configured to use the BCM notation. GPIO 2 is configured as an output pin and this is where the buzzer is connected to. The Morse code alphabet is stored in list **Morse_Code**. Function **DO_DOT** implement a single dot with a duration of one unit time. Function **DO_DASH** implement a single dash with duration of 3 unit times. Function **DO_SPACE** implements a space character with a duration of 7 unit times. The rest of the program is executed in a loop where a text is read from the keyboard and the buzzer sounds in such a way to represent the Morse code of this text. The program terminates if the user enters the text **QUIT**.

You should run the program from the command mode as follows:

pi@raspberrypi:~ $ **python3 morse.py**

```
#-----------------------------------------------------------------
#
#                  MORSE CODE EXERCISER
#                  ====================
#
# This project can be used to learn the Morse code. A buzzer is
# connected to GPIO 2 of the Zero 2 W.
#
# The program reads a text from the keyboard and then sounds the
# buzzer to simulate sending or receiving the Morse code of this
# text.
#
# In this project the Morse code speed is assumed to be 10 wpm,
# but can easily be changed by changing the parameter wpm.
#
# File  : morse.py
# Date  : December, 2022
# Author: Dogan Ibrahim
#-----------------------------------------------------------------
import RPi.GPIO as GPIO               # import GPIO module
```

```
import time                             # import time module
GPIO.setwarnings(False)

Pin = 2
words_per_minute = 10                   # define words per min
wpm = 1200/words_per_minute             # unit time in milliseconds
unit_time = wpm / 1000

GPIO.setmode(GPIO.BCM)                  # set BCM pin numbering
GPIO.setup(Pin, GPIO.OUT)               # Configure GPIO 2 as output

Morse_Code = {
         'A': '.-',
         'B': '-...',
         'C': '-.-.',
         'D': '-..',
         'E': '.',
         'F': '..-.',
         'G': '--.',
         'H': '....',
         'I': '..',
         'J': '.---',
         'K': '-.-',
         'L': '.-..',
         'M': '--',
         'N': '-.',
         'O': '---',
         'P': '.--.',
         'Q': '--.-',
         'R': '.-.',
         'S': '...',
         'T': '-',
         'U': '..-',
         'V': '...-',
         'W': '.--',
         'X': '-..-',
         'Y': '-.--',
         'Z': '--..'
         }

#
# This function sends a DOT (unit time)
#
def DO_DOT():
   GPIO.output(Pin, 1)
```

```
   time.sleep(unit_time)
   GPIO.output(Pin, 0)
   time.sleep(unit_time)
   return

#
# This function sends a DASH ( 3*unit time)
#
def DO_DASH():
   GPIO.output(Pin, 1)
   time.sleep(3*unit_time)
   GPIO.output(Pin, 0)
   time.sleep(unit_time)
   return

#
# This function sends inter-word space (7*unit time)
#
def DO_SPACE():
   time.sleep(7*unit_time)
   return

#
# Main program code
#
text = ""
while text != "QUIT":
   text = input("Enter text to send: ")
   if text != "QUIT":
      for letter in text:
         if letter == ' ':
            DO_SPACE()
         else:
            for code in Morse_Code[letter.upper()]:
               if code == '-':
                  DO_DASH()
               elif code == '.':
                  DO_DOT()
               time.sleep(unit_time)
   time.sleep(2)
```

Figure 5.25: Program: **morse.py**.

Recommended Modifications: An LED can be connected to the GPIO pin instead of the buzzer so that the Morse code can be seen in visual form.

5.8 Project 7: Electronic dice

Description: In this project 7 LEDs are arranged in the form of the faces of a dice, and a pushbutton switch is used. When the button is pressed, the LEDs turn ON to display numbers 1 through 6 as if on a real dice. The display is turned OFF after 3 seconds, ready for the next game. The aim of this project is to show how a dice can be constructed with 7 LEDs.

Block diagram: The block diagram of the project is shown in Figure 5.26.

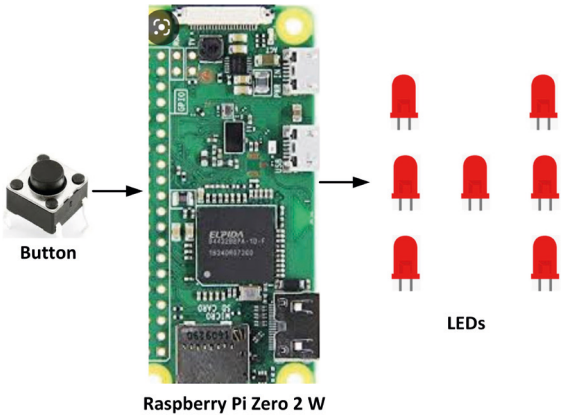


Figure 5.26: Block diagram of the project.

Figure 5.27 shows the LEDs that should be turned ON to display the 6 dice face numbers.

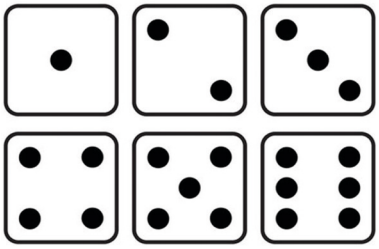


Figure 5.27: LED Dice.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.28. Here, 8 GPIO pins are collected to form a PORT. The following pins are used for the LEDs (there are 7 LEDs, but 8 port pins are used in the form of a byte where the most-significant bit position is not used):

PORT bit	7	6	5	4	3	2	1	0
GPIO:	9	10	22	27	17	4	3	2

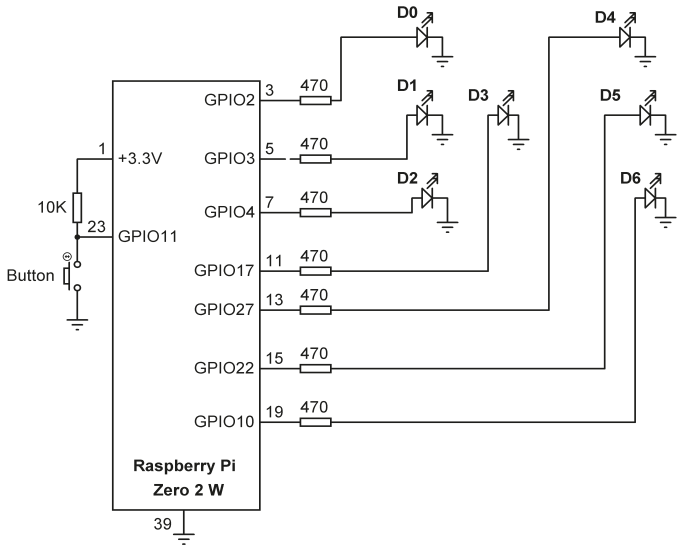


Figure 5.28: Circuit diagram of the project.

The pushbutton switch is connected to port pin GPIO 11 (pin 23).

Project PDL: The project PDL is shown in Figure 5.29.

```
BEGIN
       Import GPIO library
       Import time library
       Import random library
       Configure GPIO 11 as input
       Create DICE_NO list for LED bit patterns
       CALL Configure_Port to configure the PORT as output
              Declare callback handler (DICE) on pin GPIO 11
       DO FOREVER
              Wait for the button to be pressed
       ENDDO
END

BEGIN/Configure_Port
       IF port is output THEN
              CALL GPIO.setup to configure the port as output
       ELSE
              CALL GPIO.setup to configure the port as input
       ENDIF
END/Configure_Port


BEGIN/Port_Ouput
```

```
       CALL GPIO.output to send the byte to the port
END/Port_Output

BEGIN/DICE
       Generate a random number between 1 and 6
       Get the LED bit pattern corresponding to this number
       CALL Port_Output to send the data to the LEDs
       Wait 3 seconds
       Turn OFF all LEDs
END/DICE
```

Figure 5.29: Project PDL.

Table 5.1 gives the relationship between a dice number and the corresponding LEDs to be turned ON to imitate the faces of a real dice. For example, to display number 1 (i.e., only the middle LED is ON), you have to turn LED D3 ON. Similarly, to display number 4, you have to turn ON D0, D2, D4 and D6.

Required number	LEDs to be turned on
1	D3
2	D0, D6
3	D0, D3, D6
4	D0, D2, D4, D6
5	D0, D2, D3, D4, D6
6	D0, D1, D2, D4, D5, D6

Table 5.1: Dice number and LEDs to be turned ON.

The relationship between the required number and the data to be sent to the PORT to turn on the correct LEDs is given in Table 5.2. For example, to display dice number 2, you have to send hexadecimal 0x22 to the PORT. Similarly, to display number 5, you have to send hexadecimal 0x5D to the PORT, and so on.

Required number	PORT data (Hex)
1	0x08
2	0x41
3	0x49
4	0x55
5	0x5D
6	0x77

Table 5.2: Required number and PORT data.

Program listing: The program is called **dice.py** and the listing is shown in Figure 5.30.

At the beginning of the program, the **RPi.GPIO**, **time** and **random** modules are imported to the project. Then, port pins are declared as a list in variable PORT, and the direction of each pin is declared as output ("O") in a list variable called DIR. The bit pattern to be sent to the LEDs corresponding to each dice number is stored in hexadecimal format in a list called DICE_NO (see Table 5.2).

The pin numbering is configured to use the BCM notation. GPIO 11 is configured as an input pin and the pushbutton switch is connected to this pin to simulate the throwing of a dice. A callback routine called DICE is created so that when the button is pressed, the program jumps to this function. The callback function is set up to trigger when the button is pressed (i.e., when it goes from logic 1 to 0). Switch debouncing is also used in the callback routine. Inside the callback function, a random number is generated between 1 and 6. Then, the list called DICE_NO is used to find the LEDs that should be turned ON, and the required bit pattern is sent to the PORT to display the dice number. The program displays the dice number for 3 seconds and then all the LEDs are turned OFF to indicate that the program is ready for the next game.

```
#-----------------------------------------------------------------
#
#                    ELECTRONIC DICE
#                    ===============
#
# This program is an electronic dice. GPIO 11 of Zero 2 W is
# configured as an input and a push-button switch is connected to
# this port pin. When the button is pressed a random dice number is
# displayed between 1 and 6 on the LEDs.
#
# 7 LEDs are mounted on the breadboard in the form of the face of
# a real dice. The following GPIO pins are used for the LEDs (bit
# 7 is mot used):
#
#  Port pin: 7  6  5  4  3 2 1 0
#  GPIO    :   10 22 27 17 4 3 2
#
# The following PORT pins are used to construct the dice:
#
# D0    D4
# D1 D3 D5
# D2    D6
#
# Program: dice.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import RPi.GPIO as GPIO                        # import GPIO library
```

```
import time                             # import time library
import random                           # import random library
GPIO.setwarnings(False)

PORT = [9, 10, 22, 27, 17, 4, 3, 2]

DICE_NO = [0, 0x08, 0x41, 0x49, 0x55, 0x5D, 0x77]

#
# This function configures the port directions
#
def Configure_Port():
   for i in range (0, 8):
         GPIO.setup(PORT[i], GPIO.OUT)
   return

#
# This function sends a byte (8-bit) data to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading 0b
   diff = 8 - len(b)                    # find the difference
   for i in range (0, diff):
      b = "0" + b                       # insert leading 0s

   for i in range (0, 8):
      if b[i] == "1":
         GPIO.output(PORT[i], 1)
      else:
         GPIO.output(PORT[i], 0)
   return

#
# The program jumps here after the button is pressed
#
def DICE(dummy):
   n = random.randint(1, 6)             # generate a random number
   pattern = DICE_NO[n]                         # find the pattern
   Port_Output(pattern)                         # turn ON required LEDs
   time.sleep(3)                        # wait for 3 seconds
   Port_Output(0)                       # turn OFF all LEDs
   return

#
```

```
# Start of main program
#
Dice_Pin = 11
GPIO.setmode(GPIO.BCM)

GPIO.setup(Dice_Pin, GPIO.IN)
#
# Configure PORT as outputs
#
Configure_Port()
#
# Setup callback to function DICE when the button is pressed
#
GPIO.add_event_detect(Dice_Pin, GPIO.FALLING, bouncetime=50,
                      callback=DICE)
#
# Program waits here for the button to be pressed, then a random
# number is generated between 1 and 6 and is displayed on the LEDs
#
while True:
   pass                                                  # Do nothing
```

Figure 5.30: Program: **dice.py**.

5.9 Project 8: LED brightness control

Description: In this project, an LED and two buttons are connected to GPIO ports of the Zero 2 W. Pressing button **BRIGHTER** makes the LED brighter. Similarly, pressing button **DIMMER** makes the LED dimmer. PWM waveform is used to control the LED brightness where the Duty Cycle of the waveform is changed each time a button is pressed.

Background: PWM waves are frequently used in power control applications. The waveform is basically a positive squarewave with variable ON and OFF times. As shown in Figure 5.31, the total of the ON and OFF times is known as the period of the waveform. The ratio of the ON time to the period is known as the Duty Cycle and it is represented as a percentage. i.e.,

$$\text{Duty Cycle} = (T / P) \times 100\%$$

where T is the ON time, and P, the period (ON time + OFF time).

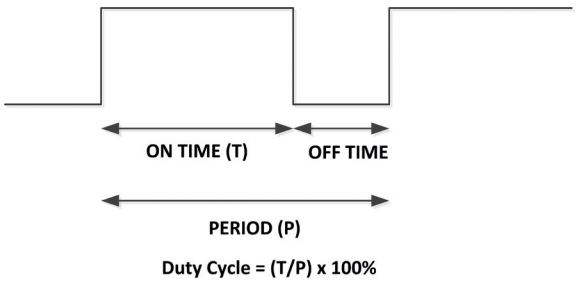


Figure 5.31: PWM waveform.

By varying the duty cycle from 0% to 100% you can easily control a load, say, a motor. For example, at 50% duty cycle the load receives half of the total power. Similarly, at 100% duty cycle the load receives full power.

The average value of the voltage applied to the load can be calculated by considering a general PWM waveform shown in Figure 5.31. The average value *A* of waveform *f(t)* with period *T* and peak value y_{max} and minimum value y_{min} is calculated from:

$$A=\frac{1}{T}\int_0^T f(t)\,dt$$

or

$$A=\frac{1}{T}\left(\int_0^{T_{ON}} y_{\max}\,dt+\int_{T_{ON}}^{T} y_{\min\,dt}\right)$$

In a PWM waveform, *y*min = 0 and the above equation becomes

$$A=\frac{1}{T}\left(T_{ON}\ y_{\max}\right)$$

or

$$A=D\ y_{\max}$$

As it can be seen from the above equation, the average value of the voltage supplied to the load is directly proportional to the duty cycle of the PWM waveform and by varying the duty cycle you control the average load voltage. Figure 5.32 shows the average voltage for different values of the duty cycle.

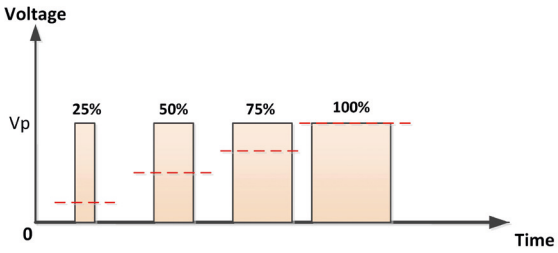


Figure 5.32: Average voltage (shown as a dashed line) supplied to a load.

It is interesting to notice that with correct low-pass filtering, the PWM can be used as a DAC if the MCU does not have a DAC channel. By varying the duty cycle you can effectively vary the average analog voltage supplied to the load.

Block diagram: The block diagram of the project is shown in Figure 5.33.

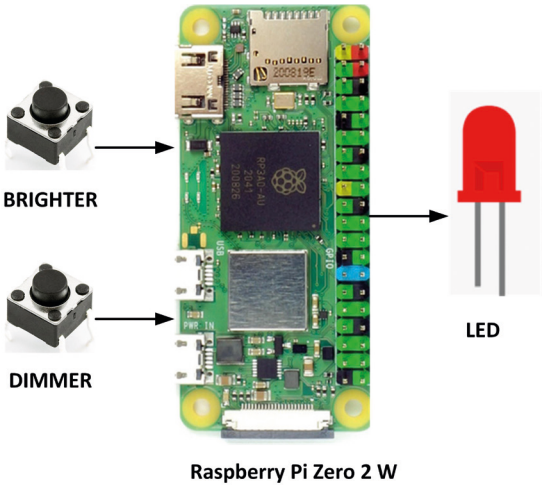


Figure 5.33: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.34. Buttons BRIGHTER and DIMMER are connected to port pins GPIO 2 (pin 3) and GPIO 3 (pin 5) respectively. The LED is connected to port pin GPIO 4 (pin 7).

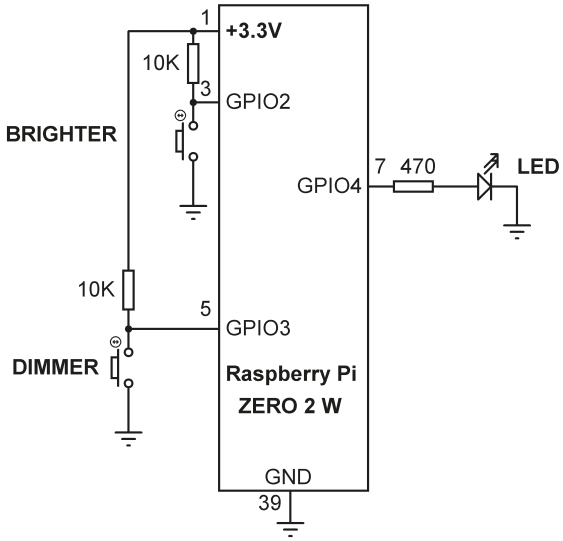


Figure 5.34: Circuit diagram of the project.

Program listing: The program is called **pwm.py**, and its listing is shown in Figure 5.35. At the beginning of the program, the **RPi.GPIO** module is imported to the project. Ports GPIO 2 and GPIO 3 are configured as inputs and LED port (GPIO 4) is configured as output. The default state of the buttons is logic 1 and they go to logic 0 when pressed.

GPIO pin 4 is configured as a PWM port with a frequency of 200 Hz using the following statement:

```
P = GPIO.PWM(LED, 200)
```

Then, the PWM is started with 50% Duty Cycle using the statement:

```
p.start(Duty)
```

where **Duty** is set to 50 initially.

Inside the program loop the state of both buttons are checked. Every time button **BRIGHTER** is pressed, the Duty Cycle is incremented by 5 until it reaches to maximum (100%). Similarly, every time button **DIMMER** is pressed, the Duty Cycle is decremented by 5 until it reaches the minimum (0%). The program runs until stopped by the user.

```
#-------------------------------------------------------------------------
#               LED BRIGHTNESS CONTROL
#               ======================
#
# In this program two buttons and an LED are used. Pressing button
```

```
# BRIGHTER makes the LED brighter. Simialrly, pressing button DIMMER
# makes the LED dimmer. PWM waveform is used to drive the LED where
# the Duty Cycle is changed (increased or decreased by 5) each time
# a button is pressed. PWM frequency is set to 200 Hz
#
# File  : pwm.py
# Date  : December, 2022
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import RPi.GPIO as GPIO                # import GPIO module
GPIO.setwarnings(False)
import time

GPIO.setmode(GPIO.BCM)
BRIGHTER = 2                           # button BRIGHTER
DIMMER = 3                             # button DIMMER
LED = 4                                # LED

GPIO.setup(LED, GPIO.OUT)              # configure LED output
GPIO.setup(BRIGHTER, GPIO.IN)          # configure BRIGHTER input
GPIO.setup(DIMMER, GPIO.IN)            # configure DIMMER input

p = GPIO.PWM(LED, 200)                 # generate PWM waveform
Duty = 50                              # Initial Duty Cycle
p.start(Duty)                          # set duty cycle to 50%

while True:                            # wait here
   p.ChangeDutyCycle(Duty)             # set/change Duty Cycle
   if GPIO.input(BRIGHTER) == 0:       # if BRIGHTER pressed
       if Duty < 100:                  # if not max Duty Cycle
          Duty = Duty + 5              # increase Duty Cycle
          time.sleep(0.25)             # wait a bit
       else:
          Duty = 100                   # set to max Duty Cycle

   elif GPIO.input(DIMMER) == 0:       # if DIMMER is pressed
       if Duty > 0:
          Duty = Duty - 5              # decrease Duty Cycle
          time.sleep(0.25)             # wait a bit
       else:
          Duty = 0                     # set to min Duty Cycle
```

Figure 5.35: Program: **pwm.py**.

The PWM library supports the following commands:

P = GPIO.PWM(channel, frequency)	-	Configure channel for PWM with specified frequency
p.start(Duty)	-	Start PWM with specified Duty Cycle
p.stop()	-	Stop PWM
p.ChangeFrequency(frequency)	-	Change PWM frequency
p.ChangeDutyCycle(Duty)	-	Change Duty Cycle

5.10 Project 9: Lucky day of the week

Description: In this project, 7 LEDs are positioned in the form of a circle and are connected to the Zero 2 W. Each LED is assumed to represent a day of the week. Pressing a button generates a random number between 1 and 7 and lights up only one of the LEDs. The day name corresponding to this LED is assumed to be your lucky day of the week!

Block diagram: Figure 5.36 shows the block diagram of the project.

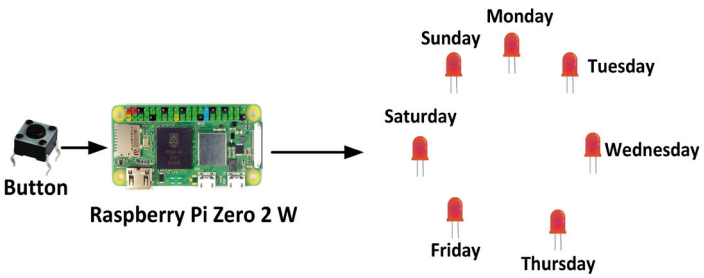


Figure 5.36: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.37, where 7 LEDs are connected to the Zero 2 W through current limiting resistors. The button is connected to GPIO11 (pin 23). Normally, the output of the button is at logic 1 and goes to logic 0 when the button is pressed.

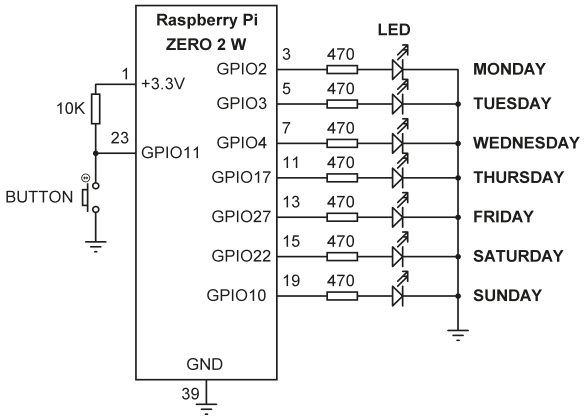


Figure 5.37: Circuit diagram of the project.

Program listing: Figure 5.38 shows the program listing (Program: **LuckyDay.py**). At the beginning of the program, all the 8 LED GPIO pins are combined into a single port and is addressed as a single 8-bit port using function **PORT_Output**. Notice that the system date and time are jointly used as the seed value to the random number generator function so that different values are available every time the program is started. An integer random number is generated between 1 and 7 and this number is used to turn ON one of the LEDs corresponding to a day of the week.

```
#-------------------------------------------------------------
#              LUCKY DAY OF THE WEEK
#              =====================
#
# In this program 7 LEDs are connected to Zero 2 W where each
# LED represents a day of the week. Pressing a button
# turns ON one of the LEDs randomly and this corresponds to
# your lucky day of the week
#
# Author: Dogan Ibrahim
# File  : LuckyDay.py
# Date  : December, 2022
#-------------------------------------------------------------
import RPi.GPIO as GPIO                # import GPIO library
import time                            # import time library
GPIO.setwarnings(False)                # disable warning messages
GPIO.setmode(GPIO.BCM)                 # set BCM pin numbering
import random                          # random no
from datetime import datetime

PORT = [9, 10, 22, 27, 17,  4,  3,  2] # port connections
DIR = ["0","0","0","0","0","0","0","0"]        # port directons
BUTTON = 11
```

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

#
# This function configures the port pins as outputs ("0") or
# as inputs ("I")
#
def Configure_Port():
   for i in range(0, 8):
      if DIR[i] == "0":
         GPIO.setup(PORT[i], GPIO.OUT)
      else:
         GPIO.setup(PORT[i], GPIO.IN)
   return

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                          # convert into binary
   b = b.replace("0b", "")             # remove leading "0b"
   diff = 8 - len(b)                   # find the length
   for i in range (0, diff):
      b = "0" + b                      # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         GPIO.output(PORT[i], 1)
      else:
         GPIO.output(PORT[i], 0)
   return
#
# Configure PORT to all outputs
#
Configure_Port()

#
# Main program loop, check if Button is pressed
#
print("Press the Button to display your lucky number...")
dt = datetime.today()
seconds = dt.timestamp()
random.seed(seconds)

while GPIO.input(BUTTON) == 1:         # If Button not pressed
   pass
```

```
r = random.randint(1, 7)              # Generate random number
r = pow(2, r-1)                       # LED to be turned ON
Port_Output(r)                        # Send to LEDs
```

Figure 5.38: Program: **LuckyDay**.

5.11 Project 10: Using an I²C LCD: Seconds counter

Description: In this project an I2C type LCD is connected to the Zero 2 W. The program counts up in seconds and displays on the LCD. The aim of this project is to show how an I2C type LCD can be used in Raspberry Pi Zero 2 W projects.

Block Diagram: Figure 5.39 shows the block diagram of the project.

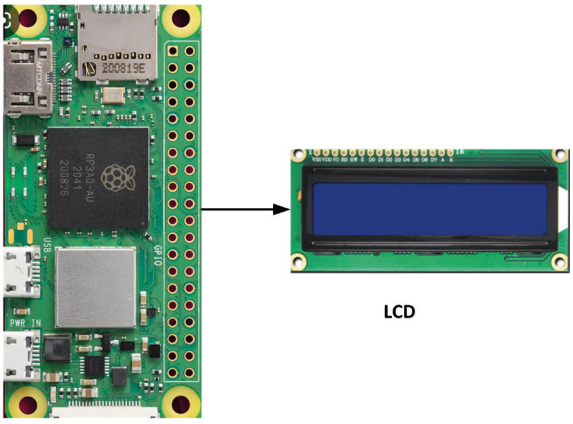


Figure 5.39: Block diagram of the project.

Circuit Diagram: The I2C LCD has 4 pins: GND, +V, SDA, and SCL. SDA is connected to pin GPIO 2 and SCL is connected to pin GPIO 3. The +V pin of the display should be connected to +5 V (pin 2) on the Zero 2 W. Notice that Zero 2 W pins are not +5 V tolerant, but the I2C LCD operates with +5 V where its SDA and SCL pins are pulled to +5 V. It is not a good idea to connect the LCD directly to the Zero 2 W as it can damage the latter"s I/O circuitry. There are several solutions here. One solution is to remove the I2C pull-up resistors on the LCD module. The other option is to use an I2C device which does operate at +3.3 V. The other solution is to use a bidirectional +3.3 V to +5 V logic level converter chip. In this project, you will use the TXS0102 bidirectional logic level converter chip like the one shown in Figure 5.40. The circuit diagram of the project is shown in Figure 5.41.

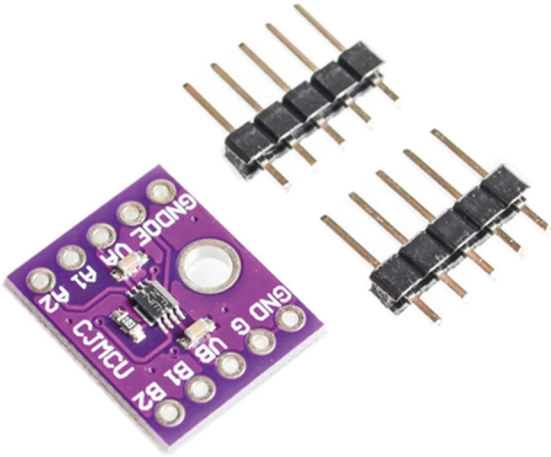


Figure 5.40: Logic level converter.

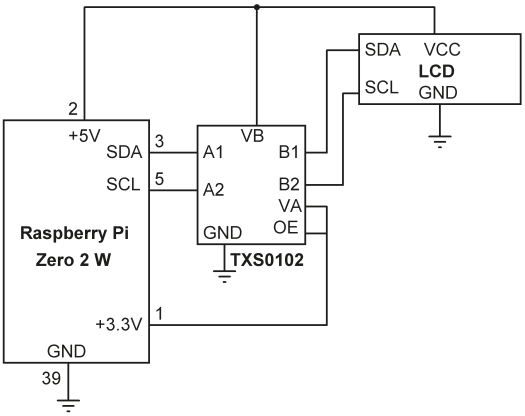


Figure 5.41: Circuit diagram of the project.

The LCD used in this project is based on the I2C interface. I2C is a multi-slave, multi-master, single-ended serial bus used to attach low-speed peripheral devices to microcontrollers. The bus consists of only two wires called SDA and SCL where SDA is the data line and SCL is the clock line and up to 1008 slave devices can be supported on the bus. Both lines must be pulled up to the supply voltage by suitable resistors. The clock signal is always generated by the bus master. The devices on the I2C bus can communicate at 100 kHz or 400 kHz.

Figure 5.42 shows the front and back of the I2C based LCD. Notice that the LCD has a small board mounted at its back to control the I2C interface. The LCD contrast is adjusted through the small potentiometer mounted on this board. A jumper is provided on this board to disable the backlight if required.

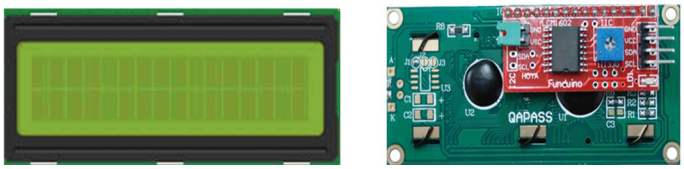


Figure 5.42: I2C-based LCD (front and back views).

Program Listing: Before using the I2C pins of the Zero 2 W you have to enable the I2C peripheral interface on the device. The steps for this are as follows:

- Start the configuration menu from the command prompt:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Go down the menu to **Interface Options**
- Go down and select **I2C**
- Enable the I2C interface
- Select **Finish** to complete

Now you have to install the I²C library on your Zero 2 W. The steps are as follows:

- Enter the following commands from the command menu:

  pi@raspberrypi:~ $ **sudo apt install -y i2c-tools python3-smbus**
  pi@raspberrypi:~ $ **sudo reboot**

- Connect the I²C display to your Zero 2 W and enter the following command to test the installation and LCD hardware address:

  pi@raspberrypi:~ $ **sudo i2cdetect –y 1**

  You should see a table similar to the one shown below. A number in the table means that the LCD has been recognized correctly and the I²C slave address of the LCD is shown in the table. In this example, the LCD address is 27:

```
1  2 3  4  5  6  7  8  9  a  b  c  d  e  f
00:                 -- -- -- -- -- -- -- -- -- -- -- --
10:    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:    -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30:    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:    -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
70:     --  --  --  --  --  --  --  --  --   --  --  --  --  --  --
```

Now install an I2C LCD library so you can send commands and data to your LCD. There are many Python libraries available for the I2C type LCDs. The one chosen here is called the **RPLCD** and it can be installed as follows:

- pi@raspberrypi:~ $ **sudo pip3 install RPLCD**

The I2C LCD library supports the following functions (see the RPLCD I2C LCD library documentation for more details):

lcd_clear()	clear LCD and set to home position
cursor_pos = (row, column)	position cursor
lcd.write_string(text)	display text
lcd.write_string(text\r\n)	display text followed by new line
lcd.home()	home cursor
lcd.cr()	insert carriage-return
lcd.lf()	insert linefeed
lcd.crlf()	insert carriage-return and linefeed

Testing the I2C bus and LCD

It is worthwhile testing the LCD display before developing any controller program. Create the simple test program shown in Figure 5.43 (Program: **lcdtest.py**) which sends the text **MY LCD** to the LCD.

If you have used a text editor (like **nano**) to create the program, you can run it by entering the following command:

pi@raspberrypi:~ $ **python3 lcdtest.py**

Alternatively, if you have used the **Thonny** IDE, then just run the program.

```
#---------------------------------------
#
#      LCD TEST PROGRAM
#      ================
#
# Author: Dogan Ibrahim
# File  : lcdtest.py
# Date  : December, 2022
#---------------------------------------
from RPLCD.i2c import CharLCD
lcd=CharLCD('PCF8574', 0x27)

lcd.clear()
```

```
lcd.home
lcd.write_string("MY LCD")
```

Figure 5.43: LCD test program.

You are now all set to write your program. Figure 5.44 shows the program listing (**lcd.py**). At the beginning of the program, libraries RPi.GPIO, timer, and the LCD driver library are all imported to the program. The heading **SECONDS COUNTER** is displayed for 2 seconds. The program then clears the LCD screen and enters into an infinite loop. Inside this loop, variable **cnt** is incremented every second and the total value of **cnt** is displayed on the LCD continuously.

```
#------------------------------------------------------------
#             I2C LCD SECONDS COUNTER
#             ======================
#
# In this program an I2C LCD is connected to the Zero 2 W.
# The program counts up in seconds and displays on the LCD.
#
# At the beginning of the program the text SECONDS COUNTER is
# displayed for 2 seconds
#
# Program: lcd.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#------------------------------------------------------------
import time
from RPLCD.i2c import CharLCD
lcd=CharLCD('PCF8574', 0x27)                  # Init LCD

lcd.clear()                                   # clear LCD
lcd.write_string("SECONDS COUNTER")           # display string
time.sleep(2)                                 # wait 2 seconds

cnt = 0                                       # initialize cnt
lcd.clear()                                   # clear lcd

while True:                                   # infinite loop
  cnt = cnt + 1                               # increment count
  lcd.cursor_pos = (0, 0)                     # Top row
  lcd.write_string(str(cnt))                  # display cnt
  time.sleep(1)                               # wait one second
```

Figure 5.44: Program: **lcd.py**.

5.12 Project 11: Analog temperature sensor thermometer

Description: In this project, an analog temperature sensor chip is used to measure and then display the ambient temperature on the monitor, every second. Because the Zero 2 W does not have any analog-to-digital converters (ADC) on-board, an external ADC chip is used in this project. The aim of this project is to show ways of connecting an external ADC chip to a Zero 2 W and how the temperature can be read and displayed on the monitor using an analog temperature sensor chip.

Block Diagram: Figure 5.45 shows the block diagram of the project.

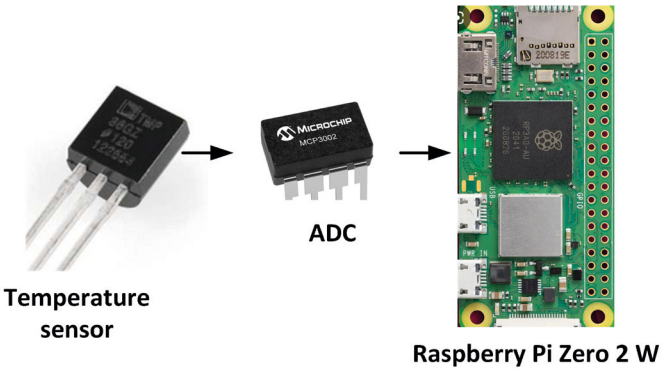


Figure 5.45: Block diagram of the project.

Circuit Diagram: The dual MCP3002 ADC chip is used in this project to provide analog input capability to the Zero 2 W. This chip has the following features:

- 10-bit resolution (0 to 1023 quantization levels)
- On-chip sample and hold
- SPI bus compatible
- Wide operating voltage (+2.7 V to +5.5 V)
- 75 Ksps sampling rate
- 5 nA standby current, 50 µA active current

The MCP3002 is a successive approximation 10-bit ADC with on-chip sample and hold amplifier. The device is programmable to operate as either differential input pair or as dual single-ended inputs. The device is offered in 8-pin package. Figure 5.46 shows the pin configuration of the MCP3002.

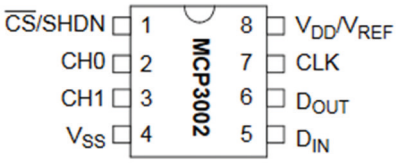


Figure 5.46: Pin configuration of the MCP3002.

The pin definitions are as follows:

Vdd/Vref:	Power supply and reference voltage input
CH0:	Channel 0 analog input
CH1:	Channel 1 analog input
CLK:	SPI clock input
DIN:	SPI serial data in
DOUT:	SPI serial data out
CS/SHDN:	Chip select/shutdown input

In this project, the supply voltage and the reference voltage are set to +3.3 V. Thus, the digital output code is given by:

$$\text{Digital output code} = 1024 \times V_{in} / 3.3$$

or $$\text{Digital output code} = 310.30 \times \text{Vin}$$

each quantization level corresponds to 3300 mV / 1024 = 3.22 mV. Thus, for example, input data "00 0000001" corresponds to 3.22 mV, "00 0000010" corresponds to 6.44 mV, and so on.

The MCP3002 ADC has two configuration bits: SGL/DIFF and ODD/SIGN. These bits follow the sign bit and are used to select the input channel configuration. The SGL/DIFF is used to select single ended or pseudo-differential mode. The ODD/SIGN bit selects which channel is used in single ended mode and is used to determine polarity in pseudo-differential mode. In this project, you are using channel 0 (CH0) in single-ended mode. According to the MCP3002 datasheet, SGL/DIFF and ODD/SIGN must be set to 1 and 0, respectively.

Figure 5.47 shows the circuit diagram of the project. A type TMP36DZ analog temperature sensor chip is connected to CH0 of the ADC. TMP36DZ is a 3 terminal small sensor chip with pins: Vs, GND, and Vo. Vs is connected to +3.3V, GND is connected to system ground, and Vo is the analog output voltage. The temperature in degrees Centigrade is given by:

$$\text{Temperature} = (\text{Vo} - 500) / 10$$

Where Vo is the sensor output voltage in millivolts.

The CS, Dout, CLK, and Din pins of the ADC are connected to SPI pins CE0, MISO, SCLK, and MOSI pins of the Raspberry Pi 3, respectively.

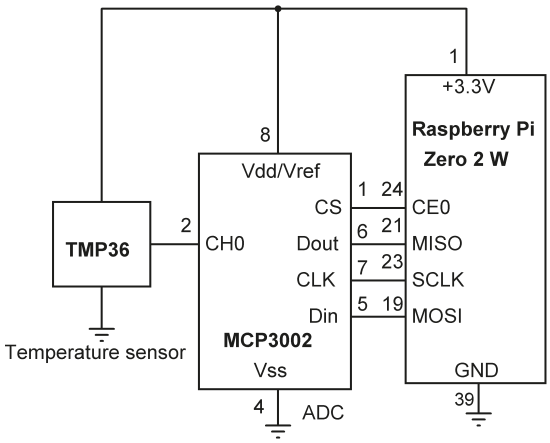


Figure 5.47: Circuit diagram of the project.

Program listing: Figure 5.48 shows the program listing (program: **tmp36.py**). The function **get_adc_data** is used to read the analog data, where the channel number (**channel_no**) is specified in the function argument as 0 or 1. Notice that you have to send the start bit, followed by the SGL/DIFF and ODD/SIGN bits and the MSBF bit to the chip.

It is recommended to send leading zeroes on the input line before the start bit. This is often done when using microcontroller-based systems that are required to send 8 bits at a time.

The following data can be sent to the ADC (SGL/DIFF = 1 and ODD/SIGN = channel_no) as bytes with leading zeroes for more stable clock cycle. The general data format is:

0000 000S DCM0 0000 0000 0000

Where, S = start bit, D = SGL/DIFF bit, C = ODD/SIGN bit, M = MSBF bit

For channel 0: 0000 0001 1000 0000 0000 0000 (0x01, 0x80, 0x00)

For channel 1: 0000 0001 1100 0000 0000 0000 (0x01, 0xC0, 0x00)

Notice that the second byte can be sent by adding 2 to the channel number (to make it 2 or 3) and then shifting 6 bits to the left as shown above to give 0x80 or 0xC0.

The chip returns 24 bit data (3 bytes) and you must extract the correct 10 bit ADC data from this 24 bit data. The 24 bit data is in the following format ("X" is a don"t-care bit):

XXXX XXXX XXXX DDDD DDDD DDXX

Assuming that the returned data is stored in 24 bit variable ADC, you have:

ADC[0] = "XXXX XXXX"
ADC[1] = "XXXX DDDD"
ADC[2] = "DDDD DDXX"

Thus, you can extract the 10 bit ADC data with the following operations:

(ADC[2] >> 2) so, low byte = "00DD DDDD"

and

(ADC[1] & 15) << 6) so, high byte = "DD DD00 0000"

Adding the low byte and the high byte you get the 10-bit converted ADC data as:

DD DDDD DDDD

The SPI bus on the Raspberry Pi supports the following functions:

Function	**Description**
open (0,0)	Open SPI bus 0 using CE0
open (0,1)	Open SPI bus 0 using CE1
close()	disconnect the device from the SPI bus
writebytes([array of bytes])	Write an array of bytes to SPI bus device
readbytes(len)	Read **len** bytes from SPI bus device
xfer2([array of bytes])	Send an array of bytes to the device with CEx asserted at all times
xfer([array of bytes])	Send an array of bytes de-asserting and asserting CEx with every byte transmitted

The module **spidev** must be imported at the beginning of the program before any of the above functions are called. Also, you must enable the SPI interface on your Zero 2 W in the configuration menu. The steps are:

- Get into command mode (e.g., from Putty)
- Enter the following command:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Select the **Interface Options**

- Enable **SPI** interface
- Finish and exit the configuration menu

At the beginning of the program in Figure 5.48, modules RPi.GPIO and spidev are imported to the program and an instance of the SPI is created. Function **get_adc_data** reads the temperature from sensor chip MCP3002 and returns a digital value comprised between 0 and 1023. This value is then converted into millivolts, 500 is subtracted from it, and the result is divided by 10 to find the temperature in degrees Centigrade. The temperature is displayed on the monitor every second.

```
#-----------------------------------------------------------------
#               ANALOG TEMPERATURE SENSOR
#               =========================
#
# In this project a TMP36 type analog temperature chip is used
# to measure the ambient temperature. The temperature is read
# using a MCP3002 type ADC chip. The result is converted into
# degrees Centigrade and is displayed on the monitor
#
# Program: tmp36.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import RPi.GPIO as GPIO
import spidev
import time
GPIO.setwarnings(False)

#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)                          # we are using CE0 for CS
spi.max_speed_hz = 4000

GPIO.setmode(GPIO.BCM)

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv
```

```
#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0                  # convert to mV
   Temp = (mV - 500.0) / 10.0                  # temperature in C
   print("Temperature = %5.2f " %Temp)         # display temperature
   time.sleep(1)                               # wait one second
```

Figure 5.48: Program: **tmp36.py**.

A typical display on the monitor is shown in Figure 5.49.

```
pi@raspberrypi:~ $ python3 tmp36.py
Temperature = 20.65
Temperature = 20.65
Temperature = 20.32
Temperature = 20.32
Temperature = 20.32
Temperature = 20.65
Temperature = 20.32
Temperature = 20.97
Temperature = 21.61
Temperature = 21.94
Temperature = 22.26
Temperature = 21.94
```

Figure 5.49: Typical display.

5.13 Project 12: Analog temperature sensor thermometer with LCD output

Description: This project is similar to the previous one but here the temperature is displayed on LCD every 2 seconds

Block diagram: Figure 5.50 shows the block diagram of the project

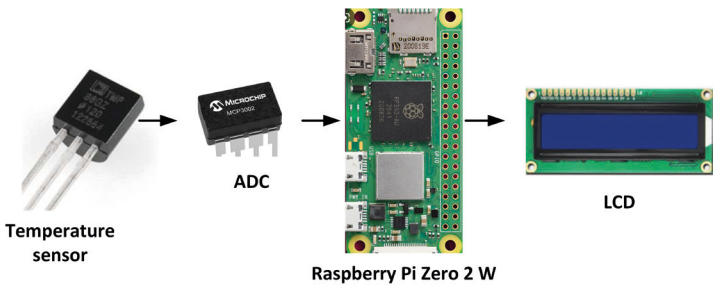


Figure 5.50: Block diagram of the project.

Circuit diagram: The circuit diagram is shown in Figure 5.51. The I^2C LCD is connected

to Zero 2 W using a voltage translator chip. The ADC chip is connected as in the previous project.

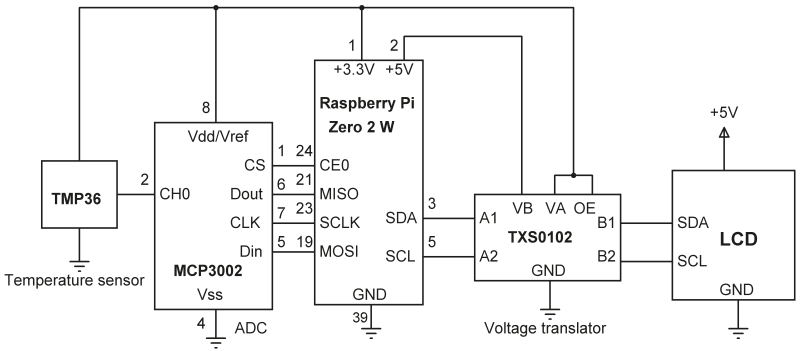


Figure 5.51: Circuit diagram of the project.

Program listing: Figure 5.52 shows the program listing (Program: **tmp36lcd.py**). The program is similar to the one in Figure 5.48 but here an I2C LCD is used to display the temperature. At the beginning of the program, the LCD library and other libraries used in the program are imported. The LCD is configured to 16 characters and 2 rows. Inside the program loop, the temperature is read from TMP36DZ via the ADC and gets converted into degrees Celsius and subsequently displayed on the LCD.

```
#-----------------------------------------------------------------
#               ANALOG TEMPERATURE SENSOR - LCD OUTPUT
#               ======================================
#
# In this project a TMP36 type analog temperature chip is used
# to measure the ambient temperature. The temperature is read
# using a MCP3002 type ADC chip. The result is converted into
# degrees Centigrade and is displayed on I2C LCD
#
# Program: tmp36lcd.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import RPi.GPIO as GPIO
import spidev
import time
from RPLCD.i2c import CharLCD
lcd=CharLCD('PCF8574', 0x27)
GPIO.setwarnings(False)

#
# Create SPI instance and open the SPI bus
```

```
#
spi = spidev.SpiDev()
spi.open(0,0)                           # we are using CE0 for CS
spi.max_speed_hz = 4000

GPIO.setmode(GPIO.BCM)

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv

lcd.clear()
lcd.home()
lcd.write_string("TEMP - TMP36")
time.sleep(2)

#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   lcd.clear()
   lcd.home()
   lcd.write_string("Temperature")            # heading
   lcd.cursor_pos = (1, 0)                    # cursor at (1, 0)
   mV = adc * 3300.0 / 1023.0                 # convert to mV
   Temp = (mV - 500.0) / 10.0                 # temperature in C
   T = str(Temp)[:5]
   lcd.write_string(T)                        # display temperature
   time.sleep(2)                              # wait two seconds
```

Figure 5.52: Program: **tmp36lcd.py**.

Figure 5.53 shows an example display.

Figure 5.53: Typical display.

5.14 Project 13: Reaction timer

Description: This is a reaction timer project. The user presses a button as soon as he/she sees an LED lighting. The time delay between seeing the light and pressing the button is measured and displayed on the monitor. The LED then turns OFF and the process is repeated after a random delay of 1 to 10 seconds. The aim of this project is to show how the time can be read and how a simple reaction timer project can be designed.

Block Diagram: Figure 5.54 shows the block diagram of the project.

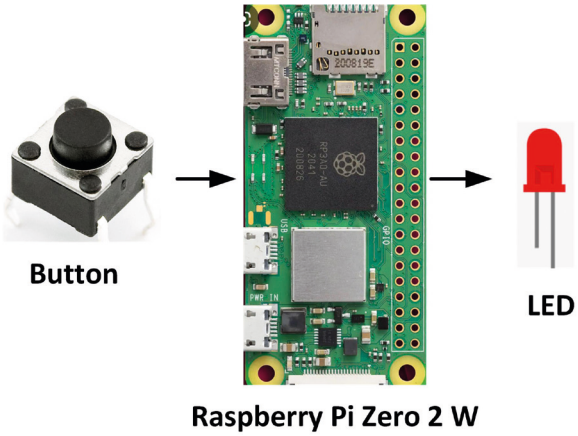


Figure 5.54: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is simple (Figure 5.55), consisting of an LED and a pushbutton switch. The LED and the button are connected to GPIO 2 and GPIO 3 port pins respectively. The button is connected such that by default its output is at logic 1, and goes to logic 0 when pressed.

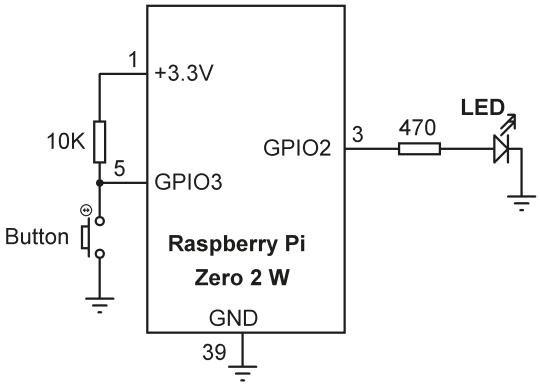


Figure 5.55: Circuit diagram of the project.

Program listing: The program is called **reaction.py** and its listing is given in Figure 5.56. At the beginning of the program, RPi.GPIO, time, and random libraries are imported. The

button is configured as an input and the LED, as an output. The program runs in a loop where the system time is recorded as soon as the LED is turned ON. The program waits for the user to press the button, and the system time is read again at this moment. The difference between this time and the first time is displayed as the reaction time of the user. This process repeats after a random delay of 1 to 10 seconds. Notice that the floating-point function **time.time()** returns the time in seconds since the epoch.

```
#-------------------------------------------------------
#                 REACTION TIMER
#                 ==============
#
# This is a reaction timer program. The user presses a button
# as soon as he/she see a light. The time between seeing the
# light and pressing the button is measured and is displayed
# in milliseconds as the reaction time of the user. The light
# comes ON after a random number of seconds between 1 and 10
# seconds.
#
# Program: reaction.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import RPi.GPIO as GPIO
import time
import random

LED = 2                                 # LED at GPIO 2
Button = 3                              # Button at GPIO 3

GPIO.setwarnings(False)                 # Disable warnings
GPIO.setmode(GPIO.BCM)                  # BCM mode

#
# LED is output, button is input
#
GPIO.setup(Button, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

GPIO.output(LED, 0)                     # LED OFF to start with

#
# Start of main program
#
while True:
   T = random.randint(1, 10)
   time.sleep(T)
```

```
GPIO.output(LED, 1)
start_time = time.time()            # start time
while (GPIO.input(Button) == 1):    # wait until button pressed
   pass
end_time = time.time()
diff_time = 1000.0*(end_time - start_time)
diff_int = int(diff_time)
print("Reaction time=%d " %diff_int)
GPIO.output(LED, 0)
time.sleep(3)
```

Figure 5.56: Program: **reaction.py**.

A typical display is shown in Figure 5.57.

```
pi@raspberrypi:~ $ python3 reaction.py
Reaction time=3364
Reaction time=435
Reaction time=592
Reaction time=8683
```

Figure 5.57: Typical display.

5.15 Project 14: Vehicle parking aid

Description: This is a vehicle parking aid project to help a person park a car safely and easily. A pair of ultrasonic transmitter/receiver sensors is mounted in the front and back of a vehicle to sense the distance to the objects . A buzzer sounds if the sensors are too close to the objects in front of them. In this project, the safe distance is assumed to be 10 cm. The aim of this project is to show how ultrasonic sensors can be attached to a Zero 2 W and how distance can be measured using these sensors.

Block Diagram: Figure 5.58 shows the block diagram of the project.

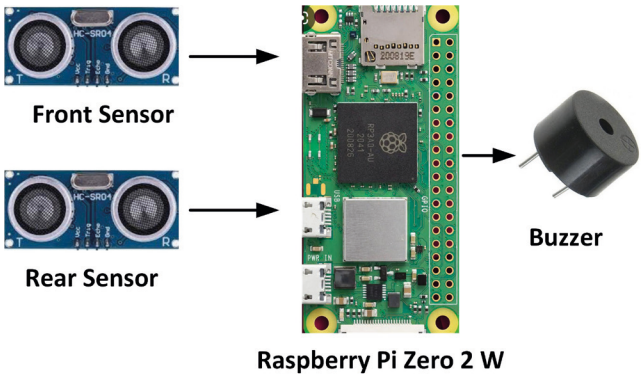


Figure 5.58: Block diagram of the project.

Circuit Diagram: A pair of U/S sensors are used to sense the distance both at the front and at the rear of a vehicle. The outputs of the ultrasonic sensors are at +5 V and therefore are not compatible with the inputs of Zero 2 W. Resistive potential divider circuits are used to lower the voltage to +3.3 V. The voltage at the output of the potential divider resistor is:

$$V_o = 5 \times 2000 / (2000 + 1000) = 3.3 \text{ V}$$

In this project, type HC-SR04 ultrasonic transmitter/receiver modules are used (see Figure 5.59). These modules have the following specifications:

- Operating voltage (current): 5 V (2 mA) operation
- Detection distance: 2 cm – 450 cm
- Input trigger signal: 10 µs TTL
- Sensor angle: under 15 degrees

The sensor modules have the following pins:

Vcc:	+V power
Trig:	Trigger input
Echo:	Echo output
Gnd:	Power ground

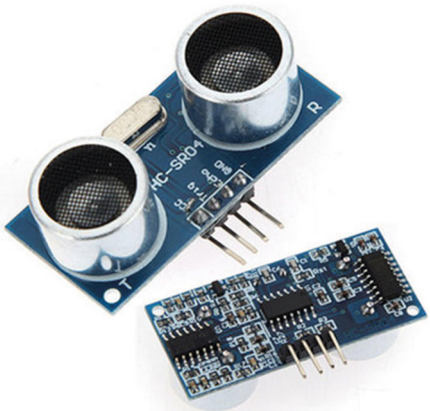


Figure 5.59: Ultrasonic (U/S) transmitter/receiver module.

The principle of operation of the ultrasonic sensor module is as follows:

- A 10-µs trigger pulse is sent to the module
- The module then sends eight 40-kHz square wave signals and automatically detects the returned (echoed) pulse signal
- If an echo signal is returned the time to receive this signal is recorded

The distance to the object is calculated as:

Distance to object (in meters) = (time to received echo in seconds × speed of sound) / 2

The speed of sound is 340 m/s, or 0.034 cm/μs

Therefore,

Distance to object (in cm) = (time to received echo in μs) × 0.034 / 2

or,

Distance to object (in cm) = (time to received echo in μs) × 0.017

Figure 5.60 shows the principle of operation of the ultrasonic sensor module. For example, if the time to receive the echo is 294 microseconds, then the distance to the object is calculated as:

Distance to object (cm) = 294 × 0.017 = 5 cm

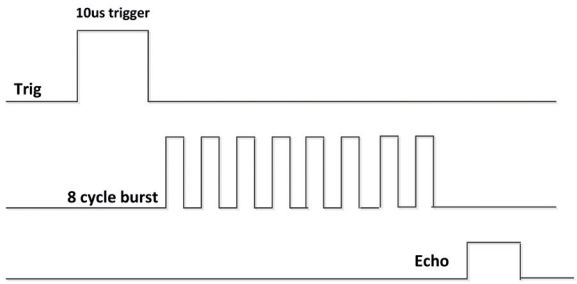


Figure 5.60: Operation of the ultrasonic sensor module.

Figure 5.61 shows the circuit diagram of the project. The trig and echo pins of the Front ultrasonic sensor are connected to GPIO 2 and GPIO 3 respectively. Similarly, the trig and echo pins of the Rear ultrasonic sensor are connected to GPIO 4 and GPIO 17, respectively. To drop the voltage levels to +3.3 V, the Echo outputs of the ultrasonic sensors are connected to the Zero 2 W through potential divider resistors. The buzzer is connected to GPIO 27.

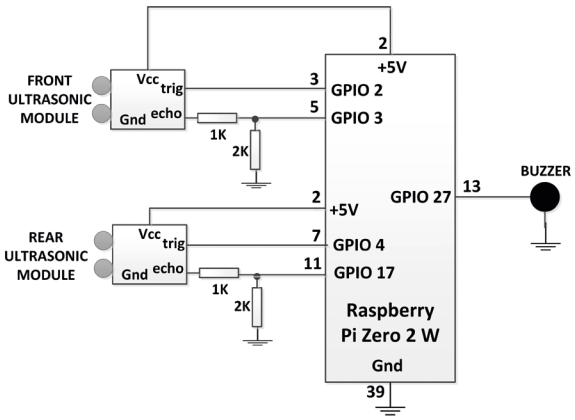


Figure 5.61: Circuit diagram of the project.

Program listing: Figure 5.62 shows the program listing (program **parking.py**). The front sensor pins are named **trig_f** and **echo_f**. Similarly, the rear sensor pins are named **trig_r** and **echo_r**. The distances from both sensors to obstacles are measured using function **GetDistance,** where the **trig** and **echo** pin names of the sensor whose distance to the obstacles is to be measured. A 10-microsecond trigger pulse is sent and then the time taken to receive the echo pulse is measured. Here, the **time.time()** function is used after sending the trigger pulse and the same function is used as soon as the echo pulse is received. The difference between the two times is the time taken to receive the echo pulse. This time is divided by 2 and the distance to the object expressed in cms is found as follows:

Speed of sound = 340 m/s, or 34000 cm/s
Distance to object (cm) = 34000 × time / 2

where "time" is in seconds and the time taken to receive the echo pulse. We can re-write the above equation as:

Distance to object (cm) = 17000 × time

If the distances of either sensors to the objects is less than or equal to the **Allowed_Distance** (which is set to 10 cm) then the buzzer is sounded to indicate that the vehicle is too close to objects (either at the front or at the rear).

Since the parking sensor is to be operated away from a PC, it is necessary to auto start the program when power is applied to the Raspberry Pi. The program name **parking.py** must be included in file **/etc/rc.local** in the following format for it to launch as soon as the Raspberry Pi starts after a power-up or after a reboot:

python /home/pi/robot2.py &

```
#-----------------------------------------------------------------------
#                          PARKING SENSORS
#                          ===============
#
# This is a parking sensors project. Ultrasonic tranamitter/receiver
# sensors are attached to the front and rear of a vehicle. In addition
# an active buzzer is connected to the Zero 2 W. The program senses
# the objects in the front and rear of the vehicle and sounds the buzzer
# if the vehicle is too close to the objects. In this project a distance
# less than 10cm is considered to be too close.
#
# File  : parking.py
# Date  : December, 2022
# Author: Dogan Ibrahim
#-----------------------------------------------------------------------
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
Allowed_Distance = 10                  # Distance in cm

#
# Front Ultrasonic sensor pins
#
trig_f = 2                             # GPIO 2
echo_f = 3                             # GPIO 3

#
# Rear ultrasonic sensor pins
#
trig_r = 4                             # GPIO 4
echo_r = 17                            # GPIO 17

#
# BUZZER pin
#
Buzzer = 27                            # GPIO 27

#
# Configure trig and buzzer as outputs, echos as inputs
#
GPIO.setup(trig_f, GPIO.OUT)
GPIO.setup(trig_r, GPIO.OUT)
GPIO.setup(echo_f, GPIO.IN)
GPIO.setup(echo_r, GPIO.IN)
```

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

#
# Turn ON the Buzzer
#
def BUZZERON():
  GPIO.output(Buzzer, 1)
  return

#
# Turn OFF the Buzzer
#
def BUZZEROFF():
  GPIO.output(Buzzer, 0)
  return

def GetDistance(trig, echo):
  GPIO.output(trig, 0)                          # Wait to settle
  time.sleep(0.08)
  GPIO.output(trig,1)                           # Send trig
  time.sleep(0.00001)                           # Wait 10 microseconds
  GPIO.output(trig, 0)                          # Remove trig
  while GPIO.input(echo) == 0:                  # Wait until echo is received
    start_time = time.time()                    # Start time

  while GPIO.input(echo) == 1:                  # Echo is received
    end_time = time.time()                      # End time

  pulse_width = end_time - start_time           # Pulse duration
  distance = pulse_width * 17150                # Distance in cm
  return distance                               # Return distance

#
# Start of the main program loop. Measure the distance to obstacles
# at the front and rear of the vehicle and activate the buzzer if the
# distance is below the allowed distance
#
BUZZEROFF()

while True:
  obstacle_f = GetDistance(trig_f, echo_f)     # distance to front obstacles
  obstacle_r = GetDistance(trig_r, echo_r)     # distance to rear obstacles
  if (obstacle_f <= Allowed_Distance or obstacle_r <= Allowed_Distance):
    BUZZERON()                                  # Turn Buzzer ON
  else:
```

```
    BUZZEROFF()                                    # Turn Buzzer OFF
```

Figure 5.62: Program: **parking.py**.

After applying power, wait until the Raspberry Pi Zero 2 W boots and the program should launch automatically. You should remove your Python program name from file **/etc/rc.local** after testing and completing your project so that the program does not start every time you restart your Pi.

5.16 Project 15: Real-Time graph of the temperature and humidity

Description: In this project you will read the temperature and the humidity using a sensor with the Zero 2 W and then plot the change in the temperature and humidity in real-time as the data is captured continuously. The aim of this project is to show how a sensor data captured by the Zero 2 W can be plotted in real-time. The project also shows how to use a temperature and humidity sensor with the Zero 2 W (See Section 4.29 for more details on plotting graphs).

Block diagram: The block diagram of the project is shown in Figure 5.63.

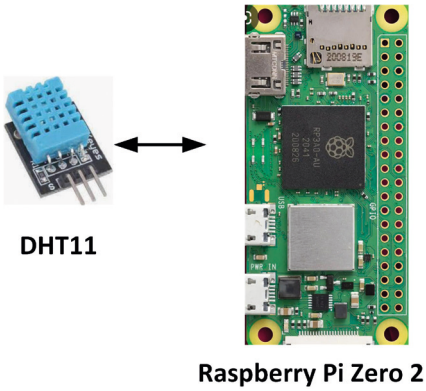


Figure 5.63: Block diagram of the project.

Circuit diagram: A type DHT11 temperature and relative humidity sensor chip (see Figure 5.64) is used in this project. This is normally a 3-pin sensor (there is also a 4-pin version of this sensor where one of the pins is not used) with pins GND, +V, and Data. GND and +V are connected to Ground and the +3.3 V power supply pins of the Raspberry Pi Zero 2 W. The Data pin must be connected to +V through a 10 k-ohm resistor. In this project, a 3-pin version of the DHT11 is used with built-in 10 k-ohm pull-up resistor. As shown in Figure 5.65, the Data pin of the sensor is named as S and it is connected to GPIO 2 of the Zero 2 W.

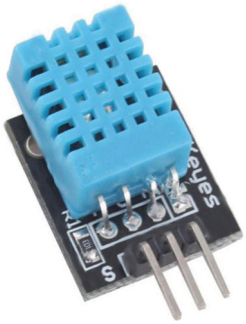


Figure 5.64: DHT11 sensor.

The DHT11 uses a capacitive humidity sensor, and a thermistor to measure the ambient temperature. Data output is available from the chip at a rate of about one second. The basic features of DHT11 are:

- 3 to 5 V operation
- 2.5 mA current consumption (during a conversion)
- Temperature reading in the range 0–50 ºC with an accuracy of ±2 ºC
- Humidity reading in the range 20-80% with 5% accuracy
- Breadboard-compatible with 0.1-inch pin spacings

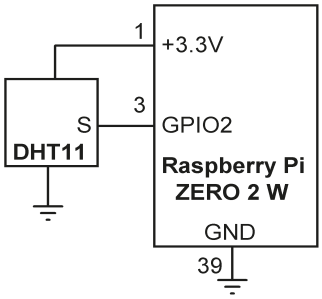


Figure 5.65: Circuit diagram of the project.

Program listing: In this program, the **Adafruit DHT11 library** and **matplotlib** modules are used. These modules should be installed into Python before they can be used. The instructions for are as follows:

- Go to command mode and enter:

  pi@raspberrypi:~ $ **git clone https://github.com/adafruit/Adafruit_Python_DHT.git**

- Change directory to:

  pi@raspberrypi:~ $ **cd Adafruit_Python_DHT**

- Enter the following commands:

```
pi@raspberrypi:~/Adafruit_Python_DHT $  sudo apt-get install build-
essential python3-dev
pi@raspberrypi:~/Adafruit_Python_DHT $ sudo python3 setup.py install
pi@raspberrypi:~/Adafruit_Python_DHT $  cd ..
```

- Install matplotlib:

```
pi@raspberrypi:~ $ sudo apt install python3-matplotlib python3-tk
```

Figure 5.66 shows the program listing (program: **graph.py**). At the beginning of the program, the **matplotlib**, **numpy**, **time**, and **Adafruit** modules are imported. The sensor type (variable **sensor**) is set to be DHT11, and it is connected to port GPIO 2. The X and Y axes are labeled and a title is given to the graph. The graph is set to be interactive by the statement **ion().** The remainder of the program runs in a **for** loop which executes as variable **i** changed from 0 to 100 in steps of 2, and this corresponds to the time axis (the length of the data collection time and hence the length of the X-axis can be changed if desired). The humidity and temperature are then read from the DHT11 and stored in variables **humidity** and **temperature** respectively. These values are then converted into floating and are stored in variables **t** and **h,** ready to be plotted. Scatter graphs are then drawn in real-time as the temperature and humidity data are received from the DHT11. A 5-second delay is introduced between each loop.

```
#----------------------------------------------------------------------
#       REAL TIME GRAPH OF HUMIDITY AND TEMPERATURE
#       ===========================================
#
# This program reads the ambient temperature and himidity from
# a DHT11 type sensor and displays them on the monitor in real-time
# as a graph.
#
# In this program data is collected every 5 seconds
#
# Program: graph.py
# Date   : December, 2022
# Author : Dogan Ibrahim
#----------------------------------------------------------------------
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import numpy as np
import time
import Adafruit_DHT

sensor = Adafruit_DHT.DHT11
```

```
GPIO = 2

#
# Start of main program. Humidity and temperature are read from
# DHT11 and are plotted in real time as the data are being read
#
plt.axis([0,100,0,100])
plt.title('Humidity and Temperature')
plt.xlabel('Time')
plt.ylabel('Hum. & Temp')
plt.clf()

j=1
plt.ion()

for i in range (0,102,2):
    humidity,temperature = Adafruit_DHT.read_retry(sensor,GPIO)
    x = float(i)
    t = float(temperature)
    h = float(humidity)
    plt.scatter(x,t,color='blue',label='Tempeature')
    plt.scatter(x,h,color='black',label='Humidity')
    plt.draw()
    if j == 1:
        j=0
        plt.legend()
    plt.pause(0.0001)
    time.sleep(5)
```

Figure 5.66: Program: **graph.py**.

The real-time changes in temperature and humidity are shown in Figure 5.67. In this image, the top graph is the humidity (RH) and the bottom one is the temperature (T).

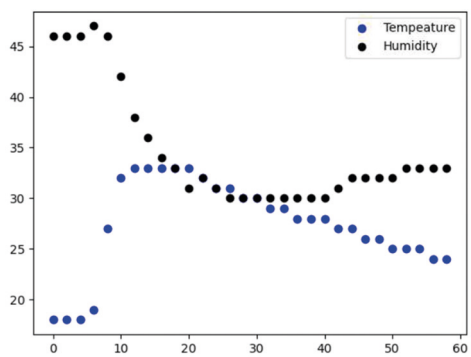


Figure 5.67: Change of temperature and humidity.

5.17 The Sense HAT interface

The Sense HAT is a small plug-in board developed by the Raspberry Pi Foundation in collaboration with the UK Space Agency and the European Space Agency (ESA). The board includes a number of sensors and that"s why it is called "Sense". The word "HAT" stands for *hardware attached on top* to indicate that the board is attached or plugged in on top of the Raspberry Pi. The Sense HAT gives the flexibility to carry out various environmental measurements using its built-in sensors and the board was specially developed for the Astro Pi challenge and competition. An emulator-based version of the Sense HAT is also available to enable students to carry out experiments without having the physical board.

The Sense HAT board (Figure 5.68) consists of 7 main components and an LED matrix. The components on the board are controlled via the I^{2C} bus interface. The following are the main components on the board:

Component	**I2C bus address**	**Function**
HTS221	0x5F	humidity sensor
LPS254H	0x5C	Pressure / temperature sensor
LSM9DS1	0x1C,0x6A	Accelerometer + magnetometer
SKRHABE010	-	joystick
LED2472G	0x46	LED matrix controller
LED matrix	-	-
ATTINY88	-	Atmel microcontroller

The Sense HAT board is normally plugged into the 40-way connector of the Raspberry Pi. In order to interface external components to the Raspberry Pi in addition to the Sense HAT board, you need to connect the Sense HAT to the Raspberry Pi using either a ribbon cable or jumper wires allowing other Raspberry Pi pins to remain accessible. In other words, you need to know which pins of the Sense HAT board are used by Raspberry Pi, and which pins of Raspberry Pi are free.

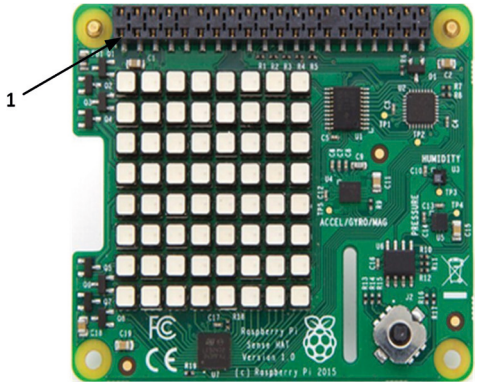


Figure 5.68: The Sense HAT board.

In addition to the I2C control lines, the ATTINY88 microcontroller on the board can be programmed via the SPI bus control lines (MOSI, MISO, SCK, CE0) provided on the board.

The following pins are used by the Sense HAT 40-way connector:

Pin number	Raspberry Pi port	Function
3	GPIO2	SDA (I2C)
5	GPIO3	SCL (I2C)
1	+3.3 V	power
19	GPIO10	MOSI (SPI)
21	GPIO9	MISO (SPI)
23	GPIO11	SCK (SPI)
24	GPIO8	CE0 (SPI
9	GND	power ground
2	+5 V	power
16	GPIO23	INT
18	GPIO24	INT
22	GPIO25	PROG
27	ID_SD	EEPROM
28	ID_SC	EEPROM

The Sense HAT board can be connected to the Zero 2 W using only the following 9 pins of the 40-way connectors:

Sense HAT pin	Zero 2 W Pin	Function
3	3 (GPIO2)	SDA (I2C)
5	5 (GPIO3)	SCL (I2C)
1	1 (+3.3 V)	power
9	9 (GND)	power ground
2	2 (+5 V)	power
16	16 (GPIO23)	joystick
18	18 (GPIO24)	joystick
27	27 (ID_SD)	EEPROM
28	28 (ID_SC)	EEPROM

Note: You can also plug in the Sense HAT board directly on top of the Zero 2 W board instead of making the above connections.

5.17.1 Programming the Sense HAT

The Sense HAT is installed by default on your latest Raspberry Pi SD card. You may, however, enter the following command to install the latest version of the Sense HAT:

```
pi@raspberrypi:~ $ sudo apt-get install sense-hat
```

Before developing a project using the Sense HAT board, the Sense HAT library must be imported into your Python program. Also, the **sense** object must be created at the beginning of the program, meaning the following two statements must be included at the beginning of your programs:

```
from sense_hat import SenseHat
sense = SenseHat()
```

The remainder parts of this Chapter is devoted to developing simple projects with the Sense HAT using the Raspberry Pi Zero W.

5.17.2 Project 16: Displaying text on Sense HAT

Description: In this project, you will learn how to display as well as to scroll text messages on the Sense HAT. The statement **show_message** is used to scroll a text message. In the following code, the message **Sense HAT** is scrolled on the LED matrix. Notice that the message is displayed only once:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message("Sense HAT")
```

If you get an error message saying that the **RPi-Sense FB device cannot be detected**, then do the following:

- pi@raspberrypi:~ $ **sudo nano /boot/config.txt**
- go to the end of the file and enter the following statement:
- dtoverlay=rpi-sense
- Press **CNTRL+X** followed by **Y** to save the change
- Reboot the Zero 2 W
- pi@raspberrypi:~ $ **sudo reboot now**

Notice that there are two versions of the Sense HAT board. Version 1.0 has no color sensor while Version 2.0 does have it. You may get a warning message saying that it failed to initialize the color sensor if you are using Version 1.0.

You can also display a single letter using the statement: **sense.show_letter**, for example: **sense.show_letter("A")**. Notice that the letter is displayed permanently.

In addition to displaying text in default mode, you can use the following options.

scroll_speed: This floating point number changes the speed that the text scrolls. The default value is 0.1. A bigger number slows down the scroll speed.

text_colour: Used to change the text color. The color is specified as (Red, Green, Blue) where each color can take a value between 0 and 255 and you can mix the colors to obtain any other colour. For example, (255, 0, 0) is red and so on.

back_colour: used to change the color of the background. Color is defined as in the text_color option.

In the following example, the same text as above is scrolled slowly, in red, with a yellow background color:

```
from sense_hat import SenseHat
sense = SenseHat()
sense.show_message("Sense HAT", scroll_speed=0.3,
      text_colour=[255,0,0], back_colour=[255,255,0])
```

Notice that in the above program the text is displayed only once, but the background color remains yellow.

If, for example, you wish to repeat displaying the text — say, every two seconds — then the required program is as shown in Figure 5.69 (program: **txt.py**). Notice how the continuation line is used in Python.

```
#------------------------------------------------------------
#                         Display Text
#                          ------------
#
# This program displays the text Sense HAT every 2 seconds.
# the text colour is RED and back ground colour is YELLOW
#
# Author: Dogan Ibrahim
# File  : txt.py
# Date  : December, 2022
#------------------------------------------------------------
from sense_hat import SenseHat
import time
sense = SenseHat()

while True:
   sense.show_message("Sense HAT",scroll_speed=0.3,\
text_colour=[255,0,0],back_colour=[255,255,0])
   time.sleep(2)
```

Figure 5.69: Program: **txt.py**.

The **sense.clear()** statement can be used to turn OFF all the LEDs. This may be necessary to ensure that all the LEDs are turned OFF at the beginning of a program. Similarly, a color can be passed to the **clear** statement to set all the LEDs to the same color, like so:

```
red = (255, 0, 0)
sense.clear(red)
```

The brightness of the LED matrix can be changed by toggling the **low_light** statement. In the following examples the brightness is toggled:

sense.low_light = True

or

sense.low_light = False

The displayed text (or image) can be rotated by using the statement **set_rotation(n)** where **n** is the rotation angle in degrees, taking the values of 0, 90, 180, or 270. The following statement rotates character s by 90 degrees and displays it on the LED matrix:

```
sense.set_rotation(90)
sense.show_letter("s")
```

Text (or image) can be flipped horizontally or vertically by using the statements **flip_h** or **flip_v** respectively. In the following example the character **X** is flipped horizontally and is then displayed:

```
sense.flip_h
sense.show_letter("X")
```

5.17.13 Project 17: Test your math skills: multiplication

Description: This project is aimed at younger readers who may want to test their multiplication skills. The program displays two numbers which are required to be multiplied together. The result of the multiplication is hidden for 10 seconds and time is given to the user to find the correct answer. After 10 seconds the correct answer is displayed so that the user can check it against his/her answer. Only numbers from 1 to 99 are considered for simplicity.

Figure 5.70 shows the program listing (program: **mult.py**). Two integer random numbers are generated between 1 and 99 and are stored in variables **no1** and **no2**. Variable **question** holds the question as a string and this is displayed in green as shown in the following example:

25 x 10 =

The program waits for 10 seconds and after this time the result 250 is displayed in red. After 2 seconds the LEDs are cleared and the program continues displaying two new numbers.

```
#-------------------------------------------------------------
#                      Multiplication Test
#                        -------------------
#
# This program displays two numbers between 1 and 99 and waits
# for 10 seconds until the user finds the correct answer. The
# correct answer is then displayed so that the user can check with
```

```
# his/her answer
#
# Author: Dogan Ibrahim
# File   : mult.py
# Date   : December, 2022
#-------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time
import random
spd = 0.2                                   # Scroll speed
red = (255, 0, 0)                           # Red colour
green = (0, 255, 0)                         # Green colour

try:

  while True:
    no1 = random.randint(1,99)              # First number
    no2 = random.randint(1, 99)             # Second number
    question = str(no1) + "x" + str(no2) + "="
    sense.show_message(question, scroll_speed = spd, text_colour=(green))
    time.sleep(10)
    result = str(no1 * no2)
    sense.show_message(result, scroll_speed = spd, text_colour=(red))
    time.sleep(2)
    sense.clear()
    time.sleep(1)

except KeyboardInterrupt:
    exit()
```

Figure 5.70: Program: **mult.py**.

5.17.14 Project 18: Learning the times tables

Description: This project helps children to practice their multiplication tables or "times tables" (not: timetables). A number (which can be changed) is hardcoded into the program. The program displays the times table for the selected number. For example, if the hardcoded number is 5 then the following is displayed on the LED matrix:

5×1=5
5×2=10
5×3=15
5×4=20
5×5=25
5×6=30
5×7=35

```
5×8=40
5×9=45
5×10=50
5×11=55
5×12=60
```

Figure 5.71 shows the program listing (program: **timestab.py**). Variable **Tablefor** stores the number whose times table is required. A loop is formed which goes from 0 to 11. Inside this loop variable **j** takes values from 1 to 12. Variable **result** stores the result of the multiplication at each iteration of the loop. String variable **disp** stores the data to be displayed by the LED matrix at each iteration. Users can easily change the value of **Tablefor** to generate times table for another number.

```
#--------------------------------------------------------------
#                      Times Table
#                       -----------
#
# This program generates a times table. The table is selected at the
# beginning of the program by setting variable Tablefor.
#
# Author: Dogan Ibrahim
# File  : timestab.py
# Date  : December, 2022
#--------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time

spd = 0.2                                              # Scroll speed
red = (255, 0, 0)                                      # Red colour
Tablefor = 5                                           # Table for 5

try:

  for k in range(12):                                  # Do 0 to 11
    j = k + 1                                          # 1 to 12
    result = Tablefor * j
    disp = str(Tablefor) + "x" + str(j) + "=" + str(result)
    sense.show_message(disp, scroll_speed = spd, text_colour=(red))
    time.sleep(1)
    sense.clear()

except KeyboardInterrupt:
    exit()
```

Figure 5.71: Program: **timestab.py**.

5.17.15 Project 19: Display temperature, humidity, and pressure

Description: In this project you learn all about displaying the ambient temperature, humidity and pressure on the Raspberry Pi Sense HAT.

Figure 5.72 shows the program listing (program: **thp.py**). The program runs in a loop every two seconds where the temperature, humidity, and pressure readings are displayed on the scrolling LED. Notice that the readings are all in floating-point format and the **round()** function is used to configure them to have one digit after the decimal point.

```
#-------------------------------------------------------------
#              TEMPERATURE,HUMIDITY & PRESSURE
#              -------------------------------
#
# This program reads the temperature, humidity and pressure and
# displays on the scrolling LEDs. The data is displayed in the
# following format:
#
#  T=nn.nC H=nn.n% P=nnnn.nmb
#
# Author: Dogan Ibrahim
# Date  : December, 2022
# File  : thp.py
#-------------------------------------------------------------
from sense_hat import SenseHat
sense=SenseHat()
import time

while True:
   T = round(sense.get_temperature(), 1)       # Get temperature
   H = round(sense.get_humidity(), 1)          # Get humidity
   P = round(sense.get_pressure(), 1)          # Get pressure
   enviro = "T="+str(T)+ "C H="+str(H)+ "% P="+str(P)+"mb "
   sense.show_message(enviro, scroll_speed = 0.2)
   time.sleep(2)
```

Figure 5.72: Program **thp.py**.

We could have also displayed the data on the PC screen by running the following program code. Figure 5.73 shows the output of the program:

```
from sense_hat import SenseHat
import time
sense = SenseHat()
while True:
     T = sense.get_temperature()
     H = sense.get_humidity()
```

```
P = sense.get_pressure()
TT = round(T, 1)
HH = round(H, 1)
PP = round(P, 1)
print("Temperature: %s, Humidity: %s, Pressure:%s"
   %(TT, HH, PP))
time.sleep(1)
```

```
Temperature: 24.9, Humidity: 43.3, Pressure: 997.8
Temperature: 24.9, Humidity: 43.4, Pressure: 997.8
Temperature: 25.0, Humidity: 43.2, Pressure: 997.8
Temperature: 25.0, Humidity: 43.4, Pressure: 997.8
Temperature: 25.0, Humidity: 43.4, Pressure: 997.8
Temperature: 25.1, Humidity: 43.3, Pressure: 997.9
Temperature: 25.0, Humidity: 43.6, Pressure: 997.8
Temperature: 25.0, Humidity: 43.3, Pressure: 997.9
```

Figure 5.73: Displaying the data on the PC screen.

We could also display the temperature or the humidity as integer variables on non-scrolling LEDs by using the **Disp** function.

5.17.16 Project 20: ON-OFF temperature controller

Description: This is an on-off temperature controller project. The Sense HAT is connected to the Zero 2 W to measure the ambient temperature. Additionally, a small buzzer is connected to one of the ports of the Zero 2 W. The set temperature value is hardcoded in the program. If the ambient temperature is lower than the set temperature, then the buzzer is activated and the LED matrix displays the ambient temperature in red. If, on the other hand, the ambient temperature exceeds the set temperature value, the buzzer is deactivated and the ambient temperature is displayed in blue. The buzzer in this project can easily be replaced with a relay whose contacts apply mains power to a heater. The heater will turn ON if the ambient temperature is lower than the set value.

Block diagram: Figure 5.74 shows the block diagram of the project.

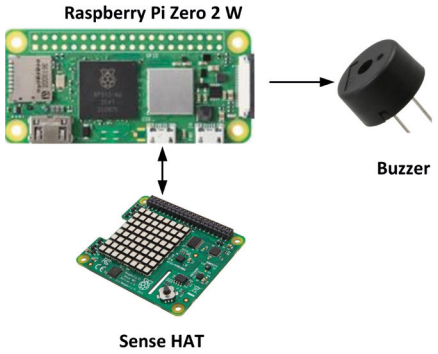


Figure 5.74: Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 5.75, where the buzzer is connected to port pin GPIO 4 of the Zero 2 W. Both the buzzer and the Sense HAT board are connected to the Zero 2 W using jumper wires.

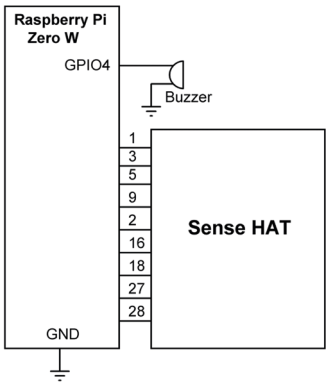


Figure 5.75: Circuit diagram of the project.

Program listing: In this program, the library function created by the author, named **Disp()** is used. This function has 3 arguments: number to be displayed, color, and mode (0 or 1, 1 to clear the display). This function is contained in the Python program called **display.py** which can be found in software archive for this book hosted on the Elektor website. Calling function **Disp()** displays a number without scrolling the display. Figure 5.76 shows the full program listing of **display.py**. Make sure that **display.py** is in the same directory as your main program.

```
#--------------------------------------------------------------
#                      FUNCTION TO DISPLAY NUMBERS
#                      ---------------------------
#
# This function displays a two-digit number on the LED matrix
# without scrolling the display. The number to be displayed and
# its colour are entered as the arguments of the function.The
# third parameter controls whether or not to clear the display
# before displaying the number. Setting this parameter to 1
# will clear the display
#
# Author: Dogan Ibrahim
# Date  : March 2022
# File  : display.py
#--------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()

def Disp(no, colour, mode):
#
```

```
# Number patterns for all the numbers 0 to 9
#
       numbers = [
       [[0,1,1,0],             # 0
       [1,0,0,1],
       [1,0,0,1],
       [1,0,0,1],
       [1,0,0,1],
       [1,0,0,1],
       [1,0,0,1],
       [0,1,1,0]],

       [[0,0,1,0],             # 1
       [0,1,1,0],
       [0,0,1,0],
       [0,0,1,0],
       [0,0,1,0],
       [0,0,1,0],
       [0,0,1,0],
       [0,1,1,1]],

       [[0,1,1,0],             # 2
       [1,0,0,1],
       [0,0,0,1],
       [0,0,0,1],
       [0,0,1,0],
       [0,1,0,0],
       [1,0,0,0,],
       [1,1,1,1]],

       [[1,1,1,1],             # 3
       [0,0,1,1],
       [0,0,1,1],
       [1,1,1,1],
       [1,1,1,1],
       [0,0,1,1],
       [0,0,1,1],
       [1,1,1,1]],

       [[0,0,1,0],             # 4
       [0,1,1,0],
       [1,1,1,0],
       [1,0,1,0],
       [1,1,1,1],
       [0,0,1,0],
       [0,0,1,0],
```

```
        [0,0,1,0]],

        [[1,1,1,1],             # 5
        [1,0,0,0],
        [1,0,0,0],
        [1,1,1,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [1,1,1,1]],

        [[1,1,1,1],             # 6
        [1,0,0,0],
        [1,0,0,0],
        [1,1,1,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,1,1,1]],

        [[1,1,1,1],             # 7
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1]],

        [[0,1,1,0],             # 8
        [1,0,0,1],
        [1,0,0,1],
        [1,1,1,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [0,1,1,0]],

        [[1,1,1,1],             # 9
        [1,0,0,1],
        [1,0,0,1],
        [1,1,1,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [1,1,1,1]]
```

```
    ]

    blank = [0,0,0]
    blanks=[0,0,0,0]
    Disp = []                               # List to store patterns

    for index in range(0, 8):
            if (no >= 10):                                  # If >= 10
                    intno = int(no / 10)                    # MSD digit
                    Disp.extend(numbers[intno][index])
            else:
                    Disp.extend(blanks)
            remno = int(no % 10)                    # LSD digit
            Disp.extend(numbers[remno][index])

    for index in range(64):
            if(Disp[index]):
                    Disp[index]=colour              # Colour
            else:
                    Disp[index]=blank

    if mode == 1:
        sense.clear()                               # Clear LEDs

    sense.set_pixels(Disp)                          # Display number
```

Figure 5.76: Program: **display.py**.

The program listing is shown in Figure 5.77 (program: **tempcont.py**). At the beginning of the program, the modules used in the program are imported to the program. The buzzer is assigned to number 4 which will correspond to GPIO 4. The set temperature value is stored in variable **SetTemperature** and is hardcoded as "24" in his example. The buzzer is turned OFF at the beginning of the program. The remainder of the program runs in an endless loop. Inside this loop, the ambient temperature is read from the Sense HAT, and this temperature is compared with the setpoint value. If the ambient temperature is less than the set value, then the buzzer is turned ON and the ambient temperature is displayed in red as non-scrolling. If, on the other hand, the ambient temperature exceeds the set value, the buzzer is turned OFF and the ambient temperature is displayed in blue.

```
#-----------------------------------------------------------------------
#              ON-OFF TEMPERATURE CONTROLLER
#              -----------------------------
#
# This is an ON-OFF temperature control project. In this project
# a buzzer is connected to port pin GPIO 4 of the Zero 2 W in
# addition to the Sense HAT. The Sense HAT is connected using
```

```
# jumper wires. The buzzer is turned ON if the ambient temperature
# is below the setpoint temperature. At the same time, the ambient
# temperature is displayed in red colour. If on the other hand the
# ambient temperature is higher than the setpoint value then the
# buzzer is turned OFF and the display is in blue colour.
#
# The buzzer in this program can be replaced with a relay for
# example to control a heater
#
# Author: Dogan Ibrahim
# Date  : December, 2022
# File  : tempcont.py
#-------------------------------------------------------------------
from display import Disp                                # import Disp
from sense_hat import SenseHat                          # import Sense HAT
sense=SenseHat()
import time                                             # import time
import RPi.GPIO as GPIO                                 # import GPIO

GPIO.setwarnings(False)                                 # disable warnings
GPIO.setmode(GPIO.BCM)                                  # set GPIO mode

Buzzer = 4                                              # Buzzer at GPIO4
SetTemperature = 24                                     # setpoint temp
red = (255, 0 ,0)                                       # red colour
blue = (0, 0, 255)                                      # blue colour

GPIO.setup(Buzzer, GPIO.OUT)                            # Buzzer is output
GPIO.output(Buzzer, 0)                                  # Buzzer OFF

while True:
   T = int(sense.get_temperature_from_humidity())       # get temperature
   if(T < SetTemperature):                              # T < setpoint?
      Disp(T, red, 0)                                   # display in red
      GPIO.output(Buzzer, 1)                            # Buzzer ON
   else:
      Disp(T, blue, 0)                                  # display in blue
      GPIO.output(Buzzer, 0)                            # Buzzer OFF

   time.sleep(5)                                        # wait 5 secs
```

Figure 5.77: Program: **tempcont.py**.

The buzzer used in this project can easily be replaced with a relay and a heater can be connected to the relay contacts to permit powering from the mains. The room temperature is then effectively governed by the program.

Chapter 6 • Communication over Wi-Fi

6.1 Overview

Two of the key features of the Raspberry Pi Zero 2 W are its Wi-Fi and Bluetooth communication capabilities. Your Raspberry Pi Zero 2 W is equipped with a 2.4-GHz 802.11 b/g/n wireless LAN module. Without such features, you"d have to resort to external communication modules. Network communication is handled using either UDP or TCP-type protocols. In this chapter, you will be learn how to write Python programs using both the UDP and TCP type protocols using the on-board Wi-Fi module.

6.2 UDP and TCP

Communication over a Wi-Fi link takes place based on client-and-server structures, while "sockets" are used to send and receive data packets. The server-side device usually waits for connection from the client(s). Once a connection is established, two-way communication can start. Two protocols are chiefly used for sending and receiving data packets over a Wi-Fi link: UDP and TCP. TCP is a connection-based protocol which guarantees the delivery of packets. Packets are given sequence numbers, and the receipt of all the packets are acknowledged to avoid them arriving in the wrong order. As a result of this confirmation, TCP is usually slow but it is reliable as it guarantees the delivery of packets. UDP, on the other hand, is not connection-based. Packets do not have sequence numbers, and consequently there is no guarantee that the packets will arrive at their destinations, or in the correct sequence. UDP has less overhead than TCP and as a result it is faster. Table 6.1 lists some of the differences between the TCP and UDP protocols.

TCP	UDP
Packets have sequence numbers and delivery of every packet is acknowledged	No delivery acknowledgement
Slow	Fast
No packet loss	Packets may be lost
Large overhead	Small overhead
Requires more resources	Requires less resources
Connection based	Not connection based
More difficult to program	Easier to program
Examples: HTTP, HTTPS, FTP	Examples: DNS, DHCP, Computer games

Table 6.1: TCP and UDP packet communications.

6.2.1 UDP communication

Figure 6.1 shows the UDP communication over a Wi-Fi link.

Server tasks

1. Create UDP socket
2. Bind the socket to server address
3. Wait until datagram packet arrives from client
4. Process the datagram packet
5. Send a reply to the client, or close the socket
6. Go back to Step 3 (if not closed)

Client tasks

1. Create UDP socket (and optionally Bind)
2. Send message to the server
3. Wait until response from the server is received
4. Process reply
5. Go back to step 2, or close the socket

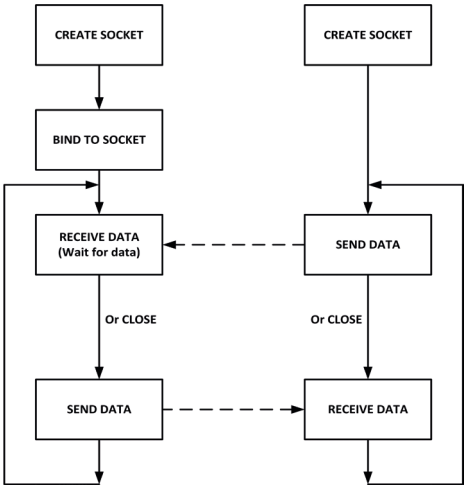


Figure 6.1: UDP communication.

6.2.2 TCP communication

Figure 6.2 shows the TCP communication over a Wi-Fi link:

Server tasks

1. Create UDP socket
2. Bind the socket to server address
3. Listen for connections
4. Accept connection
5. Wait until datagram packet arrives from client
6. Process the datagram packet
7. Send a reply to the client, or close the socket
8. Go back to Step 3 (if not closed)

Client tasks

1. Create UDP socket
2. Connect to server
3. Send message to the server
4. Wait until response from the server is received
5. Process reply
6. Go back to step 2, or close the socket

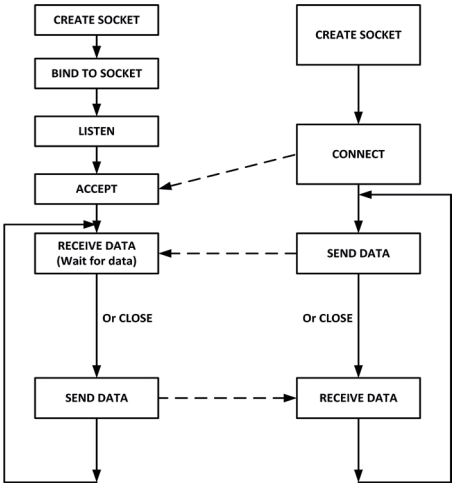


Figure 6.2: TCP communication.

6.3 Project 21: Sending a text message to a smartphone using TCP/IP

Description: In this Case Study, a TCP/IP based communication is established with an Android smartphone. The program reads text messages from the keyboard and sends them to the smartphone. The aim of this project is to show how TCP/IP communication can be established with an Android smartphone.

Background information: Port numbers range from 0 to 65535. Numbers from 0 to 1023 are reserved and called as established ports. For example, port 23 is the Telnet port, port 25 is the SMTP port etc. In this chapter you"ll use port number 5000 in your programs.

Block diagram: Figure 6.3 shows the project block diagram where the Zero 2 W and smartphone communicate over a Wi-Fi router.

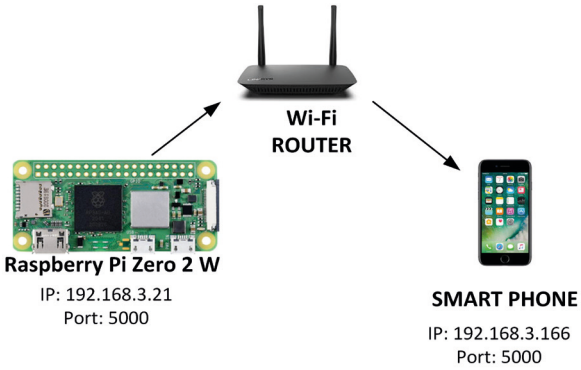


Figure 6.3: Block diagram of the project.

Raspberry Pi Zero 2 W program listing: In this project, the Zero 2 W is the server. Figure 6.4 shows the program listing (Program: **tcpserver.py**). At the beginning of the program, a TCP/IP socket is created (**sock.SOCK_STREAM**) and is then bound to port 5000. The program listens for a connection. Notice that it is possible for the server to listen to multiple clients, but it can of course communicate with only one at any time. When the client makes a connection, this is accepted by the server. The server then reads a message from the keyboard and sends it to the client over the Wi-Fi link. Notice that the **setsockopt()** statement makes sure that the program can be used again without having to wait for the socket timeout of 30 seconds.

```
#================================================================
#      SEND TEXT MESSAGES USING TCP/IP
#      ===============================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcpserver.py
# Date  : December, 2022
#================================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.21", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

yn = 'y'
```

```
while yn == 'y':
  msg = input("Enter your message: ")          # read a message
  client.send(msg.encode('utf-8'))             # send the message

  yn = input("Send more messages?: ")
  yn = yn.lower()

print("\nClosing connection to client")
sock.close()
```

Figure 6.4: Program: **tcpserver.py**.

Testing

There are many TCP apps available free of charge for smartphones. In this project, the **TCP Client by JOY S.R.L.** app is utilized on an Android smartphone. This app is available free of charge in the **Play Store** (see Figure 6.5).

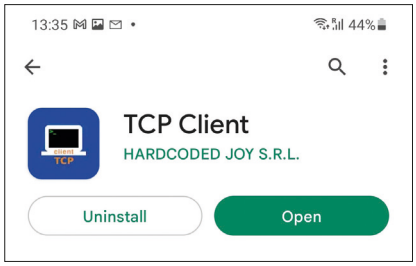


Figure 6.5: The app used in the project.

The program is run as follows:

- Run the server program first:

  pi@raspberrypi:~ $ **python3 tcpserver.py**

- Run the Android app and configure it as shown in Figure 6.6, where 192.168.3.21 is the IP address of the Zero 2 W.

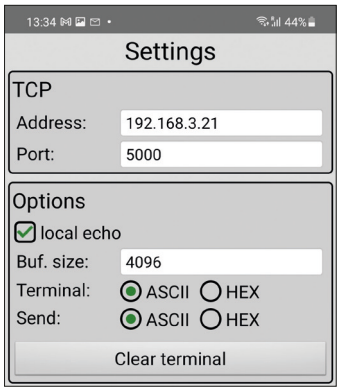


Figure 6.6: Configuring the TCP Client app.

- Click the icon at the top left corner of the app (disconnected) to connect to the Zero 2 W over TCP/IP.

- You should see a connection message on your Zero 2 W screen as well as on the IP address of the remote Android smartphone. Now enter a message and press the Enter key. In this example, the message **HELLO FROM RASPBERRY PI ZERO 2 W** is sent to the client (Figure 6.7). Enter **n** to terminate the Zero 2 W program. Figure 6.8 shows the message displayed on the smartphone.

```
pi@raspberrypi:~ $ python3 tcpserver.py
Connected to client:  ('192.168.3.166', 34126)
Enter your message: HELLO FROM RASPBERRY PI ZERO 2 W
Send more messages?: N

Closing connection to client
pi@raspberrypi:~ $
```

Figure 6.7: Entering the message on the keyboard.

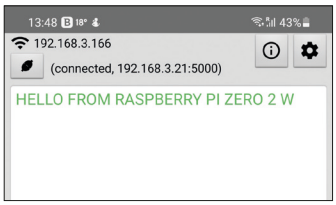


Figure 6.8: Message displayed on smartphone.

6.4 Project 22: Two-way communication with the smartphone using TCP/IP

Description: This project is similar to the previous one but here, two-way communication is established between the Zero 2 W and the smartphone.

The block diagram of the project is same as Figure 6.3.

Program listing: Figure 6.9 shows the program listing (Program: **tcp2way.py**). Here, port 5000 is used as in the previous project. The program has been changed to send and receive messages from the smartphone. Socket function **recv(byte count)** sends message over the TCP/IP link to the connected node.

```
#=============================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      =======================================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcp2way.py
# Date  : December, 2022
#=============================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.21", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

yn = 'y'

try:
   while yn == 'y':
      msg = input("Enter your message: ")       # read a message
      msg = msg +"\n"
      client.send(msg.encode('utf-8'))          # send the message

      msg = client.recv(1024)
      print("Received message: ")
      print(msg.decode('utf-8'))

      yn = input("Send more messages?: ")
      yn = yn.lower()

except KeyboardInterrupt:
      print("\nClosing connection to client")
      sock.close()
```

Figure 6.9: Program: **tcp2way.py**.

Testing

You will be using the Android app as in Figure 6.5. Start the Zero 2 W server program and then exchange messages between the Zero 2 W and the smartphone. An example communication session is shown in Figure 6.10. In this example, the Zero 2 W sends message **MESSAGE FROM ZERO 2 W**. In reply, the Android smartphone sends the message: **message from Android**.

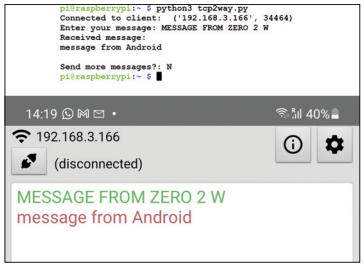


Figure 6.10: Example communication between the Zero 2 W and the Android app.

6.5 Project 23: Communicating with a PC using TCP/IP

Description: In this project, TCP/IP-based communication is established between the Raspberry Pi Zero 2 W and a PC (laptop/tablet) running Python. Messages are exchanged between the Zero 2 W and the PC. The aim of this project is to show how TCP/IP communication can be established with a PC.

Background Information: In the project, the Raspberry Pi Zero 2 W is the server and the PC is the client. The programs at either side are developed using the Python programming language. Python 3.10 is used on the PC. If you do not have Python on your PC, you could install it from the following website:

https://www.python.org/downloads/

Block diagram: Figure 6.11 shows the block diagram where the Zero 2 W and the PC communicate over a Wi-Fi router.

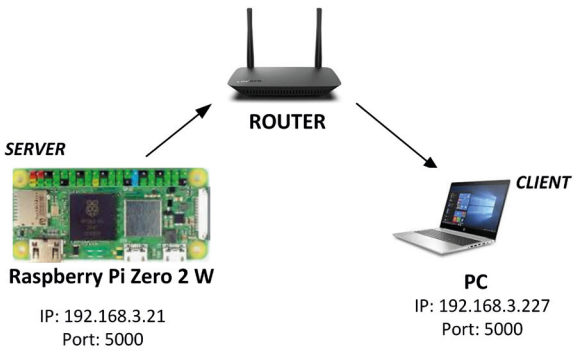


Figure 6.11: Block diagram.

Zero 2 W program Listing: The Zero 2 W program listing is shown in Figure 6.12 (Program: **tcppc.py**). The program is very similar to the one given in Figure 6.9, i.e., program: **tcp2way.py**. You should terminate the program by entering **Cntrl+C**.

```
#================================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      =======================================
#
# This is the TCP/IP server program. It communicates with a PC
# running TCP/IP on the same port
#
# Author: Dogan Ibrahim
# File  : tcppc.py
# Date  : December, 2022
#================================================================
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.21", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

try:
   while True:
      msg = input("Enter your message: ")       # read a message
      msg = msg +"\n"
      client.send(msg.encode('utf-8'))          # send the message

      msg = client.recv(1024)
      print("Received message: ", msg.decode('utf-8'))

except KeyboardInterrupt:
      print("\nClosing connection to client")
      sock.close()
      time.sleep(1)
```

Figure 6.12: Program: **tcppc.py**.

PC Program Listing: The PC program listing is shown in Figure 6.13 (program: **client.py**). The program creates a socket and then connects to the server. Next, messages are exchanged between the client and the server.

```
#============================================================
#               TCP/IP CLIENT
#               =============
#
# This is the client program on the PC.The program exchanges
# messages with the server on the Raspberry Pi Zero 2 W
#
# Author: Dogan Ibrahim
# File  : client.py
# Date  : December 2022
#============================================================
import socket
import time
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.connect(("192.168.3.21", 5000))

try:
    while True:
        msg = sock.recv(1024)
        print("Received message: ", msg.decode('utf-8'))
        data = input("Enter message to send: ")
        sock.send(data.encode('utf-8'))

except KeyboardInterrupt:
    print("Closing connection to server")
    sock.close()
    time.sleep(1)
```

Figure 6.13: PC program listing.

The steps to run the program are as follows.

- Run the server program on the Raspberry Pi Zero 2 W.
- Run the client program on the PC.
- Write messages as desired.

Figure 6.14 shows a typical run of the two programs.

```
pi@raspberrypi:~ $ python3 tcp2way.py
Connected to client:  ('192.168.3.227', 50213)
Enter your message: Message from Raspberry Pi Zero 2 W
Received message:  Message from the PC
Enter your message: ^C
Closing connection to client
```

```
D:\>python3 \ELEKTOR-RPI-ZERO2W\PROGRAMS\client.py
Received message:  Message from Raspberry Pi Zero 2 W

Enter message to send: Message from the PC
Received message:
Enter message to send: Closing connection to server
```

Figure 6.14: Example run of the program.

Note: You may find that after exiting the program you may not be able to run it again. This is because the socket stays open for about 30 seconds and the error message saying **Address is already in use** may be displayed. You can check the state of the port with the following command:

pi@raspberrypi:~ $ **netstat –n | grep 5000**

If the display includes the text **ESTABLISHED**, it means that the socket has not been closed properly and you will have to restart your Zero 2 W to run the program again. If, on the other hand, you see the message **TIME_WAIT,** simply wait about 30 seconds before restarting the program.

6.6 Project 24: Controlling an LED connected to the Zero 2 W from the smartphone, using TCP/IP

Description: In this project, an LED is connected to the Zero 2 W. The LED is turned ON and OFF by sending commands ON and OFF respectively from an Android smartphone. The aim of this project is to show how an LED connected to the Zero 2 W can be controlled from an Android smartphone by sending commands using the TCP/IP protocol over a Wi-Fi link. In this project, your Zero 2 W is the server and your smartphone acts as the client.

Block diagram: Figure 6.15 shows the block diagram of the project.

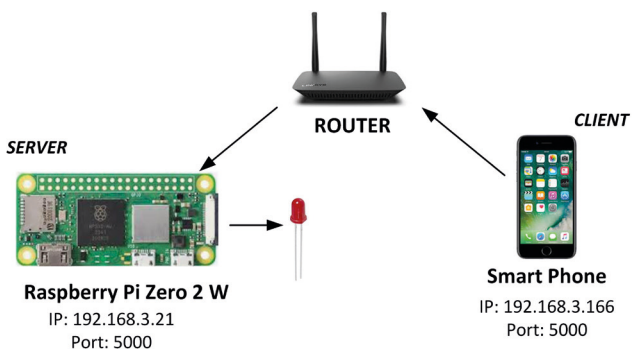


Figure 6.15: Block diagram of the project.

The LED is connected to port pin GPIO 2 (pin 3) through a 470-ohm current-limiting resistor.

Program Listing: Figure 6.16 shows the program listing (program: **serverled.py**). As in the previous program, a socket is created and port 5000 is used. The LED is assigned to port GPIO 2, configured as output, and turned OFF at the beginning of the program. The server waits for a connection from the client and then accepts the connection and displays message **Connected**. It then waits to receive a command from the client. If the command is **ON**, the LED is turned ON. If, on the other hand, the command is **OFF**, the LED is turned OFF. Sending command **X** terminates the server connection and exits the program.

```
#============================================================
#      CONTROL LED FROM SMART PHONE
#      ============================
#
# In this program TCP/IP is used where Zero 2 W is the server
# and smart phone is the client. An LED connected to Zero 2 W
# GPIO 2 and is controlled from the smart phone
#
# Author: DOgan Ibrahim
# File  : serverled.py
# Date  : December, 2022
#============================================================
import socket
import RPi.GPIO as GPIO
import time

LED = 2                                 # LED port
GPIO.setwarnings(False)                 # No warnings
GPIO.setmode(GPIO.BCM)                  # BCM numbering
GPIO.setup(LED, GPIO.OUT)               # LED is output
GPIO.output(LED, 0)                     # LED OFF

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.21", 5000))       # Zero 2 W IP, port
sock.listen(1)
client, addr = sock.accept()
print("Connected")

data = [' '] * 10


while data != b'X\n':                   # Terminate?
  data = client.recv(1024)
  if data == b'ON\n':                   # ON
     GPIO.output(LED, 1)
```

```
  elif data == b'OFF\n':                # OFF
     GPIO.output(LED, 0)

print("Closing connection")
GPIO.cleanup()
sock.close()
time.sleep(1)
```

Figure 6.16: Program: **serverled.py**.

The program is evaluated using the Android app **TCP client** (Figure 6.5) used in Project 21. The server program started, then the client is started. Figure 6.17 shows sending the **ON** command to turn ON the LED.

Figure 6.17: Command sent to turn ON the LED.

6.7 Project 25: Sending a text message to a smartphone using UDP

Description: In this project, UDP-based communication is established with an Android smartphone. The program reads text messages from the keyboard and sends them to the smartphone. The aim of the project is to show how UDP communication can be established with an Android smartphone.

The block diagram is same as in Figure 6.3.

Program Listing: In this project, your Zero 2 W is the server and your smartphone is the client. Figure 6.18 shows the program listing (program: **udpserver.py**). At the beginning of the program, a UDP socket is created (**sock.SOCK_DGRAM**) and is subsequently bound to port 5000. The server program then reads text messages sent from the smartphone and displays them on the screen. Messages sent by the Zero 2 W are displayed on the smartphone.

```
#=======================================================
#      SEND TEXT MESSAGES USING UDP
#      ============================
#
# This is the UDP server program running on Zero 2 W.
# The program exchanges text messages with an Android
# smart phone
#
# Author: Dogan Ibrahim
# File  : udpserver.py
# Date  : December, 2022
#=======================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.21", 5000))

try:
  while True:
    data, addr = sock.recvfrom(1024)
    print("Received msg:", data.decode('utf-8'))
    msg = input("Message to send: ")
    sock.sendto(msg.encode('utf-8'), addr)

except KeyboardInterrupt:
  print("\nClosing connection to client")
  sock.close()
```

Figure 6.18: Program: **udpserver.py**.

There are many UDP apps available free of charge for both Android and iOS smartphones. In this pΩroject **UDP Sender/Receiver by JC Accounting & Innovative Technologies Inc.** for Android smartphones is used with good results (Figure 6.19).

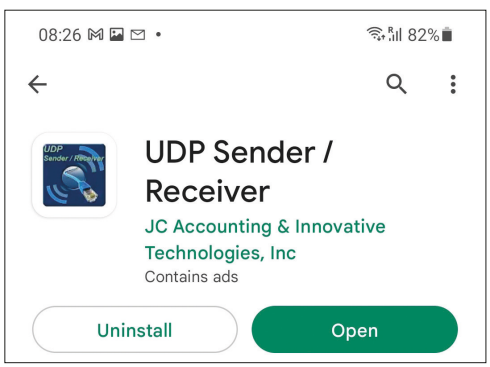


Figure 6.19: UDP Sender/Receiver app.

The steps to test the program are as follows:

- Start the server program on your Raspberry Pi Zero 2 W:

  pi@raspberrypi:~ $ **python3 udpserver.py**

- Start the smartphone app and configure it as shown in Figure 6.20.

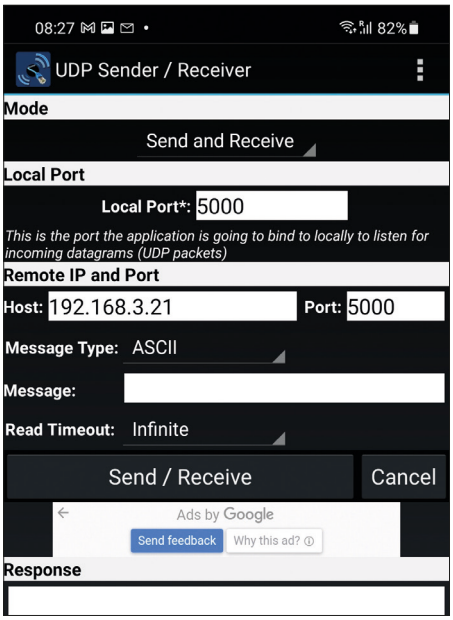


Figure 6.20: Configuring the smartphone app.

- Write a message on the mobile phone app and click **Send/Receive**. The message: **Message from smartphone** was sent as an example (Figure 6.21).

- Write a message on Zero 2 W and this message will be displayed on the smartphone. **Message from Zero 2 W** was sent as an example (Figure 6.21).

- Enter **Cntrl+C** on the Zero 2 W to close the socket.

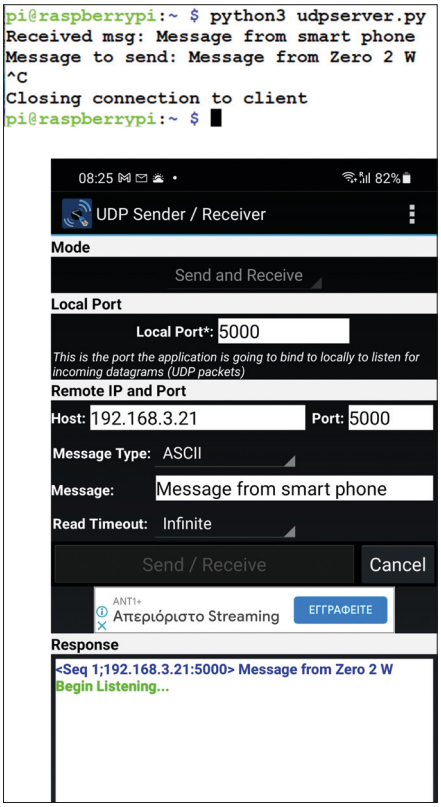


Figure 6.21: Sending and receiving messages.

6.8 Project 26: Controlling an LED connected to the Raspberry Pi Zero 2 W from the smartphone, using UDP

Description: In this project, an LED is connected to the Zero 2 W port pin GPIO 2 (pin 3) through a 470-ohm current limiting resistor. The LED is turned ON and OFF by sending commands ON and OFF respectively from an Android smartphone. The aim of this project is to show how an LED on the Zero 2 W can be controlled from a smartphone by sending commands using the UDP protocol over a Wi-Fi link. Here, the Raspberry Pi Zero 2 W is the server and the smartphone is a client.

The LED can easily be replaced with a relay, for example, to control electrical appliances from a smartphone.

The block diagram of the project is same as in Figure 6.14, but here port 2000 is used instead of 5000.

Program Listing: Figure 6.22 shows the program listing (program: **udpled.py**). As in the previous program, a socket is created and the server waits to receive commands from a client to control the LED. If the command is **ON**, the LED is turned ON. If, on the other hand, the command is **OFF**, the LED is turned OFF. Command **X** terminates the server program.

```
#======================================================================
#      CONTROL LED FROM SMART PHONE
#      ============================
#
# In this program UDP is used where Zero 2 W is the server
# and smart phone is the client. An LED connected to the server
# and is controlled from the smart phone
#
# Author: DOgan Ibrahim
# File  : udpled.py
# Date  : December, 2022
#======================================================================
import socket
import RPi.GPIO as GPIO
import time

LED = 2                                     # LED port
GPIO.setwarnings(False)                     # No warnings
GPIO.setmode(GPIO.BCM)                      # BCM numbering
GPIO.setup(LED, GPIO.OUT)                   # LED is output
GPIO.output(LED, 0)                         # LED OFF at start

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.21", 2000))           # Bind to Zero 2 W IP,port

data = [' '] * 10
while data != b'X':
  data, addr = sock.recvfrom(1024)
  if data == b'ON':                         # ON command
     GPIO.output(LED, 1)                    # LED ON
  elif data == b'OFF':                      # OFF command
     GPIO.output(LED, 0)                    # LED OFF

print("Closing connection")
GPIO.cleanup()
sock.close()
time.sleep(1)
```

Figure 6.22 Program: **udpled.py**.

The program can be evaluated using the **UDP Sender/Receiver** app used in Figure 6.18.

The steps to evaluate the program are as follows:

- Construct the circuit on your Zero 2 W with the LED.

- Start the server program on Raspberry Pi Zero 2 W:

```
pi@raspberrypi:~ $ python3 udpled.py
```

- Start the smartphone app as shown in Figure 6.19.
- Configure the app as shown in Figure 6.23.

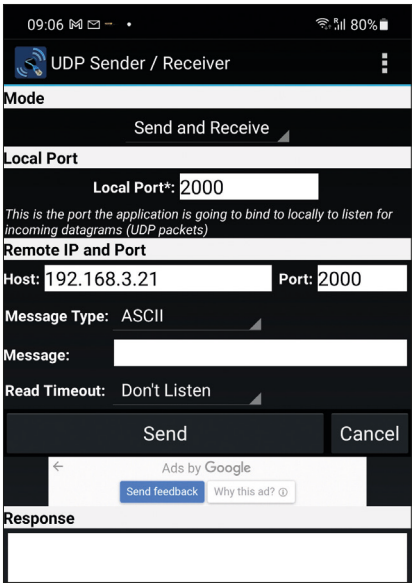


Figure 6.23: Configure the app.

- Write command **ON** and press **Send** on the smartphone (Figure 6.24). The LED should turn ON. Similarly, write **OFF** and the LED should turn OFF. Sending **X** should terminate the Zero 2 W program.

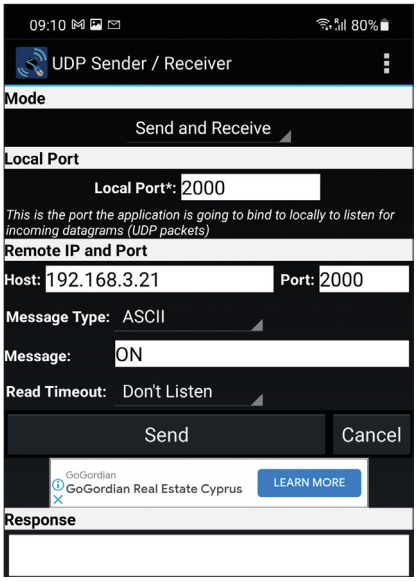


Figure 6.24: Turning ON the LED.

6.9 Using Flask to create a Web Server to control Raspberry Pi Zero 2 W GPIO ports from the Internet

Flask is a simple microframework written in Python, for Python. It is free of charge and can be used to create a web server on Raspberry Pi Zero 2 W and other members of the family (excluding the Pico) to control its GPIO ports over the Internet. The big advantage of Flask is that it does not require special tools or libraries, has no database, or any other third-party libraries.

Flask should already be available in Python on your Raspberry Pi Zero W. If not, it can be installed with the following command:

```
pi@raspberrypi:~ $ sudo apt-get install python-flask
```

It"s wise to create a new folder on your Raspberry Pi Zero W and store all of your Flask-related documents there. So go ahead and create a folder called **MyFlask** under your default directory **/home/pi**:

```
pi@raspberrypi:~ $ mkdir MyFlask
```

Make MyFlask your default directory:

```
pi@raspberrypi:~ $ cd MyFlask
```

You are now ready to create your first web server application using Flask. To evaluate Flask on your Zero 2 W development board, use the **nano** text editor and create a file called **flasktest.py** with the following lines in it:

```
        from flask import Flask                    # import module flask
        app = Flask(__name__)                      # create a flask object called
app

        @app.route("/")
        def index():                               # run index when called
           return "Hello from Flask"               # msg to display when run

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

Now, run the above program:

pi@raspberrypi:~ $ **sudo python3 flasktest.py**

You should see messages similar to the ones shown in Figure 6.25.

```
pi@raspberrypi:~/MyFlask $ sudo python3 flasktest.py
 * Serving Flask app "flasktest" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployr
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 291-931-053
```

Figure 6.25: Flask messages.

Now, open a web browser (e.g. **Google Chrome**) from a computer connected to the same Wi-Fi router and then enter the IP address of your Raspberry Pi Zero 2 W — in this example: 192.168.3.21. You should see the **Hello from Flask** message appear on a web page as shown in Figure 6.26.

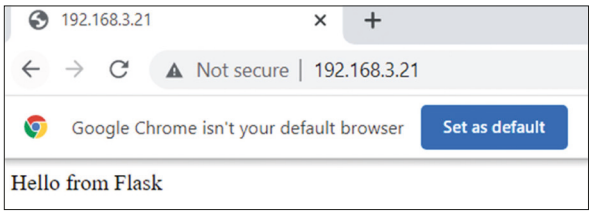


Figure 6.26: Message on the web page.

You can now create an HTML page and pass variables from a Python program. Create a folder called **templates** under **MyFlask,** move to directory **templates** and create a file called **index.html** using the **nano** text editor with the following lines (notice that the variables inside the double curly brackets will have data passed to them from the Python program):

```
<head>
    <title>{{ title }}</title>
</head>
<body>
   <h1>Hello from Flask</h1>
  <h2>The time on the server is: {{ time }}</h2>
</body>
```

You will now modify our **flasktest.py** program under directory **MyFlask** as follows:

```
from flask import Flask, render_template
import time

app = Flask(__name__)

@app.route("/")

def index():
  now = time.ctime()
  DataToPass = {
            "title" : "TESTING FLASK",
      "time": now
         }
  return render_template("index.html", **DataToPass)

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

The current date and time set is obtained using the function call **time.ctime()** and gets stored in a variable now. Then, a dictionary called **DataToPass** is created and values of title and time are stored in this dictionary. These values will be passed to the items in double curly brackets in the web page defined by index.html. When the function returns**,** variables inside the dictionary are returned to the web browser through the dictionary.

Now, run the program **flasktest.py** with the command

sudo python3 flasktest.py

Next, go to a web browser and enter the IP address of your Raspberry Pi Zero 2 W (192.168.3.21 for author"s Zero 2 W). You should see a display similar to the one shown in Figure 6.27.

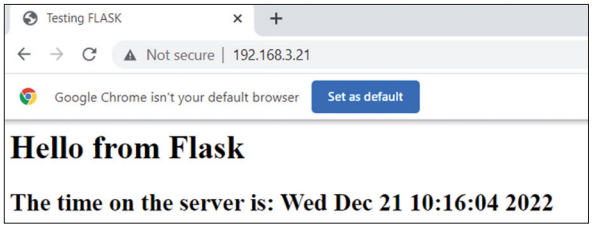


Figure 6.27: Web page displaying the date and time.

Now that you have learned how to pass variables from a Python program to a web page, you can monitor the status of a GPIO pin, or control a GPIO pin from a web page.

6.10 Project 27: Web Server — Controlling an LED connected to the Raspberry Pi Zero 2 W, using Flask

Description: In this project, an LED is connected to port GPIO 2 of the Raspberry Pi Zero 2 W through a 470-ohm current-limiting resistor. The LED is turned ON or OFF via remote web pages using Flask. The aim of this project is to show how Flask can be used to control an LED connected to the Zero 2 W.

HTML Template Program Listing: The HTML template **index.html** in folder **/home/pi/MyFlask/templates** is simple and it consists of a **title** and two buttons: **ON** and **OFF**. The **title** is in double curly brackets and therefore it expects data to be passed to it from Python. Two buttons are defined called **LED ON** and **LED OFF** with green and red colors, respectively. The **LED ON** button is referenced as **/LED/on** and the **LED OFF** button, as **/LED/off**. Figure 6.28 shows the program listing.

```
<head>
  <title>{{ title }}</title>
</head>

<body>
  <h3>
     <a href="/LED/on"><button type="button"><font color="green">LED ON
</button></a>
     <a href="/LED/off"><button type="button"><font color="red">LED OFF
</button></a>
  </h3>
</body>
```

Figure 6.28: HTML template program listing.

Raspberry Pi Program Listing: Figure 6.29 shows the Python program listing for the Raspberry Pi Zero 2 W in folder **/home/pi/MyFlask** (program: **flasktest.py**). The program has the basic Flask type template as shown earlier with some additional code. Port pin GPIO 2 is configured as an output and the LED is turned OFF at the beginning of the program. The **title** to be passed to **index.html** is named **LED CONTROL** and function **index** is used to pass this string. Notice that another **app.route** is created with parame-

ters **device** and **action**. In this example the **device** is **LED** and its actions are **on** and **off**. Function **action** checks the **device** and if it is **LED** then the **actuator** is set to **LED**. For every actuator you must have an action. If the **action** is **on**, the LED is turned **ON** by the statement **GPIO.output(actuator, GPIO.HIGH),** otherwise, if the **action** is **off**, the LED is turned **OFF**.

```
#=======================================================
#              CONTROLLING LED FROM WEB PAGE
#              ============================
#
# This program turns the LED ON or OFF from a web browser
# activated from any computer on the same Wi-Fi router as
# the Zero 2 W. The LED is controlled by clicking buttons
# when the web page is started
#
# Author: Dogan Ibrahim
# File  : flasktest.py
# Date  : December, 2022
#=======================================================
from flask import Flask,render_template
import RPi.GPIO as GPIO

app=Flask(__name__)

#
# Define GPIO2 as output and turn OFF LED at beginning
#
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
LED = 2
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED, 0)

@app.route('/')
def index():
  DataToPass = {
    'title': "LED CONTROL"
  }
  return render_template('index.html', **DataToPass)

@app.route("/<device>/<action>")
def action(device, action):
  if device == "LED":
    actuator = LED
```

```
  if action == "on":
    GPIO.output(actuator, GPIO.HIGH)
  if action == "off":
    GPIO.output(actuator, GPIO.LOW)
  return render_template('index.html')

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

Figure 6.29 Program: **flasktest.py**.

To run the program, you should follow the steps given below:

- Connect an LED to GPIO 2 through a current limiting resistor.
- Run program **flasktest.py**

  pi@raspberrypi:~/MyFlask $ **sudo python3 flasktest.py**

- Activate a web browser from a computer connected to the same Wi-Fi router and enter the IP address of your Raspberry Pi Zero 2 W. As shown in Figure 6.29, you should see two buttons to control the LED. Clicking the buttons turns the LED ON or OFF.

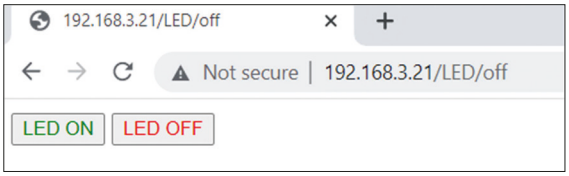


Figure 6.30 Controlling the LED from a web page.

- Terminate your program by entering **Cntrl+C**.

Modified Program

You could remove file **index.html** from the project and use only **flasktest.py** to control the LED. The modified program (**flasktest2.py**) is shown in Figure 6.31. To control the LED, you should enter the following web commands after starting **flasktest2.py** (Here 192.168.3.21 is the IP address of the Raspberry Pi Zero 2 W). Run program **flasktest2.py** as **sudo python3 flasktest2.py**:

- To turn the LED ON: 192.168.3.21/LED/on
- To turn the LED OFF: 192.168.3.21/LED/off

```
#=========================================================
#              CONTROLLING LED FROM WEB PAGE
#              ============================
#
# This program turns the LED ON or OFF from a web browser
# activated from any computer on the same Wi-Fi router as
# the Zero 2 W. The commands are (assuming that the IP
# address of Raspberry Pi is: 192.168.3.21):
#
# 192.168.1.202/LED/on         turn LED ON
# 192.168.1.202/LED/off                turn LED OFF
#
# Author: Dogan Ibrahim
# File  : flasktest2.py
# Date  : December, 2022
#=========================================================
from flask import Flask,render_template
import RPi.GPIO as GPIO

app=Flask(__name__)

#
# Define GPIO2 as output and turn OFF LED at beginning
#
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
LED = 2
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED, 0)

@app.route("/<device>/<action>")
def action(device, action):
  if device == "LED":
    actuator = LED

  if action == "on":
    GPIO.output(actuator, GPIO.HIGH)
    return "LED turned ON"
  if action == "off":
    GPIO.output(actuator, GPIO.LOW)
    return "LED turned OFF"

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

Figure 6.31 Modified program (flasktest2.py).

The modified program displays messages LED turned ON and LED turned OFF. Figure 6.32 shows the web command to turn ON the LED.

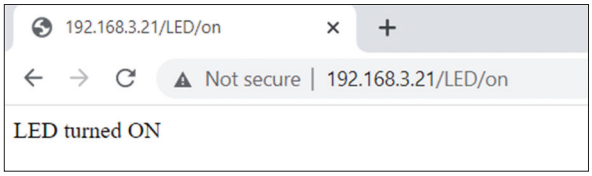


Figure 6.32 Turning ON the LED.

6.11 Communicating with the Raspberry Pi Pico W over Wi-Fi

The Raspberry Pi Pico W (called "Pico" from here on) is a very low-cost $6 microcontroller module based on the RP2040 microcontroller chip with dual-core Cortex-M0+ processor with on-board Wi-Fi module. Figure 6.33 shows the front view of the Pico hardware module which is basically a small board. At the middle of the board is the tiny 7 × 7mm size type RP2040 microcontroller chip housed in a QFN-56 package.

At the two edges of the board there are 40 gold-colored metal GPIO (General-Input-Output) pins with holes. You should solder pins to these holes so that external connections can easily be made to the board. The holes are marked starting with number 1 at the top left corner of the board and the numbers increase downwards up to number 40 which is at the top right-hand corner of the board. The board is breadboard-compatible (i.e., 0.1-inch pin spacing), and after soldering the pins, it can be plugged on a breadboard for easy connection to the GPIO pins using jumper wires. Next to these holes you will see bumpy circular cut-outs which can be plugged in on top of other modules not having any physical pins fitted.

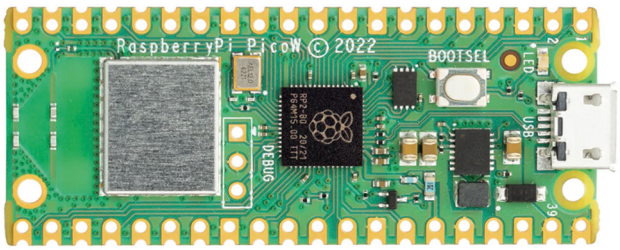


Figure 6.33 Front view of the Pico hardware module.

At one edge of the board there is the micro-USB B port for providing power to the board and for programming it. Next to the USB port there is an on-board user LED that can be used during program development. Next to this LED there is a button named BOOTSEL that"s used during programming of the microcontroller as you will see in next chapters. Next to the processor chip, there are three holes where external connections can be made to. These are used to debug your programs using Serial Wire Debug (SWD). At the other edge of the board is the single-band 2.4-GHz Wi-Fi module (802.11n). Next to the Wi-Fi module is the on-board antenna.

Figure 6.34 shows the back view of the Pico hardware module. Here, all the GPIO pins are identified with letters and numbers. You will notice the following types of letters and numbers:

GND	-	power supply ground (digital ground)
AGND	-	power supply ground (analog ground)
3V3	-	+3.3 V power supply (output)
GP0 – GP22	-	digital GPIO
GP26_A0 – GP28_A2	-	analog inputs
ADC_VREF	-	ADC reference voltage
TP1 – TP6	-	test points
SWDIO, GND, SWCLK	-	debug interface
RUN	-	default RUN pin. Connect LOW to reset the RP2040
3V3_EN	-	this pin by default enables the +3.3 V power supply. +3.3 V can be disabled by connecting this pin LOW
VSYS	-	system input voltage (1.8 V to 5.5 V) used by the on-board SMPS to generate +3.3 V supply for the board
VBUS	-	micro-USB input voltage (+5 V)

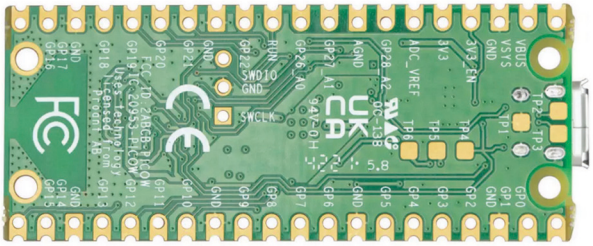


Figure 6.34: Back view of the Pico hardware module.

Some of the GPIO pins are used for internal board functions. These are:

GP29 (input)	-	used in ADC mode (ADC3) to measure VSYS/3
GP24 (input)	-	VBUS sense - HIGH if VBUS is present, else LOW
GP23 (output)	-	Controls the on-board SMPS Power Save pin

The specifications of the Pico hardware module are as follows:

- 32-bit RP2040 Cortex-M0+ dual-core processor operating at 133 MHz
- 2 MBbyte Q-SPI Flash memory
- 264 Kbyte SRAM memory
- 26 GPIO (+3.3V compatible)
- 3× 12-bit ADC pins

- Accelerated floating point libraries on-chip
- On-board single-band Infineon CYW43439 wireless chip, 2.4-GHz wireless interface (802.11b/g/n) and Bluetooth 5.2 (not supported at the time of writing)
- Serial Wire Debug (SWD) port
- Micro-USB port (USB 1.1) for power (+5 V) and data (programming)
- 2× UART, 2× I^2C, 2× SPI bus interface
- 16× PWM channels
- 1× Timer (with 4 alarms), 1× Real Time Counter
- On-board temperature sensor
- On-board LED at GPIO0, controlled by the 43439 module
- Castellated module allowing soldering direct to carrier boards
- 8× Programmable IO (PIO) state machines for custom peripheral support
- MicroPython, C, C++ programming
- Drag & drop programming using mass storage over USB

The Pico GPIO hardware is +3.3 V compatible which makes it important to be careful not to exceed this voltage when interfacing external input devices to the GPIO pins. +5 V to +3.3 V logic converter circuits or resistive potential divider circuits must be used if it is required to interface devices with +5 V outputs to the Pico GPIO pins.

The Pico can be programmed using MicroPython or C/C++ languages. In this section, you will be using the MicroPython with the Thonny editor to program your Pico. It is assumed that you have Pico development boards with MicroPython ready installed. It will also be useful if you are familiar with Thonny running on the Pico. An excellent book entitled *Raspberry Pi Pico W* is available at the Elektor web site and interested readers should purchase it for developing Raspberry Pi Pico-based projects.

Figure 6.35 shows the pin configuration of the Pico.

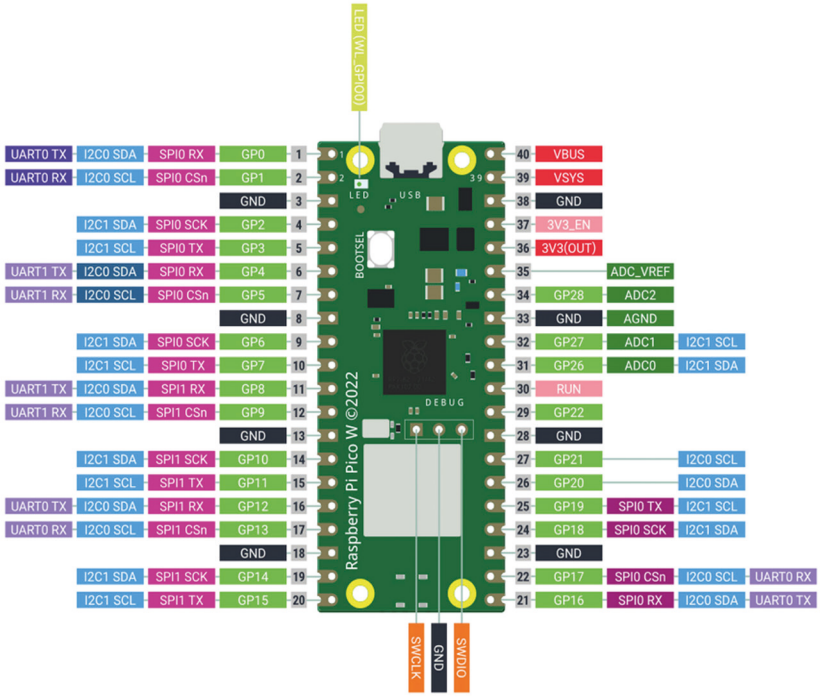


Figure 6.35 Pico pin configuration.

6.12 Project 28 – Raspberry Pi Zero 2 W and Raspberry Pi Pico W communication – controlling a relay over Wi-Fi

Description: In this project, you have a Raspberry Pi Zero 2 W and Raspberry Pi Pico W working together. A pushbutton is connected to the Raspberry Pi Pico W, and a +3.3 V relay is connected to the Raspberry Pi Zero 2 W. Pressing the button on the Pico sends a command to the Zero 2 W over the Wi-Fi to activate the relay. The relay remains active for 5 seconds. In this project, the Zero 2 W and the Pico communicate using the UDP protocol. The Zero 2 W is the server while the Pico is the client.

Block diagram: Figure 6.36 shows the block diagram of the project.

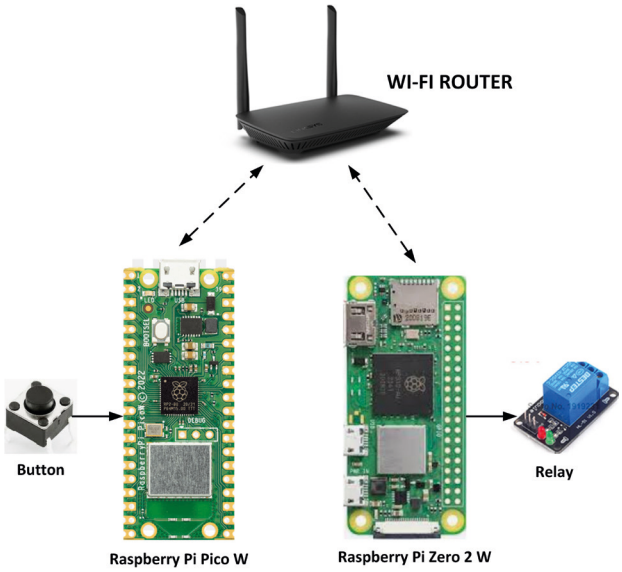


Figure 6.36 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 6.37 with the button and relay connected to the Pico and Zero 2 W, respectively.

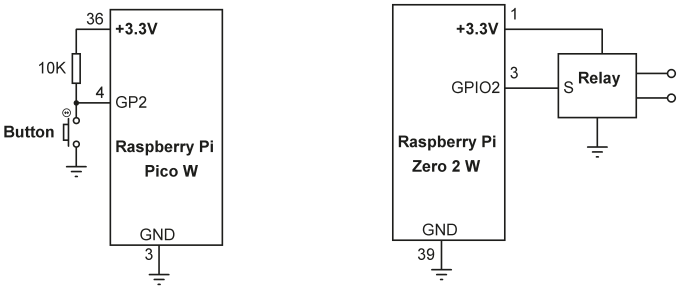


Figure 6.37 Circuit diagram of the project.

Pico program listing: Figure 6.38 shows the Pico program listing (Program: **picoudp.py**). At the beginning of the program, an LED is assigned to port GP2 and is turned OFF. Function **Connect()** is called to connect to local Wi-Fi. Then, a socket is created with port number 2000 and IP address 192.168.3.21. When the Button is pressed, the program sends "1" to the Zero 2 W so that the LED can be turned ON. This process is repeated after a 1-second delay.

```
#--------------------------------------------------------------
#      RASPBERRY PI PICO W - RASPBERRY PI ZERO 2 W COMMS
#      ================================================
#
# In this project a pushbutton is connected to GP2 of PICO W.
# Presingthe button sends a command to Zero 2 W to activate
# a relay. UDP protocol is used in this project
#
# Author: Dogan Ibrahim
# File  : picoudp.py
# Date  : December, 2022
#--------------------------------------------------------------
from machine import Pin
import network
import socket
import utime
global wlan

BUTTON = Pin(2, Pin.IN)                                  # Button at GP2

#
# This function attempts to connect to Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    while wlan.isconnected() == False:
       print("Waiting to be connected")
       wlan.active(True)
       wlan.connect("TP-Link_6138_EXT", "24844994")
       utime.sleep(5)

connect()
print("Connected")
UDP_PORT = 2000                                  # Port used
UDP_IP = "192.168.3.21"                          # Zero 2W IP
cmd = b"1"                                       # Cmd to turn ON
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    while BUTTON.value() == 1:                   # Not pressed
       pass
    while BUTTON.value() == 0:                   # Not released
       pass
    sock.sendto(cmd, (UDP_IP, UDP_PORT))         # Send cmd
```

```
    print("Command sent")                    # Message
    utime.sleep(1)                           # Wait 1 sec
```

*Figure 6.38 The Raspberry Pi Pico W program listing (**picoudp.py**).*

Zero 2 W program listing: Figure 6.39 shows the Zero 2 W program listing (Program: **zeroudp.py**). At the beginning of the program, the libraries used are imported, while Relay is configured at port GPIO 2 and deactivated. Then, a socket is created and the program binds to it with the Zero 2 W IP address. The program then waits to receive a command from the Pico. The received command is stored in variable **data** and if it is 1, the Relay is activated for 5 seconds. At the end of this time the Relay is deactivated and the program repeats waiting for a command.

```
#=======================================================
#      RASPBERRY PI PICO W - RASPBERRY PI ZERO 2 W COMMS
#      ============================
#
# This is the UDP server program running on Zero 2 W. The
# program receives a command from PICO W and activates a
# relay connected to GPIO 2 for 5 seconds
#
# Author: Dogan Ibrahim
# File  : zeroudp.py
# Date  : December, 2022
#=======================================================
import RPi.GPIO as GPIO
GPIO.setwarnings (False)
GPIO.setmode(GPIO.BCM)
import socket
import time

RELAY = 2                               # Relay at port 2
GPIO.setup(RELAY, GPIO.OUT)             # Relay is output
GPIO.output(RELAY, 0)                   # Deenergize Relay

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.21", 2000))

try:
  while True:
    data, addr = sock.recvfrom(1024)    # GEt command
    if data == b'1':                    # Command is 1?
       GPIO.output(RELAY, 1)            # Activate Relay
       time.sleep(5)                    # 5 seconds delay
       GPIO.output(RELAY, 0)            # Deactivate Relay
```

```
except KeyboardInterrupt:              # Keyboard interrupt
  print("\nClosing connection to client")
  sock.close()
```

*Figure 6.39 Raspberry Pi Zero 2 W program listing (**zeroudp.py**).*

Evaluating the project
The steps to evaluate the project are:

- Run the server on the Zero 2 W:

  pi@raspberry: ~$ **python3 zeroudp.py**

- Run the Pico program in Thonny by clicking the green Run button. You should see the message **Connected** when Pico connects to the local router.
- Push the button on Pico. The message **Command** sent will be displayed on the Pico terminal. A packet will be sent to the Zero 2 W which will turn ON the LED for 5 seconds.
- Enter **Cntrl+C** to terminate the program.

6.13 Project 29 — Storing ambient temperature and atmospheric pressure data in the Cloud

Description: In this project the ambient temperature and atmospheric pressure are read and stored in the Cloud. A BMP280 type sensor module is used in this project.

BMP280
The BMP280 is an absolute barometric pressure and temperature sensor chip, originally developed for mobile applications. Its small dimensions and low power consumption allow easy implementation in battery-powered devices such as mobile phones, GPS modules, and watches. The BMP280 is based on Bosch"s proven piezo-resistive pressure sensor technology featuring high accuracy and linearity as well as long-term stability and high EMC robustness. The device is optimized in terms of power consumption, resolution and filter performance. The basic specifications of the BMP280 include:

- Pressure range: 300 to 1100 hPa
- Relative accuracy: ±0.12 hPa
- Absolute accuracy: ±1 hPa
- Digital interface: I^2C, SPI
- Temperature range: –40 ºC to +85 ºC
- Current consumption: 2.7 µA

Block diagram: The block diagram of the project is shown in Figure 6.40.

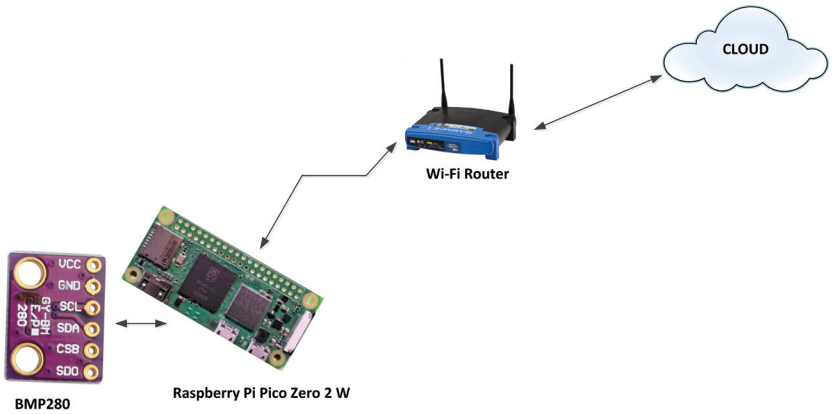


Figure 6.40 Block diagram of the project.

Circuit diagram: Figure 6.41 shows the circuit diagram. SCL and SDA pins of BMP280 are connected to the SDA (pin 3) and SCL (pin 5) of your Zero 2 W. 10 k-ohm pull-up resistors are used for the I^2C bus as the BMP280 board has no pull-up resistors (some BMP280 modules may already have on-board pull-up resistors). The sensor is powered from +3.3 V.

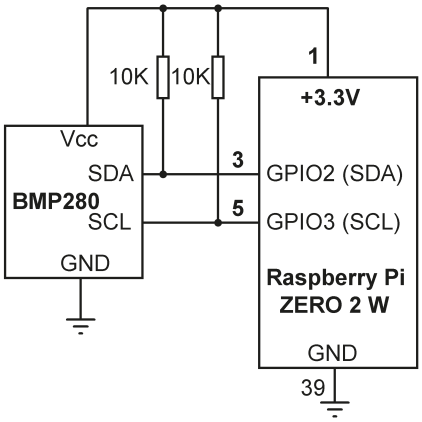


Figure 6.41 Circuit diagram of the project.

The Cloud

There are several cloud services that can be used to store data (for example, **SparkFun**, **ThingSpeak, Cloudino, Bluemix,** etc). In this project, **ThingSpeak** is used. This is a free cloud service where sensor data can be stored and retrieved using simple HTTP requests. Before using the ThingSpeak, you have to create an account from their website and then log in to this account.

Go to the ThingSpeak web site:

https://thingspeak.com/

Click **Get Started For Free** and create an account if you don"t already have one. Then, you should create a new channel by clicking on… **New Channel**! Fill in the form as shown in Figure 6.42. Give the name **Raspberry Pi Zero 2 W** to the application, give a description, and create two **fields** called **Atmospheric Pressure** and **Temperature**. You can optionally fill in the other items as well if you wish.

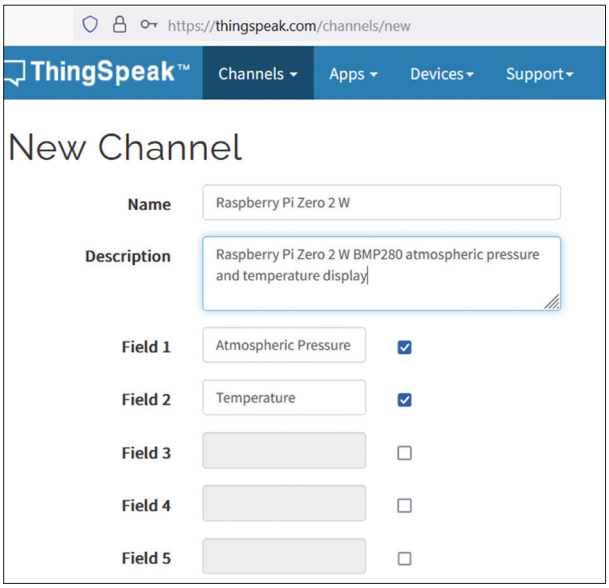
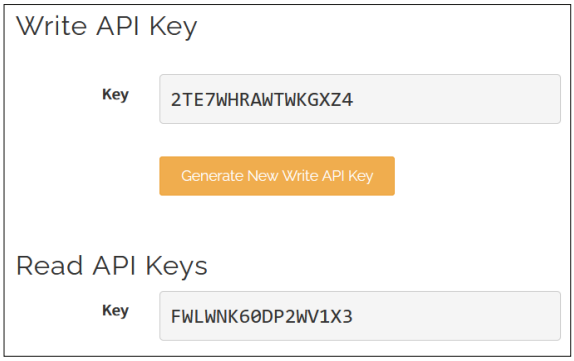


Figure 6.42: Create a New Channel (only part of the ThingSpeak form shown).

Click **Save Channel** at the bottom of the form. Your channel is now ready to be used with your data. You will now see tabs with the following names. You can click on these tabs and view the contents to make corrections, if necessary:

- **Private View**: This tab displays private information about your channel which only you can see.
- **Public View**: If your channel is public, use this tab to display selected fields and channel visualizations.
- **Channel Settings**: This tab shows all the channel options you set at creation. You can edit, clear, or delete the channel from this tab.
- **API Keys**: This tab displays your channel API keys. Use the keys to read from and write to your channel.
- **Data Import/Export**: This tab enables you to import and export channel data.

- You should click the **API Keys** tab and save your **Write API** and **Read API** keys and the **Channel ID** in a safe place as you will need to use them in our program. The API Keys and the Channel ID in this project were:



Also, note your **Channel ID**: 2004389

Also, select the **Public View** and navigate to **Sharing**. You may select option **Share channel view with everyone** so that everyone can access your data remotely.

Program listing: In this program, you will use a BMP280 library for Raspberry Pi. The steps to install the library are as follows:

```
pi@raspberrypi:~ $ sudo pip install bmp280
```

After constructing the circuit, you should check to make sure that the BMP280 is detected by the Zero 2 W. Enter the following command:

```
pi@raspberrypi:~ $ sudo i2cdetect –y 1
```

You should see the hardware address of the BMP280 chip displayed as: 76 (see Figure 6.43).

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@raspberrypi:~ $
```

Figure 6.43: BMP280 hardware address detected.

Figure 6.44 shows the program listing (Program: **Cloud.py**). At the beginning of the program, the libraries used get imported to the program. The ThingSpeak **Write Key** and **Host Address** are defined. The main program loop starts with the **while** statement. Inside

this loop, the IP address of the **ThingSpeak** website is extracted and a connection is made to this site at port 80. Then, the atmospheric pressure and temperature readings are obtained from the BMP280 module and are included in the **path** statement. The **sock.send** statement sends an HTTP GET request to the **ThingSpeak** site and uploads the pressure and temperature values. This process is repeated every 30 seconds.

Figure 6.45 shows the pressure and temperature data plotted by **ThingSpeak**. The Chart Options can be clicked on to change various parameters of the charts. For example, Figure 6.46 shows the temperature as a column display. In Figure 6.47, the pressure is shown as a step graph. A title and X-axis label are added in Figure 6.48 to the pressure graph. Figure 6.49, finally, shows the atmospheric pressure displayed in a clock format (click **Add Widgets** for this type display).

Because the Channel was saved as Public, you can view the graph from a web browser by entering the Channel ID. In this project, the link to view the data graphs from a web browser is:

https://api.thingspeak.com/channels/2004389

You can also export some or all of the fields in CSV format by clicking **Export recent data**, so that it can be analyzed by external statistical packages such as Excel.

```
#-------------------------------------------------------------------------
#        ATMOSPHERIC PRESSURE AND TEMPERATURE ON THE CLOUD
#       ================================================
#
# The ambient temperature and pressure sensor BMP280 is connected to Zero 2 W.
# The project reads the temperature and atmospheric pressure and sends
# to the Cloud where it can be accessed from anywhere. In addition, change
# of the temperature and the pressure can be plotted in the cloud.
#
#
# The program uses the Thingspeak cloud service
#
# Author: Dogan Ibrahim
# File  : Cloud.py
# Date  : January, 2022
#-------------------------------------------------------------------------
import socket
import time
from bmp280 import BMP280
from smbus import SMBus

bus = SMBus(1)
bmp280 = BMP280(i2c_dev=bus)
```

```
APIKEY = "2TE7WHRAWTWKGXZ4"                    # Thingspeak API key
host = "api.thingspeak.com"                    # Thigspeak host

#
# Send data to Thingspeak. This function sends the temperature and
# humidity data to the cloud every 30 seconds
#
while True:
   sock = socket.socket()
   addr = socket.getaddrinfo("api.thingspeak.com",80)[0][-1]
   sock.connect(addr)
   p=bmp280.get_pressure()
   t=bmp280.get_temperature()                  # Temperature in C
   path = "api_key="+APIKEY+"&field1="+str(p)+"&field2="+str(t)
   sock.send(bytes("GET /update?%s HTTP/1.0\r\nHost: %s\r\n\r\n"
%(path,host),"utf8"))
   time.sleep(5)
   sock.close()
   time.sleep(25)
```

Figure 6.44: Program: **Cloud.py**.

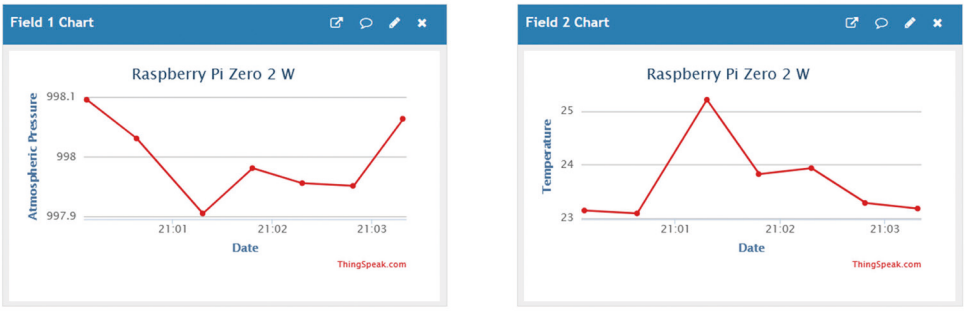


Figure 6.45: Plotting the pressure and temperature.

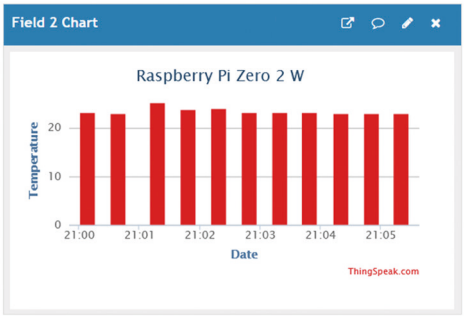


Figure 6.46: Displaying temperature as columns.

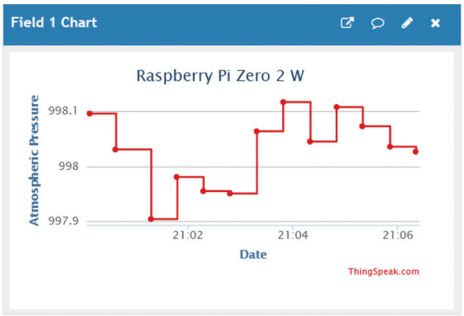


Figure 6.47: Displaying pressure as steps.

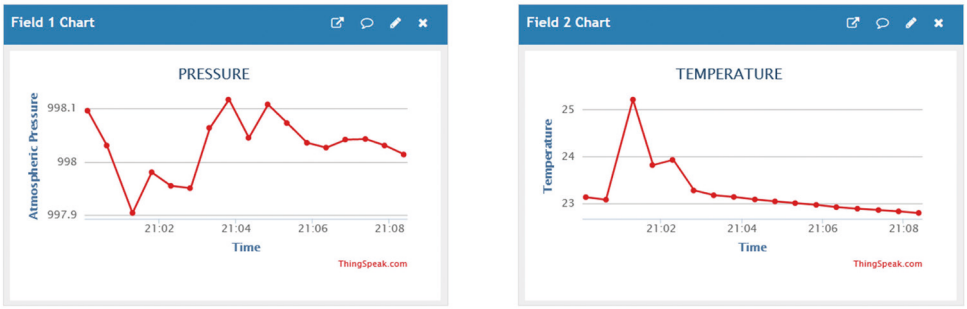


Figure 6.48: Adding title and x-axis label.

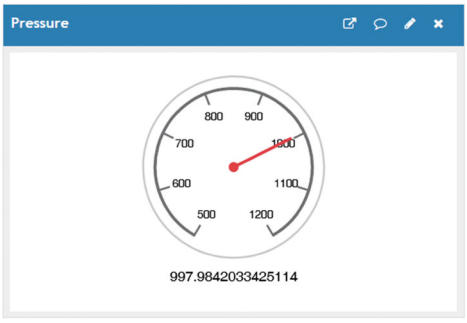


Figure 6.49: Displaying pressure in a clock format.

6.14 Useful network commands

In this section, you will explore some of the useful network commands that can be accessed from the Zero 2 W command mode.

6.14.1 Ping

This is one of the useful commands that is used to detect devices on a network. The general form of the command is:

ping hostname or **ping IPaddress**

The display is repeated every second. You can restrict the number of attempts with the **–c** option. For example, to restrict to 5 attempts, use **ping hostname –c 5**. An example is shown in Figure 6.50.

```
pi@raspberrypi:~ $ ping www.bbc.co.uk -c 5
PING bbc.map.fastly.net (199.232.16.81) 56(84) bytes of data.
64 bytes from 199.232.16.81 (199.232.16.81): icmp_seq=1 ttl=49 time=100 ms
64 bytes from 199.232.16.81 (199.232.16.81): icmp_seq=2 ttl=49 time=147 ms
64 bytes from 199.232.16.81 (199.232.16.81): icmp_seq=3 ttl=49 time=97.5 ms
64 bytes from 199.232.16.81 (199.232.16.81): icmp_seq=4 ttl=49 time=156 ms
64 bytes from 199.232.16.81 (199.232.16.81): icmp_seq=5 ttl=49 time=114 ms

--- bbc.map.fastly.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 97.460/122.995/156.002/24.154 ms
pi@raspberrypi:~ $
```

Figure 6.50: The ping command.

6.14.2 hostname

This command has several options. Command **hostname** shows the host name. **hostname –I** shows the IP address assigned to our Pi, **hostname – d** shows the DNS name. An example is shown in Figure 6.51.

```
pi@raspberrypi:~ $ hostname
raspberrypi
pi@raspberrypi:~ $ hostname -I
192.168.3.21
pi@raspberrypi:~ $
```

Figure 6.51: The hostname command.

6.14.3 ifconfig

This command displays network information about your RPi. The options are:

ifconfig	full network display
ifconfig wlan0	show Wi-Fi interface details
ifconfig eth0	show Ethernet interface details

An example is shown in Figure 6.52.

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether dc:a6:32:00:e4:28  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 26  bytes 2768 (2.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 26  bytes 2768 (2.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.21  netmask 255.255.255.0  broadcast 192.168.3.255
        inet6 fe80::2eae:24a3:22a1:979c  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:00:e4:29  txqueuelen 1000  (Ethernet)
        RX packets 2849  bytes 769716 (751.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 746  bytes 88229 (86.1 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

pi@raspberrypi:~ $
```

Figure 6.52: Command: ifconfig.

You can shutdown or bring up our Wi-Fi network interface with the commands: **sudo ifconfig wlan0 down** and **sudo ifconfig wlan0 up** respectively.

6.14.4 route

This command is used to configure and display Wi-Fi router information. An example is shown in Figure 6.53. In this example, your Wi-Fi router IP address is 192.168.3.0, having the mask 255.255.255.0.

```
pi@raspberrypi:~ $ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         dogan.akay      0.0.0.0         UG    303    0        0 wlan0
192.168.3.0     0.0.0.0         255.255.255.0   U     303    0        0 wlan0
pi@raspberrypi:~ $
```

Figure 6.53 Command: route.

6.14.5 netstat

This command displays information about your network and ports. An example is shown in Figure 6.54. The following options are useful (enter **netstat –help** to display a list of all options):

netstat –i	display interface table
netstat –s	display network statistics
netstat –l	display listening server sockets
netstat – o	display timers

```
pi@raspberrypi:~ $ netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500        0      0      0 0             0      0      0      0 BMU
lo       65536       27      0      0 0            27      0      0      0 LRU
wlan0     1500     4003      0      0 0          1065      0      0      0 BMRU
pi@raspberrypi:~ $
```

Figure 6.54: Command: netstat.

6.14.6 host

The host command is used to find the IP address of a particular DNS. An example is shown in Figure 6.55.

```
pi@raspberrypi:~ $ host www.bbc.co.uk
www.bbc.co.uk is an alias for www.bbc.co.uk.pri.bbc.co.uk.
www.bbc.co.uk.pri.bbc.co.uk is an alias for bbc.map.fastly.net.
bbc.map.fastly.net has address 199.232.16.81
pi@raspberrypi:~ $
```

Figure 6.55: Command: host.

6.15 Setting-up Wi-Fi on your Raspberry Pi Zero 2 W

There are many ways to set up Wi-Fi on the Zero 2 W and these are described briefly in this section.

6.15.1 During the installation of the Raspberry Pi operating system

As described in section 1.3, the local Wi-Fi router SSID and password can be specified during the installation of the Raspberry Pi operating system.

6.15.2 Modifying the Wi-Fi details on the SD card

The SD card stores the Raspberry Pi operating system. You can easily modify the Wi-Fi SSID and password by editing a file on the SD card. The steps are as follows:

- Power down your Zero 2 W orderly.
- Remove the SD card and insert it into your PC.
- Open the **Notepad** text editor on your PC.
- Create a new, empty file with the **Notepad** and save it in the boot folder of the SD card with the name **ssh** (without any extension), where this file will enable the SSH to be used to access your Raspberry Pi Zero 2 W remotely. In Windows, this is the only folder you will see which contains items like: loader. bin, start.elf, kernel.img, etc.
- Close the file.
- Create a new file called **wpa_supplicant.conf** and copy the following lines to this file (replace the **MySSID** and **MyPassword** with the details of your own Wi-Fi router):

```
country=GB
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
```

```
        scan_ssid=1
        ssid="MySSID"
        psk="MyPassword"
}
```

- Save the file to the boot folder on your SD card with the name: **wpa_supplicant.conf**.
- Insert the SD card back into your Raspberry Pi Zero 2 W and power-up the device.
- The Raspberry Pi Zero 2 W should boot and access the local Wi-Fi router.

6.15.3 Setting via the Task Bar

If you are directly connected to your Zero 2 W using a monitor, there should be a network icon near the clock at the top right-hand side of the Desktop. Click on the icon to see all the available Wi-Fi networks. Select your network from the list and enter the network password.

6.15.4 Using the raspi-config tool

If you are directly connected to your Zero 2 W using a monitor, you can use the **raspi-config** tool to set up the local network details. The steps are:

- Start the raspi-config:

```
pi@raspberrypi:~ $ sudo raspi-config
```

- Select option 1 (**System Options**) and press Enter.
- Select option S1 (**Wireless LAN**) and enter your network details (Figure 6.56).

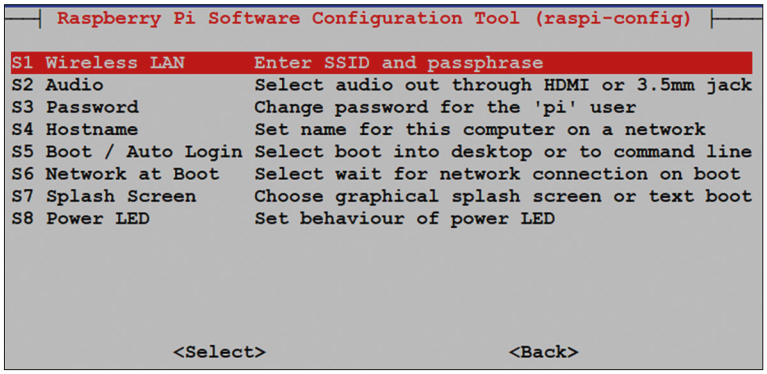


Figure 6.56: Select Wireless LAN.

- **Finish** and **Exit** the tool.

6.15.5 Manual setup

If you are directly connected to your Zero 2 W using a monitor, you can edit a file to set up your network details. The steps are as follows:

- Create/edit the following file:

  pi@raspberrypi:~ $ **sudo nano /etc/wpa_supplicant/wpa_supplicant.conf**

- Enter (or change) to the following details:

```
country=GB
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
        scan_ssid=1
        ssid="MySSID"
        psk="MyPassword"
}
```

- Enter Cntrl+X followed by Y to save and exit the file.
- Reboot your Zero 2 W, which will come up connected to you Wi-Fi:

  pi@raspberrypi:~ $ **sudo reboot**

6.16 Finding the IP address of your Zero 2 W

There are times when you cannot log in to your Zero 2 W remotely since you may not know the IP address allocated by the Wi-Fi router. Fortunately, there are several ways to find your RPi"s IP address.

6.16.1 Using a smartphone app

- Install a network scan app on your smartphone and scan your network to find out the connected devices to your Wi-Fi router. For example, the **WiFi Router Warden-Analyzer** is a freely available app (Figure 6.57) for Android smartphones (apps may contain many advertisements).

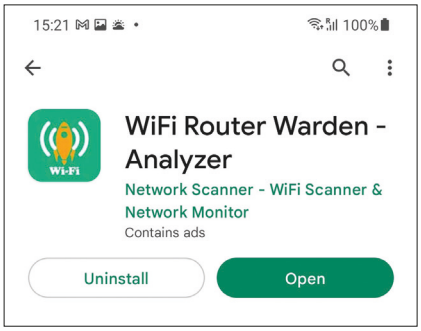


Figure 6.57: WiFi Router Warden-Analyzer app.

- Open the app and tick on **Who is Using My WiFi?** (Figure 6.58)

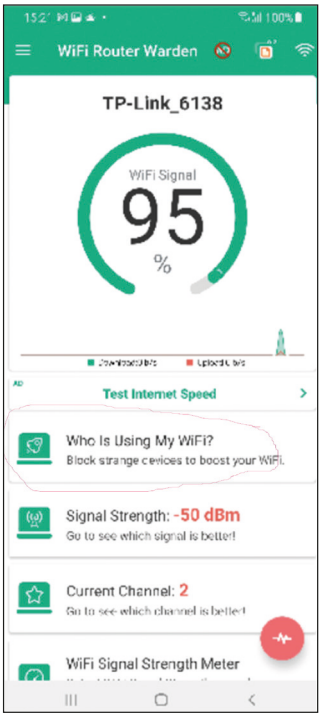


Figure 6.58: Select Who is Using My WiFi?.

- Yu should see a list of the devices connected to your Router. Note the one with the **Name: raspberrypi**. In this example, the IP address was 192.168.3.21 (Figure 6.59)

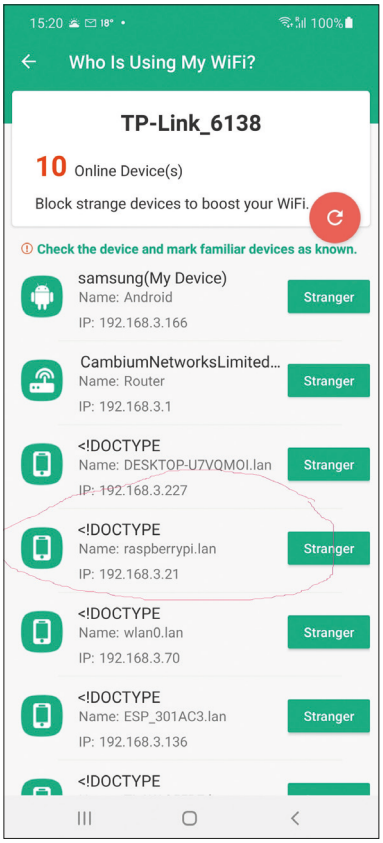


Figure 6.59: Note the one called "raspberrypi".

6.16.2 Using a PC program

There are many PC programs that can be used to scan a Wi-Fi router. Here, you will be using the one called **Wireless Network Watcher**. It can be downloaded from the following website (scroll down to click on Download). At the time of writing this book, the setup program was called: **wnetwatcher_setup.exe**:

http://www.nirsoft.net/utils/wireless_network_watcher.html

Download and click on the setup file to run the program. Figure 6.60 shows an example output from the program (only part of the output is shown here. The output displays more information about each node on the right hand side of the screen). From this output, you can see that the IP address of our Pi is 192.168.3.21.

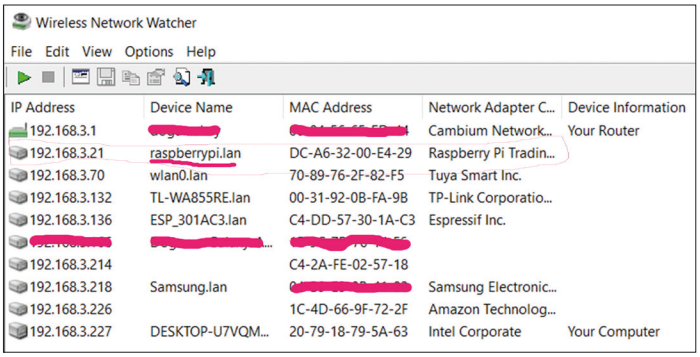


Figure 6.60: Example display of Wireless Network Watcher.

6.16.3 Using nmap

Nmap is a program that enables you to gauge the security of the network your RPi is connected to. This program has many options, but here you will only look at the option that can be used to list all the devices connected to the same Wi-Fi as your Pi. The steps are:

- install nmap

  pi@raspberrypi:~ $ **sudo apt install nmap**

- After the installation, assuming the IP address of your Zero 2 W is: 192.168.3.21, enter the following command:

  p@raspberrypi:~ $ **sudo nmap –sn 192.168.3.21/34**

- You should get a display showing all the devices connected to your Wi-Fi, together with their names and MAC addresses. Part of the display is shown in Figure 6.61.

```
MAC Address: 00:04:56:65:ED:44 (Cambium Networks Limited)
Nmap scan report for TL-WA855RE.lan (192.168.3.132)
Host is up (0.11s latency).
MAC Address: 00:31:92:0B:FA:9B (Unknown)
Nmap scan report for ESP_301AC3.lan (192.168.3.136)
Host is up (0.13s latency).
MAC Address: C4:DD:57:30:1A:C3 (Espressif)
```

Figure 6.61: Displaying the devices on the same network.

6.17 Project 30 – Fetching and displaying the real-time weather data on the screen

Description: In this project, you fetch the local data from the Open Weather Map and display some of its parameters on the screen. The aim of this project is to show how Weather Map can be accessed from the Internet.

Open Weather Map

The Open Weather Map API can be used to fetch the real-time weather data anywhere on earth. In this project, as an example, you will fetch the current real-time weather data in London and then display the important weather parameters.

Before using the Open Weather Map, you have to register to it and get an API key free of charge (up to 1000 API calls per day are free!). The steps are as follows:

- Go to Open Weather Map website:

  https://openweathermap.org/

- Click **Sign In** at the top menu and end then click **Create an Account** (Figure 6.62)

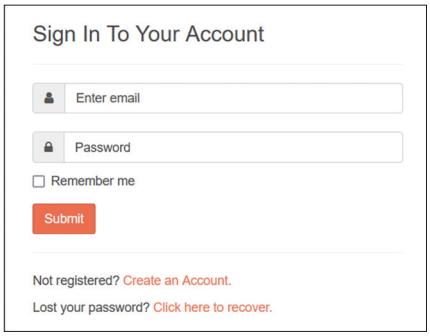


Figure 6.62: Click Create an Account.

- Login to your email to confirm the registration.
- Click to **Subscribe** to **Current Weather Data** (Figure 6.63)

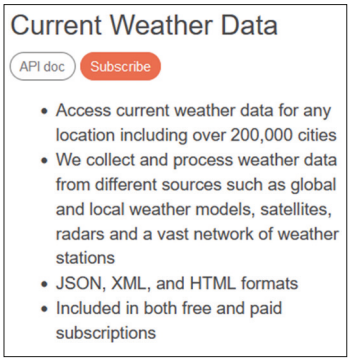


Figure 6.63 Subscribe to Current Weather Data.

- You should now click Get API key to get an API. The API key will be sent to your email address. It is something like: 05ff9a8a5d66e4f3115c50d50ff7dAA9. The key will be activated in a few hours.

- Now, you have to install the Python library (called: **pyowm**) for Open Weather Map. Enter the following command to install the library:

  pi@raspberrypi:~ $ **pip install pyowm**

- You are now ready to write your Python program and fetch and display the current weather every 15 seconds (until stopped by the user). See the following website for a detailed list of available commands:

  https://pyowm.readthedocs.io/en/latest/v3/code-recipes.html

Program listing: Figure 6.64 shows the program listing (Program: **LondonWeather.py**). At the beginning of the program, you imported the **pyowm** library. After entering the API key assigned to you, go ahead and display the weather data as shown in the program. The display is updated every 15 seconds until stopped by the user.

```
#=======================================================
#     DISPLAY LONDON WEATHER IN REAL-TIME
#
# This program fetches the real-time weather data
# for a give city from Open Weather MAp and displays
# on the screen every 15 seconds
#
# Author: Dogan Ibrahim
# File  : LondonWeather.py
# Date  : January, 2023
#=======================================================
from pyowm import OWM          # import Open Weather Map
import time                    # import time

#
# Enter your API key below
#
weather = OWM('05ff9a8a5d66e4f3115c5YOUR API')
manager = weather.weather_manager()

#
# Now we enter the city and country details
#
cty = manager.weather_at_place('London,GB')
MyWeather = cty.weather

while True:
   W = MyWeather.wind()
   print("       Wind: ", W['speed'], " m/s")
   print("       Wind: ", W['deg'], " Degrees")
```

```
print("   Humidity: " , MyWeather.humidity)

print("       Rain: " , MyWeather.rain)

T =  MyWeather.temperature('celsius')
print("   Min Temp: ", T['temp_min'], " C")
print("   Max Temp: ", T['temp_max'], " C")
print(" Feels like: ", T['feels_like'], " C")

print("   Pressure: ", MyWeather.barometric_pressure(), " hPa")

print("     Clouds: " , MyWeather.clouds)

print("Visibility : ", MyWeather.visibility_distance, " km")

print("    Sunrise: ", MyWeather.sunrise_time(timeformat='date'))
print("     Sunset: ", MyWeather.sunset_time(timeformat='date'))

time.sleep(15)
```

Figure 6.64: Program: **LondonWeather.py**.

Run the program as follows (notice that the program is run from directory **pyowm**):

pi@raspberrypi:~/pyowm $ **python3 LondonWeather.py**

A sample output from the program is shown in Figure 6.65.

```
pi@raspberrypi:~/pyowm $ python3 LondonWeather.py
       Wind:  3.6  m/s
       Wind:  220  Degrees
   Humidity:  84
       Rain:  {}
   Min Temp:  10.58  C
   Max Temp:  12.11  C
 Feels like:  10.83  C
   Pressure:  {'press': 1006, 'sea_level': None}  hPa
     Clouds:  98
Visibility :  10000  km
    Sunrise:  2022-12-25 08:05:25+00:00
     Sunset:  2022-12-25 15:55:25+00:00
```

Figure 6.65: Sample output.

6.18 Using TFT displays with the Raspberry Pi Zero 2 W

A thin-film transistor display is a type of liquid crystal display that makes use of thin-film transistor technology in order to improve qualities such as contrast and addressability. TFT is also called *active-matrix LCD technology*. In TFT technology, an individual transistor is used to drive each individual pixel, allowing for much faster response times. The TFT in the LCD controls individual pixels in the display by setting the level of the electric field across the three (red, green, blue) liquid crystal capacitors in the pixel in order to control the

polarization of the crystal material which determines the amount of light that reaches the color filter from the backlight. TFT displays are available in many different physical sizes and also by different number pixel numbers.

In this section, you will use a 1.8-inch TFT display within a Raspberry Pi Zero 2 W project.

6.18.1 TFT display used

The TFT display used in the project in this book is shown in Figure 6.66. This display has the following specifications:

Size:	1.8 inch
Resolution:	128 × 160
Driver IC:	ST7735
Interface:	4-wire SPI
SD:	On-board SD card adapter

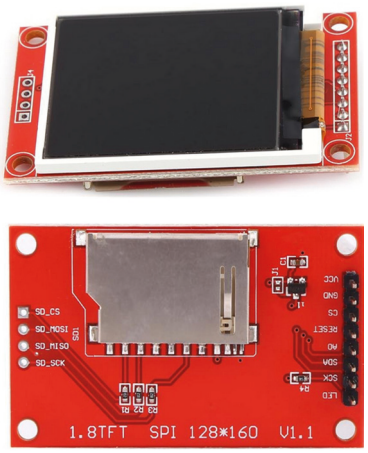


Figure 6.66: TFT display used.

The display has an 8-bit connector on one side for the display logic, and a 4-bit connector on the other side for the SD card interface. You will be using the display logic interface which has the following pins:

8-pin connector

LED	Display backlight control
SCK	Display SPI Clock
SDA	Display SPI Data
A0	Display address select (data/command, DC)
RESET	Display reset (RST)
CS	Display chip select (CS)
GND	Power supply ground
VCC	Power supply

6.18.2 Connecting a TFT display to the Raspberry Pi Zero 2 W

The TFT display is connected to the Zero 2 W as follows (see Figure 6.67), where SPI0 is used for the communication:

TFT display	Pico port pin
SCK	GPIO11 – SCLK
SDA (or MOSI)	GPIO10 – MOSI
A0 (or DC)	GP17
RESET (or RST)	GP27
CS	GPIO8 – CE0
GND	GND
VCC	+3.3 V
LED	+3.3 V

Figure 6.67: TFT display-to-RPi Zero 2 W interface.

6.18.3 ST7735 TFT display driver library

Before using the 1.8-inch TFT display with your Zero 2 W, you have to install the display driver ST7735 to your Zero 2 W. The steps are as follows:

- Start **raspi-config** and enable the **SPI** interface under **Interface Options**:

```
pi@raspberrypi:~ $ sudo raspi-config
```

- Install the following:

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt install python3-spidev python3-pip
python3-pil python3-numpy
pi@raspberrypi:~ $ sudo python3 -m pip install st7735
```

Figure 6.68 shows the pixel co-ordinates of the TFT display used.

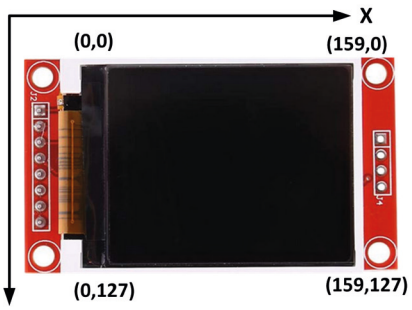


Figure 6.68: Co-ordinates of the display used.

6.18.4 Example display — writing text

Figure 6.69 shows an example program (Program: **TFTexample.py**) for displaying text on the TFT. Here, text **RASPBERRY PI ZERO 2 W** is displayed. At the beginning of the program, the PIL library and the ST7735 libraries are imported, Then, the display interface is configured and text is written starting from TFT coordinate (5, 60). For more examples on how to use the TFT library, head over to the following web site:

https://github.com/pimoroni/st7735-python

```
#=========================================================
#             DISPLAY TEXT ON TFT
#             ===================
#
# Author: Dogan Ibrahim
# File  : TFTexample.py
# Date  : January, 2023
#=========================================================
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
import ST7735

msg = "RASPBERRY PI ZERO 2 W"
disp = ST7735.ST7735(port=0,cs=0,rst=27,dc=17,width=128,height=160,
rotation=90,invert=False,offset_left=0,offset_top=0)
disp.begin()

width = disp.width
height =disp.height
img=Image.new('RGB', (160,128),color=(0,0,0))
draw=ImageDraw.Draw(img)
font=ImageFont.load_default()

draw.text((5,60),msg, font=font)
disp.display(img)
```

Figure 6.69: Program: **TFTexample.py**.

Figure 6.70 shows the text on the display. The display background is set to black using the statement **color=(0, 0, 0).**

Figure 6.70: Displayed text.

Some useful TFT commands are given below:

Rectangle
To draw a rectangle with the desired outline color and fill color, use the following statement. x0 any y0 are the top-left coordinates and xm and ym, are the bottom-right coordinates:

```
draw.rectangle((x0, y0, xm, ym), outline = (0, 0, 0), fill = (0, 0, 0))
```

For example, the statement **draw.rectangle((5, 5, 150, 120), outline = (0, 255, 0))** draws a green empty rectangle with the top left coordinate at (5, 5) and bottom right coordinate at (150, 120).

Ellipse
To draw an ellipse, use the following statement:

```
draw.ellipse((x0, y0, xm, ym), outline = (0, 0, 0), fill = (0, 0, 0))
```

Line
To draw a line, use the following statement:

```
draw.line((x0, y0, xm, ym), fill = (0, 0, 0))
```

Polygon
To draw a polygon, use the following statement:

```
draw.polygon([(x1, y1), (x2, y2), (x3, y3), x4, y4)], outline = (0, 0, 0),
fill = (0, 0, 0))
```

6.18.5 Example display — displaying various shapes

Figure 6.71 shows a program (Program: **TFTshapes.py**) which displays various shapes. The display is shown in Figure 6.72.

```
#============================================================
#             DISPLAY VARIOUS SHAPES ON TFT
#             ============================
#
# Author: Dogan Ibrahim
# File  : TFTshapes.py
# Date  : January, 2023
#============================================================
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
import ST7735

msg = "RASPBERRY PI ZERO 2 W"
disp = ST7735.ST7735(port=0,cs=0,rst=27,dc=17,width=128,height=160,
rotation=90,invert=False,offset_left=0,offset_top=0)
disp.begin()

width = disp.width
height =disp.height
img=Image.new('RGB', (160,128),color=(0,0,0))
draw=ImageDraw.Draw(img)
font=ImageFont.load_default()

draw.rectangle((5, 5, 150, 120), outline = (0, 255, 0))
draw.ellipse((20, 20, 100, 80), outline = (0, 0, 255))
draw.line((10, 100, 60, 100), fill = (0, 255, 0))
draw.polygon([(100, 80), (100, 110), (130, 90), (145, 80)], outline = (0, 255,
255),
fill = (0, 255, 0))
draw.text((7,7),msg, font=font,fill=(0,255,0))
disp.display(img)
```

Figure 6.71: Program: **TFTshapes.py**.

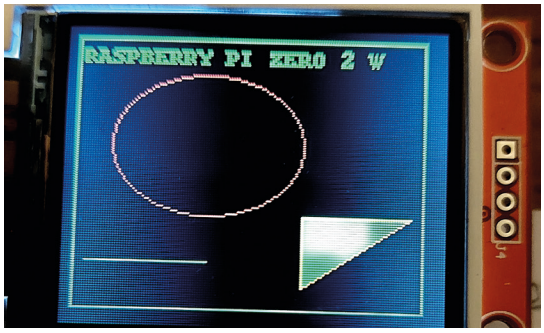


Figure 6.72: Displayed shapes.

6.18.6 Project 31 — Displaying the local weather data on TFT

Description: This project is similar to Project 30, but here the local weather in London is displayed in real time on the TFT display and is updated every 5 seconds.

The circuit diagram of the project is same as in Figure 6.67.

Program listing: Figure 6.73 shows the program listing (Program: **TFTweather.py**). Make sure the required libraries are imported to the program as illustrated in Figure 6.64. After configuring the TFT display and Open Map Weather, an endless loop is formed. Inside this loop the heading **LONDON WEATHER** is displayed inside a double rectangle. Then, the following weather parameters are extracted and displayed:

- Humidity
- Minimum temperature
- Maximum temperature
- Wind speed
- Wind direction
- Atmospheric pressure
- Current date and time

The display is updated every 5 seconds.

```
#=======================================================
#     DISPLAY LONDON WEATHER IN REAL-TIME ON TFT
#
# This program fetches the real-time weather data
# for a give city from Open Weather MAp and displays
# on the screen every 5 seconds
#
# Author: Dogan Ibrahim
# File  : TFTweather.py
# Date  : January, 2023
#=======================================================
from pyowm import OWM          # import Open Weather Map
import time                    # import time
from PIL import Image
from PIL import ImageDraw
from PIL import ImageFont
import ST7735
from datetime import datetime

#
# Enter your API key below
#
weather = OWM('05ff9a8a5d66e4f3115c50d YOUR API')
manager = weather.weather_manager()
```

```
#
# TFT configuration
#
disp=ST7735.ST7735(port=0,cs=0,rst=27,dc=17,width=128,height=160,
rotation=90,invert=False,offset_left=0,offset_top=0)

disp.begin()
width=disp.width
height=disp.height

while True:
  img = Image.new('RGB', (160, 128), color=(0, 0, 0))
  draw = ImageDraw.Draw(img)
  font = ImageFont.load_default()

  draw.rectangle((3, 3, 150, 124), outline = (0, 0, 255))
  draw.rectangle((6, 8, 147, 120), outline = (0, 0, 255))

  draw.text((30, 20), "LONDON WEATHER", fill = (0,255, 0))

#
# Now we enter the city and country details
#
  cty = manager.weather_at_place('London,GB')
  MyWeather = cty.weather

  h =      "Humidity (%) = " + str(MyWeather.humidity)
  temp = MyWeather.temperature('celsius')
  wind = MyWeather.wind()
  p = MyWeather.barometric_pressure()

  tmin =   "Temp Min (C) = " + str(temp['temp_min'])
  tmax =   "Temp Max (C) = " + str(temp['temp_max'])

  wspeed = "Wind (m/s)   = " + str(wind['speed'])
  wdir =   "Wind (deg)   = " + str(wind['deg'])

  press =  "Pressure     = " + str(p['press'])

  draw.text((20, 40), h, fill = (255, 255, 255))
  draw.text((20, 50), tmin, fill = (255, 255, 255))
  draw.text((20, 60), tmax, fill = (255, 255, 255))
  draw.text((20, 70), wspeed, fill = (255, 255, 255))
  draw.text((20, 80), wdir, fill = (255, 255, 255))
  draw.text((20, 90), press, fill = (255, 255, 255))
```

```
d = datetime.now().strftime("%d-%m-%y %H:%M:%S")
draw.text((20, 105), str(d), fill = (0, 255, 0))
disp.display(img)
time.sleep(5)
```

Figure 6.73: Program: **TFTweather.py**.

Figure 6.74 shows the local weather data in London displayed on the TFT.

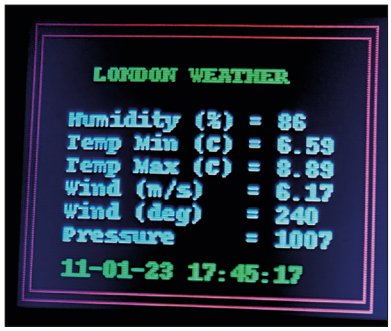


Figure 6.74: Displaying the local weather data.

CHAPTER 7 • Using Node-Red with the Raspberry Pi Zero 2 W

7.1 Overview

Node-RED is an open-source visual editor for wiring the Internet of Things, produced by IBM. Node-RED comes with substantial number of nodes to manage a variety of tasks. The required nodes are selected and joined together to perform a certain task. Node-RED is based on flow-type programming where nodes are configured and joined together to form the application program. There are nodes for doing fairly complex tasks, including web access, Twitter, E-mail, HTTP, Bluetooth, MQTT, controlling the GPIO ports, etc. The nice thing about Node-RED is that the programmer does not need to learn to do complex programs. For example, an email can be sent by joining a few nodes together and writing a few lines of code. In this chapter, you will develop some projects using Node-RED running on your Raspberry Pi Zero 2 W.

7.2 Installing and running Node-RED on the Raspberry Pi Zero 2 W

Node-RED should already be installed on your Zero 2 W. If not, you can easily install it by entering the following command:

pi@raspberrypi:~ $ **bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered) --node16**

You can start Node-RED by entering the following command in the command mode (characters entered by the user are in bold for clarity):

pi@raspberrypi:~ $ **node-red-start**

When started, you should see some messages scrolling on your screen. A part of these messages is shown in Figure 7.1.

```
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.238:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use   node-red-stop                         to stop Node-RED
Use   node-red-start                        to start Node-RED again
Use   node-red-log                          to view the recent log output
Use   sudo systemctl enable nodered.service  to autostart Node-RED at every boo
Use   sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
12 Jan 09:20:14 - [info]
Welcome to Node-RED
===================
12 Jan 09:20:14 - [info] Node-RED version: v3.0.2
12 Jan 09:20:14 - [info] Node.js  version: v12.22.12
12 Jan 09:20:14 - [info] Linux 5.15.61-v7+ arm LE
```

Figure 7.1: Part of the messages displayed when Node-RED started.

A list of the important commands is given below:

node-red-start: start Node-RED

node-red-stop:	stop Node-RED
node-red-log:	view the recent log output
sudo systemctl enable nodered.service:	autostart Node-RED after every boot
sudo systemctl disable nodered.service:	disable autostart after every boot

Notice that this is a "service" type application and therefore it can be accessed from anywhere on the network, as well from other computers. To do this, you have to know the IP address of your Raspberry Pi Zero 2 W. This can be found by entering the following command at the command prompt:

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

The IP address of your Raspberry Pi Zero 2 W in this example is: 192.168.1.238.

You can now enter the following command on your web browser to display the Node-RED startup screen:

```
http://192.168.1.238:1880
```

When started, you should see the Node-RED startup screen as in Figure 7.2. When Node-RED is started as a service, pressing Cntrl+C does not stop the service since it keeps running in the background.

Figure 7.2: Node-RED startup screen (partial screen shown).

Notice that, if you are using a browser running on the same computer as Node-RED, you can access Node-RED using the following url:

```
http://localhost:1880
```

7.3 Node-RED interface to external world

Node-RED is normally accessed from a web browser. Figure 7.3 shows the Node-RED interface to the external world. The Raspberry Pi Zero 2 W accesses the Node-RED software through the local Wi-Fi router (or Internet). Sensors, actuators, and any other hardware are interfaced directly to the Raspberry Pi GPIO ports through the 40-pin header mounted on the Zero 2 W. The Raspberry Pi communicates with the external hardware by receiving commands from the Node-RED through the local Wi-Fi router.

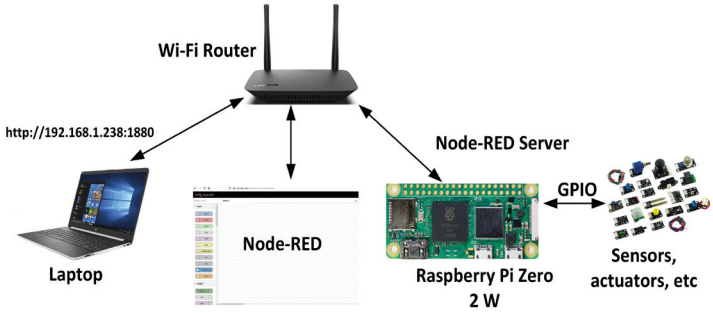


Figure 7.3: Node-RED Interface to external world.

7.4 Project 32: Hello World!

Possibly the easiest way to understand how to use Node-RED is to look at a simple flow program. In this program you will display the message **Hello World!** In the Debug window. The steps are given below.

- Start Node-RED on your Raspberry Pi Zero 2 W:

  pi@raspberrypi:~ $ **node-red-start**

- Enter the following url into your web browser to display the Node-RED screen (you will have to enter the IP address of your own Zero 2 W):

  http://192.168.1.202:1880

- From the node palette, Click on **inject** node at the left-hand side and drag it to the workspace.

- Double-click on this node, click on **Payload**, select **String** and enter the following string **Hello World!** as shown in Figure 7.4.

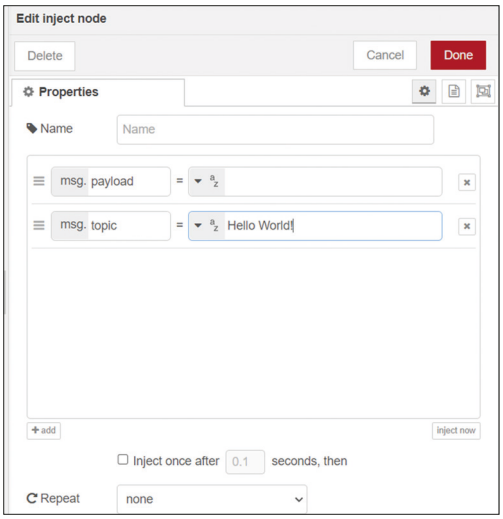


Figure 7.4: Enter Hello World!

- Click **Done**.

- Add node debug by clicking on **debug** node and dragging it to the right-hand side of the inject node already in the workspace.

- Wire the two nodes together. Place the mouse cursor over the **inject** node"s output port (a small grey square on the right-hand side of the node), then left-click and drag a wire over to the input port of the Debug node. A wire should now connect the two nodes as shown in Figure 7.5.

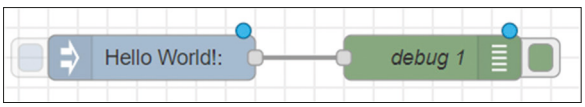


Figure 7.5: Connecting the two nodes together.

- Click the **Deploy** button at the top right-hand side of the screen.

- Click the small grey square at the right-hand side of node **debug** to activate the node. You should see the message **Successfully activated: debug** displayed by the console window.

- Select the **Debug messages** tab at the top right-hand side.

- Click the **inject** node's button, which is the small square coming out from the left-hand side of the node. Clicking the button will inject the message into the flow and towards the debug node. The output should be displayed in the debug window as shown in Figure 7.6.

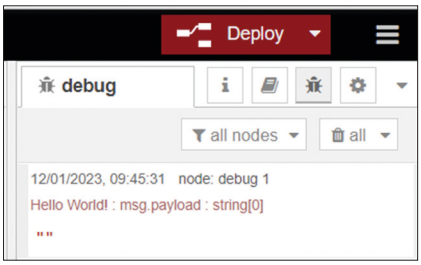


Figure 7.6: Displaying the output in the debug window.

- Notice that, if you place a wrong node on the workspace, you can click on the node and then use the keyboard Delete button to remove it.

In this example, you first created an inject node. The inject node can be used to manually trigger a flow by clicking the node's button within the workspace. It can also be used to automatically trigger flows at regular intervals. You then stored the string **Hello World!** in this node. Next, you created a debug node and connected the two nodes together and activated the debug node. By triggering the inject node, the string is sent to the debug node and is subsequently displayed in the debug window. The Debug node can be used to display messages in the Debug sidebar within the editor. In addition to the displayed messages, the debug sidebar includes information about the time the message was received as well as the ID of the Debug node that sent the message. Clicking on the ID reveals the node that sent the message.

Modified Program

You can modify the flow program, for example to display the string every 5 seconds. The steps to do this are given below:

- Double-click on **inject** node.
- Set the Repeat Interval to every 5 seconds (Figure 7.7) and click **Done**.

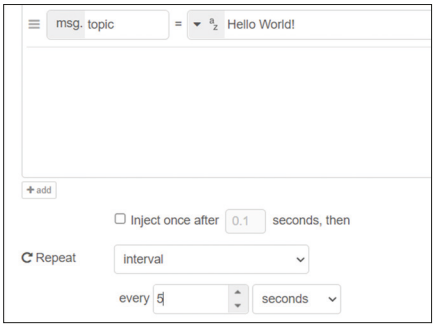


Figure 7.7: Setting the repeat interval to 5 seconds.

- Click **Deploy**.

- Activate the **debug** node.
- Click the **inject** node"s button. You should see the message displayed every 5 seconds like in Figure 7.8.

Figure 7.8: Displaying the text every 5 seconds.

7.5 Core nodes

When Node-RED is installed on the Raspberry Pi Zero 2 W, it starts with categories of nodes known as the *core nodes*. Let"s now briefly look at the nodes in each category.

7.5.1 Input nodes

inject: this node injects a timestamp or text into a message. The node can be configured either to inject manually or at regular intervals.

catch: If a node gives an error while managing a message, it is likely that the flow will halt. The catch node can be used to catch the errors and return messages about the error.

mqtt in: this node connects to a MQTT broker and subscribes to messages from the specified topic.

http in: this node creates an HTTP end-point for creating web services.

websocket: this node provides an endpoint for a browser so that a websocket connection can be established with Node-RED.

tcp: this node is used to establish TCP communications on a specified port. It accepts incoming TCP requests.

udp: this node is used to establish communication using the UDP protocol. It accepts incoming UDP packets.

serial in: this node reads serial data from the serial port of the local device. The node can be configured to read at specific times.

7.5.2 Output nodes

debug: this node is used to view messages on the Debug window. The node can be configured to display either the entire message or just the msg.payload.

mqtt out: this node connects to a MQTT broker and publishes messages.

http response: this node sends responses back to the HTTP requests received from HTTP input node.

http request: this node sends HTTP requests and returns the response.

websocket: this node sends msg.payload to the configured websocket.

tcp: this node replies to a configured TCP port.

udp: this node sends UDP messages to the configured UDP host where the IP address and port number are specified. Broadcast is supported.

serial out: this node sends data to the defined serial port of the local device. The node can be configured to send some control characters such as newline after a message.

7.5.3 Function nodes

function: this is a processing node, used to write JavaScript compatible functions.

template: this node is useful for constructing messages and configuration files where name:value pairs are inserted into the template.

delay: this node delays messages by a random or specific time.

trigger: this node can be used as a watchdog timer. It can also be used to create two output messages with time intervals when an input message is received.

comment: this node is used to insert comments.

http request: this node is used to construct and then send a HTTP request to a given URL.

tcp request: this node sends msg.payload to a TCP server and expects a response. The node is configurable — for example, it can be configured to wait for a specific character, or to return immediately.

switch: this node is used to route messages based on their properties.

change: this node can be used to set, change, or delete the properties of incoming messages.

range: this node maps numerical input data to new output data.

csv: this node parses msg.payload and converts to or from CSV format. The input can be a string in which case a JavaScript object is output. If on the other hand the input is a JavaScript object, then a CSV formatted string is outputted.

html: this node is used to extract data from an html type document in msg.payload.

json: this node converts to or from a JSON object and JavaScript object.

xml: this node converts to or from XML format and JavaScript object.

random: this node generates a random number between a high and a low value.

smooth: this node is used for various functions such as max, min, mean, high, and low pass filter.

rbe: this is the "rbe" (Report By Exception) node which generates message if its input is different from the previous input.

7.5.4 Social nodes

email in: this node is used to read new emails as they arrive at the local host. It can be configured to read repeatedly.

twitter in: this node is used to return tweeter messages.

email out: this node is used to send email messages.

twitter out: this node Tweets the msg.payload on the configured user account. Text and binary (image) messages can be sent.

7.5.5 Storage nodes

tail: this node tails a file and injects the contents into the flow.

file in: this node reads the specified file and sends it contents as msg.payload.

file: this node writes the msg.payload to the specified file.

7.5.6 Analysis nodes

sentiment: this node analyses the msg.payload and scores incoming words using the AFINN-165 wordlist and attaches a "sentiment.score" property to the msg.

7.5.7 Advanced nodes

watch: this node watches a folder for changes and sends events when files are added, changed. created or deleted.

feedparse: this node monitors an RSS/atom feed for new entries and if there are new entries, it delivers them as messages.

exec: this node calls to a system command and provides stdout, stderr, and the return code.

7.5.8 Raspberry Pi nodes

rpi gpio in: this is the Raspberry Pi input node. Depending on the state of the input, this node generates a msg.payload with either a 0 or 1. You"ll be using this node in the Raspberry Pi-based projects in later chapters.

rpi gpio out: this is the Raspberry Pi output node. The selected hardware pin of the Raspberry Pi is set High or Low depending whether msg.payload is 0 or 1. You"ll be using this node in the Raspberry Pi-based projects in later chapters.

rpi mouse: this is the Raspberry Pi mouse button node. The node generates a 0 or 1 when the selected mouse button is clicked and released. A USB mouse is required.

rpi keyboard: this node is used to capture keystrokes from a USB keyboard.

7.6 Project 33: Dice number

In this project, you will explore the **random** node. The program will display a random number between 1 and 6 every time it is clicked to start. The steps are as follows:

- Create an **injection** node with the title **Your Chance**.
- Create a **random** node and name it as **Dice**. Double-click to edit it and set to generate whole integer numbers between 1 and 6 as shown in Figure 7.9. Click **Done**.

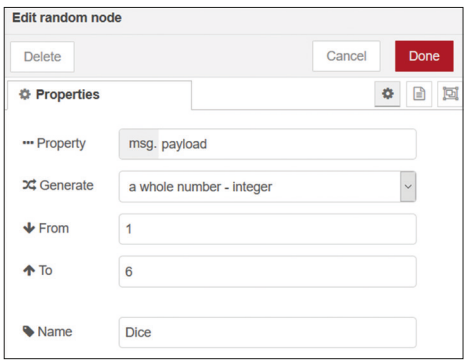


Figure 7.9: Generating random numbers between 1 and 6.

- Create a **debug** node.

- Join all three nodes as shown in Figure 7.10 and click **Deploy**.

Figure 7.10: Joining all three nodes.

- Click the **inject** node"s button. You should see a number between 1 and 6 displayed in the Debug window. Every time you click the inject node"s button, a random number will be generated and displayed between 1 and 6 as pictured in Figure 7.11.

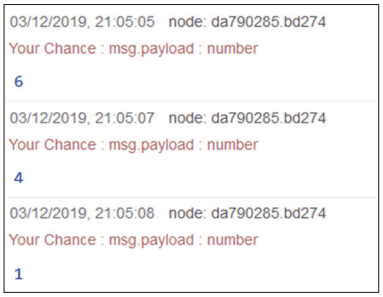


Figure 7.11: The Debug window.

7.7 Project 34: Double dice numbers

Description: In many dice games such as backgammon, two dice are thrown at a time by each player before making a move. The program given in the previous project is modified in this project, so that two random dice numbers are generated and displayed every time the **inject** node's button is clicked. This program uses two **random** nodes and a **join** node, and the steps to design the flow are given below.

- Create an **injection** node with the title **Your Chance**.

- Create two **random** nodes with the names **Dice1** and **Dice2**. Double-click to edit them and set to generate whole integer numbers between 1 and 6 for both nodes as in the previous project.

- Create a **join** node with the name **join payloads**. Double-click this node and edit as shown in Figure 7.12. Notice that the message parts is set to 2 and are separated with a space character. Click **Done**.

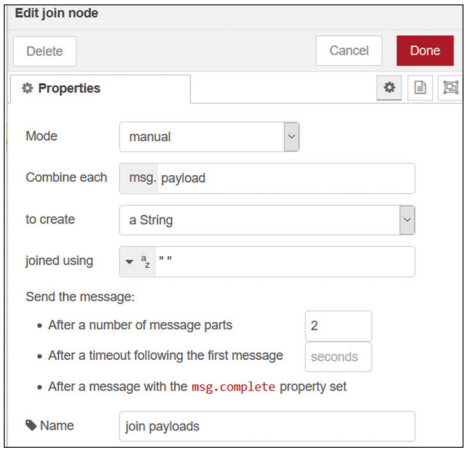


Figure 7.12: Editing the "join" node.

- Create a debug node as before and activate it.

- Join the five nodes as shown in Figure 7.13 and click **Deploy**.

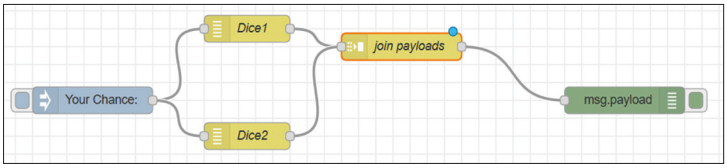


Figure 7.13: Joining the five nodes together.

- Click the **inject** node"s button to generate two random numbers next to each other in the Debug window as shown in Figure 7.14.

Figure 7.14: The Debug window.

Notice here that, if the **After a number of message parts** in node "join" was set to 1, then the two dice numbers would not have been displayed adjoining as shown in Figure 7.15.

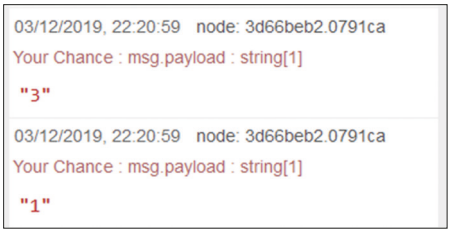


Figure 7.15: Displaying the two numbers separately.

7.8 Project 35: LED control

Description: In this project, you will connect an LED to one of the Raspberry Pi Zero 2 W GPIO pins and then turn the LED ON or OFF from a Node-RED flow program. The aim of this project is to show how an LED can be connected to a GPIO port pin and how it can be controlled from a Node-RED flow program.

Circuit Diagram: In this project, an LED is connected to port pin GPIO17 (pin 11) of the Zero 2 W through a 470-ohm current-limiting resistor.

Node-RED flow program: Figure 7.16 shows the flow program. In this program, two **inject** nodes and an **rpi gpio out** node are used.

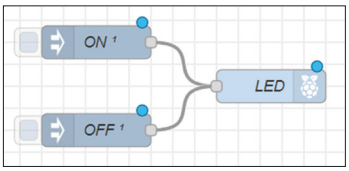


Figure 7.16: Flow program of the project.

The steps are as follows:

- Create an **inject** node with the configuration as shown in Figure 7.17.

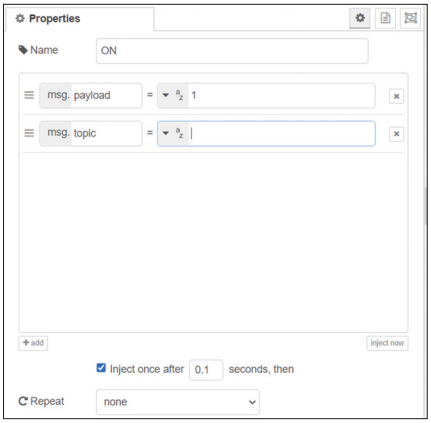


Figure 7.17: Create an inject node.

- Create another **inject** node with the configuration as shown in Figure 7.18.

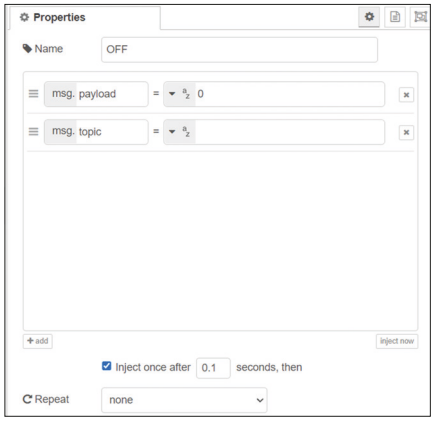


Figure 7.18: Create another inject node.

- Create a **rpi gpio out** node with the configuration as shown in Figure 7.19, set the Type to Digital output, initialize pin state to 0, and name it as LED. Click **Done**.

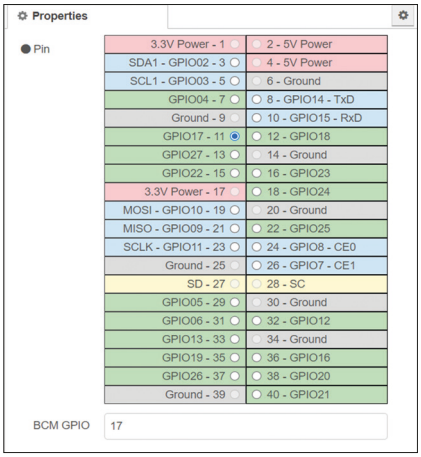


Figure 7.19: Creating an "rpi gpio out" node.

- Join the 3 nodes together as shown in Figure 7.16, and click **Deploy**.

- Connect the LED to your Raspberry Pi Zero 2 W.

- Click the button of **inject** node ON, you should see the LED turning ON. Click the button of **inject** node OFF, and this time the LED should turn OFF.

As shown in Figure 7.20, if desired, you can connect **debug** nodes to the flow so that the state of the LED can be displayed at any time in the Debug window (don't forget to click **Deploy** and then **Debug messages**). Figure 7.21 shows the LED state as the LED is turned ON or OFF.

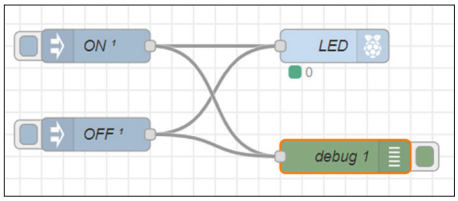


Figure 7.20: Connecting debug nodes to the flow.

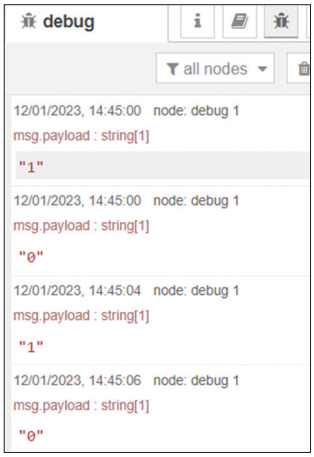


Figure 7.21: State of the LED displayed.

7.9 Project 36: Flashing an LED

Description: In this project, you will connect an LED to one of the Raspberry Pi Zero 2 W GPIO pins as in the previous project, and then flash the LED every second. The aim of the project is to show how an LED connected to a GPIO port pin can be flashed using a **trigger** node.

Node-RED flow program: Figure 7.22 shows the flow program. In this program, an **inject** node, **a trigger** node, and a **rpi gpio out** node are used.

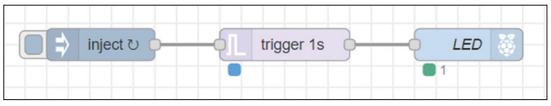


Figure 7.22: Flow program of the project.

The steps are as follows:

- Create an **inject** node which runs regularly every 2 seconds, whose configuration is shown in Figure 7.23. Click **Done**.

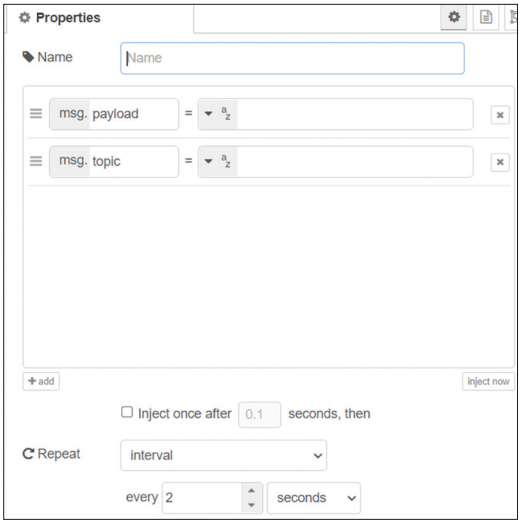


Figure 7.23: Creating an "inject" node.

- Create a **trigger** node to send out 1, wait for 1 second and then send out 0, with the configuration as shown in Figure 7.24. Click **Done**.

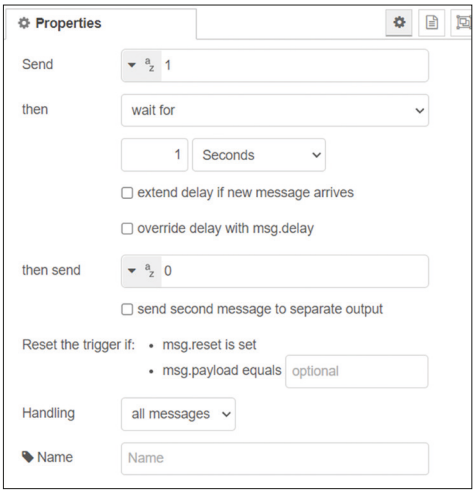


Figure 7.24: Creating a trigger node.

- Create the **rpi gpio out** node as in the previous project. Join the three nodes and click **Deploy**.

- Click the **inject** node's button. You should see the LED flashing every second.

In this project, the **inject** node activates the **trigger** node every two seconds, which in turn controls the LED by turning it ON for a second, and OFF for a second.

You could insert a **debug** node to the flow program so that the state of the LED can be displayed. Additionally, you can see the timing accuracy of the project in the Debug window.

7.10 Project 37: Pushbutton switch input

Description: This is a quite simple project in which a pushbutton switch and an LED are connected to the Zero 2 W. The project turns ON the LED when the button is pressed. The aim of this project is to show how a button can be used in a Node-RED based project.

The button used in this project is a small component with four pins as shown in Figure 7.25. Actually, the button is a 2-pin device with the pins paralleled for convenience.

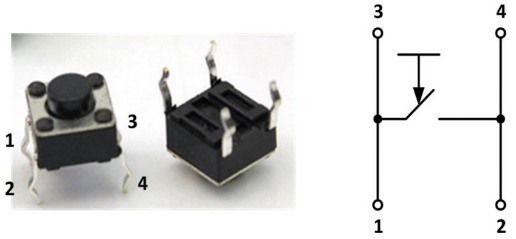


Figure 7.25: Small button and its connection diagram.

Mechanical switches have bouncing problems. When a mechanical switch is operated, its mechanical contacts bounce rapidly (i.e., move from one state to another state) until they settle. Although this settling time may seem to be small (around 10 ms), it is actually a long time considering a microcontroller"s processing time. Special switch debouncing circuits can be used to eliminate this effect called "contact bounce." Luckily, the **rpi gpio in** the node provides a parameter to set the switch debouncing time so that the contacts are not read before this time expires. This makes sure that the correct switch output state (ON or OFF) is read by the microcontroller.

Circuit Diagram: The LED is connected to port pin GPIO17 through a 470-ohm current-limiting resistor. Port pin GPIO2 is connected to one side of the button at GND potential, while the other side of the button is connected to +3.3 V through a 10 k-ohm resistor (Figure 7.26)

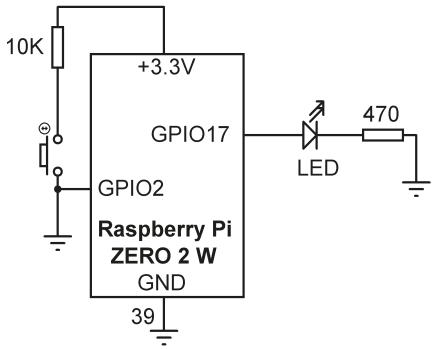


Figure 7.26: Circuit diagram.

Node-RED flow program: Figure 7.27 shows the flow program. In this program two nodes are used.

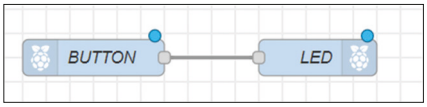


Figure 7.27: Flow program of the project.

The steps are:

- Create an **rpi gpio in** node and name it as **BUTTON**, set the pin name to GPIO2, set the resistor to pulldown.
- Create a **rpi gpio out** node and name it as **LED**, set the pin name to GPIO17, and initialize the pin level to 0.
- Join the two nodes as shown in Figure 7.27, and click **Deploy**.
- You should see that the LED is initially OFF. Pressing the button should change the state of port GPIO2 and the LED will turn ON. You should see the state of port GPIO2 on the flow diagram under node BUTTON.

7.11 Using the ALEXA in Node-RED projects with the Raspberry Pi Zero 2 W

Alexa is a sound-based virtual assistant device developed by Amazon in 2014, and first used in Amazon Echo and Amazon Echo Dot. Alexa interacts with the user through sound and it can obey the commands given by a speaker. For example, Alexa can play a required piece of music, it can provide weather reports, ask questions, give traffic reports, supply current news, give recipes, translate words to other languages, and many more. Alexa can also be used to control a device through sound commands and therefore it can be used in home automation. The device is activated by giving the "attention, please" expression: "Alexa". The user can then communicate with the system. Alexa is connected to the user"s local Wi-Fi router and gets the answers to the user"s questions from the Amazon cloud. The

latest model of Alexa at the time of authoring this book was the 3rd generation of devices. Figure 7.28 shows the 3rd Gen Alexa Dot device.

Figure 7.28: Third-Gen Alexa Dot.

In this section, you will combine Node-RED with Alexa and the Raspberry Pi Zero 2 W and learn how to control the GPIO ports of the Zero 2 W by issuing spoken commands to Alexa.

7.11.1 Project 38: Controlling an LED using Alexa

Description: In this project, an LED is connected to one of the Raspberry Pi Zero 2 W GPIO ports. The LED is turned ON and OFF by giving spoken commands to Alexa. The aim of this project is to show how Alexa can be used with Node-RED in Zero 2 W projects to control a device remotely.

Block Diagram: The block diagram of the project is shown in Figure 7.29.

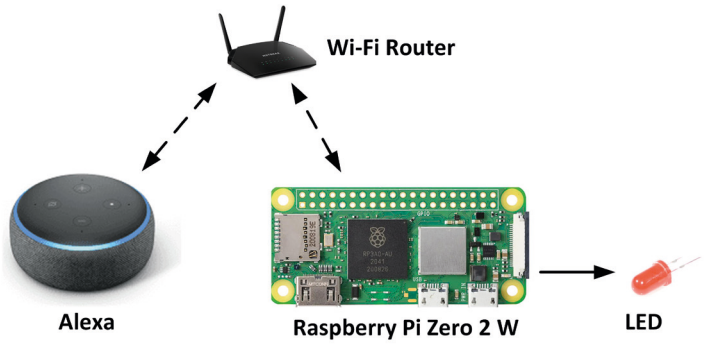


Figure 7.29: Block diagram of the project.

Circuit Diagram: In this project, a small LED is connected to port pin GPIO17 (pin 11) of the Zero 2 W through a 470-ohm current-limiting resistor.

Node-RED flow program: Before developing your flow program, you have to install Alexa node to your Node Palette. Because the Alexa nodes "listen" on port 80, it is important that you start your Node-RED on the Raspberry Pi with a super user command. i.e.,

```
pi@raspberrypi:~ $ sudo node-red-start
```

Then, the steps to install Alexa are as follows:

- Start the node-red screen on your PC web browser by entering the Zero 2 W IP address followed by :1880.
- Click **Menu** → **Manage palette**, click **Install**.
- Enter **node-red-contrib-amazon-echo** and click **install**.
- You should see two new nodes called **amazon echo hub** and **amazon echo device** added to your Node Palette.

If you find that your Raspberry Pi GPIO nodes are not listed in the Nodes Palette, you can re-install them using the following steps:

- Click **Menu** → **Manage palette**, click **Install**.
- Enter **node-red-node-pi-gpio** and click **install**.

You are now ready to develop our flow program.

Figure 7.30 shows the flow program which consists just four nodes: an **amazon echo hub** node, an **amazon echo device** node, a **function node**, and an **rpi gpio out** node.

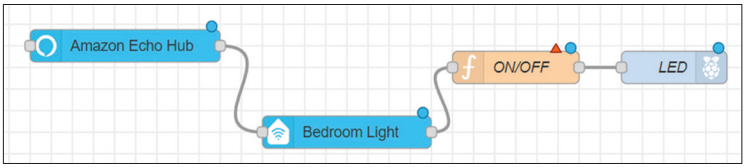


Figure 7.30: Flow program of the project.

The steps to perform are as follows.

- Create an **amazon echo hub** node and make sure that the **Port** is set to 80 and **Process Input** is set to No.
- Create an **amazon echo device** node and set the **Name** to **Bedroom Light** (this is the name that Alexa will associate with the LED — you should choose your own name here).
- Create a **function** node and name it as **ON/OFF**. Enter the following statements inside this node. This node outputs 1 if the message coming from Alexa is **on**, or it outputs 0 if the message coming from Alexa is **off** (Figure 7.31).

```
var out = 0;
if(msg.payload == "on")
  out = 1;
else
if(msg.payload == "off")
  out = 0;
msg.payload = out;
return msg;
```

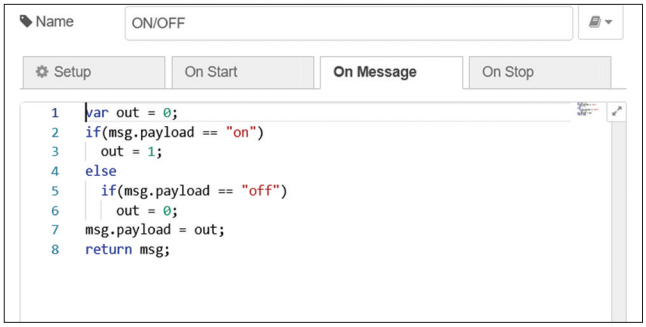


Figure 7.31: Node "Function" contents.

- Create an **rpi gpio out** node and name it as **LED**. Set the **Pin** to GPIO17 **Type:** digital output, and **Initialize** the pin state to logic 0.

- Join all the nodes as in Figure 7.30 and click **Deploy**.

Note: Alexa Echo uses port 80 for communication, You may find that port 80 is not available on your Zero 2 W and the message **unable to start port 80** may be displayed under the Amazon Echo Hub. Also, you may get the following error message in Debug window:

"Error: listen EACCES: permission denied 0.0.0.0.:80".

If you get this error message, redirect port 80 to say 8080 using the following commands. Also change the port number to 8080 by double-clicking on node **Amazon Echo Hub**:

sudo iptables -t nat -I OUTPUT -p tcp -d 127.0.0.1 --dport 80 -j REDIRECT --to-ports 8080

sudo iptables -t nat -I PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 8080

Note that the above port redirection commands are not retained after a reboot.

To test the project, ask Alexa: "**Alexa, discover devices**". After a minute or so Alexa will confirm that it has detected your device (**Bedroom Light** in this project).

To turn the LED ON, ask Alexa: "**Alexa, turn on bedroom light**". Alexa will respond with OK and the light will turn ON.

To turn the LED OFF, ask Alexa "**Alexa, turn off bedroom light**". Alexa will respond with OK and the light will turn OFF.

The nodes can be used to turn on/off a device or to dim a light and it is supported by the 1st, 2nd, and 3rd generation Alexa devices. Notice that you can add as many **amazon echo devices** as you wish to your design for the control of various equipment. An example project isgiven below where you get to add a buzzer to your design.

7.11.2 Project 39: Controlling an LED and a buzzer using Alexa

Description: In this project, an LED and a buzzer are connected to the Raspberry Pi Zero 2 W GPIO ports. The LED and the buzzer are turned ON and OFF by issuing spoken commands to Alexa. The aim of this project is to show how multiple devices can be controlled by Alexa.

Block Diagram: The block diagram of the project is shown in Figure 7.32.

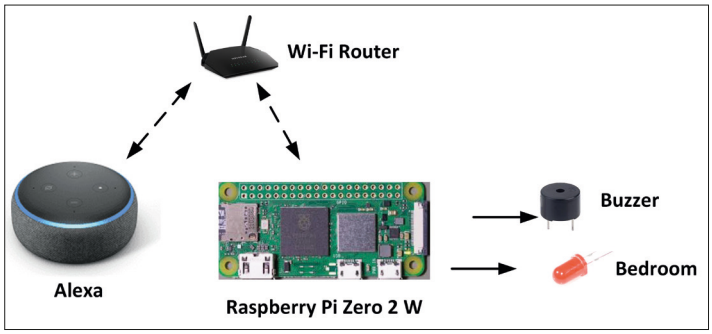


Figure 7.32: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 7.33. The LED is connected to GPIO17 (pin 11) and the buzzer to GPIO27 (pin 13).

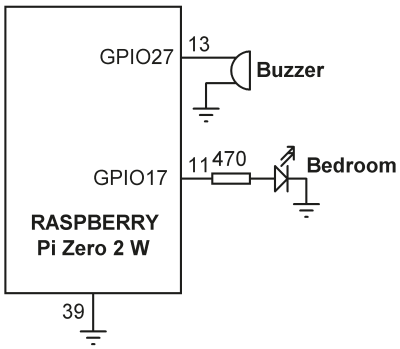


Figure 7.33: Circuit diagram of the project.

Node-RED flow program: Figure 7.34 shows the flow program. Notice that you have added another **amazon echo device** node to our program and named it **Buzzer**. Now, you have two devices with the names: **Bedroom** and **Buzzer**. This second node drives another **function** node which has exactly the same content as the previous one. The second **rpi gpio out** node is used to drive the buzzer where its **Pin** is set to GPIO27 and its **Type** is digital output as before. Because the new amazon echo device is not known to Alexa, you have to ask Alexa to discover nodes again: "**Alexa, discover devices**". After this, you can control the LED and the buzzer by asking the Alexa:

To turn the LED ON, ask Alexa: "**Alexa, turn on bedroom**"
To turn the LED OFF, ask Alexa: "**Alexa, turn off bedroom**"
To turn the Buzzer ON, ask Alexa: "**Alexa, turn on buzzer**"
To turn the Buzzer OFF, ask Alexa: "**Alexa, turn off buzzer**"

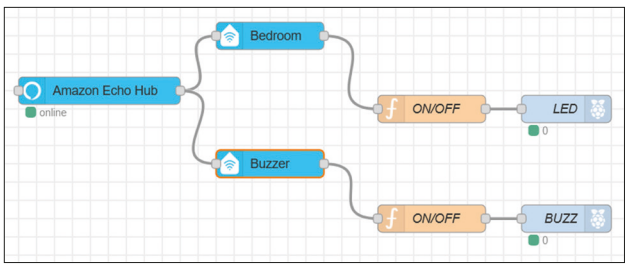


Figure 7.34: Flow program of the project.

Notice that if you change the names of the devices in nodes Amazon-echo-device, you will have to go to Alexa apps and remove the old device names from Alexa.

Also notice that you could have used another Alexa node called **node-red-contrib-alexa-home-skill** for more sophisticated Alexa based control tasks. This node, called the Alexa Home node, requires the user to be registered and have an account before using it. Alexa Home node supports the following commands:

- TurnOnRequest
- TurnOffRequest
- SetPercentageRequest
- IncrementPercentageRequest
- DecrementPercentageRequest
- SetTargetTemperatureRequest
- IncrementTargetTemperatureRequest
- DecrementTargetTemperatureRequest
- GetTemperatureReadingRequest
- GetTargetTemperatureRequest
- SetLockState
- GetLockState
- SetColorRequest
- SetColorTemperatureRequest

Interested readers can get much more information from the web link:

https://alexa-node-red.bm.hardill.me.uk/docs

7.11.3 Project 40: Controlling an LED and a buzzer using Alexa – using a trigger node

Description: In the previous Alexa based projects, the LED is turned ON and OFF by sending spoken commands to Alexa. In some applications, you may want to turn the LED ON by sensing a command, then you may want the LED to turn OFF automatically after some time. For example, before entering a room, you may ask Alexa to turn the light ON and after 15 seconds you may want the light to turn OFF automatically. This shows how it can be done in this project.

Node-RED flow program: This can be done easily by adding a **trigger** node to Figure 7.30. In the example flow program shown in Figure 7.35, the LED is turned OFF after 15 seconds. The trigger node is configured to send 1 for 15 seconds and then it sends 0. Figure 7.36 shows the **trigger** node contents. Notice that the output of the **trigger** node is reset if msg.payload is equal to 0 (i.e., if Alexa is asked to turn the Bedroom Light OFF).

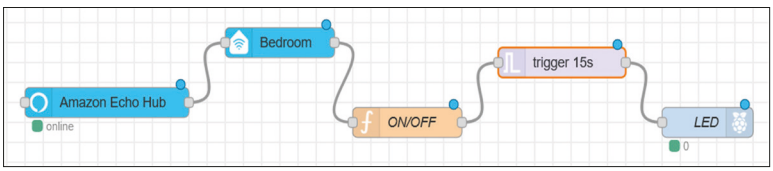


Figure 7.35: Modified flow program.

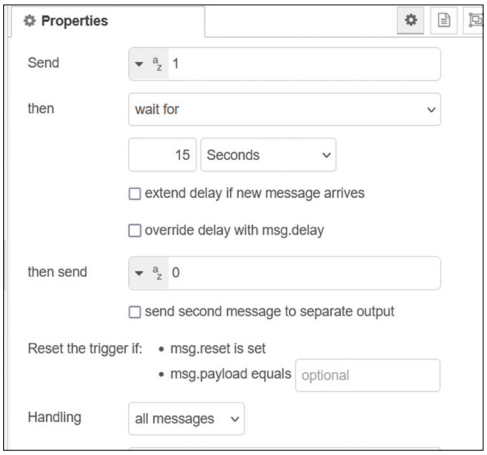


Figure 7.36: Configuring the trigger node.

Chapter 8 • Using MQTT with The Raspberry Pi Zero 2 W

8.1 Overview

MQTT stands for *Message Queuing Telemetry Transport*. It is a message protocol created for machine-to-machine (M2M) communication and based on publishing and subscribing. MQTT is especially useful when it comes to sending data and controlling information with low bandwidth and high response times. This protocol is especially worth considering when it is required to send data to actuators and retrieve data from sensors. MQTT was first developed by IBM in 1999 mainly for satellite communications, and since then has become the standard communications protocol for the Internet of Things (IoT) applications. MQTT uses your existing home network to send messages to your IoT devices and receive responses from them. The official website of MQTT is: http://mqtt.org.

MQTT works on top of the TCP/IP protocol and is faster than sending HTTP requests since a message can be as small as two bytes and there are no headers as with HTTP. Additionally, in MQTT, messages are distributed automatically to the interested clients.

In this chapter, you will be using MQTT with Node-RED on the Raspberry Pi Zero 2 W.

8.2 How MQTT works

It is worthwhile to learn how MQTT works before it is used in any of your projects. In the MQTT protocol, there is a sender called the **Publisher**, and a receiver called the **Subscriber**. Between these two, a server called a **Broker** is used to function as the intermediary. Figure 8.1 shows the basic structure of an MQTT based system.

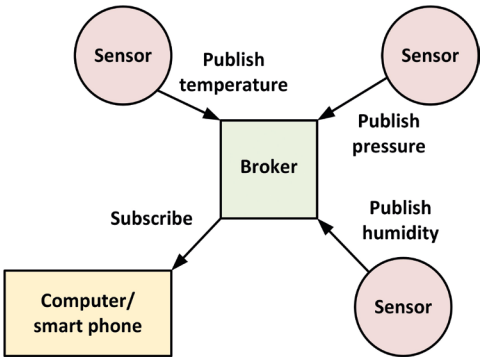


Figure 8.1: Basic MQTT structure.

In Figure 8.1, for example, the temperature sensor publishes (or sends) its measured value to the broker, who accepts and saves the data. The broker then sends this measured value to other devices which have subscribed to receive this data.

In MQTT there are five basic concepts that you should understand:

- Publisher
- Subscriber
- Messages
- Topics
- Broker

Publisher: The publisher is the node that sends data or the message on a topic. For example, a publisher can be a sensor node that sends the ambient temperature values.

Subscriber: The subscriber can be a computer, a smartphone, a microcontroller or any other processor that subscribes to receive the data or the message. For example, a device publishes on a topic, another device subscribes to the same topic and receives the published message.

Messages: Messages are the information exchanged within the MQTT network. A message can be data or a command. For example, a message can be the temperature value, or a command to turn ON a switch.

Topics: Topics are important concepts of MQTT. These are the ways you register interest for incoming messages or how you specify where you want to publish the message. i.e., the message exchange takes place via the topics. Topics in MQTT are represented with character strings, separated by a forward slash. Each forward slash indicates a topic level. An example is shown below; this one creates a topic for an LED in your kitchen:

```
home/kitchen/led
```

In the above example, home, kitchen, and led are the topic levels and they are separated by the topic level separators. The topics are case sensitive and for example, Home/kitchen/led is not the same as above.

Single Level Wildcards: You can use wildcards in topics enabling several sensors to be queried at the same time. A "**+**" sign is used to identify a single-level wildcard. An example is given below:

home/+/led	topic with singe-level wildcard
home/bedroom/led	wildcard replaced with bedroom
home/livingroom/led	wildcard replaced with livingroom
home/garden/led	wildcard replaced with garden

Multi-Level Wildcards: You can also use multi-level wildcard, represented by the sign "#". This must always be used at the end of a topic. An example is given below:

home/kitchen/#	topic with multi-level wildcard
home/bedroom/led	kitchen replaced with bedroom
home/livingroom/light	livingroom and light replaced

Broker: the broker receives all messages, filters them, decides who are interested in them, and then sends the message to all subscribed clients.

Figure 8.2 illustrates the operation of the MQTT, which can be summarized as follows:

- The processor (and sensor) publishes ON and OFF messages on topic **home/kitchen/led**.
- You have a Raspberry Pi Zero 2 W that controls an LED. The Zero 2 W is subscribed to topic: **/home/kitchen/led**.
- When a new message is published on topic **home/kitchen/led**, the Zero 2 W receives the ON or OFF messages and turns the LED ON or OFF.

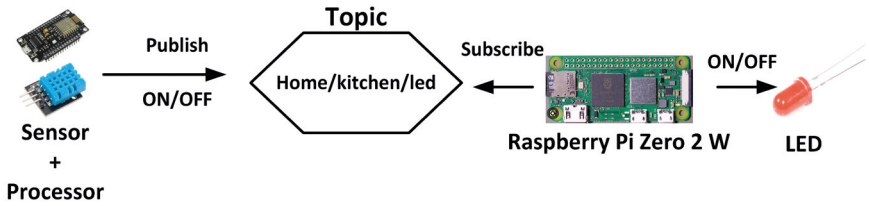


Figure 8.2: MQTT operation.

8.3 The Mosquitto Broker

The broker is one of the important parts of MQTT. There are several brokers that one can use. In most home automation projects, the **Mosquitto** broker is used. The steps to install this broker on the Raspberry Pi Zero 2 W is as follows:

```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get install mosquitto  mosquitto-clients
```

You can make Mosquitto to auto-start on boot by entering the following command:

```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

To evaluate the Mosquitto installation, enter the command **mosquitto –v** as shown in Figure 8.3. This command displays the version of the Mosquitto running on your system (don't worry about the *Error: Address already in use*).

```
pi@raspberrypi:~ $ mosquitto -v
1673614346: mosquitto version 2.0.11 starting
1673614346: Using default config.
1673614346: Starting in local only mode. Connections will only be possible
clients running on this machine.
1673614346: Create a configuration file which defines a listener to allow
 access.
1673614346: For more details see https://mosquitto.org/documentation/authe
ion-methods/
1673614346: Opening ipv4 listen socket on port 1883.
1673614346: Error: Address already in use
1673614346: Opening ipv6 listen socket on port 1883.
1673614346: Error: Address already in use
pi@raspberrypi:~ $
```

Figure 8.3: Testing the Mosquitto installation.

You can evaluate if your broker is working correctly, as follows:

Subscribe to the topic **"TestTopic"** by entering the following command. **Mosquitto_sub** tells that you want to subscribe to a topic, and the name following **–t** is the topic name. Now, every time you publish (i.e., send a message) to **TestTopic**, the message will appear in the window:

pi@raspberrypi:~ $ **mosquitto_sub –t «TestTopic»**

Now, because the terminal is listening for messages from your broker, you have to open another session with your Zero 2 W so that you can publish messages. After opening another window, enter the following command to publish to TestTopic:

pi@raspberrypi:~ $ **mosquitto_pub –t "TestTopic" –m "Hello There!"**

Here, the topic name is after **–t**, and the message is after **–m**. After hitting the Enter you will see the message appear on your subscriber terminal as shown in Figure 8.4.

```
pi@raspberrypi:~ $
pi@raspberrypi:~ $ mosquitto_sub -t "TestTopic"
Hello There!

pi@raspberrypi:~ $ mosquitto_pub -t "TestTopic" -m "Hello There!"
pi@raspberrypi:~ $
```

Figure 8.4: Broker example.

8.4 Using MQTT in home automation and IoT projects

To be able to use MQTT in home automation and in IoT projects, you need the following:

- a Raspberry Pi, e.g., a Raspberry Pi Zero 2 W.
- Node-RED and MQTT nodes.
- A Raspberry Pi, an Arduino, an ESP32, an ESP8266, or any other compatible microcontroller.

Figure 8.5 shows the basic MQTT system setup. The sensors, actuators, relays, switches, lamps, LEDs, and so on are connected to the system via an Arduino, ESP32, ESP8266, or any other compatible processor (system).

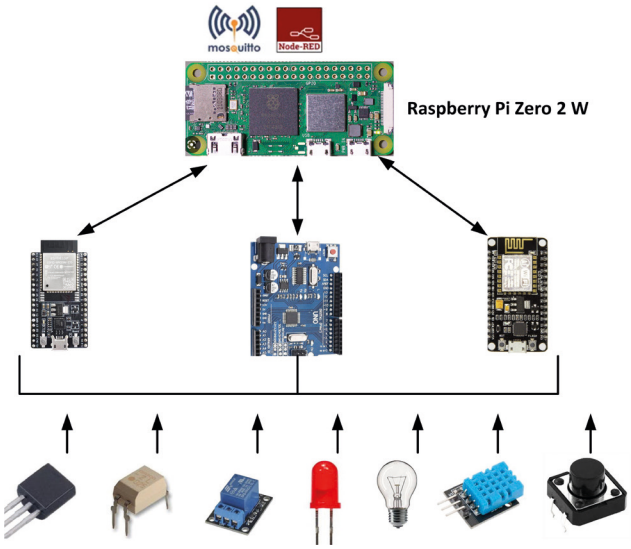


Figure 8.5: MQTT system setup.

Node-RED offers two MQTT nodes (**mqtt in** and **mqtt out**) which can be found in the network palette as shown in Figure 8.6.

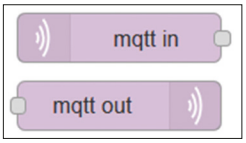


Figure 8.6: MQTT nodes.

You can send a message to the MQTT broker using an **inject** node of Node-RED as shown in Figure 8.7. Notice that the small green box under the **mqtt out** node indicates that the connection from the **mqtt out** node to the broker has been established. In Figure 8.7, the topic is set to **TestTopic**, and the **inject** node **Payload** is set to message **Hello From Me!!**. This message is displayed on the MQTT subscriber as shown in the image. In this flow program, the **mqtt out** node **Server** was set to **localhost** and the **Port** number was set to 1883 (network ports 1883 and 8883 are reserved by default):

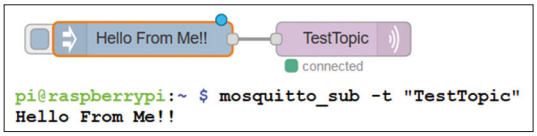


Figure 8.7: Sending a message to the MQTT broker.

Notice that, by using different topics you can send (publish) different messages, and these messages will be received by the subscribers having the same topics.

A quite simple example is given below which illustrates how an LED can be controlled using the MQTT.

8.5 Project 41: Controlling an LED using MQTT

Description: In this project, you will explore how an LED can be controlled using MQTT with Node-RED. The aim of this project is to show how the MQTT can be used in a remarkably simple project.

Circuit Diagram: In this project, an LED is connected to Raspberry Pi Zero 2 W port pin GPIO17 (pin 11) through a 470-ohm current-limiting resistor.

Node-RED flow program: Figure 8.8 shows the flow program of the project. Basically, two **inject** nodes are used with the Payloads set to 1 and 0 respectively (see Figure 8.9 for one of the nodes). Use an **mqtt out** node with the name **TestTopic, Server** set to **localhost**, and port set to **1883**. Connect the two **inject** nodes as shown in Figure 8.9. When, for example, the button of the upper **inject** node is clicked, then a 1 is sent to the broker via the **mqtt out** node. The **mqtt in** node in the lower flow is given the same topic name as the **mqtt out** node, and this node drives the **rpi gpio out** node which controls the LED accordingly. Name the **rpi gpio out** node as **LED** and configure it for GPIO17 with the initial pin state set to digital 0. Click **Deploy**. Click the left part of the upper **inject** node to turn ON the LED, Similarly, click the left part of the bottom **inject** node to turn OFF the LED. The Raspberry Pi Zero 2 W console displays the data received by the GPIO port since it is subscribed to the same topic, as shown in Figure 8.11.

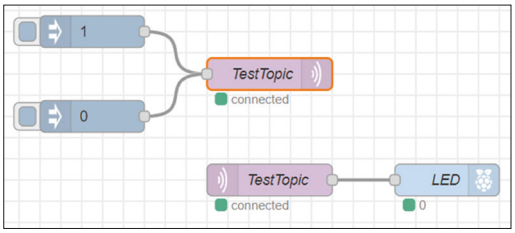


Figure 8.8: Flow program of the project.

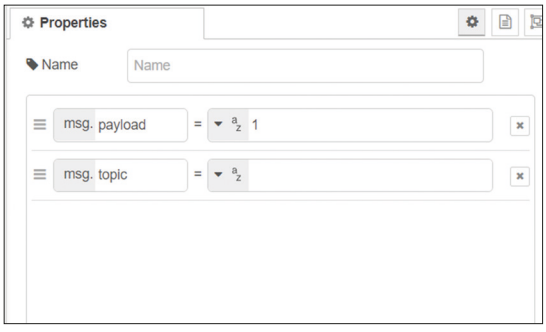


Figure 8.9: Setting the payload to 1.

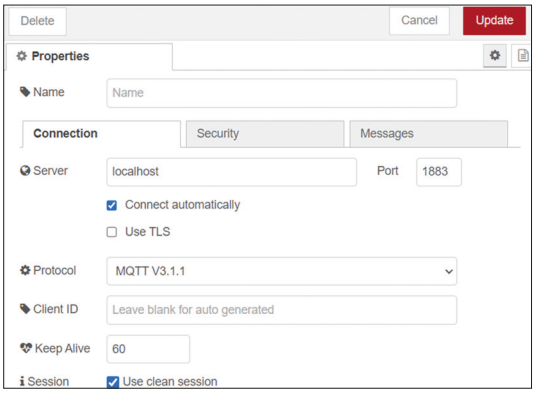


Figure 8.10: Configuring the mqtt nodes.

```
pi@raspberrypi:~ $ mosquitto_sub -t "TestTopic"
1
0
1
0
```

Figure 8.11: Raspberry Pi console subscribed to the same topic.

In more complex projects, instead of using inject nodes, you usually have processors with sensors connected to their inputs. The data from these sensors gets sent to the broker via the **mqtt out** node. The **mqtt in** node then receives and processes this data (e.g., display or activate an actuator).

In the next section, you will learn how to use the ESP8266 NodeMCU development board as the client in an MQTT application.

8.6 Project 42: Controlling an LED using ESP8266 NodeMCU with MQTT – LED connected to Raspberry Pi Zero 2 W

Description: This is quite a simple project where an LED is connected to one of the GPIO ports of the Raspberry Pi Zero 2 W. Additionally, a pushbutton switch is connected to the ESP8266 NodeMCU. The LED is toggled when the button is pressed. The aim of this project is to show how the MQTT can be used with the ESP8266 NodeMCU and the Zero 2 W. Here, the NodeMCU is the client (publisher) and the Zero 2 W is the subscriber which receives the messages.

Block Diagram: Figure 8.12 shows the block diagram of the project.

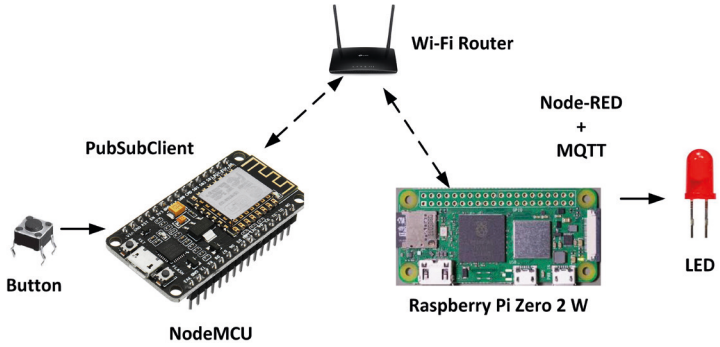


Figure 8.12: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 8.13. The button is connected to port pin GPIO2 of the ESP8266 NodeMCU. Similarly, the LED is connected to port pin GPIO2 of the Raspberry Pi Zero 2 W.

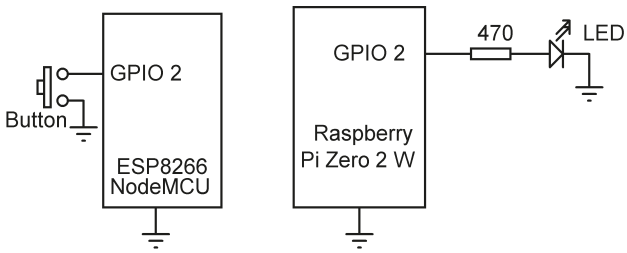


Figure 8.13: Circuit diagram of the project.

ESP8266 NodeMCU Program Listing: Figure 8.14 shows the program listing (program: **mqttesp8266**). NodeMCU employs the Wi-Fi link to communicate with the Raspberry Pi Zero 2 W in MQTT-based applications. Therefore, you need to know the SSID name and the password of your Wi-Fi router and the IP address of your Raspberry Pi Zero 2 W. In this example you have the following settings (you will have to enter your own settings):

SSID: BTHomeSpot-XNH
Password: 49350baeb
Raspberry Pi IP address: 192.168.1.238

You have to enter these values into your program. Notice that the Button is connected to port pin GPIO2 of the ESP8266:

```
#include <ESP8266WiFi.h>
       #include <PubSubClient.h>

const char* ssid = "BTHomeSpot-XNH";
const char* password = "49350bbbb";
const char* mqtt_server = "192.168.1.202";
```

```
const byte Button = 2;
const char *ID = "Example_Button";
bool SwitchState = 0;
```

In this example, you have chosen the MQTT topic as: **kitchen/light**

```
const char *Topic = "kitchen/light/"
```

you have to setup a Wi-Fi client, and a PubSubClient:

```
WiFiClient wclient;
PubSubClient client(wclient);
```

Inside the **setup()** function you will initialize the Arduino IDE Serial Monitor at 115200 Baud so that you can trace the execution of the code easily. You will also have to configure port pin GPIO2 where the button is connected to as input and enable the pull-up resistor, and then enable the Wi-Fi on your ESP8266 NodeMCU:

```
void setup()
{
                Serial.begin(115200);
        pinMode(Button, INPUT);
        digitalWrite(Button, HIGH);
        delay(100);
        setup_wifi();
        client.setServer(mqtt_server, 1883);
}
```

The Wi-Fi is set up inside the setup routine as follows (readers who used the ESP8266 before should be familiar with the following code):

```
void setup_wifi()
{
          Serial.print("\nESP32 NodeMCU connecting to: ");
        Serial.println(ssid);
        WiFi.begin(ssid, password);

        while (WiFi.status() != WL_CONNECTED)
        {
                delay(500);
                Serial.print(".");
        }
        Serial.println("\nConnected to Wi-Fi");
}
```

The main program loop attempts to reconnect to the client by calling function **reconnect** if the connection is lost, and reads the status of the button to check if it is pressed.

The main loop code is:

```
       void loop()
       {
               if (!client.connected()) reconnect();
               client.loop();

               if(digitalRead(Button) == 0)            // If button is pressed
               {
                       ButtonState = !ButtonState;     // Toggle ButtonState
                       if(ButtonState == 1)            // ON
                       {
                               client.publish(Topic, "on");
                               Serial.println((String)Topic + " is ON");
                       }
               else
               {
                       client.publish(Topic, "off");
                       Serial.println((String)Topic + " is OFF");
               }

               while(digitalRead(Button) == 0)         // Await switch release
               {
                       yield();
                       delay(20);
               }
       }
```

```
/*********************************************************************
 *           ESP8266 NodeMCU MQTT PROGRAM
 *           ============================
 * In this program a push-button switch is conencted to port
 * GPIO2 of the EXP8266 NodeMCU. Similarly, an LED is connected
 * to port GPIO2 of the Zero 2 W. The NodeMCU is the client.
 * Pressing the button toggles the LED.
 *
 * Author: Dogan Ibrahim
 * File  : mqttesp8266
 * Date  : Janury, 2022
 *
 *********************************************************************/
#include <ESP8266WiFi.h>
```

```
#include <PubSubClient.h>

const char* ssid = "BTHomeSpot-XNH";                // WiFi SSID
const char* password = "49350bbbb";                 // WiFi password
const char* mqtt_server = "192.168.1.238";          // Zero 2 W IP
const byte Button = 2;                              // Button pin
const char *ID = "Example_Button";
bool ButtonState = 0;
const char *Topic = "kitchen/light/";               // MQTT TOPIC

WiFiClient wclient;
PubSubClient client(wclient);

//
// Connect to local Wi-Fi router. This function connects the
// NodeMCU to the local WiFi router. Because the Serial Monitor
// is enabled, we can see the progress in the monitor
//
void setup_wifi()
{
    Serial.print("\nESP32 NodeMCU connecting to: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED)
    {
          delay(500);
          Serial.print(".");
    }
    Serial.println("\nConnected to Wi-Fi");
}


//
// This is the setup routine. Here the Serial Monitor is enabled,
// Button is configured as digital input, and the WiFi is setup.
// Notice that the MQTT server (Zero 2 W) operate on port 1883
//
void setup()
{
      Serial.begin(115200);                         // Enable Serial Monitor
      pinMode(Button, INPUT);                       // Button is input
      digitalWrite(Button, HIGH);                   // Pull Button HIGH
      delay(100);                                   // Small delay
      setup_wifi();                                 // Setup WiFi
      client.setServer(mqtt_server, 1883);          // Define client
```

```
}


//
// Reconnect to client if the connection is lost
//
void reconnect()
{
    while (!client.connected())
    {
          Serial.print("Attempting MQTT connection...");
          if (client.connect(ID))
          {
              Serial.println("connected");
              Serial.print("Publishing to: ");
              Serial.println(Topic);
              Serial.println('\n');
          }
          else
          {
              Serial.println(" Trying to connect again in 5 seconds");
              delay(5000);
          }
    }
}

//
// Main program loop. Check the Button status and publish on or off
// depending on this status
//
void loop()
{
      if (!client.connected()) reconnect();
      client.loop();

      if(digitalRead(Button) == 0)                  // If button pressed
      {
          ButtonState = !ButtonState;               // Toggle ButtonState
          if(ButtonState == 1)                      // If ON
          {
              client.publish(Topic, "on");          // Publish "on"
              Serial.println((String)Topic + " is ON");
          }
       else
       {
            client.publish(Topic, "off");           // Publish "off"
```

```
            Serial.println((String)Topic + " is OFF");
        }

         while(digitalRead(Button) == 0)           // Wait switch to release
         {
            delay(50);
         }
}
}
```

Figure 8.14: **mqttesp8266** *program listing.*

Node-RED flow program: Figure 8.15 shows the flow program for the Raspberry Pi Zero 2 W which consists of just three nodes: an **mqtt in** node, a **function** node, and an **rpi gpio out** node.

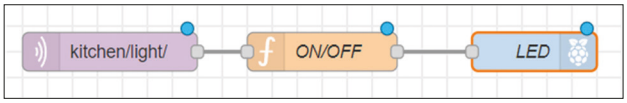


Figure 8.15: Flow program of the project.

The steps are as follows:

- Create an **mqtt in** node and set its **Server** to localhost, **Port** to 1883, and the **Topic** to kitchen/light/.
- Create a **function** node named ON/OFF and enter the following statements inside this function:

```
if(msg.payload == "on")
  msg.payload = 1;
else if(msg.payload == "off")
  msg.payload = 0;
return msg;
```

- Create an **rpi gpio out** node and set the **Pin** to GPIO2, **Type** to digital output, and **Initialize** to 0.
- Connect all the nodes as in Figure 8.15 and click **Deploy**.

Testing
The project can be evaluated as follows:

- Power on the ESP8266 NodeMCU and open the Serial Monitor and set its Baud rate to 115200.

- You should see messages on the Serial Monitor saying that the NodeMCU is connected to your Wi-Fi (see Figure 8.16) and that it can publish to the MQTT with the topic kitchen/light/

```
ESP32 NodeMCU connecting to: BTHomeSpot-XNH
.................
Connected to Wi-Fi
Attempting MQTT connection...connected
Publishing to: kitchen/light/
```

Figure 8.16: Connection messages on the Serial Monitor.

- Press the button. You should see the following message displayed on the Serial Monitor (see Figure 8.17). At the same time, the LED will turn ON:

  kitchen/light/ is ON

- Release the button and press again. You should see the following message displayed on the Serial Monitor. At the same time, the LED will turn OFF:

  kitchen/light/ is OFF

```
ESP32 NodeMCU connecting to: BTHomeSpot-XNH
.................
Connected to Wi-Fi
Attempting MQTT connection...connected
Publishing to: kitchen/light/


kitchen/light/ is ON
kitchen/light/ is OFF
kitchen/light/ is ON
kitchen/light/ is OFF
kitchen/light/ is ON
kitchen/light/ is OFF
```

Figure 8.17: Messages on the Serial Monitor.

- If you subscribe your Raspberry Pi Zero 2 W console to topic kitchen/light/, then you should see the messages displayed there as well (see Figure 8.18).

```
pi@raspberrypi:~ $ mosquitto_sub -t "kitchen/light/"
off
on
off
on
off
```

Figure 8.18: Messages on the Raspberry Pi console.

Chapter 9 • Communication over Bluetooth

9.1 Overview

In the previous chapter, you explored the writing of programs aiming to use Wi-Fi and then communicate with other devices over LAN using the UDP and TCP protocols. In this chapter, you will develop programs for using Bluetooth communication.

Bluetooth is a short-range communication technology commonly used to exchange data with other devices, mainly developed with IoT applications in mind. All smartphones nowadays support communication through Bluetooth. Bluetooth operates at 2.4 GHz with data rates lower than those of Wi-Fi. Bluetooth is not as secure as Wi-Fi but it is easier to use. The power consumption of Bluetooth is low compared to Wi-Fi and it offers shorter ranges overall than Wi-Fi. Bluetooth is a packet-based protocol with a master-slave architecture where one master may communicate with up to seven slaves. The effective range of Bluetooth depends on propagation conditions, antenna configurations, power supply conditions, material coverage and so on. Most Bluetooth applications are for indoor use where the signals are attenuated owing to walls, floors, and ceilings, usually resulting in shorter ranges than hoped for.

The Raspberry Pi Zero 2 W supports both classic Bluetooth and Bluetooth Low Energy (BLE). BLE is intended for reduced-power applications while providing reasonable range. Most smartphones including Android, iOS, Windows Phone, Blackberry, macOS and many others, support BLE. BLE uses the same 2.4-GHz radio frequency as the classic Bluetooth and dual-mode devices, therefore can share the same, single antenna. Classic Bluetooth can handle large amounts of data quickly, whereas BLE has been developed to handle smaller amounts of data.

9.2 Project 43: Exchanging text with a smartphone

Description: In this project, Bluetooth communication is established between the Zero 2 W and an Android smartphone. Text messages are exchanged between the two devices.

Block diagram: Figure 9.1 shows the block diagram of the project.

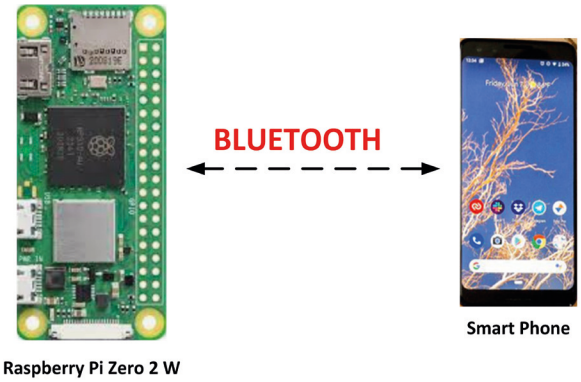


Figure 9.1: Block diagram of the project.

Enabling Bluetooth
Before using your smartphone for Bluetooth applications, you have to enable it. Depending on the model of the smartphone you have, this is usually done from the **Settings** menu.

Similarly, before using Bluetooth on your Zero 2 W, you have to enable it. There are two ways you can enable Bluetooth on the Zero 2 W: using the graphical desktop (GUI mode) or using the command mode.

Using the Graphical Desktop
The steps for enabling Bluetooth on the Zero 2 W by way of the graphical desktop are given below.

- Enable Bluetooth on your smartphone.
- If you have a monitor connected directly to the Zero 2 W, skip the next item.
- Start the VNC server on your Zero 2 W and log in using the VNC Viewer.
- Click on the blue Bluetooth icon on your Zero 2 W screen at the top right hand side, and turn Bluetooth ON if it is not already ON. Then, select **Make Discoverable.** You should see the Bluetooth icon flashing. Click **Add Device**.
- Select **raspberrypi** in the Bluetooth menu (**raspberrypi** is the default Bluetooth name of your Zero 2 W) on your mobile device (you may have to scan on your mobile device). You should see the **Connecting** message on your smart device.
- Click **Pair** to accept the pairing request on your Zero 2 W as shown in Figure 9.2.

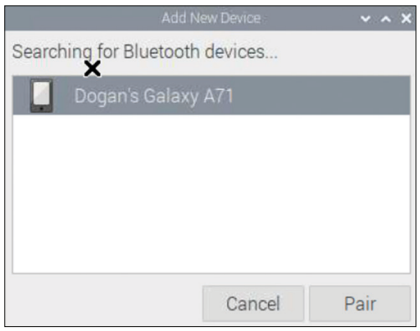


Figure 9.2: Bluetooth pairing request on Zero 2 W.

- You should now see the message **Pairing Successfully** on your Zero 2 W.

Using Command Mode

You can enable Bluetooth on your Zero 2 W using the command mode. Additionally, you can make Bluetooth discoverable, scan for nearby Bluetooth devices and then connect to a Bluetooth device. The steps are given below (characters typed by the user are in bold for clarity):

- Find the Bluetooth MAC address of your smartphone. For Android phones, the steps are usually:

  - Go to the **Settings** menu
  - Tap **About Phone**
  - Tap **Status information**
  - Scroll down to see your **Bluetooth address** (e.g., Figure 9.3). In this example, the MAC address was **50:50:A4:0F:62:3F**

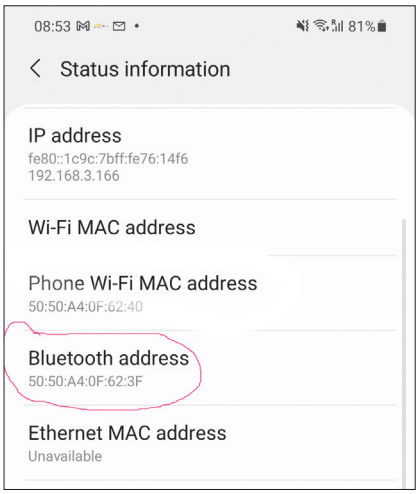


Figure 9.3: Bluetooth MAC address.

- Make your Bluetooth discoverable with the following command:

  pi@raspberrypi: ~ $ **sudo hciconfig hci0 piscan**

- Start the Bluetooth tool on your Zero 2 W from the command mode:

  pi@raspberrypi:~ $ **bluetoothctl**

- Turn Bluetooth ON:

  bluetooth]# **power on**

- Configure Bluetooth to run:

```
bluetooth]# agent on
[bluetooth]# default-agent
```

- Make the device discoverable:

```
[bluetooth]# discoverable on
```

- Scan for nearby Bluetooth devices; this may take several minutes:

```
[bluetooth]# scan on
```

- Enter the command **devices** to detect any nearby Bluetooth devices (see Figure 9.4). You may have to wait couple of minutes for the display to update. Make a note of the MAC address of the device you wish to connect to (Android mobile phone in this project) as you will be using this address to connect to the device.

```
[Bluetooth]# devices
```

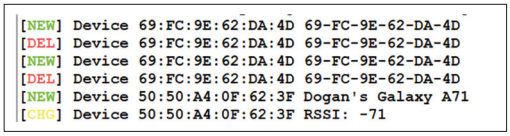


Figure 9.4: Nearby Bluetooth devices.

In this example, the author's smartphone is **Galaxy A71** and the Bluetooth MAC address is: **50:50:A4:0F:62:3F**

- Pair the device:

```
[bluetooth]# pair 50:50:A4:0F:62:3F
```

- Connect to your smartphone:

```
[bluetooth]# connect 50:50:A4:0F:62:3F
```

- Enter **yes** to confirm **passkey**.
- Accept pairing on your smartphone.
- You should see message **device 50:50:A4:0F:62:3F Connected : yes** displayed.
- Exit from the Bluetooth tool by entering **Cntrl+Z**.

You can find the Bluetooth MAC address of your Zero 2 W by entering the following command:

```
pi@raspberrypi:~ $ hciconfig | grep "BD Address"
```

You can change the Bluetooth broadcast name by the following command:

```
pi@raspberrypi:~ $ sudo hciconfig hci0 name "new name"
```

To see your Bluetooth broadcast name, enter:

```
pi@raspberrypi:~ $ sudo hciconfig hci0 name
```

Some other useful Zero 2 W Bluetooth commands are:

- To reset Bluetooth adapter: **sudo hciconfig hci0 reset**
- To restart Bluetooth: **sudo invoke-rc.d bluetooth restart**
- To list Bluetooth adapters: **hciconfig**

Python Bluetooth Library
You will need to install the Python Bluetooth library before developing your program. This is done by entering the following command in the command mode:

```
pi@raspberrypi:~ $ sudo apt-get install bluez python3-bluez
```

Accessing from the Mobile Phone
To be able to access the Zero 2 W from a smartphone app, make the following changes to your Zero 2 W from the command line:

- Start **nano** to edit the following file:

  ```
  pi@raspberrypi:~ $ sudo nano /etc/systemd/system/
  dbus-org.bluez.service
  ```

- Add **–C** at the end of the **ExecStart=** line. Also add another line after the ExecStart line. The final two lines should look like:

  ```
  ExecStart=/usr/lib/bluetooth/bluetoothd -C
  ExecStartPost=/usr/bin/sdptool add SP
  ```

- Exit and save the file by entering **Ctrl+X** followed by **Y**
- Reboot the Zero 2 W:

  ```
  pi@raspberrypi:~ $ sudo reboot
  ```

Program listing: Figure 9.5 shows the program listing (Program: **bluetxt.py**). Do not call your program: **Bluetooth.py**! The Bluetooth code is similar to TCP/IP code. At the beginning of the program, the modules socket and Bluetooth are imported to the program. The program then creates a Bluetooth socket, binds, and listens on this socket, and then waits to accept a connection. The remainder of the program is executed in a loop where the program issues the statement **ClientSock.recv** and waits to read data from the smartphone. The received data is decoded and displayed on the screen. The user is then expected to send text message to the smartphone. This message is displayed on the smartphone screen. This process is repeated until halted by the user.

```
#=======================================================
#              BLUETOOTH COMMUNICATION
#              =======================
#
# In this project text messages are exchanged with a smart
# phone using the Bluetooth protocol
#
# Author: Dogan Ibrahim
# File  : bluetxt.py
# Date  : December, 2022
#=======================================================
import socket
import bluetooth

#
# Start of main program loop.Configure Bluetooth, create a
# port, listen for client connections, and accept connection
#
port = 1
ServerSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
ServerSock.bind(("", port))
ServerSock.listen(1)
ClientSock, addr = ServerSock.accept()

#
# Now receive text from smart phone and display
#
try:

   while True:
      data = ClientSock.recv(1024)                    # receive text
      print("Received data: ", data.decode('utf-8'))
      msg = input("Enter data to send: ")             # TExt to send
      ClientSock.send(msg.encode('utf-8'))            # Send text
```

```
except KeyboardInterrupt:                          # Keyboard int
      ServerSock.close()                           # Close socket
```

Figure 9.5: Program: **bluetxt.py**.

Testing
The project can be evaluated by the following steps:

- Make sure that Bluetooth is enabled on both the smartphone and the Zero 2 W and the devices are paired.

- You will be using the freely available Bluetooth app on your smartphone to communicate with your Zero 2 W. In this project, the app called **Bluetooth Terminal HC05 by mightyIT** is used (Figure 9.6).

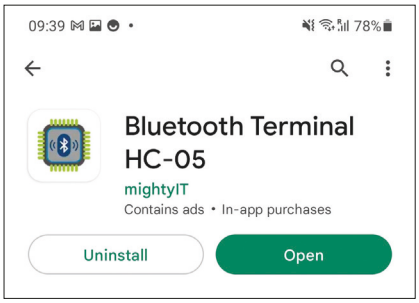


Figure 9.6: Android Bluetooth app used in the project.

- Start the Zero 2 W program:

  pi@raspberrypi:~ $ **python3 bluetxt.py**

- Start the smartphone app and select the paired Zero 2 W device (e.g., **raspberrypi**).

- Enter a message in column **Enter ASCII Command** and press **Send ASCII**. The message will be sent to the Zero 2 W.

- Enter a message on the Zero 2 W. This message will be sent and displayed on the smartphone top part of the screen.

- Figure 9.7 shows a sample message exchange between the Zero 2 W and the smartphone. In this example, the smartphone sends the message: **message from the smartphone** and the Zero 2 W sends message: **message from Zero 2 W**.

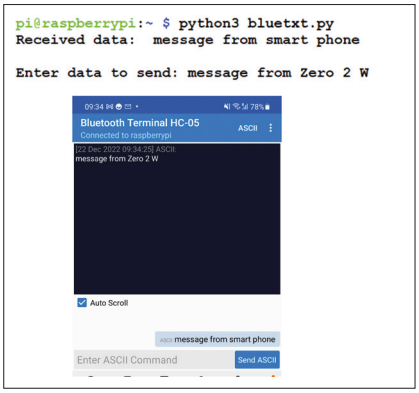


Figure 9.7: Message exchange between the Zero 2 W and smartphone.

- Enter **Cntrl+C** to close the socket and terminate the program.

9.3 Project 44: Bluetooth control of LED from a smartphone

Description: In this project, an LED is connected to port GPIO 17 (pin 11) of Zero 2 W through a 470-ohm current-limiting resistor. The LED is controlled by sending commands from an Android smartphone using Bluetooth communication.

The following commands can be sent from the Android smartphone to control the LED:

L1 Turn the LED ON
L0 Turn the LED OFF

Block diagram: Figure 9.8 shows the block diagram of the project.

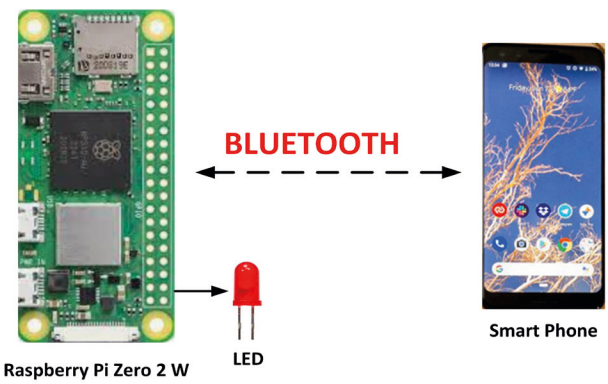


Figure 9.8: Block diagram of the project.

Program Listing: Figure 9.9 shows the program listing of the project (program: **blueled.py**; do not call your program **Bluetooth.py**!). The Bluetooth code is similar to TCP/IP code. At the beginning of the program, the modules socket, RPi.GPIO, and Bluetooth are imported to the program. The LED port is defined and configured as output. The program

then creates a Bluetooth socket, binds, and listens on this socket, and then waits to accept a connection. The remainder of the program is executed in a loop where the program issues the statement **ClientSock.recv** and waits to read data from the smartphone. Note that the smartphone app automatically appends carriage-return (CR) and line-feed (LF) characters to the end of the data (i.e. **\r\n**).

```
#=======================================================
#              LED CONTROL BY BLUETOOTH
#              ========================
#
# In this project an LED is connected to GPIO 17.The LED
# LED is controlled by sending commands from an Android
# smart phone using a Bluetooth apps.
#
# Valid comamdns are:
# L1 Turn ON the LED
# L0 Turn OFF the LED
#
# Author: Dogan Ibrahim
# File  : blueled.py
# Date  : December, 2022
#=======================================================
import socket
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
import bluetooth

#
# LED is at GPIO 17, configure as output and turn OFF
#
LED = 17                                # LED at port 17
GPIO.setup(LED, GPIO.OUT)               # LED as output
GPIO.output(LED, 0)                     # LED OFF

#
# Start of main program loop.Configure Bluetooth, create a
# port, listen for client connections, and accept connection
#
port = 1
ServerSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
ServerSock.bind(("", port))
ServerSock.listen(1)
ClientSock, addr = ServerSock.accept()

#
```

```
# Now receive comamnds and decode
#
try:

   while True:
     data = ClientSock.recv(1024)              # receive command
     if data == b'L1\r\n':                     # L1?
         GPIO.output(LED, 1)                   # turn ON LED
     elif data == b'L0\r\n':                   # L0?
          GPIO.output(LED, 0)                  # turn OFF LED

except KeyboardInterrupt:                      # Interrupt
     ServerSock.close()
```

Figure 9.9: Program: **blueled.py***.*

Testing
The same Android app as in the previous project is used. The steps are:

- Make sure that the Bluetooth is enabled on both the smartphone and Zero 2 W and that they are already paired.
- Start the Zero 2 W program:

  pi@raspberrypi:~ $ **python3 blueled.py**

- Start the app as before. To send command **L1**, enter L1 and click **Send ASCII.** The LED should turn ON.

Suggestion for additional work

You could enter the program name in the following format inside file **/etc/rc.local** so that the program automatically starts every time the Zero 2 W re-starts:

python /home/pi/blueled.py &

When you finish your project, don"t forget to remove the above line from file **/etc/rc.local**, otherwise the program will run every time your Zero 2 W is restarted. You should also shut down your Zero 2 W orderly instead of just unplugging the power cable. The command to shut down properly is:

pi@raspberrypi:~ $ **sudo shutdown now**

9.4 Arduino Uno: Raspberry Pi Zero 2 W Bluetooth communication

The Arduino Uno has no built-in Bluetooth module so you have to use an external Bluetooth module to enable the Uno to communicate with other devices via Bluetooth. You can, however, use a serial Bluetooth module such as the HC-06. In the next section, you will embark on developing a project and learn how to connect an HC-06 type low-cost Bluetooth module to your Arduino Uno and then communicate with the Zero 2 W.

9.4.1 Project 45: Communicating with an Arduino Uno over Bluetooth

Description: In this project, a button is connected to the Arduino Uno. Also, a +3.3 V relay is connected to the Zero 2 W. Pressing the button on the Arduino sends command "**1**" to the Zero 2 W which then activates the relay for 5 seconds. The aim of this project is to show how your Arduino Uno and Zero 2 W can communicate by using an external Bluetooth module on the Arduino Uno.

The HC-06 Bluetooth module

The HC-06 is a low-cost, popular, 4-pin, serially controlled module with the following pins (see Figure 9.10):

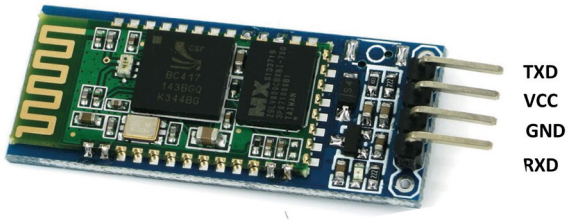


Figure 9.10: The HC-06 Bluetooth module.

The HC-06 is a serial-controlled module with the following basic specifications:

- +3.3 V to +6 V operation
- 30-mA unpaired current (10-mA matched current)
- Built-in antenna
- Band: 2.40 GHz – 2.48 GHz
- Power level: +6 dBm
- Default communication: 9600 baud, 8 data bits, no parity, 1 stop bit
- Signal coverage 30 feet (approx. 10 m)
- Safety feature: Authentication and encryption
- Modulation mode: Gaussian frequency-shift keying (FSK)

Block diagram: Figure 9.11 shows the block diagram of the project.

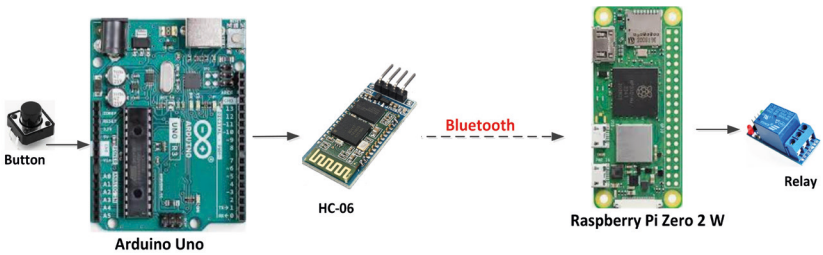


Figure 9.11: Block diagram of the project.

Circuit diagram: The circuit diagram is shown in Figure 9.12. The button is connected to pin 2 of the Arduino Uno through a pull-up resistor. The relay is connected to port GPIO 4 of the Zero 2 W. The HC-06 is a serial device with TX and RX pins. Only the RX input of the HC-06 are used since in this project you only send data to the Bluetooth module. The output pin voltage of Arduino Uno is +5 V but the HC-06 is not +5 V compatible. Therefore, a resistive voltage divider circuit is used to lower the Arduino voltage to +3.3 V. The TX pin of HC-06 is connected to pin 3 of Arduino Uno (pin 3 is software-configured as a "software serial port").

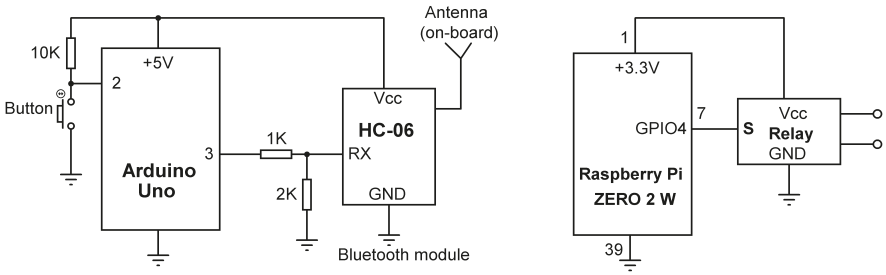


Figure 9.12: Circuit diagram of the project.

Connecting to Arduino over Bluetooth

You should find the MAC address of your HC-06 and then use this address to connect to it. The default HC-06 passcode is **1234**. The steps to find the MAC address are:

- Construct the Arduino circuit (Figure 9.12) and apply power so that HC-06 can be accessed. The red LED on HC-06 will flash to indicate that it is not currently connected to any device.

- Make your Zero 2 W Bluetooth discoverable:

  pi@raspberrypi: ~ $ **sudo hciconfig hci0 piscan**

- Start the Bluetooth tool:

  pi@raspberrypi:~ $ **bluetoothctl**

- Turn Bluetooth ON:

  [bluetooth]# **power on**

- Configure Bluetooth to run:

  [bluetooth]# **agent on**
  [bluetooth]# **default-agent**

- Make device discoverable:

  [bluetooth]# **discoverable on**

- Scan for nearby Bluetooth devices, you may have to wait several minutes:

  [bluetooth]# **scan on**

- Enter command **devices** to see the nearby Bluetooth devices. You may have to wait several minutes for the display to update. You should see the HC-06 listed with its MAC address.

  [Bluetooth]# **devices**

  In this example, the author"s HC-06 was identified with the MAC address: **98:D3:91:F9:6C:19**

- After finding out the MAC address, you may get information about the HC-06 by entering the following command:

  [Bluetooth]# **info 98:D3:91:F9:6C:19**

  Which may display as in Figure 9.13.

```
[bluetooth]# info 98:D3:91:F9:6C:19
Device 98:D3:91:F9:6C:19 (public)
        Name: HC-06
        Alias: HC-06
        Class: 0x00001f00
        Paired: yes
        Trusted: yes
        Blocked: no
        Connected: no
        LegacyPairing: yes
        UUID: Serial Port
        RSSI: -29
```

Figure 9.13: Getting information on HC-06.

If **Trusted: no** is displayed, enter command: **trust 98:D3:91:F9:6C:19**.

Exit the Bluetooth tool by entering **Cntrl+Z** and then enter the following statement to make a connection in command mode (enter your own HC-06 MAC address):

- pi@raspberrypi:~ $ **sudo rfcomm connect hcio 98:D3:91:F9:6C:19 &**
- Enter **Cntrl+C** to exit

You should be connected now and the red LED on HC-06 should stop flashing. You are now ready to develop your programs for the Arduino Uno and Zero 2 W concurrently or separately.

RPi Zero 2 W program: Figure 9.14 shows the Zero 2 W program (Program: **zeroprog.py**). In this program, you will not be using the Bluetooth library. When you connect to the HC-06, a virtual serial terminal named **/dev/rfcomm0** is created on the Zero 2 W. You can read the commands sent by the HC-06 by opening this serial port and then viewing the traffic.

Serial port **/dev/rfcomm0** is opened with the baud rate set to 9600 (the default baud rate of the HC-06 is 9600) using the following statement:

```
ser = serial.Serial(port="/dev/rfcomm0", baudrate=9600)
```

Data sent by HC-06 is then read using the statement:

```
data = ser.read()
```

If the received data is b"1" then the Relay can be activated for 5 seconds by the following statements:

```
GPIO.output(RELAY, 1)
Time.sleep(5)
GPIO.output(RELAY, 0)
```

```
#=======================================================
#              RELAY CONTROL BY BLUETOOTH
#              ==========================
#
# In this Case Study a Relay is connected to GPIO 4.The
# Relay is controlled by sending command from an Arduino
# Uno using a Bluetooth apps.
#
# Valid command is: "1"
#
# Author: Dogan Ibrahim
# File  : zeroprog.py
# Date  : December, 2022
#=======================================================
```

```
import RPi.GPIO as GPIO
import serial
import time
GPIO.setmode(GPIO.BCM)

#
# Relay is on GPIO 2, configure as output and turn OFF
#
RELAY = 4                               # Relay at port 4
GPIO.setup(RELAY, GPIO.OUT)             # Relay as output
GPIO.output(RELAY, 0)                   # Relay OFF

#
# Attach to virtual serial port /dev/rfcomm0
#
ser = serial.Serial(port='/dev/rfcomm0', baudrate=9600)

#
# Now receive commands and decode
#
try:

   while True:
     data = ser.read()                  # receive comman
     if data == b'1':                   # 1?
         GPIO.output(RELAY, 1)          # activate Relay
         time.sleep(5)                  # 5 seconds
         GPIO.output(RELAY, 0)          # Relay OFF

except KeyboardInterrupt:               # Interrupt
     GPIO.output(RELAY, 0)              # deactivate Relay
     GPIO.cleanup()
```

Figure 9.14: Program: **zeroprog.py**.

Arduino Uno program: Figure 9.15 shows the Arduino Uno program (Program: **ardprog**). Software serial port is used in this program where pin 3 is configured as the TX pin and pin 4 as the RX pin (RX is not used in this project). **Button** is then assigned to port 2. Inside the **setup()** function, the serial port baud rate is set to 9600 and **Button** is configured as an input pin. Inside the main program loop, the program waits until the button is pressed and then released. A "**1**" is then sent to the HC-06, with the Zero 2 W responding by turning ON the LED upon reception of this command.

```
/*================================================
      ARDUINO UNO - RASPBERRY PI ZERO 2 W COMMS
      =========================================
In this project a button is connectd to Arduino Uno
pin 2. HC-06 Bluetooth module is connected to pin 3
The program sends command "1" over the Bluetooth when
the button is pressed

Autjor: Dogan Ibrahim
File  : ardprog
Date  : December, 2022
================================================*/
#include <SoftwareSerial.h>
SoftwareSerial MySerial(4, 3);        // rx, tx

int Button = 2;                       // Button at pin 2

void setup()
{
  MySerial.begin(9600);               // Baud rate
  pinMode(Button, INPUT);             // Button is input
}

//
// MAin program looop
//
void loop()
{
  while (digitalRead(Button) == 1);   // Button not pressed
  while (digitalRead(Button) == 0);   // Button not released
  MySerial.print("1");                // Send "1"
  delay(1000);
}
```

Figure 9.15: Program: **ardprog**.

Testing

- Run the Zero 2 W program:

  pi@raspberrypi:~ $ **python3 zeroprog,py**

- Compile and upload the Arduino Uno program.
- Press and release the button. The LED should turn ON for 5 seconds.

9.5 Project 46: Play audio (like music) on a Bluetooth speaker via Zero 2 W

Description: In this project you will play music on an external Bluetooth speaker via your Raspberry Pi Zero 2 W. You will discover how to store your MP3 music files on the Zero 2 W and then play them on the Bluetooth speaker.

Before you can send audio to a Bluetooth speaker, you have to have a program on your Zero 2 W that can play audio files like MP3's. In this project, you will be using the popular **VLC Media Player** program on your Zero 2 W. The steps to install VLC are:

- pi@reaspberrypi:~ $ **sudo apt-get update**
- pi@reaspberrypi:~ $ **sudo apt-get upgrade**
- pi@raspberrypi:~ $ **sudo apt-get install vlc**
- Wait until the VLC program is installed. You now need to have MP3 files so that you can evaluate your project. Download or copy some of your favorite MP3 music files to your Raspberry Pi Zero 2 W (say, to directory /home/pi or a newly created directory).

You now have to pair with your Bluetooth speaker and connect to it. You will do this from the Desktop. The steps are:

- Start **Desktop** on your Zero 2 W
- Click the **Bluetooth icon** at the top right-hand corner of Desktop and select to turn ON Bluetooth.
- Click on **Bluetooth icon** and set to make it Discoverable.
- Click on **Bluetooth icon** and click **Add Device** to pair and add your speaker, or if your speaker is already listed., click on it and click to **Connect** (in the author"s case, the Bluetooth speaker had the name **BT-888**). See Figure 9.16.

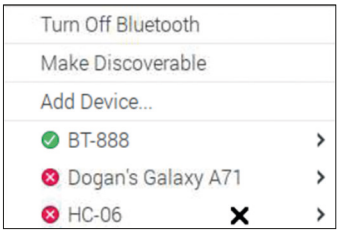


Figure 9.16: Click on Bluetooth speaker device to connect to it.

- You are now connected to your speaker. Next thing to do is to direct your audio output to the speaker. Right-click on the **Volume icon** at the top right-hand corner of **Desktop** and select your Bluetooth speaker"s name (like **BT-888** in author"s case).

- Open **File Manager** in Desktop (**Accessories** → **File Manager**) and double-click on your MP3 file. The Bluetooth speaker should start playing your chosen music.

- Click **Media** → **Quit** to stop playing the music and exit from VLC.

- Click **File** → **Close Window** to exit **File Manager**.

Suggestion: You can create a Play List using the VLC Media Player and store your favorite music files in this list for playing later.

Appendix

Bill of Materials

Components and modules used to replicate the projects described in this book.

- 8× red LED
- 8× 470 ohm resistor
- 1× MCP3002 ADC
- 1× TMP36
- 2× HC-SR04 ultrasonic module
- 1× I^2C LCD
- 1× Sense HAT
- 1× Buzzer
- 2× 10 k-ohm resistor
- 2× pushbutton switch
- 1× TXS0102
- 1× DHT11
- 1× BMP280
- 1× HC-06

Additionally

- 1× Raspberry Pi Zero 2 W
- 1× Raspberry Pi Pico W
- 1× Arduino Uno
- Jumper cables
- microUSB cables
- Breadboard

Index

The Raspberry Pi Zero 2 W *GO!* Book

A Fast-Lane Ride From Concept to Project

The core of the book explains the use of the Raspberry Pi Zero 2 W running the Python programming language, always in simple terms and backed by many tested and working example projects. On part of the reader, familiarity with the Python programming language and some experience with one of the Raspberry Pi computers will prove helpful. Although previous electronics experience is not required, some knowledge of basic electronics is beneficial, especially when venturing out to modify the projects for your own applications.

Over 30 tested and working hardware-based projects are given in the book, covering the use of Wi-Fi, communication with smartphones and with a Raspberry Pi Pico W computer. Additionally, there are Bluetooth projects including elementary communication with smartphones and with the popular Arduino Uno. Both Wi-Fi and Bluetooth are key features of the Raspberry Pi Zero 2 W.

Some of the topics covered in the book are:

- Raspberry Pi OS installation on an SD card
- Python program creation and execution on the Raspberry Pi Zero 2 W
- Software-only examples of Python running on the Raspberry Pi Zero 2 W
- Hardware-based projects including LCD and Sense HAT interfacing
- UDP and TCP Wi-Fi based projects for smartphone communication
- UDP-based project for Raspberry Pi Pico W communication
- Flask-based webserver project
- Cloud storage of captured temperature, humidity, and pressure data
- TFT projects
- Node-RED projects
- Interfacing to Alexa
- MQTT projects
- Bluetooth-based projects for smartphone and Arduino Uno communications

All programs discussed in this book are contained in an archive file you can download free of charge from the Elektor website. Head to: www.elektor.com/books and enter the book title in the Search box.

Prof Dogan Ibrahim has a BSc (Hons) degree in Electronic Engineering, an MSc degree in Automatic Control Engineering, and a PhD degree in Digital Signal Processing and Microprocessors.

Dogan has worked in many organizations and is a Fellow of the Institution of Engineering and Technology (IET) in UK as well as a Chartered Electrical Engineer. He has authored over 100 technical books and over 200 technical articles on electronics, microprocessors, microcontrollers, and related fields. Dogan is a certified Arduino professional and has many years of experience with numerous types of microprocessors and microcontrollers.

Elektor International Media
www.elektor.com

ISBN 978-3-89576-549-0
9 783895 765490