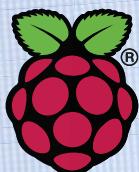


Raspberry Pi for Radio Amateurs

Program and build RPI-based ham station utilities, tools, and instruments.



Dogan Ibrahim, G7SCU

Raspberry Pi for Radio Amateurs

**Program and build RPi-based ham
station utilities, tools, and instruments**



Dogan Ibrahim, G7SCU

● 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

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

● British Library Cataloguing in Publication Data

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

ISBN 978-3-89576-404-2

Ebook 978-3-89576-405-9

Epub 978-3-89576-406-6

© Copyright 2020: Elektor International Media B.V.

Prepress Production: D-Vision, Julian van den Berg



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

LEARN ➤ DESIGN ➤ SHARE

Preface	11
CHAPTER 1 • Raspberry Pi Models	12
1.1 Overview	12
1.2 Raspberry Pi 1 Model A	12
1.3 Raspberry Pi 1 Model A+	13
1.4 Raspberry Pi 1 Model B	14
1.5 Raspberry Pi 1 Model B+	15
1.6 Raspberry Pi 2 Model B	16
1.7 Raspberry Pi Zero	17
1.8 Raspberry Pi 3 Model B	18
1.9 Raspberry Pi Zero W	18
1.10 Raspberry Pi 3 Model B+	19
1.11 Raspberry Pi 4 Model B	20
1.11.1 Raspberry Pi 4 purchase and setup options	25
1.12 Summary	29
Chapter 2 • Installing the Operating System on Raspberry Pi	30
2.1 Overview	30
2.2 Raspbian Buster installation steps on Raspberry Pi 4	30
2.3 Using networked connection	33
2.4 Remote access	35
2.5 Using the Putty	36
2.5.1 Configuring the Putty	37
2.6 Remote access of the Desktop	38
2.7 Static IP address	39
2.8 Summary	42
Chapter 3 • Using the Command Line	43
3.1 Overview	43
3.2 The command prompt	43
3.3 Useful Linux commands	43
3.3.1 System and user information	43
3.3.2 The Raspberry Pi directory structure	45
3.3.3 Resource monitoring on Raspberry Pi	57

3.3.4 Shutting down	59
3.4 Summary	60
CHAPTER 4 • A Quick Look at the Desktop	61
4.1 Overview	61
4.2 The Desktop	61
4.3 Libre Office Writer	63
4.4 Libre Office Calc	64
4.5 VLC media player.	64
4.6 Calculator.	65
4.7 File Manager.	65
4.8 SD Card Copier	66
4.9 Task Manager	67
4.10 Terminal	67
4.11 Help	67
4.12 Add/Remove software	68
4.13 Mouse and keyboard settings	68
4.14 Raspberry Pi configuration	69
4.15 Shutdown	69
4.16 Configuring Wi-Fi	69
4.17 Configuring Bluetooth.	70
4.18 Summary	70
CHAPTER 5 • Raspberry Pi Program Development	71
5.1 Overview	71
5.2 The 'nano' text editor.	71
5.3 Creating and running a Python program.	73
5.4 Summary	77
CHAPTER 6 • The GPIO.	78
6.1 Overview	78
6.2 The Raspberry Pi 4 GPIO connector	78
6.3 Interfacing to the GPIO	79
6.3.1 Loads requiring small currents.	79
6.3.2 Loads requiring higher currents	80

6.3.3 Using relays	81
6.4 The GPIO library	82
6.4.1 Pin numbering	82
6.4.2 Channel (I/O port pin) configuration	83
6.5 The Raspberry Pi project development cycle	85
6.5.1 The hardware	86
6.5.2 The software	86
6.6 Project – Alternately flashing red and green LEDs	87
6.7 Running a program automatically at startup time	90
6.8 Scheduling a program to run at specified times	91
6.9 Summary	97
CHAPTER 7 • Station Mains On/Off Power Control	98
7.1 Project	98
CHAPTER 8 • Station Clock	103
8.1 Project	103
8.2 Real-time clock	108
CHAPTER 9 • Why Multitasking?	111
CHAPTER 10 • The Station Temperature and Humidity	116
10.1 Project	116
CHAPTER 11 • Station Mains On-Off Control, Station Time, and Station Weather	120
CHAPTER 12 • Station Geographical Coordinates	125
CHAPTER 13 • Waveform Generation — Using Software	133
13.1 The MCP4921 DAC	133
13.2 Generating a squarewave signal with a peak voltage of 3.3 V	135
13.3 Generating a squarewave signal with any peak voltage	138
13.4 Generating a sawtooth-wave signal	142
13.5 Generating a triangular-wave signal	144
13.6 Generating an arbitrary-wave signal	146
13.7 Generating a sinewave signal	149
CHAPTER 14 • Waveform Generation – Using Hardware	153
14.1 Project: Fixed-Frequency Waveform Generator	153
14.2 Project: Keypad Frequency Entry, LCD Readout, Waveform Generator	160

CHAPTER 15 • Designing a Single Stage Common-Emitter Bipolar Transistor Amplifier Circuit	171
15.1 Project	171
CHAPTER 16 • Active Low-Pass Filter Design	176
16.1 Project	176
CHAPTER 17 • Morse Code Exerciser	182
17.1 Project: MCE with User-Entered Characters	182
17.2 Project: MCE sending randomly generated characters	187
17.3 Project: MCE with Rotary-Encoder WPM Setting and CD readout	190
CHAPTER 18 • Voltmeter – Ammeter – Ohmmeter - Capacitance Meter	198
18.1 Project: Voltmeter	198
18.2 Project: Ammeter	202
18.3 Project: Ohmmeter	202
18.4 Project: Capacitance Meter	204
CHAPTER 19 • Frequency Counter	209
19.1 Project: Frequency Counter	209
CHAPTER 20 • Raspberry Pi 4 Audio Input & Portable Power Supply	215
20.1 Raspberry Pi audio outputs	215
20.1.1 Testing	216
20.2 Using an external USB audio input-output device	217
20.2.1 Testing (1-2-3)	218
20.3 Powering the Raspberry Pi 4	219
CHAPTER 21 • Raspberry Pi FM Transmitter	222
21.1 Project: Raspberry Pi 4 VHF FM Transmitter	222
21.2 Project: RadioStation Click board	223
CHAPTER 22 • RF Power Meter	234
22.1 Project: RF Power Meter	234
22.2 RF attenuator	240
22.3 dB, dBm, and watt?	240
CHAPTER 23 • Raspberry Pi – Smartphone Projects	244
23.1 The MIT App Inventor	244
23.2 Setting up the MIT App Inventor	245

23.3 Project: Web Server to Control Multiple Relays	247
CHAPTER 24 • RTL-SDR and Raspberry Pi	256
24.1 Overview	256
24.2 Installing the RTL-SDR software on Raspberry Pi 4	258
24.3 The GQRX	261
24.4 The CubicSDR	269
24.5 RTL-SDR server	270
24.6 SimpleFM	272
24.7 ShinySDR	274
24.8 Other SDR-RTL software	277
24.9 The SDR – The big brother of RTL-SDR?	277
24.9.1 The HackRF One	278
24.9.2 The NooElec NESDR Smart HF bundle	278
24.9.3 The AirSpy HF+	279
24.9.4 The Quisk	280
24.10 Receiving Weather Fax (WEFAX)	281
CHAPTER 25 • Using Some of the Popular Radio Applications.	284
25.1 TWCLK	284
25.2 Klog	287
25.3 Gpredict	288
25.4 FLDIGI	289
25.5 Direwolf	291
25.6 xcwcp	294
25.7 QSSTV	295
25.8 LinPsK	298
25.9 Ham Clock	299
25.9 CHIRP	301
25.10 Xastir	303
25.11 CQRLOG	305
25.12 What next?	305
Index	307

Preface

In recent years there has been major changes in the radio equipment used by radio amateurs around the globe (and in orbit!). Although much classical HF and mobile equipment is still in use by large number of amateurs, we see the use of computers and digital techniques becoming very popular among amateur radio operators or 'hams'. In early days of digital communications, personal computers were used by the amateurs to communicate with each other. Personal computers have the disadvantage that they are rather expensive and bulky. Nowadays, anyone can purchase a £40 Raspberry Pi computer and run almost all of the amateur radio software on this computer, which is slightly bigger than the size of a credit card.

Several authors have written books and published projects for using the Arduino in amateur radio projects. Although the Arduino can be used in hardware based projects, it is rather limited since it lacks an operating system. Raspberry Pi is an alternative to Arduino because of its popular operating system, large memory, and rich on-board peripheral support, such as USB, Bluetooth, Wi-Fi, camera interface, etc. As a result of this, Raspberry Pi is well suited to be used as a digital computer for amateur radio, and most of the amateur radio software packages that were available on PCs can now be installed and used on the Raspberry Pi.

The RTL-SDR devices have become very popular by the amateurs because of their very low cost (around £12) and rich features. A basic system may consist of a USB based RTL-SDR device (dongle) with a suitable antenna, a Raspberry Pi computer, an USB based external audio input-output adapter, and software installed on the Raspberry Pi computer. With such a simple setup it is possible to receive signals from around 24 MHz to over 1.7 GHz. With the addition of a low cost upconverter device, an RTL-SDR can easily and effectively receive the HF bands.

This book has four purposes: Firstly, it is aimed to teach the basic operating principles and features of the Raspberry Pi to beginners. Secondly, many hardware based projects are given using the Raspberry Pi together with the Python programming language. Although these projects are general, they have been chosen to be useful to the amateur radio operators. Thirdly, the book explains in some detail how to use the RTL-SDR devices together with a Raspberry Pi and popular RTL-SDR software to tune into and receive signals from a wide range of bands. Lastly, the book explains how to install and use some of the commonly used amateur radio software packages on the Raspberry Pi.

I hope you enjoy reading the book.

Prof Dogan Ibrahim, G7SCU

London, 2020

CHAPTER 1 • Raspberry Pi Models

1.1 Overview

The Raspberry Pi is a low-cost, single-board, powerful computer capable of running a full operating system and also capable of doing everything that a laptop or a desktop computer can do, such as creating and editing documents, getting on the Internet, receiving and sending mails, playing games, developing programs to monitor and control its environment via electronic sensors and actuators, and many more.

Currently, there are many different models of the Raspberry Pi available, each having slightly different features. The fundamental features of all the Raspberry Pi computers are very similar, all using ARM processors, all having the operating system installed on an SD card, all having on-board memory as well as input-output interface connectors. Some models, such as the Raspberry Pi 4, Raspberry Pi 3, and Raspberry Pi Zero W have built-in Wi-Fi and Bluetooth capabilities, making them easy to get online and to communicate with similar devices having Wi-Fi or Bluetooth connectivity.

In this book we shall be concentrating on the most advanced model, which was the Raspberry Pi 4 at the time of writing. All the projects developed in this book will run on this model and virtually all predecessors. Some of the projects may not run on the lower models such as the Raspberry Pi Zero W, but necessary information will be given at the beginning of each project.

In this Chapter we shall briefly take a look at the features of the different models of the Raspberry Pi computer. Interested readers can get further information from the following web site:

https://en.wikipedia.org/wiki/Raspberry_Pi

1.2 Raspberry Pi 1 Model A

This model pictured in Figure 1.1 was released in 2013 has the following features:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	700 MHz
RAM:	256 MB
Camera interface	
USB ports:	1
HDMI ports:	1
Composite video	
SD/MMC:	SD card
GPIO:	26 pins
Current:	200 mA
Cost:	\$20

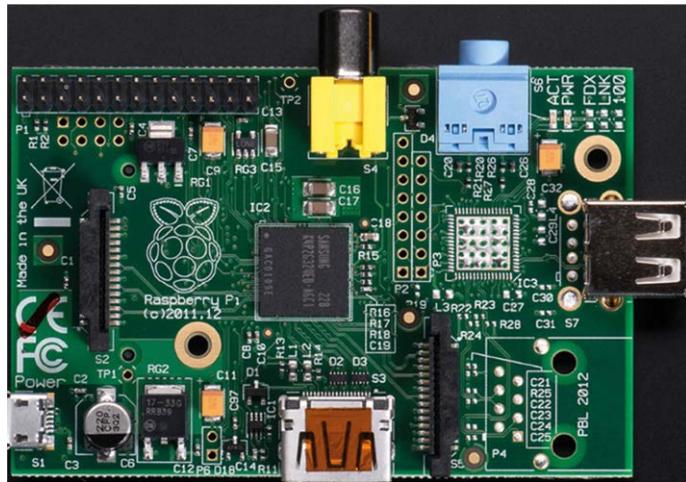


Figure 1.1: Raspberry Pi 1 Model A.

1.3 Raspberry Pi 1 Model A+

This model pictured in Figure 1.2 was released in 2014 and it has the following basic features:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	700 MHz
RAM:	256 MB
Camera interface	
USB ports:	1
HDMI ports:	1
Composite video	
SD/MMC:	microSD card
GPIO:	40 pins
Current:	200 mA
Cost:	\$20

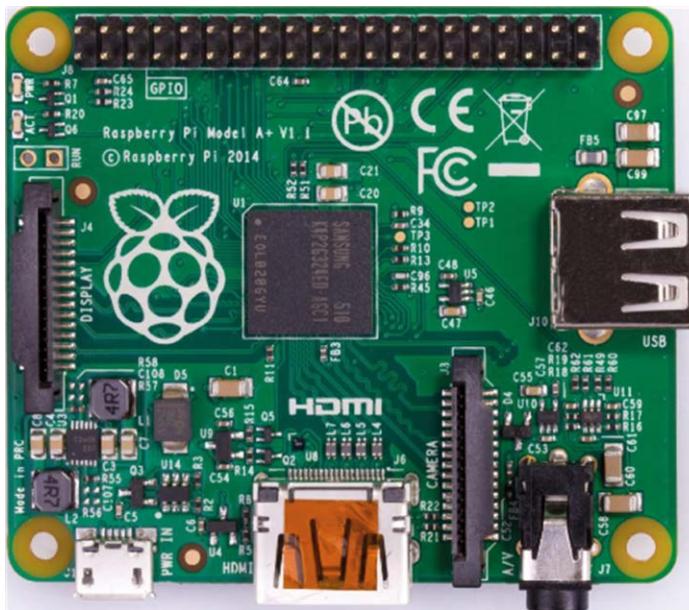


Figure 1.2: Raspberry Pi 1 Model A+.

1.4 Raspberry Pi 1 Model B

This model pictured in Figure 1.3 was released in 2012 and it has the following features:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	700 MHz
RAM:	512 MB
USB ports:	2
HDMI ports:	1
Ethernet ports:	1
Camera interface	
Composite video	
SD/MMC:	SD card
GPIO:	26 pins
Current:	700 mA
Cost:	\$25

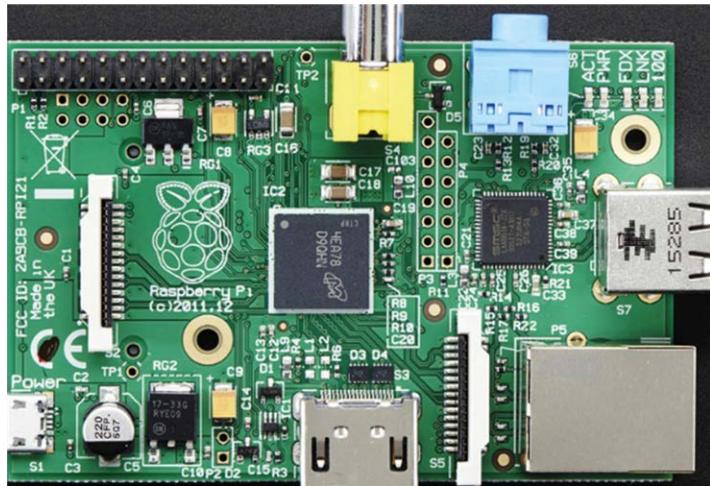


Figure 1.3: Raspberry Pi 1 Model B.

1.5 Raspberry Pi 1 Model B+

This model pictured in Figure 1.4 was released in 2014 and it has the following features:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	700 MHz
RAM:	512 MB
USB ports:	4
HDMI ports:	1
Camera interface	
Composite video	
Ethernet ports:	1
SD/MMC:	microSD card
GPIO:	40 pins
Current: 700 mA	
Cost:	\$25

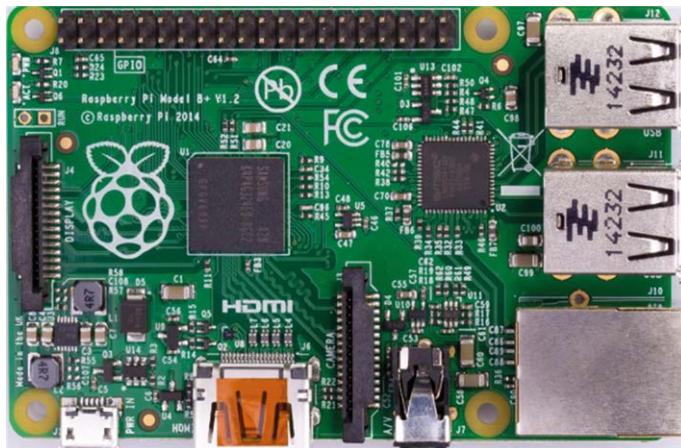


Figure 1.4: Raspberry Pi Model B+

1.6 Raspberry Pi 2 Model B

This model pictured in Figure 1.5 was released in 2015 and it has larger memory, more USB ports, and a faster processor:

SOC:	Broadcom BCM2836
Processor:	Cortex-A7
No of cores:	4
CPU clock:	900 MHz
RAM:	1 GB
USB ports:	4
Ethernet ports:	1
HDMI ports:	1
Camera interface	
Composite video	
SD/MMC:	microSD card
GPIO:	40 pins
Current:	800 mA
Cost:	\$35

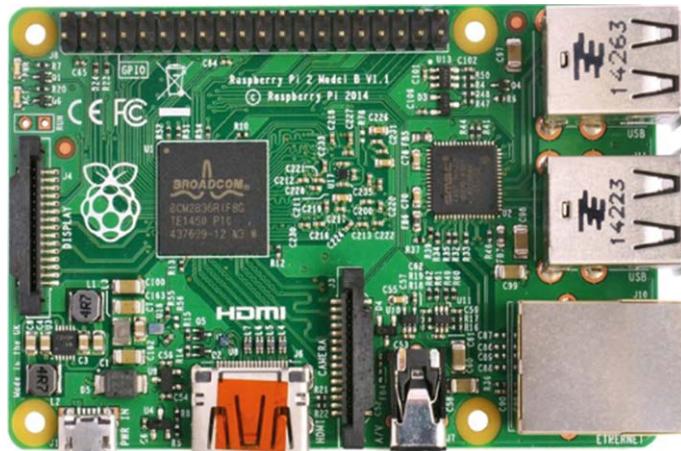


Figure 1.5: Raspberry Pi 2 Model B.

1.7 Raspberry Pi Zero

This model pictured in Figure 1.6 was released in 2015 and it is a smaller board than the others, but has a fast processor. Its main features are:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	1 GHz
RAM:	512 MB
USB ports:	1 (micro)
Camera interface	
HDMI ports:	1 (mini)
SD/MMC:	microSD card
GPIO:	40 pins
Current:	160 mA
Cost:	\$5

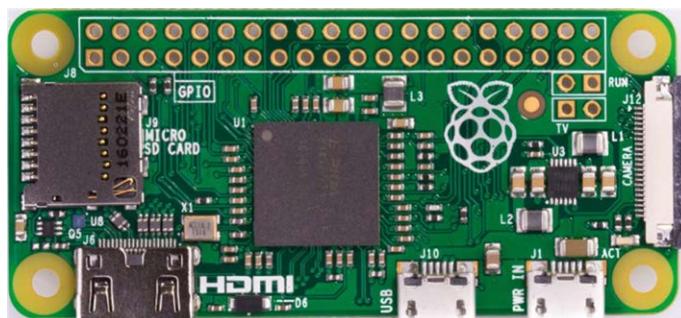


Figure 1.6: Raspberry Pi Zero.

1.8 Raspberry Pi 3 Model B

This model pictured in Figure 1.7 was released in 2016 and its main features are:

SOC:	Broadcom BCM2837
Processor:	Cortex A-53
No of cores:	4
CPU clock:	1.2 GHz
RAM:	1 GB
USB 2.0 ports:	4
Ethernet ports:	1 (10/100 Mbit/s)
HDMI ports:	1
Camera interface	
Composite video	
Wi-Fi:	b/g/n
Bluetooth:	4.1
SD/MMC:	microSD card
GPIO:	40 pins
Current:	1.34 A
Cost:	\$35



Figure 1.7: Raspberry Pi 3 Model B.

1.9 Raspberry Pi Zero W

The Raspberry Pi Zero W pictured in Figure 1.8 was released in 2017, and it is a small board (half the size of Model A+) with low current consumption but has a surprising amount of power. Its main advantages are the on-board Wi-Fi and Bluetooth connectivity. The basic features of this model are:

SOC:	Broadcom BCM2835
Processor:	ARM1176JZF-S
No of cores:	1
CPU clock:	1 GHz
RAM:	512 MB
USB ports:	2 (micro)
Camera interface	
HDMI ports:	1 (mini)
Wi-Fi	
Bluetooth	
SD/MMC:	microSD card
GPIO:	40 pins
Current:	180 mA
Cost:	\$10

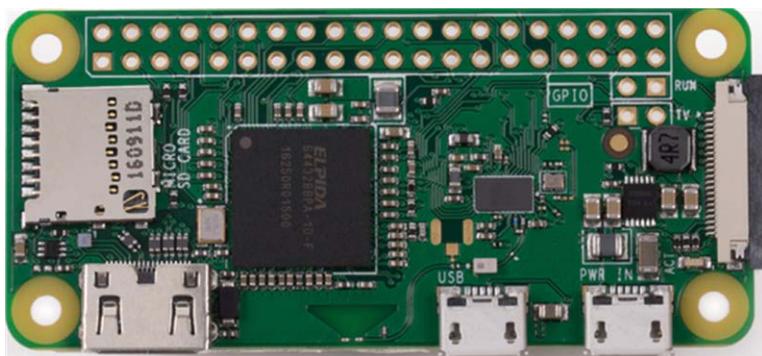


Figure 1.8: Raspberry Pi Zero W.

1.10 Raspberry Pi 3 Model B+

This model pictured in Figure 1.9 was released in 2018 and it has the second fastest processor speed of all the current models. Its main features are:

SOC:	Broadcom BCM2837B0
Processor:	Cortex A-53
No of cores:	4
CPU clock:	1.4 GHz
RAM:	1 GB
USB 2.0 ports:	4
Ethernet ports:	1 (10/100/1000 Mbit/s)
HDMI ports:	1
Camera interface	
Composite video	
Wi-Fi:	b/g/n/ac
Bluetooth:	4.2
SD/MMC:	microSD card
GPIO:	40 pins

Current: 1.13 A
Cost: \$35



Figure 1.9: Raspberry Pi 3 Model B+.

1.11 Raspberry Pi 4 Model B

This is the latest and the fastest Raspberry Pi (Figure 1.10) that was available at the time of writing this book. The main features of this computer are:

SOC:	Broadcom BCM2711
Processor:	Cortex A-72
No of cores:	4
CPU clock:	1.5 GHz
RAM:	1, 2, 4, or 8 GB
USB 2.0 ports:	2
USB 3.0 ports:	2
USB-C:	1 (power)
Ethernet ports:	1 (10/100/1000 Mbit/s)
HDMI ports:	2
Camera interface	
Composite video	
Wi-Fi:	b/g/n/ac
Bluetooth:	5.0
SD/MMC:	microSD card
GPIO:	40 pins
Current:	1.25 A
Cost:	\$35 - \$75

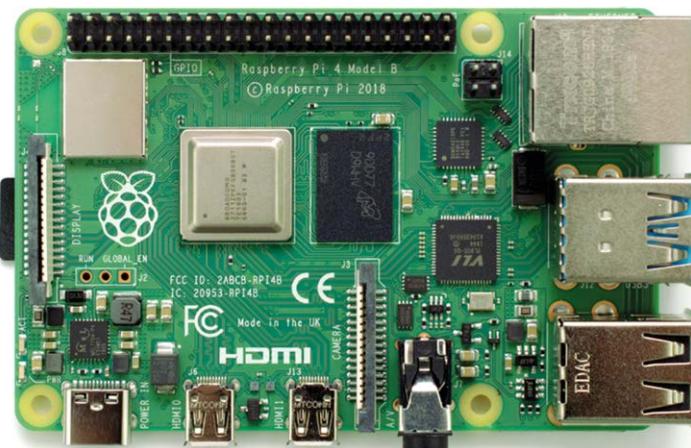


Figure 1.10: Raspberry Pi 4 Model B.

Since we will be using the Raspberry Pi 4 Model B in our projects, it is worthwhile to look at the details of this model in greater detail. In the remainder parts of this book, the name Model B will be dropped, and the board will just be named 'Raspberry Pi 4'.

The new BCM2711 SoC is a very powerful processor. One problem with this processor is that it can get very hot and a fan is recommended to cool it down. At the end of a 10-minute test, the temperature of the processor was measured to be 74.5 °C (as a comparison, the temperature of a Raspberry Pi Model B+ reached 62.6 °C during the same period). Several small fans with different shapes and sizes are available to cool the processor chip, and it is recommended that the users choose and use a suitable fan. The fans take their power from the header connector on the board. Some fans for the Raspberry Pi 4 are shown in Figure 1.11 and Figure 1.12. Although large fans are more efficient, it is recommended by the author not to use a too large fan since it can complicate access to the GPIO header or the connection of a plug-in board (e.g. a HAT shield) onto the header connector. Because of the higher power consumption, a 3A external power supply is recommended for the Raspberry Pi 4. The power supply is connected to USB-C connector of the Raspberry Pi 4 (Figure 1.13).



Figure 1.11: Large Raspberry Pi 4 fan.



Figure 1.12: Small Raspberry Pi fan.



Figure 1.13: Raspberry Pi 4 USB-C power supply.

Although the Raspberry Pi 4 clock speed is only 100 MHz higher than that of the Raspberry Pi 3 Model B+, its performance is much better as it uses the highly efficient high-speed Cortex-A72 processor. Figure 1.14 shows the Linpack Benchmark speed comparison of the different models (see: <https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b>). The high performance of the Raspberry Pi 4 is very clear from this figure.

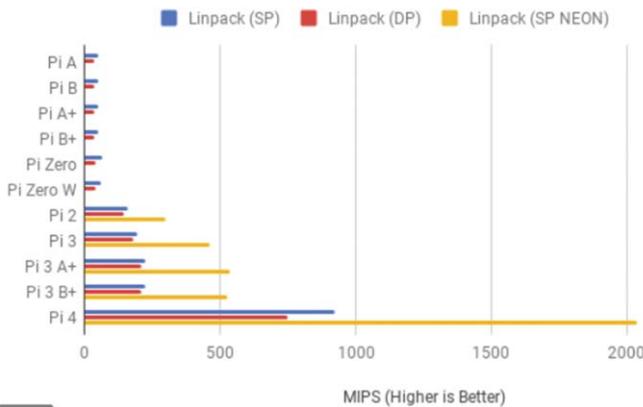


Figure 1.14: Linpack Benchmark.

Image processing time comparison is another benchmark normally used to compare different processors. Figure 1.15 shows the GIMP Image Editing Benchmark of different models. Again, the superiority of the Raspberry Pi 4 is clear from this figure. For example, compared to Raspberry Pi Zero W, the Pi 4 is about 8 times faster in processing an image.

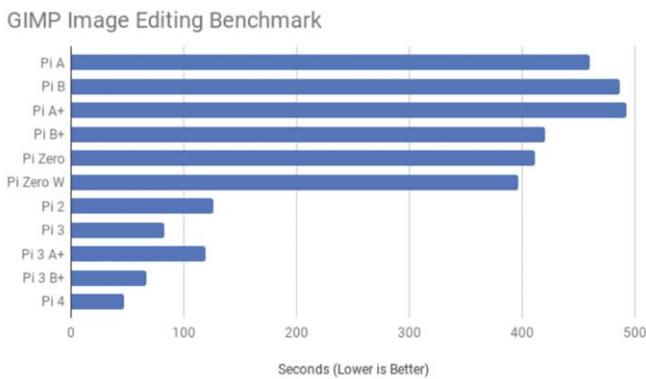


Figure 1.15: GIMP Image Editing Benchmark.

Figure 1.16 shows the component layout on the Raspberry Pi 4 (source: <https://www.seeedstudio.com>).

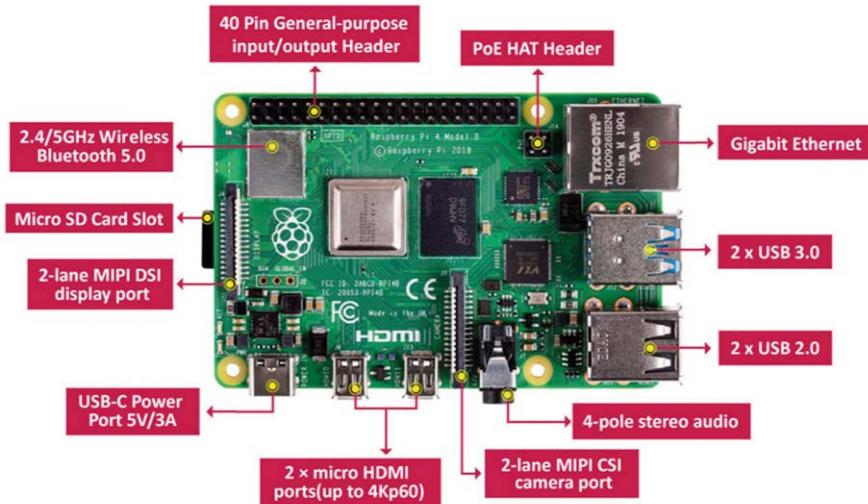


Figure 1.16: Raspberry Pi 4 component layout.

A brief description of the various components on the board is given below.

Processor: the processor is enclosed in a metal cap and it is based on Broadcom BCM2711B0, which consists of a Cortex A-72 core, operating at 1.5 GHz.

RAM: There are three versions of Raspberry Pi 4 depending on the amount of DDR4 RAM required: 1 GB, 2 GB, and 4 GB.

USB Ports: Raspberry Pi 4 includes 2 × USB 3.0, 2 × USB 2.0, and 1 × USB-C ports. USB 3.0 data transfer rate is 4,800 Mbps (megabits per second), while USB 2.0 can transfer at up to 480 Mbps, i.e. 10 times slower than the USB 2.0. The USB-C port enables the board to be connected to a suitable power source.

Ethernet: The Ethernet port enables the board to be connected directly to an Ethernet port on a router. The port supports Gigabit connections (125 Mbps).

HDMI: Two micro-HDMI ports are provided that support up to 4 K screen resolutions. HDMI adapters can be used to interface the board to standard size HDMI devices.

GPIO: A 40-pin header is provided as the GPIO (General Purpose Input Output). This is compatible with the earlier GPIO ports.

Audio and Video Port: A 3.5-mm jack type socket is provided for stereo audio and composite video interface. Headphones can be connected to this port. External amplifier devices will be required to connect speakers to this port. This port also supports composite video, enabling TV sets, projectors, and other composite video compatible display devices to be connected to the port.

CSI Port: This is the camera port (Camera Serial Interface), allowing a compatible camera to be connected to the Raspberry Pi.

DSI Port: This is the display port (Display Serial Interface), allowing a compatible display (e.g. 7-inch Raspberry Pi display) to be connected to the Raspberry Pi.

PoE Port: This is a 4-pin header, allowing the Raspberry Pi to receive power from a network connection.

Micro SD Card: This card is mounted at the card holder placed at the bottom of the board and it holds the operating system software as well as the operating system and user data.

1.11.1 Raspberry Pi 4 purchase and setup options

The user has two options:

- Purchase Raspberry Pi 4 as a kit (see Figure 1.17) including the processor board, power supply, micro SD card with the operating system already loaded, fan, cables, etc.
- Purchase the processor board, the power supply and a blank micro SD card and then build the operating system onto the SD card (the topic of next Chapter)

The choice of whether to purchase a kit or the individual components depends entirely on the user's finances and decisions.

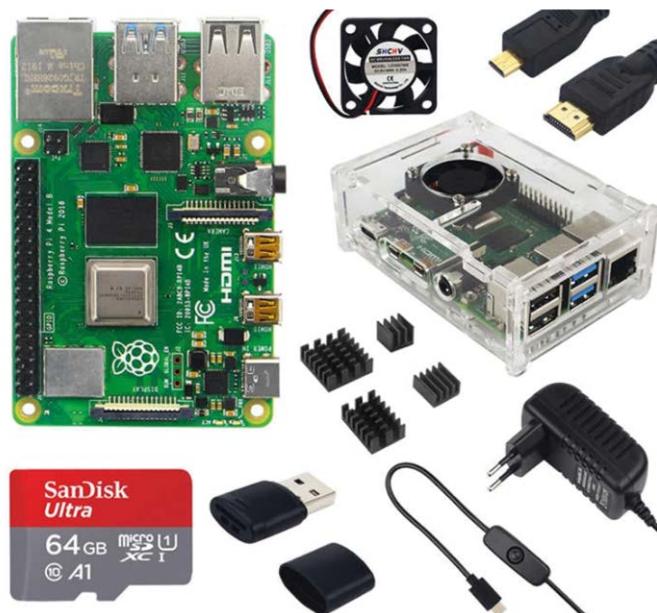


Figure 1.17: Raspberry Pi as a kit.

As shown in Figure 1.18, Raspberry Pi can be set up in two ways: using **direct connection**, or **connecting through a network**

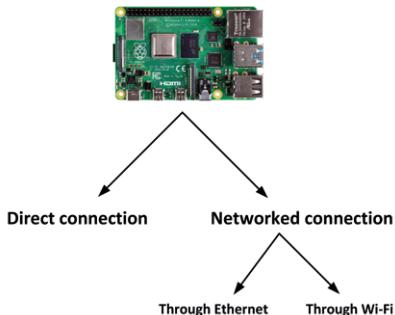


Figure 1.18: Setting up the Raspberry Pi.

Direct connection

This is perhaps the most expensive way of setting up and using the Raspberry Pi 4. In this configuration a monitor and a keyboard are connected to the Raspberry Pi 4. In this setup, the minimum required components are:

- Power supply
- Micro SD card
- Operating system software
- USB keyboard and mouse
- Micro HDMI cable to receive sound and video signals
- HDMI compatible display or TV (you may also need to have micro HDMI to DVI-D or VGA adapters. A 3.5-mm TRRS type cable and plug will be required if you will be using an old TV set with composite video)

Power Supply: As mentioned earlier, a 5 V 3 A power supply with USB-C type connector is required.

Micro SD Card: It is recommended to use a micro SD card with a capacity of at least 8 GB, although higher capacity (e.g. 16 GB or 32 GB) is better as there will be room to grow in the future. A Class 10 (or faster) card is recommended.

Operating System: You can purchase the operating system pre-loaded on a micro SD card, which requires minimum configuration before it is fully functional. The alternative is to purchase a blank micro SD card and upload the operating system on this card. The steps to prepare a new micro SD card with the operating system are given in the next Chapter.

USB Keyboard and Mouse: You can either use a wireless or wired keyboard and mouse pair. When using a wired pair, you should connect the keyboard to one of the USB ports and the mouse to another USB port. When using a wireless keyboard and mouse, you should connect the wireless dongle to one of the USB ports.

Display: A standard HDMI compatible display monitor with a micro HDMI to standard HDMI adapter can be used. Alternatively, a VGA type display monitor with a micro HDMI to VGA adapter or DVI-D adapter can be used. If you have an old TV set with composite video interface (CVBS), then you can connect it to the Raspberry Pi 3.5-mm port with a TRRS type connector.

You may also consider purchasing additional parts, such as a case, CPU fan, and so on. A case is very useful as it protects your Raspberry Pi electronics.

Figure 1.19 shows a possible direct connection setup. Here, depending on what type of display monitor we have, we can use an HDMI display, a VGA monitor, a DVI-D monitor, or a TV. Notice that depending on the external USB devices used, you can use either the USB 2.0 or the USB 3.0 ports.

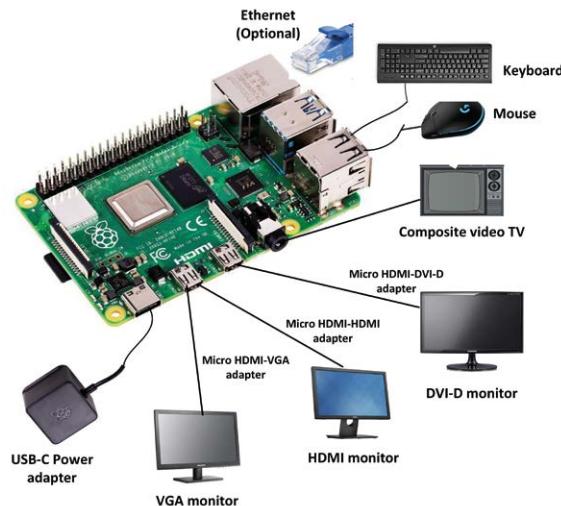


Figure 1.19: Raspberry Pi 4 setup — option 1.

Figure 1.20 shows another way of connecting to the Raspberry Pi directly. In this option, a powered hub is used to connect the USB devices.

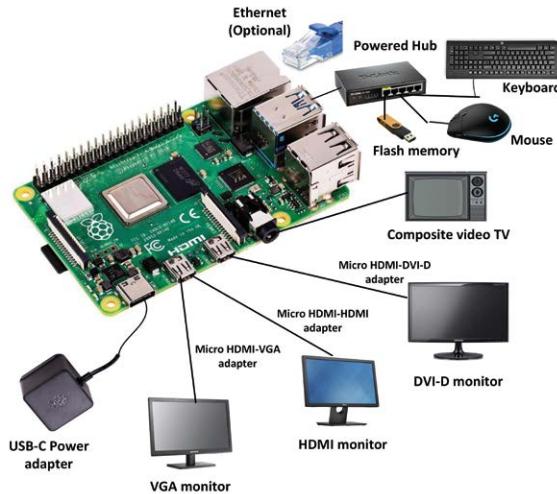


Figure 1.20: Raspberry Pi 4 setup — option 2.

Connecting through a network

Connecting through a network is recommended by the author since it is cheaper, requires less components and is easier to manage. There are two possible ways of setting up and using the Raspberry Pi through a network:

Connection through Ethernet: This option is only available if the Raspberry Pi is equipped with an Ethernet port, such as the Raspberry Pi 2/3/4. As shown in Figure 1.21, in this configuration the Ethernet port is connected directly to the Wi-Fi router (e.g. through a hub) and a PC is used to access the Raspberry Pi over the network. The disadvantage of this method is that the Raspberry Pi has to be close to the Wi-Fi router which may not always be possible or desirable.

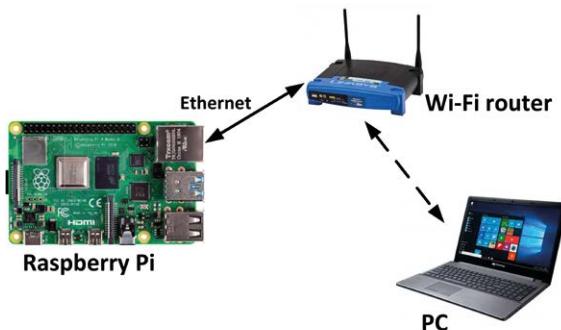


Figure 1.21: Connection through Ethernet.

Wi-Fi connection: This is perhaps the cheapest and the easiest way of using the Raspberry Pi. Here, the Raspberry Pi is connected to the external world through its on-board Wi-Fi module. Most Raspberry Pi models (e.g. Zero W, Pi 2/3/4 etc) are equipped with Wi-Fi

modules. As shown in Figure 1.22, Raspberry Pi is accessed over the Wi-Fi using a PC. The advantage of this method is that it is cheap, easy, and very flexible since the Raspberry Pi can be placed anywhere within the range of the Wi-Fi router.

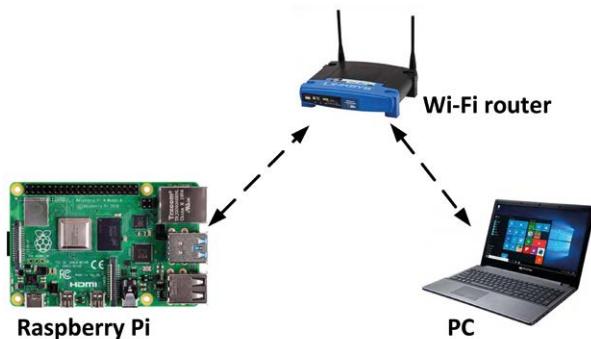


Figure 1.22: Wi-Fi connection.

1.12 Summary

In this Chapter we had a look at the basic features of the different models of the Raspberry Pi computer. It is recommended to use the small and low-cost latest model Raspberry Pi Zero W for Wi-Fi or Bluetooth based applications where 512 MB RAM and 1 GHz clock speed are sufficient. For higher speed and more memory requirements, the slightly more expensive Raspberry Pi 3 or Raspberry Pi 4 are recommended. The Raspberry Pi 4 is the model that is used in all the projects in this book.

In the next Chapter we will be seeing how to load the operating system to Raspberry Pi and how to connect to it through a network.

Chapter 2 • Installing the Operating System on Raspberry Pi

2.1 Overview

In this Chapter we will be learning how to install the latest operating system (**Raspbian Buster**) on the Raspberry Pi 4, and learn the different ways of using the Python programming language. Notice that the installation process given below applies to all Raspberry Pi models unless specified otherwise.

2.2 Raspbian Buster installation steps on Raspberry Pi 4

Raspbian Buster is the latest operating system of the Raspberry Pi. This section gives the steps for installing this operating system on a new blank SD card, ready to use with your Raspberry Pi 4. You will need a micro SD card with a capacity of at least 8 GB (16 GB is even better) before installing the new operating system on it.

The steps to install the Raspbian Buster are as follows:

- Download the Buster image to a folder on your PC (e.g. C:\RPiBuster) from the following link by clicking the Download ZIP under section **Raspbian Buster with desktop and recommended software** (see Figure 2.1). At the time of writing, the file was called: **2020-02-13-raspbian-buster-full.img**. You may have to use the Windows 7Zip software to unzip the download since some of the features are not supported by older unzip software.

<https://www.raspberrypi.org/downloads/raspbian/>

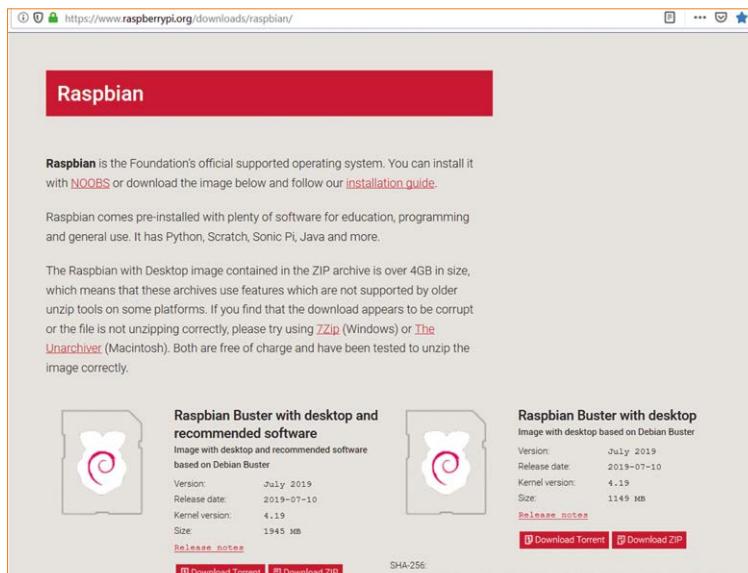


Figure 2.1: Raspbian Buster download page.

- Put your blank micro SD card into the card slot on your computer. You may need to use an adapter to do this.
- Download the Etcher program to your PC to flash the disk image. The link is (see Figure 2.2):

<https://www.balena.io/etcher/>

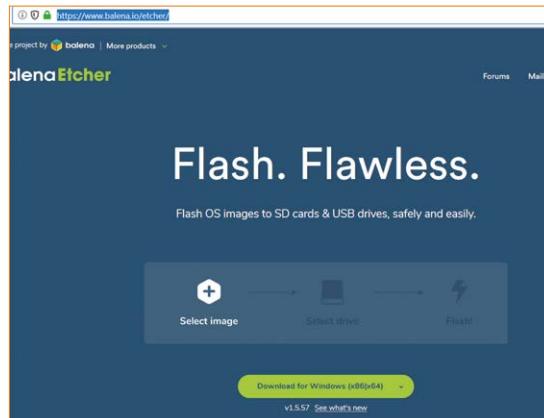


Figure 2.2: Download Etcher.

- Double click to Open Etcher, and click **Select image**. Select the Raspbian Buster file you just downloaded and unzipped.
- Click **Select target** and select the micro SD card
- Click **Flash** (see Figure 2.3). This may take several minutes, wait until it is finished. The program will then validate and unmount the micro SD card. You can remove your micro SD card after it is unmounted.

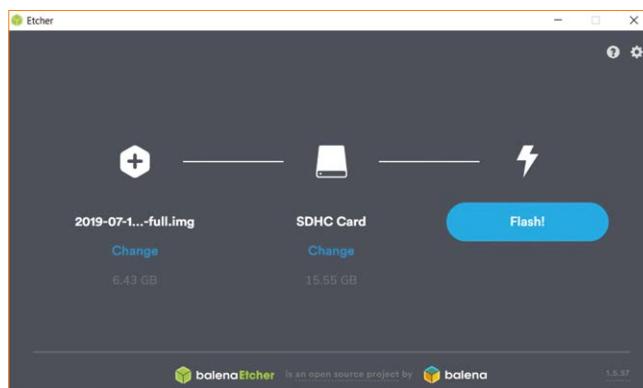


Figure 2.3: Click Flash to flash the disk image.

Now your micro SD card has been loaded with the Raspberry Pi operating system. The various options now are as follows:

Using direct connection

If you are making direct connection to your Raspberry Pi using a monitor and keyboard, just insert the SD card into the card slot and power-up your Raspberry Pi. After a short while you will be prompted to enter the login details. The default values are: username: **pi**, password: **raspberry**. We will see in the next Chapter how to change the password.

You can now start using your Raspberry Pi either in command mode or in Desktop mode. If you are in command mode, then enter the following command to start the GUI mode:

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

If you want to boot in GUI mode by default, then the steps are:

- Start the configuration tool:

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

- Move down to **Boot Options** and press Enter to select (Figure 2.4)

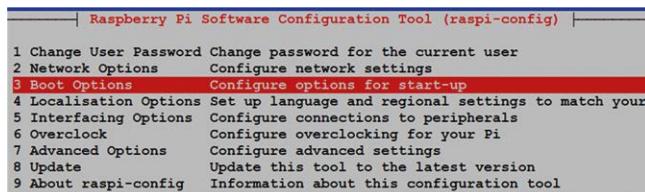


Figure 2.4: Select Boot Options.

- Select **Desktop / CLI** and then select **Desktop Autologin** to boot automatically into GUI mode.
- Click **OK** and then accept to reboot the system. The system will be in GUI mode next time it reboots.
- You can change your selections to boot in command mode if you wish by selecting **Console** in **Boot Options**.

You may now want to connect your Raspberry Pi to the Internet either to access it remotely from a PC or to surf the Internet. If your Raspberry Pi is equipped with an Ethernet port (e.g. Raspberry Pi 2/3/4), then you can directly connect it to your Wi-Fi router using an Ethernet cable. You can find the IP address of your connection by entering the command: **ifconfig** in command mode.

Alternatively, you may want to connect your Raspberry Pi to your Wi-Fi and access it remotely. First, you will have to enable the **SSH**. The steps are:

- Start the configuration tool:

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

- Move down to **Interface Options** and then select **SSH** and enable it.
- If you are in GUI mode, click the Wi-Fi icon at the top right hand of the screen and enable the Wi-Fi. Note the IP address allocated automatically to your Raspberry Pi.
- You can now access your Raspberry Pi remotely using a terminal emulation software, such as **Putty** (see Section 2.4 and 2.5).

2.3 Using networked connection

If you do not have a suitable monitor and keyboard to connect directly to your Raspberry Pi, then you will have to use a networked connection and then access your Raspberry Pi remotely using a PC. As mentioned in Chapter 1, there are two options here: **connection using an Ethernet cable**, and **connection over Wi-Fi**.

Connection using an Ethernet cable: The steps are:

- Install the **NotePad++** software to your PC from the following website:
<https://notepad-plus-plus.org/downloads/v7.8.5/>
- Insert the SD card back to your PC and start the **NotePad++** software.
- Click **Edit** → **EOL Conversion** → **UNIX/OSX Format**.
- 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 remotely. In Windows this is the only folder you will see which contains items like: loader.bin, start.elf, kernel.img, etc.
- Insert the SD card back into your Raspberry Pi.
- Connect your Raspberry Pi to one of the ports of your Wi-Fi router through an Ethernet cable and power it up.
- Find out the IP address allocated to your Raspberry Pi by accessing your Wi-Fi router. Alternatively, install the **Advanced IP Scanner** program to your PC, available at the following link:

<https://www.advanced-ip-scanner.com>

- Run the software and look for your Raspberry Pi. You do not have to install this software to run it. Click to **Run portable version**, and then click **Scan**. As shown in Figure 2.5, the IP address of author's Raspberry Pi was 192.168.1.202.

	09AA01AC491808W2.home	192.168.1.200	Nest Labs Inc.	64:16:66:93:79:43
	raspberrypi.home	192.168.1.202		DC:A6:32:00:E4:29

Figure 2.5: IP address of the Raspberry Pi.

- You can now use Putty to login to your Raspberry Pi (see Sections 2.4 and 2.5).

Notice that, you can alternatively find the IP address of your Raspberry Pi by entering the command prompt on your PC with administrator privilege (by right-clicking to accepting to run as an administrator) and then entering the command: **ping raspberrypi.home** as shown in Figure 2.6.

```
C:\WINDOWS\system32>ping raspberrypi.home  
Pinging raspberrypi.home [192.168.1.202] with 32 bytes of data:  
Reply from 192.168.1.202: bytes=32 time=1ms TTL=64  
Reply from 192.168.1.202: bytes=32 time=2ms TTL=64  
Reply from 192.168.1.202: bytes=32 time=2ms TTL=64
```

Figure 2.6: Using ping to find the Raspberry Pi IP address.

It is also possible to find the IP address of your Raspberry Pi using your smartphone. There are many apps that can be used to find out who are currently using your Wi-Fi router. e.g. **Who's On My Wi-Fi – Network Scanner** by Magdalm and many others.

Connection using Wi-Fi: This is perhaps the preferred method of accessing your Raspberry Pi, and the method used by the author. Here, as described in Chapter 1, the Raspberry Pi can be placed anywhere you like within the reach of the Wi-Fi router, and it is accessed easily from your PC using the Putty (see Section 2.4 and 2.5).

The steps are:

- Install the **Notepad++** software to your PC from the following web site:

<https://notepad-plus-plus.org/downloads/v7.8.5/>

- Insert the SD card back to your PC and start the **Notepad++** software.

- Click **Edit → EOL Conversion → UNIX/OSX Format**

- 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 remotely. In Windows this is the

only folder you will see which contains items like: loader.bin, start.elf, kernel.img. etc.

- Enter the following statements into a blank file (replace the **MySSID** and **My>Password** 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"
}
```

- Copy the file (save) to the boot folder on your SD card with the name: **wpa_supplicant.conf**.
- Insert the SD card back into your Raspberry Pi and power-up the device.
- Use the **Advanced Ip Scanner** or one of the other methods described earlier to find out the IP address of your Raspberry Pi.
- Log in to your Raspberry Pi remotely using the **Putty** software on your PC (see Section 2.3 and 2.4).
- After logging in you are advised to change your password for security reasons. You should also run the **sudo raspi-config** from the command line to enable the VNC, I²C, and SPI as they are useful interface tools which can be used in your future GPIO based work.

2.4 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 display to it. Before being able to access the Raspberry Pi remotely, we have to enable the SSH by entering the following command at a terminal session (if you have followed the steps given earlier then the SSH is already enabled and you can skip the following command):

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

Go to the configuration menu and select **Interface Options**. Go down to **P2 SSH** and enable SSH. Click **<Finish>** to exit the menu.

You should also enable VNC so that the Raspberry Pi Desktop can be accessed graphically over the Internet. This can be done by entering the following command at a 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. At this stage you may want shutdown or restart your Raspberry Pi by entering one of the following commands in command mode:

```
pi@raspberrypi:~ $ sudo shutdown now
```

or

```
pi@raspberrypi:~ $ sudo reboot
```

2.5 Using the Putty

Putty is a communications program that is used to create a connection between your PC and the Raspberry Pi. This connection uses a secure protocol called SSH (Secure Shell). Putty doesn't need to be installed as it can just be stored in any folder of your choice and run from there.

Putty can be downloaded from the following website:

<https://www.putty.org/>

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 2.7). The message shown in Figure 2.8 will be displayed the first time you access the Raspberry Pi. Click **Yes** to accept this security alert.

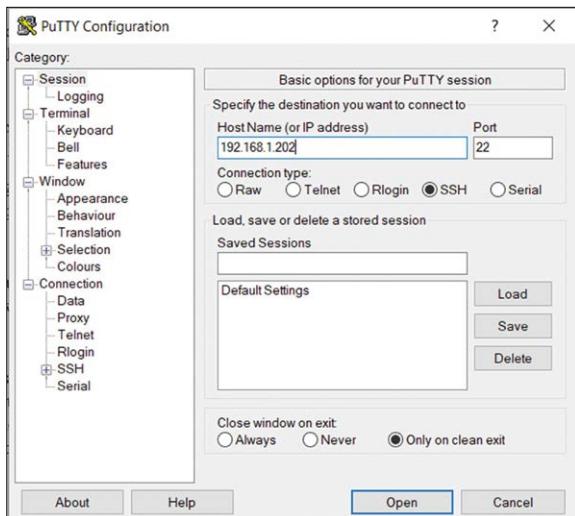


Figure 2.7: Putty startup screen.



Figure 2.8: Click Yes to accept.

You will be prompted to enter the username and password. Notice that the default username and password are:

username: **pi**
password: **raspberry**

You now have a terminal connection with the Raspberry Pi and you can type in commands, including the **sudo** privileged administrative commands.

To change your password, enter the following command:

`passwd`

You can use the cursor keys to scroll up and down through the commands you've previously entered in the same session. You can also run programs although not graphical programs.

2.5.1 Configuring the 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. The steps to configure the Putty with these settings are given below. Notice that in this example these settings are saved with the name **RPI4** so that they can be recalled whenever the Putty is restarted:

- Restart Putty.
- Select **SSH** and enter the Raspberry Pi IP address.
- Click **Colours** under **Window**.

- Set the **Default Foreground** and **Default Bold Foreground** colours 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 Colour** 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. RPI4) and click **Save**.
- Click **Open** to open the Putty session with the saved configuration.
- Next time you restart the Putty, select the saved session and click **Load** followed by **Open** to start a session with the saved configuration

2.6 Remote access of the Desktop

You can control your Raspberry Pi via Putty, and run programs on it from your Windows PC. This however will not work with graphical programs because Windows doesn't know how to represent the display. As a result of this, for example, we cannot run any graphical programs in the Desktop mode. We can get round this problem using some extra software. Two popular software utilities used for this purpose are: VNC (Virtual Network Connection), and Xming. Here, we shall be learning how to use the VNC.

Installing and using VNC

VNC consists of two parts: VNC Server and the VNC Viewer. VNC Server runs on the Raspberry Pi, and the VNC Viewer runs on the PC. VNC server is already installed on your Raspberry Pi and is enabled as described in Section 2.3 using **raspi-config**.

The steps to install and use the VNC Viewer onto your PC are given below:

- There are many VNC Viewers available, but the recommended one is the TightVNC which can be downloaded from the following website:
<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.
- Enter the following command:

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

- Start the **TightVNC Viewer** on your PC and enter the Raspberry Pi IP address (see Figure 2.9) followed by :1. Click **Connect** to connect to your Raspberry Pi.

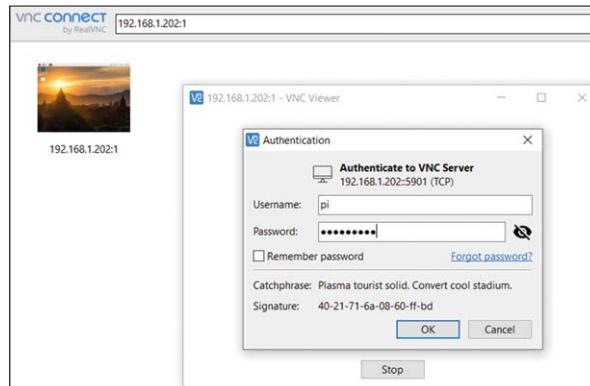


Figure 2.9: Start the TightVNC and enter the IP address.

Figure 2.10 shows the Raspberry Pi Desktop displayed on the PC screen.

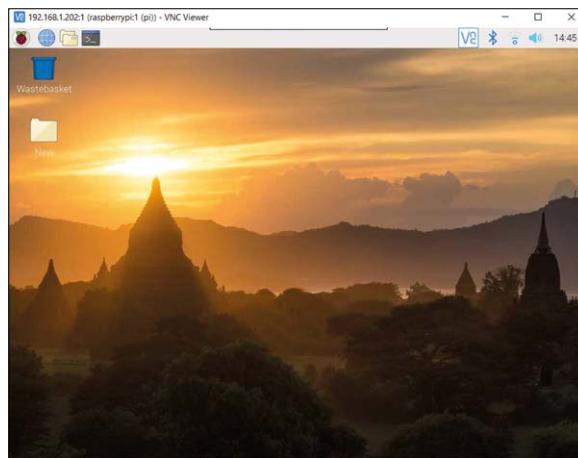


Figure 2.10: Raspberry Pi Desktop on the PC screen.

2.7 Static IP address

When we are using the Raspberry Pi with the Wi-Fi router, the IP address is automatically allocated by the router. It is possible that every time we start the Raspberry Pi, the Wi-Fi router will give the Pi another IP address and this makes it difficult to log in as we have to find the new IP address before we log in.

We can give our Raspberry Pi a static IP address so that every time it starts the same IP address is allocated from the Wi-Fi router. The IP address is given by the DHCP protocol running on the Wi-Fi router.

Before setting a static IP address we have to decide what this address will be, plus we have to make sure that no other devices in our network use this address. We can check this by either logging in to our Wi-Fi router or by displaying the devices on our network using an app on our smartphone.

The steps to assign a static IP address are as follows:

- First, check that dhcpcd service is active by entering the following command:

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

You should see the text **active: (running)** displayed as shown in Figure 2.11 (only part of the display is shown). Enter **Cntrl+C** to exit from the display.

```
pi@raspberrypi:~ $ sudo service dhcpcd status
● dhcpcd.service - dhcpcd on all interfaces
  Loaded: loaded (/lib/systemd/system/dhcpcd.service; enabled; vendor pres
  Active: active (running) since Thu 2020-06-18 23:06:12 BST; 2 weeks 5 da
    Process: 375 ExecStart=/usr/lib/dhcpcd5/dhcpcd -q -b (code=exited, status
   Main PID: 416 (dhcpcd)
     Tasks: 2 (limit: 4035)
    Memory: 4.5M
```

Figure 2.11: Check DHCP running.

- If dhcpcd is not running, enter the following commands to activate it:

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

- Now, we need to find the IP address (Default Gateway) and the Domain Name Server address of our router. This can easily be obtained either from our Wi-Fi router, or from our PC. The steps to obtain these addresses from the PC are:

- Go to **Control Panel** on your Windows 10 PC.
- Click **Network and Sharing Centre**.
- Click **Internet** as shown in Figure 2.12.

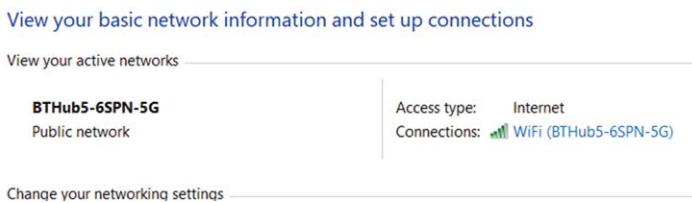


Figure 2.12: Click: Internet.

- Click **Details**. You will see a screen similar to the one shown in Figure 2.13 where you can see the Default Gateway and DNS server addresses. In this example they are both: 192.168.1.254.

DHCP Enabled	Yes
IPv4 Address	192.168.1.199
IPv4 Subnet Mask	255.255.255.0
Lease Obtained	08 July 2020 08:10:45
Lease Expires	09 July 2020 12:27:53
IPv4 Default Gateway	192.168.1.254
IPv4 DHCP Server	192.168.1.254
IPv4 DNS Server	192.168.1.254

Figure 2.13: Click: Details.

- Now, you will have to edit the following file: **/etc/dhcpcd.conf** using a text editor such as **nano**. Although you may not be familiar with **nano** yet, just follow the instructions given here (**nano** is described in a later Chapter).

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

- Go to the end of the file using the down arrow key and enter the following lines:

```
interface wlan0
static ip_address=192.168.1.120/24
static routers=192.168.1.254
static domain_name_servers=192.168.1.254

interface eth0
static ip_address=192.168.1.120/24
static routers=192.168.1.254
static domain_name_servers=192.168.1.254
```

In this example, we have chosen the static IP address to be 192.168.1.120 after making sure that there are no other devices on our network with the same IP address. The suffix **/24** is an abbreviation of the subnet mask 255.255.255.0 and you have to make sure that you only change the last digit of the IP address. i.e. choose an address in the form 192.168.1.x. **wlan0** is for the Wi-Fi link, and **eth0** is for the Ethernet link.

- Now, save the file by entering **Cntrl+X**, followed by **Y**.
- Display the file on your screen to make sure that the changes you have made are correct. Enter the command:

```
pi@raspberrypi:~ $ cat /etc/dhcpcd.conf
```

- Reboot your Raspberry Pi and it should come up with the IP address set as required.

2.8 Summary

In this Chapter we have learned how to install the latest Raspberry Pi operating system to a SD card, as well as how to start using the Raspberry Pi remotely. The instructions given here are applicable to all versions of the Raspberry Pi. Additionally, setting static IP address for your Raspberry Pi was discussed.

In the next Chapter we will be looking at some of the useful Raspberry Pi commands.

Chapter 3 • Using the Command Line

3.1 Overview

Raspberry Pi is based on a version of the Linux operating system. Linux is one of the most popular operating systems in use today. Linux is very similar to other operating systems, such as Windows and UNIX. Linux is an Open operating system based on UNIX, and has been developed collaboratively by many companies since 1991. In general, Linux is harder to manage than some other operating systems like Windows, but offers more flexibility and configuration options. There are several popular versions of the Linux operating system such as *Debian*, *Ubuntu*, *Red Hat*, *Fedora* and so on.

Linux command line instructions are text-based. In this Chapter we look at some of the useful Linux commands and see how you can manage your Raspberry Pi using these commands. Raspberry Pi 4 is used in the examples in this Chapter, but the notes are applicable to all versions of the Raspberry Pi.

3.2 The command prompt

After you login to Raspberry Pi, you see the following prompt displayed where the system waits for you to enter a command:

```
pi@raspberrypi ~$
```

Here, **pi** is the name of the user who is logged in.

raspberrypi is the name of the computer, used to identify it when connecting over the network.

- ~ character indicates that you are currently in your default directory.
- \$ character indicates that you are a normal user (not a privileged super-user)

3.3 Useful Linux commands

In this section we shall be looking at some of the useful Linux commands where examples will be given for each command. **In this book, commands entered by the user are shown in bold for clarity.** Also, it is important to remind you that all the commands must be terminated by the Enter key.

3.3.1 System and user information

These commands are useful as they tell us information about the system. Command **cat /proc/cpuinfo** displays information about the processor (command **cat** displays the contents of a file. In this example, the contents of file **/proc/cpuinfo** is displayed). Since there are 4 cores in the Raspberry Pi 4, the display is in four sections. Figure 3.1 shows an example display, where only part of the display is shown here.

```
pi@raspberrypi:~ $ cat /proc/cpuinfo
processor       : 0
model name     : ARMv7 Processor rev 3 (v7l)
BogoMIPS        : 108.00
Features        : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
vfpd32 lpaes evtstrm crc32
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xd08
CPU revision   : 3

processor      : 1
```

Figure 3.1: Command: `cat /proc/cpuinfo`.

Command **uname -s** displays the operating system Kernel name, which is Linux. Command **uname -a** displays complete detailed information about the Kernel and the operating system. An example is shown in Figure 3.2.

```
pi@raspberrypi:~ $ uname -s
Linux
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.19.57-v7l+ #1244 SMP Thu Jul 4 18:48:07
Linux
pi@raspberrypi:~ $ █
```

Figure 3.2: Commands: `uname -s` and `uname -a`.

Command **cat /proc/meminfo** displays information about the memory on your Pi. Information such as the total memory and free memory at the time of issuing the command are displayed. Figure 3.3 shows an example, where only part of the display is shown here.

```
pi@raspberrypi:~ $ cat /proc/meminfo
MemTotal:       945512 kB
MemFree:        677372 kB
MemAvailable:   796884 kB
Buffers:         22384 kB
Cached:          145152 kB
SwapCached:      0 kB
Active:          134172 kB
Inactive:        100596 kB
Active(anon):   67500 kB
Inactive(anon):  7480 kB
Active(file):   66672 kB
Inactive(file): 93116 kB
Unevictable:     0 kB
Mlocked:         0 kB
SwapTotal:      102396 kB
SwapFree:        102396 kB
Dirty:            4 kB
Writeback:        0 kB
AnonPages:       67240 kB
Mapped:          76040 kB
Shmem:           7744 kB
Slab:            19528 kB
SReclaimable:   9560 kB
```

Figure 3.3: Command: `cat /proc/meminfo`.

Command **whoami** displays the name of the current user. Figure 3.4 shows that the current user is pi.

```
pi@raspberrypi:~ $ whoami
pi
pi@raspberrypi:~ $ █
```

Figure 3.4: Command: whoami.

A new user can be added to our Raspberry Pi using the command **useradd**. In the example in Figure 3.5, user called **John** is added. A password for the new user can be added using the **passwd** command followed by the username. In Figure 3.5, the password for user **John** is set to **mypassword** (not displayed for security reasons). Notice that both the **useradd** and **passwd** are privileged commands and the keyword **sudo** must be entered before these commands. Notice that the **-m** option creates a home directory for the new user.

```
pi@raspberrypi:~ $ sudo useradd -m John
pi@raspberrypi:~ $ sudo passwd John
New password:
Retype new password:
passwd: password updated successfully
pi@raspberrypi:~ $ █
```

Figure 3.5 Commands: useradd and passwd.

We can log in to the new user account by specifying the username and the password as shown in Figure 3.6. You can type command **exit** to logout from the new account.

```
pi@raspberrypi:~ $ su John
Password:
John@raspberrypi:/home/pi $ _
```

Figure 3.6: Logging in to a new account.

3.3.2 The Raspberry Pi directory structure

The Raspberry Pi directory structure consists of a single root directory, with directories and subdirectories under the root. Different types of operating system programs and application programs are stored in different directories and subdirectories.

Figure 3.7 shows part of the Raspberry Pi directory structure. Notice that the root directory is identified by the ' / ' (slash) symbol. Under the root we have directories named such as bin, boot, dev, etc, home, lib, lost+found, media, mnt, opt, proc, and many more. The important directory as far as the users are concerned is the **home** directory. The **home** directory contains subdirectories for each user of the system. In the example in Figure 3.7, **pi** is the subdirectory for user **pi**.

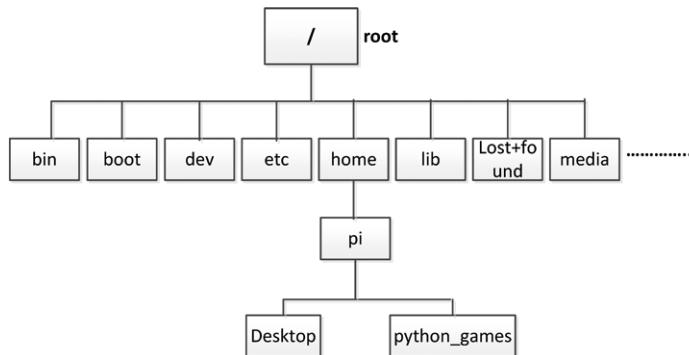


Figure 3.7: Raspberry Pi directory structure (only part of it is shown).

Some useful directory commands are given below. Command **pwd** displays the user home directory:

```
pi@raspberrypi ~$ pwd
/home/pi
pi@raspberrypi ~$
```

To show the directory structure, enter the command **ls /**:

```
pi@raspberrypi ~$ ls /
bin  dev  home  lost+found  mnt  proc  run  srv  var
boot  etc  lib   media      opt  root  sbin  sys  usr
pi@raspberrypi ~$
```

To show the subdirectories and files in our working directory, enter **ls** (the files in your directory may be different):

```
pi@raspberrypi ~$ ls
```

Desktop ocr_pi.png python_games

```
pi@raspberrypi ~$
```

Notice that the subdirectories are displayed in blue colour. There are two subdirectories called **Desktop** and **python_games**. There is also a file in the working directory called **ocr_pi.png** which is displayed in pink colour.

The **ls** command can take a number of arguments. Some examples are given below.

To display the subdirectories and files in a single row:

```
pi@raspberrypi ~$ ls -1
Desktop
```

```
myfiles
ocr_pi.png
python_games
pi@raspberrypi ~$
```

to display the file type, enter the following command. Note that directories have a "/" after their names, and executable files have a "*" character after their names:

```
pi@raspberrypi ~$ ls -F
Desktop/ myfiles/ ocr_pi.png python_games/
pi@raspberrypi ~$
```

To list the results, separated by commas:

```
pi@raspberrypi ~$ ls -m
Desktop, myfiles, ocr_pi.png, python_games
pi@raspberrypi ~$
```

we can mix the arguments as in the following example:

```
pi@raspberrypi ~$ ls -m -F
Desktop/, myfiles/, ocr_pi.png, python_games/
pi@raspberrypi ~$
```

Subdirectories are created using command **mkdir** followed by the name of the subdirectory. In the following example, subdirectory **myfiles** is created in our working directory:

```
pi@raspberrypi ~$ mkdir myfiles
pi@raspberrypi ~$ ls
Desktop myfiles ocr_pi.png python_games
pi@raspberrypi ~$
```

File permissions

One of the important arguments used with the **ls** command is **-l** (lower case letter l) which displays the file permissions, file sizes, and when they were last modified. In the example below, each line relates to one directory or file. Reading from right to left, the name of the directory or the file is on the right-hand side. The date the directory or file was created is on the left-hand side of its name. Next comes the size, given in bytes. For example, in Figure 3.8, file **mytestfile.txt** consists of 23 bytes. The characters at the beginning of each line are about the permissions. i.e. who is allowed to use or modify the file or the directory.

The permissions are divided into 3 categories:

- What the user (or owner, or creator) can do – called USER
- What the group owner (people in the same group) can do - GROUP
- What everyone else can do – called WORLD

The first word **pi** in the example in Figure 3.8 shows who the user of the file (or directory) is, and the second word **pi** shows the group name that owns the file. In this example, both the user and the group names are **pi**.

```
drwxr-xr-x  2 pi pi 4096 Aug 16 01:41 Music  -
-rw-r--r--  1 pi pi   23 Sep 23 12:55 mytestfile.txt
drwxr-xr-x  2 pi pi 4096 Aug 16 01:41 Pictures
drwxr-xr-x  2 pi pi 4096 Aug 16 01:41 Public
drwxr-xr-x  2 pi pi 4096 Aug 16 01:11 python_games
```

Figure 3.8: File permissions example.

The permissions can be analyzed by breaking down the characters into four chunks for: File type, User, Group, World. The first character for a file is '-' and for a directory it is 'd'. Next comes the permissions for the User, Group and World. The permissions are as follows:

- Read permission (r): the permission to open and read a file or to list a directory.
- Write permission (w): the permission to modify a file, or to delete or create a file in a directory.
- Execute permission (x): the permission to execute the file (applies to executable files), or to enter a directory.

The three letters **rwx** are used as a group and if there is no permission assigned then a '-' character is used.

As an example, considering the **Music** directory, we have the following permission codes:

drwxr-xr-x which translates to:

d: it is a directory
rwx: user (owner) can read, write, and execute
r-x: group can read and execute, but cannot write (e.g. create or delete)
r-x: world (everyone else) can read and execute, but cannot write

as another example, let's look at the permissions for file **mytestfile.txt**:

-rw-r--r-- which translates to:

--: it is a file;
rw-: user (owner) can read and write, but cannot execute (this is not an executable file);
r--: group can only read it, they cannot modify, delete, or execute the file;
r--: everyone else (world) can only read it, they cannot modify, delete, or execute the file.

The **chmod** command is used to change the file permissions. Before going into details of how to change the permissions, let us look and see what arguments are available in **chmod** for changing the file permissions.

The available arguments for changing file permissions are given below. We can use these arguments to add/remove permissions or to explicitly set permissions. It is important to realize that if we explicitly set permissions then any unspecified permissions in the command will be revoked:

u:	user (or owner)
g:	group
o:	other (world)
a:	all
+:	add
-:	remove
=:	set
r:	read
w:	write
x:	execute

To change the permissions of a file we type the **chmod** command, followed by one of the letters **u**, **g**, **o**, or **a** to select the people, followed by the **+** or **=** to select the type of change, and finally followed by the filename. An example is given below. In this example, file **mytestfile.txt** has the user read and write permissions. We will be changing the permissions so that the user does not have read permission on this file:

```
pi@raspberrypi ~$ chmod u-r mytestfile.txt
pi@raspberrypi ~$ ls -lh
```

The result is shown in Figure 3.9.

```
drwxr-xr-x  2 pi pi 4.0K Aug 16 01:41 Music
--w-r--r--  1 pi pi  23 Sep 23 12:55 mytestfile.txt
drwxr-xr-x  2 pi pi 4.0K Aug 16 01:41 Pictures
drwxr-xr-x  2 pi pi 4.0K Aug 16 01:41 Public
drwxr-xr-x  2 pi pi 4.0K Aug 16 01:11 python games
```

Figure 3.9: File permissions of **mytestfile.txt**.

Notice that if we now try to display the contents of file **mytestfile.txt** using the **cat** command we will get an error message:

```
pi@raspberrypi ~$ cat mytestfile.txt
cat: lin.dat: permission denied
pi@raspberrypi ~$
```

All the permissions can be removed from a file by the following command:

```
pi@raspberrypi ~$ chmod a= mytestfile.txt
```

Figure 3.10 shows the new permissions of file **mytestfile.txt**.

```
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Music -  
----- 1 pi pi 23 Sep 23 12:55 mytestfile.txt  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Pictures  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Public  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:11 python_games
```

Figure 3.10: New permissions of file mytestfile.txt.

In the following example, **rwx** user permissions are given to file **mytestfile.txt**:

```
pi@raspberrypi ~$ chmod u+rwx mytestfile.txt
```

Figure 3.11 shows the new permissions of file mytestfile.txt.

```
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Music -  
-rwx----- 1 pi pi 23 Sep 23 12:55 mytestfile.txt  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Pictures  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:41 Public  
drwxr-xr-x 2 pi pi 4.0K Aug 16 01:11 python_games
```

Figure 3.11: New permissions of file mytestfile.txt.

To change our working directory, the command **cd** is used. In the following example we change our working directory to **Music**:

```
pi@raspberrypi ~$ cd /home/pi/Music  
pi@raspberrypi ~/Music $
```

to go up one directory level, i.e. to our default working directory:

```
pi@raspberrypi ~/Music $ cd..  
pi@raspberrypi ~$
```

to change our working directory to **Music**, we can also enter the command:

```
pi@raspberrypi ~$ cd ~/Music  
pi@raspberrypi ~/myfiles $
```

to go back to the default working directory, we can enter:

```
pi@raspberrypi ~/Music $ cd ~  
pi@raspberrypi ~$
```

to find out more information about a file we can use the file command. For example:

```
pi@raspberrypi ~$ file mytestfile.txt
mytestfile.txt: ASCII text
pi@raspberrypi ~$
```

the **-R** argument of command **ls** lists all the files in all the subdirectories of the current working directory. An example is given below. Notice here in Figure 3.12 there are no files in subdirectory **Music** (notice that your listing may be different). Only part of the display is shown here.

```
./Downloads:
./Music:
./Pictures:
./Public:
./python_games:
4row_arrow.png      gem4.png      pentomino.py
4row_black.png      gem5.png      pinkgirl.png
4row_board.png      gem6.png      Plain_Block.png
4row_computerwinner.png  gem7.png  princess.png
4row_humanwinner.png  gemgem.py  RedSelector.png
4row_red.png        grass1.png  Rock.png
4row_tie.png        grass2.png  Selector.png
```

Figure 3.12: Command: ls -R.

to display information on how to use a command, we can use the **man** command. As an example, to get help on using the **mkdir** command:

```
pi@raspberrypi ~$ man mkdir
MKDIR(1)

NAME
    Mkdir – make directories

SYNOPSIS
    Mkdir [OPTION]... DIRECTORY...

DESCRIPTION
    Create the DIRECTORY(ies), if they do not already exist.

    Mandatory arguments to long options are mandatory for short options

    -m, --mode=MODE
        Set file mode (as in chmod), not a=rwx – umask
-----
-----
```

Enter Cntrl+Z to exit from the man display.

Help

The **man** command usually gives several pages of information on how to use a command. We can type **q** to exit the **man** command and return to the operating system prompt.

The less command can be used to display a long listing one page at a time. Using the up and down arrow keys we can move between pages. An example is given below. Type **q** to exit:

```
pi@raspberrypi ~$ man ls | less
<display of help on using the ls command>
pi@raspberrypi ~$
```

Date, Time, and Calendar

To display the current date and time the **date** command is used. Similarly, the **cal** command displays the current calendar. Figure 3.13 shows an example.

```
pi@raspberrypi:~ $ date
Wed 8 Jul 16:49:00 BST 2020
pi@raspberrypi:~ $ cal
      July 2020
Su Mo Tu We Th Fr Sa
      1  2  3  4
  5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
pi@raspberrypi:~ $ █
```

Figure 3.13: Commands: **date** and **cal**.

Copying a file

To make copy of a file, use the command **cp**. In the following example, a copy of file **mytestfile.txt** is made and the new file is given the name **test.txt**:

```
pi@raspberrypi ~$ cp mytestfile.txt test.txt
pi@raspberrypi ~$
```

Wildcards

We can use wildcard characters to select multiple files with similar characteristics. e.g. files having the same file-extension names. The '*' character is used to match any number of characters. Similarly, the '?' character is used to match any single character. In the example below all the files with extensions **.png** are listed:

```
pi@raspberrypi ~$ ls *.txt
mytestfile.txt  test.txt
pi@raspberrypi ~$
```

The wildcard characters [a-z] can be used to match any single character in the specified character range. An example is given below which matches any files that start with letters o, p, q, r, s, and t, and with the '.txt' extension:

```
pi@raspberrypi ~$ ls [o-t]*.txt  
test.txt  
pi@raspberrypi ~$
```

Renaming a file

you can rename a file using the **mv** command. In the example below, the name of file **test.txt** is changed to **test2.txt**:

```
pi@raspberrypi ~$ mv test.txt test2.txt  
pi@raspberrypi ~$
```

Deleting a file

The command **rm** can be used to remove (delete) a file. In the example below file **test2.txt** is deleted:

```
pi@raspberrypi ~$ rm test2.txt  
pi@raspberrypi ~$
```

the argument **-v** can be used to display a message when a file is removed. Also, the **-i** argument asks for confirmation before a file is removed. In general, the two arguments are used together as **-vi**. An example is given below:

```
pi@raspberrypi ~$ rm -vi test2.txt  
rm: remove regular file 'test2.txt'? y  
removed 'test2.txt'  
pi@raspberrypi ~$
```

Removing a directory

A directory can be removed using the **rmdir** (or **rm**) command:

```
pi@raspberrypi ~$ rmdir Music  
pi@raspberrypi ~$
```

We may also want to delete a directory and all the files and sub-directories inside this directory. The following command works down the Music directory tree and deletes all the files and sub-directories inside this directory:

```
pi@raspberrypi:~ $ rm -Rf Music  
pi@raspberrypi:~ $
```

Redirecting the output

The 'greater than' sign **>** can be used to re-direct the output of a command to a file. For example, we can re-direct the output of the **ls** command to a file called **ltest.txt**:

```
pi@raspberrypi ~$ ls > ltest.txt  
pi@raspberrypi ~$
```

The **cat** command can be used to display the contents of a file:

```
pi@raspberrypi ~$ cat mytestfile.txt
This is a file
This is line 2
pi@raspberrypi ~$
```

Using two 'greater than' signs **>>** adds to the end of a file. An example is given in Figure 3.14.

```
pi@raspberrypi:~ $ cat mytestfile.txt
this is a file
line 2

pi@raspberrypi:~ $ date >> mytestfile.txt
pi@raspberrypi:~ $ cat mytestfile.txt
this is a file
line 2

Wed  8 Jul 16:55:53 BST 2020
pi@raspberrypi:~ $ █
```

Figure 3.14: Re-directing the output.

Writing to the screen or to a file

The **echo** command can be used to write to the screen. It can be used to perform simple mathematical operations if the numbers and the operation are enclosed in two brackets, preceded by a \$ character:

```
pi@raspberrypi ~$ echo $((5*6))
30
pi@raspberrypi ~$
```

The echo command can also be used to write a line of text to a file. An example is shown below:

```
pi@raspberrypi ~$ echo a line of text > lin.dat
pi@raspberrypi ~$ cat lin.dat
a line of text
pi@raspberrypi ~$
```

Matching a string

The 'grep' command can be used to match a string in a file. An example is given below assuming that the file lin.dat contains sting a line of text. Notice that the matched word is shown in bold:

```
pi@raspberrypi ~$ grep line lin.dat
a line of text
pi@raspberrypi ~$
```

Head and Tail commands

The 'head' command can be used to display the first 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi ~$ head mytestfile.txt  
.....  
.....  
pi@raspberrypi ~$
```

Similarly, the 'tail' command is used to display the last 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi ~$ tail mytestfile.txt  
.....  
.....  
pi@raspberrypi ~$
```

Super user commands

Some of the commands are privileged and only the authorized persons can use them. Inserting the word **sudo** at the beginning of a command gives us the authority to use the command without having to login as an authorized user.

What software is installed on my Raspberry Pi?

To find out what software is installed on your Raspberry Pi, enter the following command. You should get several pages of display:

```
pi@raspberrypi ~$ dpkg -l  
.....  
.....  
pi@raspberrypi ~$
```

We can also find out if a certain software package is already installed on our computer. An example is given below, which checks whether or not software called **xpdf** (PDF reader) is installed. In this example **xpdf** is installed and the details of this software are displayed:

```
pi@raspberrypi ~$ dpkg --s xpdf  
Package: xpdf  
Status: install ok installed  
Priority: optional  
Section: text  
Installed-Size: 395  
.....  
.....  
pi@raspberrypi ~$
```

If the software is not installed, we get a message similar to the following (assuming we are checking to see if a software package called **bbgd** is installed):

```
pi@raspberrypi ~$ dpkg -s bbgd
dpkg-query: package 'bbgd' is not installed and no information is available
.....
.....
pi@raspberrypi ~$
```

Updating the cache

You should normally update the cache before installing a new software. The following command is used to update the cache:

```
pi@raspberrypi:~ $ sudo apt-get update
```

Installing and removing software packages

The command **sudo apt-get install <package name>** installs the specified package. In the following example, the package called **lcd** is installed:

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

To update all the software on your Raspberry Pi enter the following commands:

```
pi@raspberrypi:~ $ sudo apt-get update
pi@raspberrypi:~ $ sudo apt-get upgrade
pi@raspberrypi:~ $
```

To remove an installed software package, use the command **sudo apt-get remove <package name>**. In the following example, the package named **lcd** is removed:

```
pi@raspberrypi:~ $ sudo apt-get remove lcd
pi@raspberrypi:~ $
```

Removing a software package may leave behind some user files and configuration files. As an example, to remove such files after removing the package **lcd**, enter the following command:

```
pi@raspberrypi:~ $ sudo apt-get purge lcd
pi@raspberrypi:~ $
```

Additionally, when a package is installed it can also install some other packages dependent on it and such packages are not removed when the original package is removed. Use the following command to remove the packages that are no longer required:

```
pi@raspberrypi:~ $ sudo apt-get autoremove  
pi@raspberrypi:~ $
```

Once a package is installed, the installation file is not required anymore, but it is saved in the archives directory (**/var/cache/apt/archives**). The following command can be entered to delete such files and free up SD card space:

```
pi@raspberrypi:~ $ sudo apt-get clean  
pi@raspberrypi:~ $
```

Finding an installed package

The following command lists an index of all the software packages with the name 'game'.

```
pi@raspberrypi:~ $ sudo apt-cache search game
```

The list is very long, and we should use the **less** option to list a page at a time (enter **Cn-trl+Z** to exit):

```
pi@raspberrypi:~ $ sudo apt-cache search game | less
```

Screenshots

Capturing screen shots can be very useful. The command **scrot** is used to perform a screenshot. The following command captures the screen and saves in a file called **myscreen.png**. Option **-d n** causes the program to wait for **n** seconds before taking the screen shot so that you can arrange the screen as you like.

```
pi@raspberrypi:~ $ scrot -d 3 myscreen.png
```

The **-s** option is very useful as it allows you to click on the desired window, or to draw a rectangle with the mouse to define the capture area:

```
pi@raspberrypi:~ $ scrot -s -d 3 myscreen.png
```

You can also add a down counter using the **-c** option together with the **-d** option:

```
pi@raspberrypi:~ $ scrot -c -d 3 myscreen.png
```

3.3.3 Resource monitoring on Raspberry Pi

System monitoring is an important topic for managing usage of your Raspberry Pi. One of the most useful system monitoring commands is the top, which displays the current usage of system resources and displays which processes are running and how much memory and CPU time they are consuming.

Figure 3.15 shows a typical system resource display obtained by entering the following command (Enter **Ctrl+Z** to exit):

```
pi@raspberrypi ~$ top  
pi@raspberrypi ~$
```

```
top - 16:59:13 up 47 min, 2 users, load average: 0.21, 0.21, 0.18
Tasks: 118 total, 1 running, 117 sleeping, 0 stopped, 0 zombie
%Cpu(s):  0.2 us,  0.4 sy,  0.0 ni, 99.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem: 1939.5 total, 1728.7 free, 69.7 used, 141.1 buff/cache
MiB Swap: 100.0 total, 100.0 free, 0.0 used. 1784.5 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1897	pi	20	0	10188	2960	2600	R	0.7	0.1	0:00.06	top
108	root	20	0	21208	7492	6604	S	0.3	0.4	0:02.21	systemd-j+
1	root	20	0	32644	7968	6424	S	0.0	0.4	0:03.48	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	20	0	0	0	0	I	0.0	0.0	0:00.14	kworker/0+
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu+0
9	root	20	0	0	0	0	S	0.0	0.0	0:00.07	ksoftirqd+0
10	root	20	0	0	0	0	I	0.0	0.0	0:00.33	rcu_sched
11	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_bh
12	root	rt	0	0	0	0	S	0.0	0.0	0:00.01	migration+
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
15	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration+
16	root	20	0	0	0	0	S	0.0	0.0	0:00.07	ksoftirqd+0

Figure 3.15: Typical system resource display.

Some of the important points in Figure 3.15 are summarized below (for lines 1 to 5 of the display):

- There are total of 118 processes in the system.
 - Currently only one process is running, 117 processes are sleeping, no processes are stopped or in zombie state.
 - The percentage CPU utilization is 0.2us for user applications (us).
 - The percentage CPU utilization for system applications is 0.4 (sy).
 - There are no processes requiring more or less priority (ni).
 - 99.4% of the time the CPU is idle (id).
 - There are no processes waiting for I/O completion (wa).
 - There are no processes waiting for hardware interrupts (hi).
 - There are no processes waiting for software interrupts (si).
 - There is no time reserved for a hypervisor (st).
 - The total usable memory is 1939.5 Mbytes, of which 1728.7 Mbytes is free, 69.7 Mbyte is in use, and 141.4 Mbytes are used by buffers/cache.
 - Line 5 displays the swap space usage.

The process table gives the following information for all the processes loaded to the system:

- PID: the process ID number.
 - USER: owner of the process.
 - PR: priority of the process.
 - NI: the nice value of the process.
 - VIRT: the amount of virtual memory used by the process.
 - RES: size of the resident memory.
 - SHR: shared memory the process is using.

- S: process status (sleeping, running,, zombie).
- %CPU: the percentage of CPU consumed.
- %MEM: percentage of RAM used.
- TIME+: total CPU time the task used.
- COMMAND: The actual name of the command.

The **ps** command can be used to list all the processes used by the current user. An example is shown in Figure 3.16.

```
pi@raspberrypi:~ $ ps
  PID TTY      TIME CMD
 1150 pts/0    00:00:00 bash
 1897 pts/0    00:00:01 top
 2007 pts/0    00:00:00 ps
pi@raspberrypi:~ $ █
```

Figure 3.16: Command: ps

the command **ps -ef** gives a lot more information about the processes running in the system.

Killing a process

There are many options for killing (or stopping; halting) a process. A process can be killed by specifying its PID and using the following command:

```
pi@raspberrypi ~$ kill -9 <PID>
```

Disk usage

The disk free command **df** can be used to display the disk usage statistics. An example is shown in Figure 3.17. option **-h** displays in human readable form. According to Figure 3.17, the disk (SD card) size is 14 GB, where 6 GB is used, 7.4 GB is available, i.e. 45% of the space is already used.

```
pi@raspberrypi:~ $ df -h
Filesystem  Size  Used  Avail  Use%  Mounted on
/dev/root   14G   6.0G  7.4G  45%  /
devtmpfs    841M   0    841M  0%   /dev
tmpfs       970M   0    970M  0%   /dev/shm
tmpfs       970M   8.5M  962M  1%   /run
tmpfs       5.0M   4.0K  5.0M  1%   /run/lock
tmpfs       970M   0    970M  0%   /sys/fs/cgroup
/dev/mmcblk0p1 253M   40M  214M  16%  /boot
tmpfs       194M   0    194M  0%   /run/user/1000
pi@raspberrypi:~ $ █
```

Figure 3.17: Command: df

3.3.4 Shutting down

Although you can disconnect the power supply from your Raspberry Pi when you finish working with it, it is not recommended since there are many processes running on the system and it is possible to corrupt the file system. It is much better to shut down the system in an orderly manner.

The following command will stop all the processes and make the file system safe and then turn off the system safely. Notice that shutting down or restarting the system requires super-user privilege:

```
pi@raspberrypi ~$ sudo halt
```

the following command stops and then re-starts the system:

```
pi@raspberrypi ~$ sudo reboot
```

the system can also be shut down and then re-started after a time by entering the following command. Optionally, a shutdown message can be displayed if desired:

```
pi@raspberrypi ~$ sudo shutdown -r <time> <message>
```

for a quick shutdown, enter the following command:

```
pi@raspberrypi ~$ sudo shutdown now
```

3.4 Summary

This Chapter has described the use of some of the important Linux commands. You should be able to get further information on each command and other Linux commands from the Internet and from other books on Raspberry Pi and Linux.

In the next Chapter we will briefly look at the various features of the Desktop.

CHAPTER 4 • A Quick Look at the Desktop

4.1 Overview

In the last Chapter we looked at some of the useful commands that can be entered from the command line. In this Chapter we will be looking at briefly the various features of the Desktop. Arguably, this will be a quick guide to Desktop as it is not our main topic in this book. Interested readers can get further details on the use of the Desktop from the Internet and from many other sources.

4.2 The Desktop

If you are in command mode, issue the following command:

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

Then, start the **VNCViewer** program on your PC. You should see the Desktop displayed as shown in Figure 4.1.

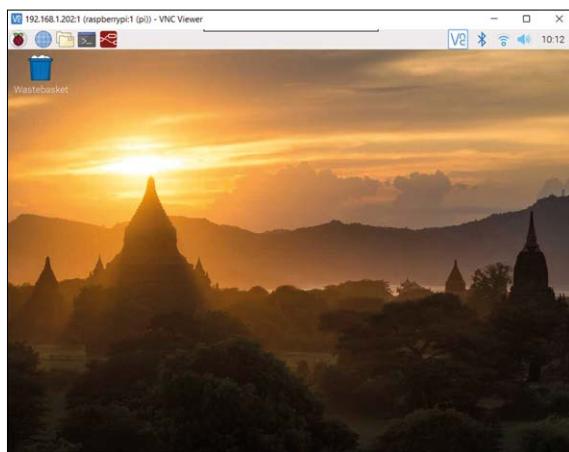


Figure 4.1: The Desktop.

On the top right-hand side of the Desktop you should see the Wi-Fi icon, Bluetooth icon, the volume control, and the current time.

You should see some icons at the top left-hand side of the screen with the following menu items:

Applications menu: This menu item has the following menu and sub-menu options (see Figure 4.2):

Programming: This menu includes various programming tools, such as:

BlueJ Java, Geany Programmer's Editor, Greenfoot Java IDE, Mathematica, mu, Node-RED, Scratch, Scratch 2, Sense HAT emu, Sonic Pi, Thonny Python IDE, Wolfram.

Education: This menu includes the SmartSim.

Office: This menu includes office software, such as

Libre Office Base, Libre Office Calc, Libre Office Draw, Libre Office Impress, Libre Office Math, Libre Office Writer.

Internet: This menu includes internet programs, such as Chromium Web Browser, Claws Mail, Minicom, VNC Viewer.

Sound & Video: This menu includes the VLC Media player.

Graphics: This menu includes the Image Viewer.

Games: This menu includes games, such as Minecraft Pi and Python Games.

System Tools: This menu item includes minicom.

Accessories: This menu item includes, such as Archiver, Calculator, File Manager, PDF Viewer, SD Card Copier, Task Manager, Terminal, Text Editor.

Help: This menu item includes help on topics such as Debian Reference, Get Started, Help, Projects, *The MagPi*.

Preferences: This menu item includes various configuration and settings, such as:

Add/Remove software, Appearance Settings, Audio Device Settings, Main Menu Editor, Mouse and Keyboard Settings, Raspberry Pi Configuration, Recommended Software, Screen Configuration.

Run: This menu item is used to run a command

Shutdown: This menu item is used to shut down the system.

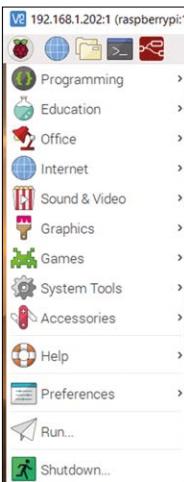


Figure 4.2: Applications menu options.

Web Browser menu: Used to start a web browser.

File Manager menu: Used to manage the files on the system.

Terminal menu: This menu item is used to enter command line commands to the system.

Node-RED: This menu item starts the Node-RED graphical software development tool

We will now look at some of the interesting and useful Desktop programs in more detail. Perhaps the easiest way to learn how to use these programs is to open them and play with them.

4.3 Libre Office Writer

This is a word processing software package similar to Windows Word which has many tools for writing and formatting documents. Figure 4.3 shows the startup screen.

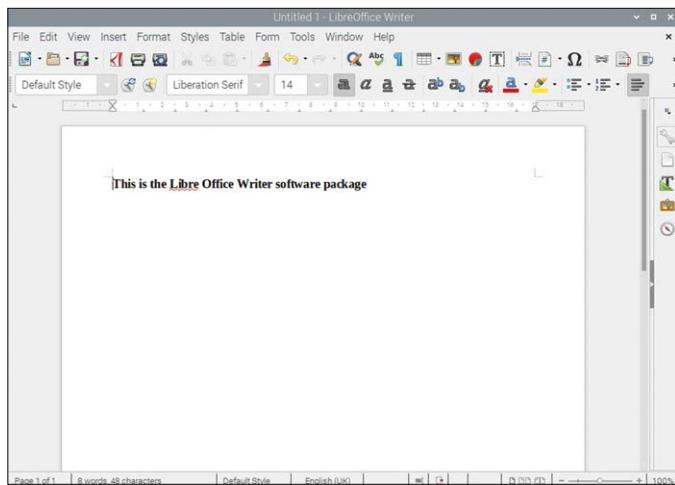


Figure 4.3: Libre Office Writer.

4.4 Libre Office Calc

This is a spreadsheet software package similar to Windows Excel. Figure 4.4 shows the startup screen.

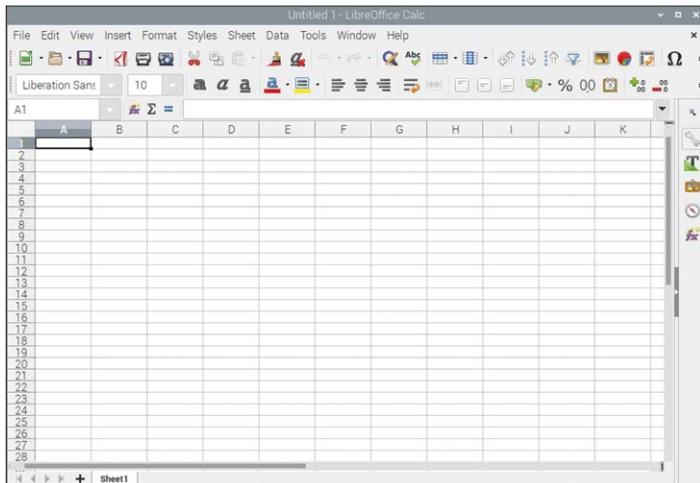


Figure 4.4: Libre Office Calc.

4.5 VLC media player

This is a media player software package that enables users play video files having many different formats. Figure 4.5 shows the startup screen.

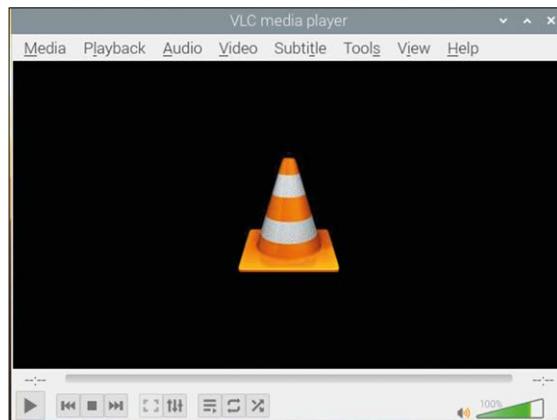


Figure 4.5: VLC media player.

4.6 Calculator

This is a simple calculator package like the Windows Calculator application, having basic and scientific functions. Figure 4.6 shows the startup screen.

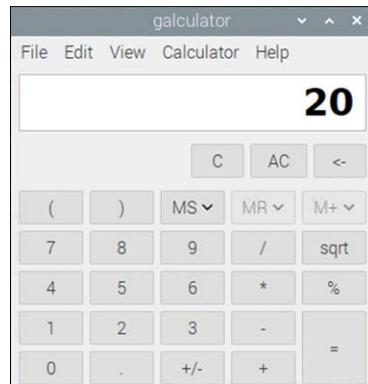


Figure 4.6: Calculator.

4.7 File Manager

This is one of the most useful programs in Desktop mode. The File Manager allows the user to manipulate the files on the SD card. Users can create, edit, delete, open, find, and sort files easily using this graphical tool with the help of the mouse. Figure 4.7 shows the startup screen.

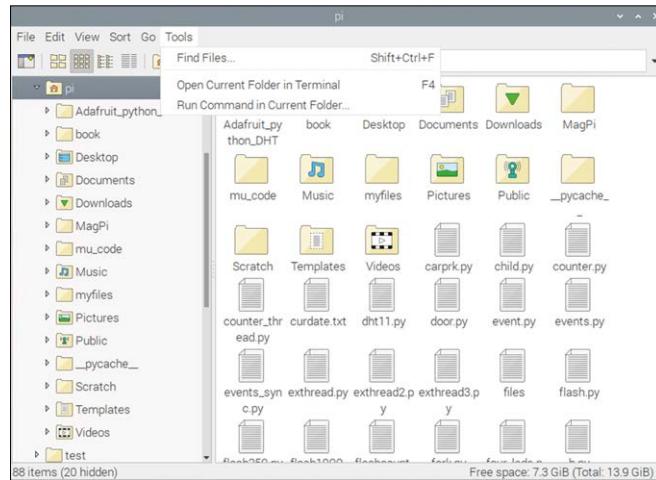


Figure 4.7: File Manager.

4.8 SD Card Copier

This is another very useful program. The SD Card Copier makes copies of the SD card where the operating system is loaded to. The nice thing about this program is that it can make copies while the SD card is in use, i.e. while the Raspberry Pi is running. You will need a blank SD card and a USB card reader/writer in order to make a copy. The USB card reader should be plugged in to one of the USB ports of your Raspberry Pi. Figure 4.8 shows the startup screen.

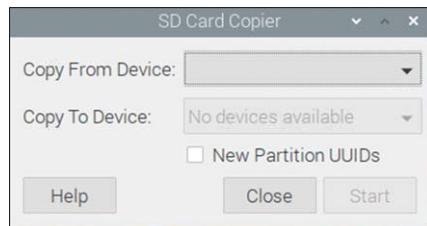


Figure 4.8: SD card copier.

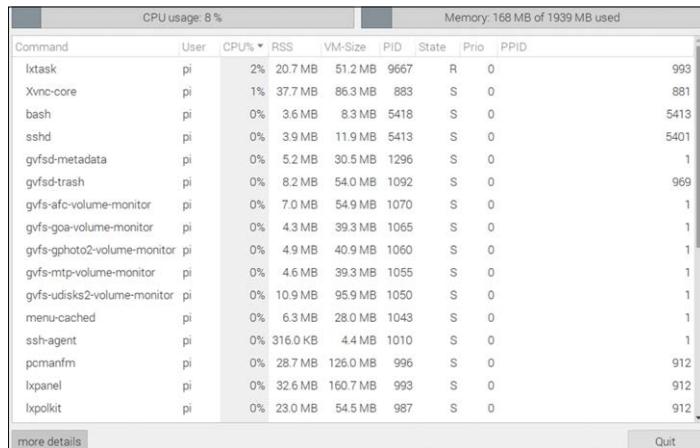
The steps to copy a Raspberry Pi 4 SD card are given below. Notice that the Raspberry Pi micro SD card is named as: **/dev/mmcblk0**. Any SD card plugged in to the USB ports takes the names **/dev/sdx** where **x** is **a** for the first card, etc.:

- Insert a blank micro SD card into a USB card reader.
- Plug-in the USB card reader/writer to one of the USB ports of the Raspberry Pi 4. In this example, it is plugged in to the top left USB port.
- Select the **Copy From Device** as: **/dev/mmcblk0** .
- Select the **Copy To Device** as: **/dev/sda** .

- Click **Start** to start copying.

4.9 Task Manager

The Task Manager shows a list of the tasks in the system, their users, percentage CPU usage of each task, memory usage, priority of the tasks, etc. Figure 4.9 shows an example display.



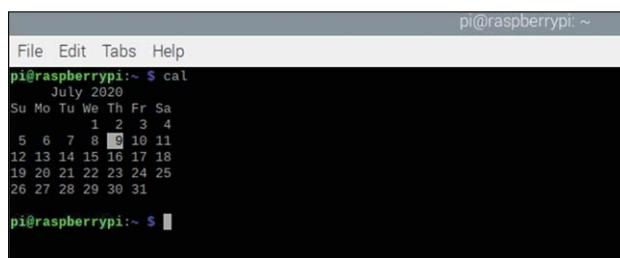
A screenshot of the Task Manager application window. The title bar says "CPU usage: 8 %". Below it, a status bar indicates "Memory: 168 MB of 1939 MB used". The main area is a table with the following columns: Command, User, CPU%, RSS, VM-Size, PID, State, Prio, and PPID. The table lists various system processes such as lxtask, Xvnc-core, bash, sshd, gvfsd-metadata, gvfsd-trash, gvfs-afc-volume-monitor, gvfs-goa-volume-monitor, gvfs-ghphoto2-volume-monitor, gvfs-ftp-volume-monitor, gvfs-udisks2-volume-monitor, menu-cached, ssh-agent, pcamnfm, lxfpanel, and lxpolkit. Most processes are run by user "pi". The table has scroll bars on the right and bottom. At the bottom left is a "more details" button, and at the bottom right is a "Quit" button.

Command	User	CPU%	RSS	VM-Size	PID	State	Prio	PPID
lxtask	pi	2%	20.7 MB	51.2 MB	9667	R	0	993
Xvnc-core	pi	1%	37.7 MB	86.3 MB	883	S	0	881
bash	pi	0%	3.6 MB	8.3 MB	5418	S	0	5413
sshd	pi	0%	3.9 MB	11.9 MB	5413	S	0	5401
gvfsd-metadata	pi	0%	5.2 MB	30.5 MB	1296	S	0	1
gvfsd-trash	pi	0%	8.2 MB	54.0 MB	1092	S	0	969
gvfs-afc-volume-monitor	pi	0%	7.0 MB	54.9 MB	1070	S	0	1
gvfs-goa-volume-monitor	pi	0%	4.3 MB	39.3 MB	1065	S	0	1
gvfs-ghphoto2-volume-monitor	pi	0%	4.9 MB	40.9 MB	1060	S	0	1
gvfs-ftp-volume-monitor	pi	0%	4.6 MB	39.3 MB	1055	S	0	1
gvfs-udisks2-volume-monitor	pi	0%	10.9 MB	95.9 MB	1050	S	0	1
menu-cached	pi	0%	6.3 MB	28.0 MB	1043	S	0	1
ssh-agent	pi	0%	316.0 KB	4.4 MB	1010	S	0	1
pcmanfm	pi	0%	28.7 MB	126.0 MB	996	S	0	912
lxfpanel	pi	0%	32.6 MB	160.7 MB	993	S	0	912
lxpolkit	pi	0%	23.0 MB	54.5 MB	987	S	0	912

Figure 4.9: Task Manager.

4.10 Terminal

This is the command-line terminal where you can enter commands as if you are in command mode. Figure 4.10 shows the screen where the command **cal** is entered.



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window contains the following text:

```
File Edit Tabs Help
pi@raspberrypi:~ $ cal
July 2020
Su Mo Tu We Th Fr Sa
      1  2  3  4
 5  6  7  8 | 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
pi@raspberrypi:~ $
```

Figure 4.10: Terminal.

4.11 Help

This menu option opens an Internet website where help on various Raspberry Pi features can be obtained. Figure 4.11 shows an example display.

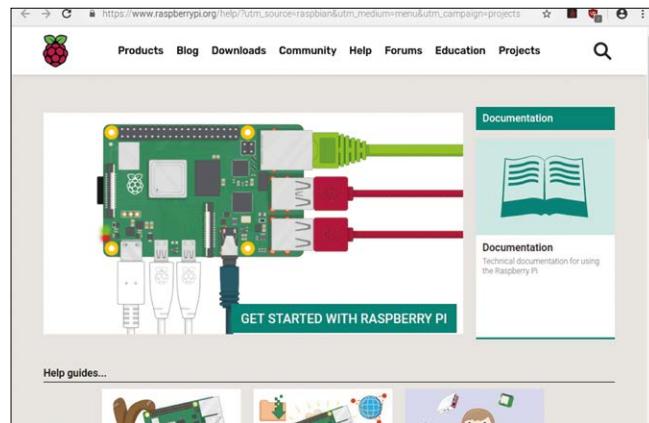


Figure 4.11: Help.

4.12 Add/Remove software

This tool is very useful as it enables you to add (install) or remove an installed software package from the system very easily. The user can select the type of software required to be added/removed. Figure 4.12 shows an example display where the admin tools was selected.

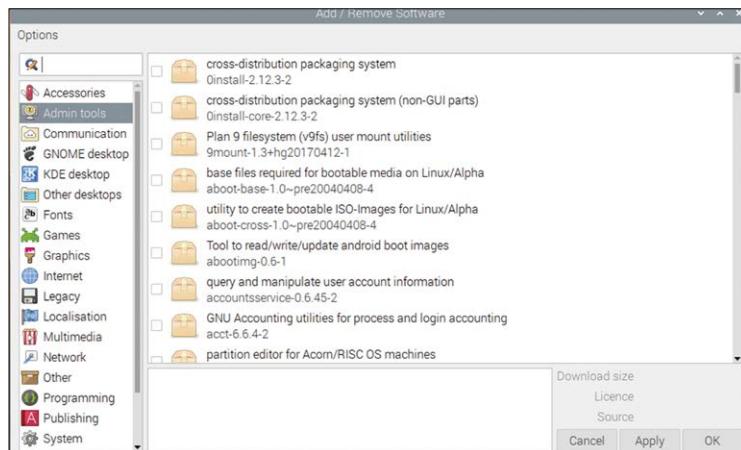


Figure 4.12: Add/Remove software.

4.13 Mouse and keyboard settings

This menu item enables the user to set the mouse and keyboard speeds. Figure 4.13 shows the startup screen.

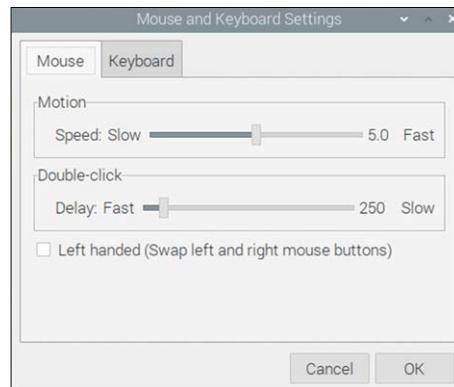


Figure 4.13: Mouse and keyboard settings.

4.14 Raspberry Pi configuration

With this menu item you can very easily configure your Raspberry Pi. The items that can be configured are:

- Password
- Hostname
- Boot options (command line or Desktop)
- Auto login
- Enabled/disabled interfaces
- GPU memory
- Timezone, keyboard type, Wi-Fi country

4.15 Shutdown

This menu item enables you to shut down the system. As shown in Figure 4.14, three options are available: Shutdown, Reboot, and Exit to command line.



Figure 4.14: Shutdown.

4.16 Configuring Wi-Fi

The Wi-Fi connectivity can be configured by placing and clicking the mouse over the Wi-Fi icon at the top right hand corner of the screen. An example display is shown in Figure 4.15. In this example, author's Raspberry Pi was connected to Wi-Fi router named: BTHub5-6SPN-5G.

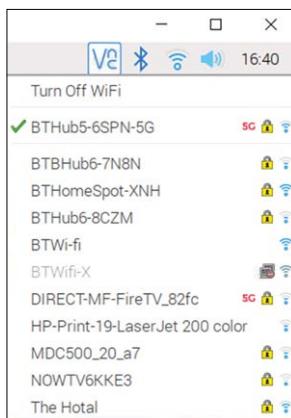


Figure 4.15: Wi-Fi menu.

4.17 Configuring Bluetooth

The Bluetooth connectivity can be configured by placing and clicking the mouse over the Bluetooth icon at the top right hand corner of the screen. An example display is shown in Figure 4.16.

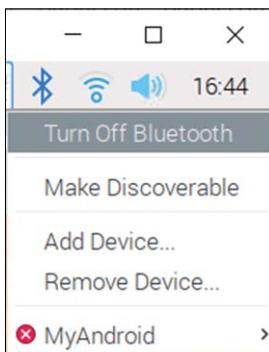


Figure 4.16: Bluetooth menu.

4.18 Summary

In this Chapter we have briefly looked at the various Desktop features. Interested readers can find more information from the Internet and from many books on the Raspberry Pi.

In the next Chapter we will be looking at the Raspberry Pi program development tools and given an example Python 3 program.

CHAPTER 5 • Raspberry Pi Program Development

5.1 Overview

In the last Chapter we have looked at some of the useful programs in Desktop mode. In this Chapter we will be looking at how to develop programs using the Raspberry Pi 4. We will be using the Python 3 programming language in this book. Although Raspberry Pi 4 is used by the author, other models of the Raspberry Pi can also be used as long as Python 3 is installed on them.

5.2 The 'nano' text editor

A text editor is a very useful tool for creating program files. Raspberry Pi supports a number of text editors such as **vi**, **nano** etc. In this section we will introduce the simple to use **nano** text editor which is normally run from the command line.

As an example, suppose that we wish to create a text file called **myfile.txt** and insert the following lines into this file:

*This is a simple text file created using nano
 This is the second line of the file
 This is the third line of the file*

The steps are as follows:

- Start the **nano** editor

```
pi@raspberrypi:~ $ nano myfile.txt
```

- Enter the above text into the file (see Figure 5.1). You should see a number of control codes at the bottom of the screen

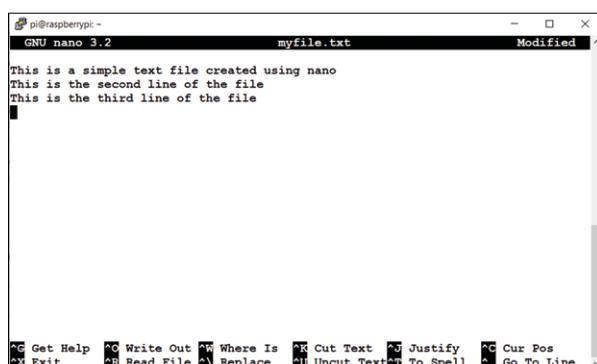


Figure 5.1: Text entered into the editor.

- Enter **Cntrl+X** followed by letter **Y** to save the file. You should now see the file listed in your directory if you enter the command:

```
pi@raspberrypi:~ $ ls myfile.txt
myfile.txt
pi@raspberrypi:~ $
```

- Let's edit the file we just created to learn some of the editor commands. Restart the **nano** editor as above by specifying the filename.
- Let's search for text starting with the word *simple*: press **Cntrl+W** and type **simple** and press **Enter** (see Figure 5.2). You should see the cursor moving to the start of word *simple*. Delete the word *simple* and change it to *difficult*.



Figure 5.2: Searching for the word 'simple'.

- Let's replace the word *third* with *fourth*: press **Cntrl+** and type **third**, and then type **fourth** when **Replace with:** is displayed. Press **Enter**. The message **Replace this instance?** will be displayed. Type **Y**. You should see that word *third* is replaced with word *fourth*.
- Let's delete the second line of text. Move the cursor to the second line and enter **Cntrl+K**. You should see that all the text in the second line is deleted.
- To recall the line just deleted, enter **Cntrl+U**.
- To get help on using the nano editor, enter **Cntrl+G**. An example help screen is shown in Figure 5.3. Enter **Cntrl+N** to display the next page, and **Cntrl+P** to display the previous page. Enter **Cntrl+X** to close the help screen.

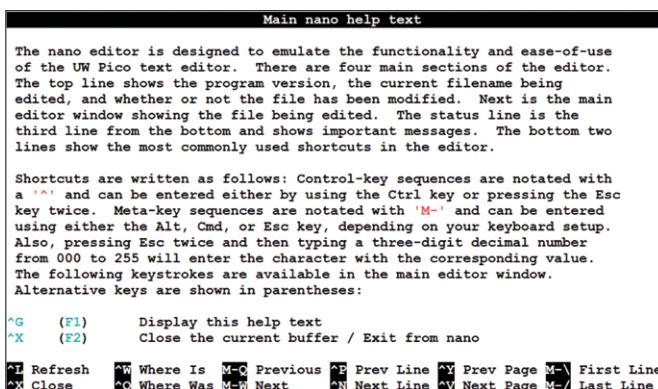


Figure 5.3: Example help screen.

- Enter **Cntrl+X** followed by **Y** to save exit the editor.
- The contents of the edited file are shown in Figure 5.4.

```
pi@raspberrypi:~ $ cat myfile.txt
This is a difficult text file created using nano
This is the second line of the file
This is the fourth line of the file

pi@raspberrypi:~ $ █
```

Figure 5.4: Contents of the edited file.

As a summary, some of the useful **nano** editor 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

5.3 Creating and running a Python program

We will be programming our Raspberry Pi 4 using the Python 3 programming language. It is worthwhile to look at the creation and running of a simple Python program on our Pi computer. In this section we will display the message **Hello From Raspberry Pi 4** on our PC screen.

As described below, there are three methods that we can employ to create and run Python programs on our Raspberry Pi 4.

Method 1 – Interactively from command mode

In this method, we will log in to our Raspberry Pi 4 remotely using the SSH and then create and run our program interactively in the command mode. This method is excellent for small programs. The steps are as follows:

- Login to the Raspberry Pi 4 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 4")
```

- The required text will be displayed interactively on the screen as shown in Figure 5.5. Enter **Cntrl+Z** to exit Python.

```
pi@raspberrypi:~ $ python3
Python 3.7.3 (default, Apr  3 2019, 05:39:12)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Raspberry Pi 4")
Hello from Raspberry Pi 4
>>> █
```

Figure 5.5: Running a program interactively:

Method 2 – Create a Python program in Command Mode

In this method, we will log in to our Raspberry Pi 4 using the SSH as before and then create a Python file. A Python file is simply a text file with the extension **.py**. We can use a text editor, e.g. the **nano** to create our file. In this example a file called **hello.py** is created using the **nano** text editor. Figure 5.6 shows the contents of file **hello.py**. This figure also shows how to run the file using Python 3. Notice that the program is run by entering the command:

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

pi@raspberrypi:~ $ cat hello.py
print("Hello from Raspberry 4")

pi@raspberrypi:~ $ python3 hello.py
Hello from Raspberry 4
pi@raspberrypi:~ $ █
```

Figure 5.6: Creating and running a Python program.

Method 3 – Create a Python program in Desktop GUI mode

In this method, we will log in to our Raspberry Pi 4 using the **VNCViewer** (if we do not have a directly connected monitor) and create and run our program in GUI mode. We will be using a program called **Thonny** which is used to create, debug, and run Python 3 programs. **Thonny** is an easy to use tool and is only available for Python 3. The nice thing about **Thonny** is that it formats the code correctly while it is entered from the terminal. For example, all the statements in the body of a **while** loop are indented correctly.

The steps to use **Thonny** are given below:

- Click **Applications menu**, then **Programming**, and select **Thonny Python IDE** as shown in Figure 5.7.



Figure 5.7: Select Thonny Python IDE.

- The Thonny startup screen will be displayed as shown in Figure 5.8. The screen is in two parts — the program is written in the upper part. The lower part is the **shell** where the results of the program are displayed. We can also run Python 3 commands interactively in the lower part of the screen. In the upper part we have the usual menu items found in most GUI type displays. Menu option **File** is used to create a new file, to open an existing file, to close, save, or print a file. Menu option **Edit** is used to undo, cut, paste, select, find-and-replace, and so on. Option **View** is used to enable to view files, heap, notes, stack, variables and so on. Menu option **Run** is used to run or debug a program. Menu option **Device** is used to soft reboot, to upload current script as main script and so on. Menu option **Tools** is used to manage packages, manage plug-ins, to configure Thonny, and so on. Finally, the **Help** menu option is used to get help on using the Thonny.

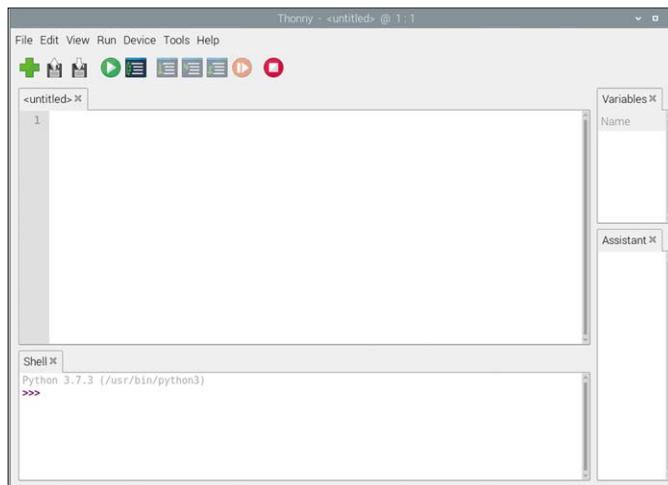


Figure 5.8: Thonny startup screen.

- Type your program in the upper part as shown in Figure 5.9.

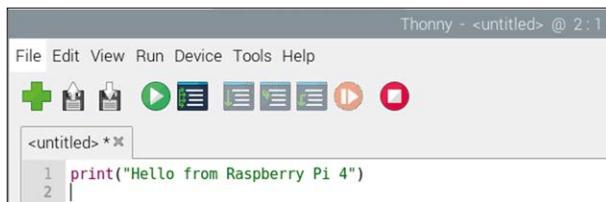


Figure 5.9: Type your program.

- Click **File** and save your program by giving it a name. You do not have to specify the file extension as this is added automatically by Thonny.
- Click **Run** and you should see the program output at the lower part of the screen as shown in Figure 5.10.



Figure 5.10: Output of the program.

Thonny provides the option of debugging a program, where we can single-step through a program and see the variables as the program is stepped through. As an example, let's debug the program given in Figure 5.9. The steps are:

- Click **Run** and then click **Debug current script (nicer)**.

- You should see the current program line highlighted in yellow colour.
- We now have the options of: **Step over**, **Step into**, and **Step out**.
- Clicking **Step over** will step through the program lines as we see them on the screen. Click this button and you should see the output of the program displayed at the lower part of the screen.
- While in Debug mode, you can also **Resume** the program so that it continues normally (the orange and white icon) or **Stop and Restart** the program from the beginning (the red and white icon).

Which method?

The choice of a method depends upon the size and complexity of a program. Small programs can be run interactively without creating a program file. Larger programs can be created as Python files either by using the **nano** text editor in command mode, or **Thonny** can be used to create them in Desktop GUI mode.

5.4 Summary

In this Chapter we have learned how to develop Python 3 programs using several methods. The choice of a method depends entirely on the user.

In the next Chapter we will be looking at the GPIO and develop some simple programs to illustrate how the GPIO can be accessed from Python programs.

CHAPTER 6 • The GPIO

6.1 Overview

In the last Chapter we have had a look at the Raspberry Pi program development tools and learned how to create and run a very simple program. In this Chapter we will be learning the details of the GPIO (General Purpose Input Output) header connector, as well as how to interface simple devices to the GPIO. An example hardware project is given to illustrate how to program the Raspberry Pi to access the GPIO ports.

6.2 The Raspberry Pi 4 GPIO connector

Before going into the details of hardware interface, it is worthwhile to look at the Raspberry Pi 4 GPIO connector. This is a 40-pin dual-in-line (DIL) 0.1 inch (2.54 mm) wide connector as shown in Figure 6.1. Other recent Raspberry Pi models have similar connectors with almost the same pin configurations

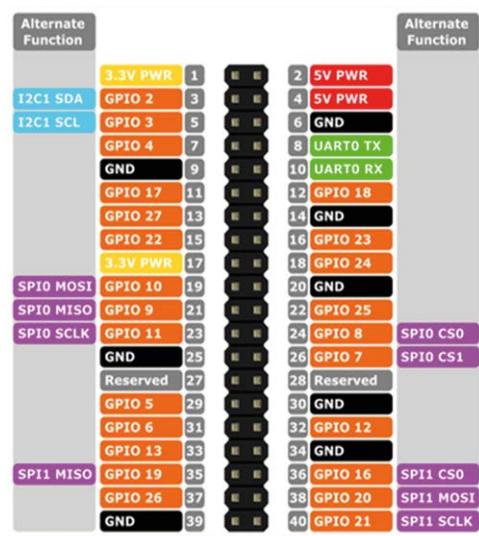


Figure 6.1: Raspberry Pi 4 GPIO connector.

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

The GPIO provides 26 general purpose bidirectional I/O pins. Some of the pins have multiple functions. For example, pins 3 and 5 are the GPIO2 and GPIO3 input/output (I/O) pins respectively. These pins can also be used as the I²C bus 'SDA' and 'SCL' pins respectively. Similarly, pins 9, 10, 11, 19 can either be used as general-purpose input/output, or as the SPI bus pins. Pins 8 and 10 are reserved for UART serial communication.

Two supply voltage outputs are provided: +3.3 V and +5.0 V. The GPIO pins operate at +3.3 V logic levels (not like many other computer circuits which operate with +5 V). A pin

can either be an input or an output. When configured as an output, the pin voltage is either 0 V (logic 0) or +3.3 V (logic 1). Raspberry Pi 4 is normally operated using an external power supply (e.g. a mains adapter or *wall wart*) with +5 V output and minimum 2 A current capacity. A 3.3 V output pin can supply up to 16 mA of current. The total current drawn from all output pins should not exceed the 51 mA limit. Care should be taken when connecting external devices to the GPIO pins as drawing excessive currents or short-circuiting a pin can easily damage your Pi. The amount of current that can be supplied by the 5 V pin depends on many factors such as the current required by the Pi itself, current taken by the USB peripherals, camera current, HDMI port current, and so on.

When configured as an input, a voltage above +1.7 V will be taken as logic 1, and a voltage below +1.7 V will be taken as logic 0. Care should be taken not to supply voltages greater than +3.3 V to any I/O pin as large voltages can easily damage your Pi. For example, the output pin of an Arduino must not be connected directly to a Raspberry Pi input pin as it can easily damage the input circuitry. Similarly, although connecting a Raspberry Pi output pin to an Arduino input pin is safe, the output voltage level of the Raspberry Pi may not be high enough to drive the Arduino input circuitry when it is at logic 1. In such circumstances a voltage level converter circuit should be used to convert either from 5 V to 3.3 V logic levels, or *vice versa* from 3.3 V to 5 V. An example logic level converter circuit is shown in Figure 6.2.

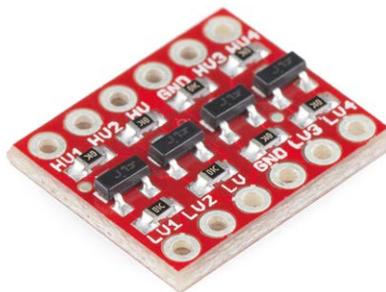


Figure 6.2: Voltage level converter circuit.

6.3 Interfacing to the GPIO

6.3.1 Loads requiring small currents

Loads requiring relatively small currents, such as LEDs, can be directly connected to the GPIO pins through current limiting resistors. LEDs are available in different sizes. Small LEDs draw around a few milliampères (mA) of current, while larger LEDs require about 15 mA to be bright. The voltage drop across an LED again depends on the type of the LED used, but in most cases we can assume a voltage drop of around 2 V. LEDs can be connected either in current source or current sink mode as described below.

Connecting in current-source mode

In this mode (see Figure 6.3), the LED is turned ON when the output port is at logic 1, and is turned OFF when it is at logic 0. Assuming 4 mA LED current and 2 V voltage drop across the LED, the required current limiting resistor is calculated to be:

$$R = (3.3 \text{ V} - 2 \text{ V}) / 4 \text{ mA} = 325 \text{ ohms}$$

The nearest physical value is 330 ohms. Choosing a smaller resistor will make the LED brighter. Similarly, a larger resistor will make the LED light dimmer.

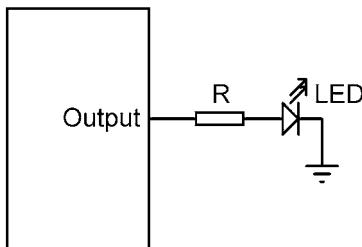


Figure 6.3: Connecting a load in current sourcing mode.

Connecting in current-sinking mode

In this mode (see Figure 6.4), the LED is turned ON when the output port is at logic 0, and is turned OFF when it is at logic 1. Assuming 4 mA LED current, 2 V voltage drop across the LED, and 0.1 V output low voltage of the Raspberry Pi, the required current limiting resistor is calculated to be:

$$R = (3.3 \text{ V} - 0.1 \text{ V} - 2 \text{ V}) / 4 \text{ mA} = 300 \text{ ohms}$$

We can choose 290 ohms or 330 ohms.

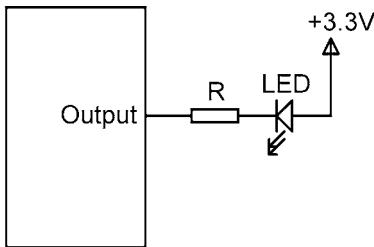


Figure 6.4: Connecting a load in current-sinking mode.

6.3.2 Loads requiring higher currents

Loads requiring currents in the range of several hundreds of milliamps can be connected to the Raspberry Pi through a transistor switch. Bipolar transistors are suited to lower currents, while MOSFET transistors are more suitable for higher currents. Figure 6.5 shows a bipolar transistor switch with the load connected to the collector circuit. Here, the load is activated when the transistor is ON, i.e. when the output of the Raspberry Pi is at logic 1 (i.e. when the transistor is switched ON). A 1-kohm base resistor is suitable in most applications. If the load is inductive then a diode should be used as shown in Figure 6.6 to protect the transistor from back emf.

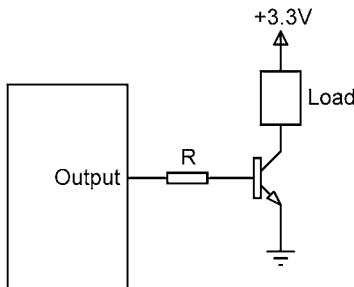


Figure 6.5: Using a bipolar transistor.

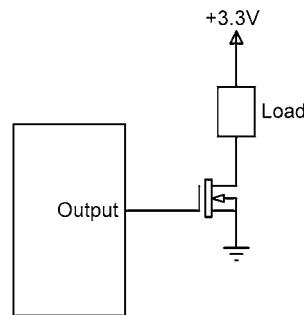


Figure 6.6: Using an inductive load.

Figure 6.7 shows a MOSFET based circuit which is suitable for larger currents, usually in the range of hundreds of millamps. Again, the load is activated when the output of the Raspberry Pi is at logic 1.

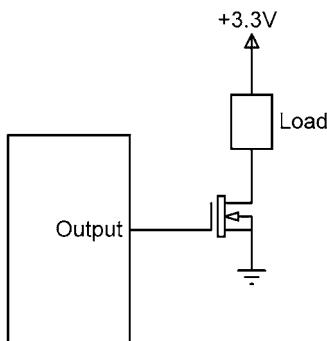


Figure 6.7: Using a MOSFET transistor.

6.3.3 Using relays

In applications where the load operates with large voltages and currents it is recommended to use relays to switch the loads ON and OFF. Either semiconductor or contact based physical relays can be used in circuits. Relays are also available as modules with built in transistors which makes them easier to use in microcontroller applications. Figure 6.8 shows such a relay module with four relays on the board. Alternatively, a transistor circuit can be used

to switch the relay ON and OFF.

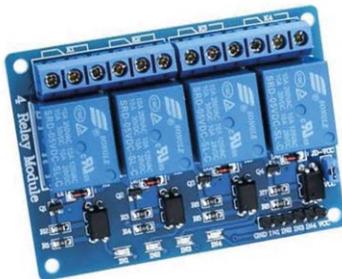


Figure 6.8: A relay module with four relays (Elegoo relay module).

6.4 The GPIO library

The GPIO library contains functions that enable the programmer to access the input output ports easily from a Python program. The GPIO library is called **RPi.GPIO** and it should already be installed on your Raspberry Pi 4. This library must be included at the beginning of your Python programs if you will be using the GPIO functions. The statement to include this library is:

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

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

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

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

6.4.1 Pin numbering

There are two ways that we can refer to the GPIO pins. The first is using the BOARD numbering, where the pin numbers on the GPIO connector of the Raspberry Pi 4 are used. Enter the following statement to use the BOARD method:

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

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

GPIO.setmode(GPIO.BCM)

6.4.2 Channel (I/O port pin) configuration

Input Configuration

You need to configure the channels (or port pins) you are using whether they are input or output channels. The following statement is used to configure a channel as an input. Here, 'channel' refers to the channel number based on the **setmode** statement above:

GPIO.setup(channel, GPIO.IN)

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

For pulldown:

GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

For pullup:

GPIO.setup(channel, GPIO.IN, pull_up_down=GPIO.PUD_UP)

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

To wait for a rising edge:

GPIO.wait_for_edge(channel, GPIO.RISING)

To wait for a falling edge:

GPIO.wait_for_edge(channel, GPIO.FALLING)

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

GPIO.wait_for_edge(channel, GPIO.BOTH)

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

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

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

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

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

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

Event detection can be removed by the following statement:

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

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

```
GPIO.add_event_detect(channel, GPIO.RISING, callback=My-  
Handler)
```

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

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

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

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

```
GPIO.add_event_detect(channel, GPIO.RISING)  
GPIO.add_event_callback(channel, MyHandler1)  
GPIO.add_event_callback(channel, MyHandler2)
```

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

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

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

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

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

When we use mechanical switches in our projects, we get what is known as the switch bouncing problem. This occurs as the contacts of the switch bounce many times until they settle to their final state. Switch bouncing could generate several pulses before it settles down. We can avoid switch bouncing problems in hardware or software. GPIO library provides a parameter called bounce-time that can be used to eliminate the switch bouncing problem. An example use of this parameter is shown below where the switch bounce time is assumed to be 10 ms:

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

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

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

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

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

Output configuration

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

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

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

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

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

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

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

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

```
GPIO.cleanup()
```

6.5 The Raspberry Pi project development cycle

The project development cycle in general includes both hardware and software development.

6.5.1 The hardware

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

While developing hardware-based projects on the Raspberry Pi we have to make connections to the 40-pin male type GPIO connector. This can easily be done using female-male type jumper leads. One side of the jumper lead can be connected to the GPIO connector, while the other side can be connected to a breadboard so that external components can easily be interfaced to the Raspberry Pi. It is recommended by the author to use a 40-way ribbon cable with a T-connector to bring all the GPIO pins to the breadboard for easy access. Figure 6.9 shows such a connector which is available on the Internet

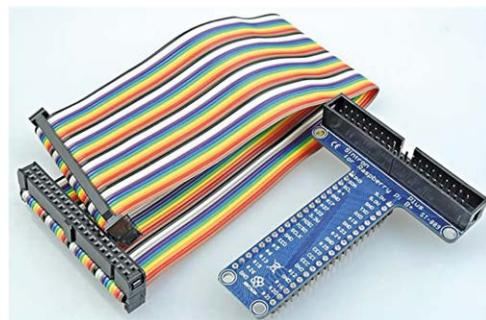


Figure 6.9: T-connector with ribbon cable (by Sintron).

6.5.2 The software

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

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

6.6 Project – Alternately flashing red and green LEDs

Description: This is a very simple project where an RGB LED is connected to the Raspberry Pi and the red and green colours are flashed alternately every 500 ms.

Aim: Although this is not strictly a ham radio project, the aim of this project is to show how a simple interface can be made to the Raspberry Pi GPIO, and also how to develop and run a program on the Raspberry Pi using the Python 3 programming language.

Background Information: As shown in Figure 6.10, the RGB LED is a 4-pin device which incorporates Red, Green and Blue LEDs. Each colour LED is assigned a pin, where the fourth pin is the ground. By activating different LEDs at different brightness, we can generate many different colours. In this project a common cathode RGB LED is used. Notice that the cathode pin of the RGB LED is the longer pin.

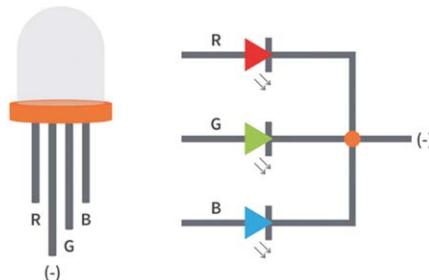


Figure 6.10: RGB LED (from CircuitBread).

Block Diagram: Figure 6.11 shows the block diagram of the project.

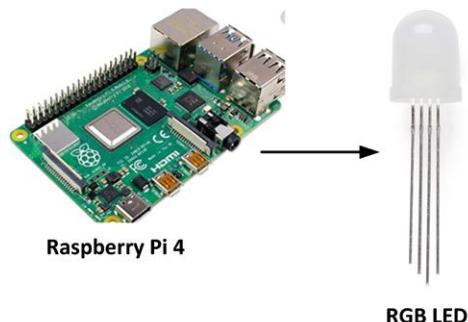


Figure 6.11: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 6.12. The Red, Green, and Blue pins are connected to port pins GPIO2, GPIO3, and GPIO4 respectively.

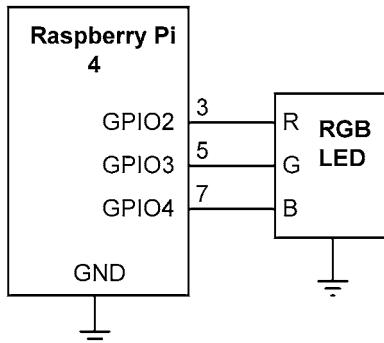


Figure 6.12: Circuit diagram of the project.

Program Listing: The program is very simple and is shown in Figure 6.13 (program: **RGB.py**). At the beginning of the program the RPi.GPIO and the time module are imported to the program. Then, RED, GREEN, and BLUE LEDs are assigned to 2, 3, and 4 respectively which corresponds to the GPIO numbers GPIO2, GPIO3 and GPIO4. The LED ports are configured as outputs. The remainder of the program runs in an endless loop. Inside this loop the RED LED is turned ON and OFF with 0.5 second delay between each output. Then the GREEN LED is turned ON and OFF with 0.5 second delay between each output. The program runs forever and can be terminated by pressing the **Cntrl+C** keys.

```
#-----
#           RED AND GREEN FLASHING LED
#
# In this project an RGB LED is connecte dto the Raspberry Pi.
# The program flashes the RED and the GREEN LEDs alternately
# every 0.5 second
#
# Author: Dogan Ibrahim
# File  : RGB.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

RED = 2                      # RED LED at GPIO2
GREEN = 3                     # GREEN LED at GPIO3
BLUE = 4                      # BLUE LED at GPIO4
GPIO.setmode(GPIO.BCM)          # Set BCM mode

GPIO.setup(RED, GPIO.OUT)        # RED LED is output
GPIO.setup(GREEN, GPIO.OUT)      # GREEN LED is output
GPIO.setup(BLUE, GPIO.OUT)       # BLUE LED is output
```

```

while True:                      # Do forever
    GPIO.output(RED, 1)           # RED LED is ON
    time.sleep(0.5)              # Wait 0.5 second
    GPIO.output(RED, 0)           # RED LED is OFF
    time.sleep(0.5)              # Wait 0.5 second
    GPIO.output(GREEN, 1)          # GREEN LED is ON
    time.sleep(0.5)              # Wait 0.5 second
    GPIO.output(GREEN, 0)          # GREEN LED is OFF
    time.sleep(0.5)              # Wait 0.5 second

```

Figure 6.13: Program listing.

You can either run the program inside Thonny, or enter the following command from the command mode:

```
pi@raspberrypi:~ $ python3 RGB.py
```

Modified program

The program given in Figure 6.13 is terminated by pressing the **Cntrl+C** keys, which admittedly isn't a clean way of terminating a program. A modified program (program: **RGB2.py**) listing is shown in Figure 6.14 where the **Cntrl+C** keys are detected as an exception by the program and the program is terminated normally after cleaning up the GPIO library.

```

#-----
#                   RED AND GREEN FLASHING LED
#
#-----#
# In this project an RGB LED is connecte dto the Raspberry Pi.
# The program flashes the RED and the GREEN LEDs alternately
# every 0.5 second. This program is terminated cleanly
#
# Author: Dogan Ibrahim
# File  : RGB2.py
# Date  : July, 2020
#-----

import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

RED = 2                      # RED LED at GPIO2
GREEN = 3                     # GREEN LED at GPIO3
BLUE = 4                      # BLUE LED at GPIO4
GPIO.setmode(GPIO.BCM)          # Set BCM mode

GPIO.setup(RED, GPIO.OUT)        # RED LED is output
GPIO.setup(GREEN, GPIO.OUT)      # GREEN LED is output
GPIO.setup(BLUE, GPIO.OUT)       # BLUE LED is output

```

try:

```
while True:                                # Do forever
    GPIO.output(RED, 1)                      # RED LED is ON
    time.sleep(0.5)                          # Wait 0.5 second
    GPIO.output(RED, 0)                      # RED LED is OFF
    time.sleep(0.5)                          # Wait 0.5 second
    GPIO.output(GREEN, 1)                    # GREEN LED is ON
    time.sleep(0.5)                          # Wait 0.5 second
    GPIO.output(GREEN, 0)                    # GREEN LED is OFF
    time.sleep(0.5)                          # Wait 0.5 second

except KeyboardInterrupt:
    GPIO.cleanup()
    print("End of program")
```

Figure 6.14: Modified program.

Figure 6.15 shows the project built on a breadboard.

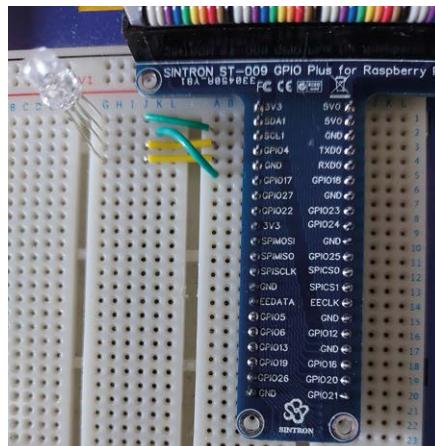


Figure 6.15: Project built on a breadboard.

6.7 Running a program automatically at startup time

There are many applications where we may want to start a program automatically as soon as the Raspberry Pi is started. This can be done in several ways. Perhaps the easiest way is to use file **/etc/rc.local**. The steps are given below where the program **RGB2.py** is run automatically just after a system startup:

- Edit file **/etc/rc.local** as a super user using the **nano** editor:

```
pi@raspberrypi:~ $ sudo nano /etc/rc.local
```

- Go to the end of the file and enter the following statements before **exit 0**. The **&** character at the end of the command runs the program as a background process. This is required if the program is in a loop otherwise the startup will never complete:

```
python3 /home/pi/RGB2&
```

- Save and exit from **nano** by entering **Cntrl+X** followed by **Y**.
- Restart your Raspberry Pi.
- You should see the RGB LED flashing as soon as the system is up and running.

Do not forget to edit file **/etc/rc.local** and remove the statement to start **RGB2.py** if it is not required any more.

6.8 Scheduling a program to run at specified times

There are many applications where we may want to run programs automatically at regular intervals, such as backup operations, time synchronization, running a process in the future, etc.

Running tasks at regular intervals is managed by the **crontab** command. This consists of a set of tables (crontab tables) and the **cron daemon**. The daemon is started by the **init** process at system startup and it wakes up every minute and checks the crontab tables to determine if there are any tasks scheduled to run.

To create a crontab table we use the **crontab** command with the **-e** option. This opens the **vi** editor (unless another editor is specified in the environment variable). Each crontab contains six fields. The values in the fields can have fixed values or a range of values, or a list of values separated by commas:

- Minute (0 - 59).
- Hour (0 - 23).
- Day of the month (1 - 31).
- Month of the year (1 - 12; alternatively specified as jan, feb, mar and so on).
- Day of the week (0 - 6; where 0=Sunday; it can also be mon, tue, wed, etc.).
- String (command) to be executed.

A ***** character in the digit position means *every*. Ranges of numbers are specified by separating them with a hyphen and the specified range is inclusive. For example, 9-12 for an Hours entry specifies execution at hours 9, 10, 11 and 12.

Skips of numbers in ranges can be specified by adding character **/** after the range. For example, 0-12/2 in the Hours field specifies execution every other hour, i.e. at the hours 0, 2, 4, 6, 8, 10.

By using a combination of * and / , we can specify steps. For example, */2 in the Hours field specifies every two hours.

Instead of the first five fields, we can specify special strings as follows:

String	meaning
@reboot	run once at startup
@yearly	run once every year ("0 0 1 1")
@monthly	run once every month ("0 0 1 *")
@weekly	run once a week ("0 0 * * 0")
@daily	run once a day ("0 0 * *")
@hourly	run once an hour ("0 * * *")

Multiple commands can be entered on a line and such commands must be separated with && characters.

Some examples of crontab lines are given below:

1. 30,50 22-23 * 6 fri-sat/home/pi/mycron.sh

Run at the 30th and 50th minutes, for the hours between 10 p.m. and midnight on Fridays and Saturdays during June.

2. @daily <command1>&&<command2>

Run command 1 and command 2 daily

3. 30 0 * * */home/pi/mycron.sh

Run at 12:30 daily

4. 0 4 12 * * /home/pi/mycroft.sh

Run at 4 a.m. on the 12th of every month

5. ***** /home/pi/mycron.sh

Run every minute

6. 0 4 15-21 * 1 /home/pi/mycron.sh

Run every month at 4 a.m. on Mondays, and on the days between 15-21. Notice that here the day of the month and the day of the week are used with no restrictions (no *) and therefore this is an 'or' condition, both will be executed

7. 0 11,16* * * /home/pi/mycron.sh

Run every day at 11:00 and 16:00 hours

```
8. 0 11-14 * * * /home/pi/mycron.sh
```

Run every day during the hours 11 a.m. -2 p.m. (i.e. 11 a.m., 12 a.m., 1 p.m., 2 p.m.)

```
9. */10 **** /home/pi/mycron.sh
```

Run every 10 minutes

```
10. @yearly /home/pi/mycron.sh
```

Run on the first minute of every year

By default, crontab sends the job to the user who scheduled the job. If you want mail not to be sent to anyone, you should specify the following line in the crontab:

```
MAIL=""
```

Any outputs in a scheduled process are usually logged in files. For example, to run **mycron.sh** daily and send the output to file **daily.txt**, enter the following command:

```
@daily /home/pi/mycron.sh > daily.txt
```

Instead of directly editing the crontab file, you can also add the entries to a cron-file first, and then install them to the cron using the crontab command and specify the filename. An example is given below:

```
pi@raspberrypi:~ $ crontab /home/pi/mycron.sh
```

This will install the **mycron.sh** to our crontab, which will also remove any old cron entries. The created crontab is stored in directory **/etc/spool/cron/<user>**

In all the examples above, we have specified the absolute path of the script file that should be executed. We can specify this path in the PATH environment variable in the crontab and then just enter the filename. An example is given below where the absolute path to the file is **/home/pi/mycron.sh**

```
PATH=/home/pi  
@daily mycron.sh
```

If the script or the command we wish to run requires privilege, then it should be prefixed with the **sudo** command.

Example

It is required to run **myscript.sh** every minute. Assume that this script file has the following line of command:

```
date >> /home/pi/myfile
```

schedule this event using the crontab command. Send the output from the command to file called **myfile**.

Solution

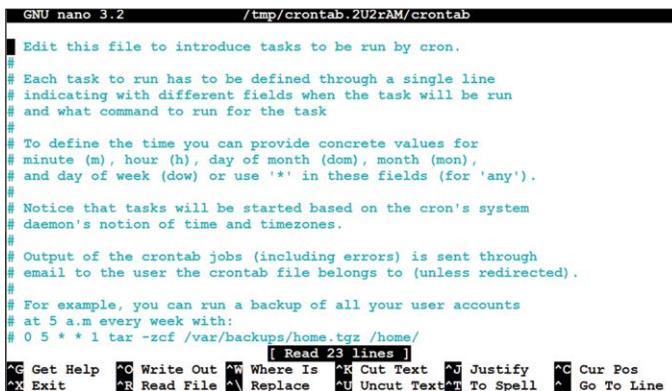
First of all, create the **myscript.sh** file and type in the above command. The default editor is **nano**. Save the file (**Cntrl+X**, followed by **Y** and Enter) and exit the editor. Next, give the file execute permission by entering the following command:

```
pi@raspberrypi:~ $ sudo chmod 755 myscript.sh
```

The steps for using crontab command to schedule this event are given below:

- Run crontab command with the **-e** flag. You should see the crontab editor screen as in Figure 6.16

```
pi@raspberrypi:~ $ crontab -e
```



The screenshot shows a terminal window titled "GNU nano 3.2" with the path "/tmp/crontab.2U2rAM/crontab". The editor displays the following text:

```
GNU nano 3.2          /tmp/crontab.2U2rAM/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
[ Read 23 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit    ^R Read File ^\ Replace   ^U Uncut Text ^T To Spell ^A Go To Line
```

Figure 6.16: Empty crontab text editor screen.

- Enter the following line to the end of the file:

```
* * * * * /home/pi/myscript.sh
```

- Exit the **nano** editor by entering:

```
Cntrl+X  
Y  
<Enter>
```

- The commands in the **myscript.sh** will now be executed every minute and the output will be sent to file **myfile**. We can verify that the commands are executed by looking at the contents of **myfile** after a few minutes:

```
pi@raspberrypi:~ $ cat myfile
Wed 13 May 16:27:01 BST 2020
Wed 13 May 16:28:01 BST 2020
Wed 13 May 16:29:01 BST 2020
pi@raspberrypi:~ $
```

- You can stop the scheduling by deleting the line entered by command **crontab -e**

The command **crontab -l** displays a list of the scheduled tasks (if there are any):

```
pi@raspberrypi:~ $ crontab -l
```

All scheduled tasks can be deleted (if there are any) using the **crontab -r** command:

```
pi@raspberrypi:~ $ crontab -r
pi@raspberrypi:~ $ crontab -l
nocrontab for pi
pi@raspberrypi:~ $
```

The **-i** option can be added to the delete command to confirm the delete action.

The crontab generator

An online tool called the **Crontab Generator** is available free of charge on the Internet that can be used to generate crontab entries easily. This tool is available at the following website:

<https://crontab-generator.org/>

Figure 6.17 shows part of the Crontab Generator. An example is given to show how to use this tool.

Complete the following form to generate a crontab line

Ctrl-click (or command-click on the Mac) to select multiple entries

Minutes	Hours	Days
<input checked="" type="radio"/> Every Minute <input type="radio"/> Even Minutes <input type="radio"/> Odd Minutes <input type="radio"/> Every 5 Minutes <input type="radio"/> Every 15 Minutes <input type="radio"/> Every 30 Minutes	<input checked="" type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 <input type="radio"/> 7 <input type="radio"/> 8 <input type="radio"/> 9	<input checked="" type="radio"/> Midnight <input type="radio"/> 1am <input type="radio"/> 2am <input type="radio"/> 3am <input type="radio"/> 4am <input type="radio"/> 5am <input type="radio"/> 6am <input type="radio"/> 7am <input type="radio"/> 8am <input type="radio"/> 9am
Months	Weekday	
<input checked="" type="radio"/> Every Month <input type="radio"/> Even Months <input type="radio"/> Odd Months <input type="radio"/> Every 4 Months <input type="radio"/> Every Half Year	<input checked="" type="radio"/> Every Weekday <input type="radio"/> Monday-Friday <input type="radio"/> Weekend Days	<input checked="" type="radio"/> Sun <input type="radio"/> Mon <input type="radio"/> Tue <input type="radio"/> Wed <input type="radio"/> Thu <input type="radio"/> Fri

Figure 6.17: The Crontab Generator.

Example

It is required to run the script file **myscript.sh** every day at 11:00 a.m. and 4:00 p.m. hours.

Solution

The steps are given below:

- Start the Crontab Generator.
- Select the Minutes as 0, Hours as 11am and 4 pm (click **Ctrl** to select more than one entry)
- Enter **myscript.sh** to the field **Command to Execute** as shown in Figure 6.18)

Complete the following form to generate a crontab line

Ctrl-click (or command-click on the Mac) to select multiple entries

Minutes	Hours	Days
<input type="radio"/> Every Minute <input type="radio"/> Even Minutes <input type="radio"/> Odd Minutes <input type="radio"/> Every 5 Minutes <input type="radio"/> Every 15 Minutes <input type="radio"/> Every 30 Minutes	<input checked="" type="radio"/> 0 <input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 <input type="radio"/> 6 <input type="radio"/> 7 <input type="radio"/> 8 <input type="radio"/> 9	<input checked="" type="radio"/> 11am <input type="radio"/> 2am <input type="radio"/> 3am <input type="radio"/> 4am <input type="radio"/> 5am <input type="radio"/> 6am <input type="radio"/> 7am <input type="radio"/> 8am <input type="radio"/> 9am <input type="radio"/> 10am
Months	Weekday	
<input checked="" type="radio"/> Every Month <input type="radio"/> Even Months <input type="radio"/> Odd Months <input type="radio"/> Every 4 Months <input type="radio"/> Every Half Year	<input checked="" type="radio"/> Every Weekday <input type="radio"/> Monday-Friday <input type="radio"/> Weekend Days	<input checked="" type="radio"/> Sun <input type="radio"/> Mon <input type="radio"/> Tue <input type="radio"/> Wed <input type="radio"/> Thu <input type="radio"/> Fri <input type="radio"/> Sat
Command To Execute		
myscript.sh		

Figure 6.18: Configure to schedule at 11:00 a.m. and 4:00 p.m..

- Click **Save output to file** and enter **myfile** so that the output will be stored in file **myfile**.
- Click button **Generate Crontab Line**
- You should see the generated line as shown in Figure 6.19, which is:

0 11,16 * * * myscript.sh > myfile

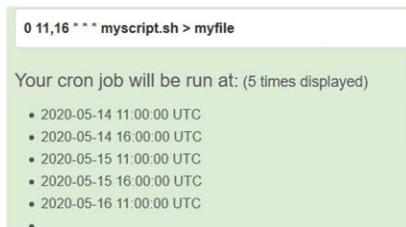


Figure 6.19: The generated line.

- The tool also displays sample dates and times that the script file will be scheduled to run. It is clear from Figure 6.19 that the script file **myscript.sh** will run at 11:00 a.m. and 4:00 p.m. every day.
- You should copy the above line to the end of the crontab table by entering the command **crontab -e** as discussed earlier.

6.9 Summary

In this Chapter we have learned how to access the GPIO of the Raspberry Pi 4. A simple hardware-based program is given to illustrate how a program can be developed and run on the Raspberry Pi 4. Additionally, we saw how to run a program automatically at the startup time, and how to schedule a program to run at specified days and times.

In the next Chapter we will be developing projects geared to ham radio operators.

CHAPTER 7 • Station Mains On/Off Power Control

7.1 Project

Description: In this project four buttons and four relays are connected to the Raspberry Pi. The relays change state when their corresponding buttons are pushed. For example, various mains-operated equipment can be connected to the 230 V_{AC} mains supply (US: 115 V_{AC} 'line') through the relays, and such equipment can be switched ON or OFF easily by pressing its corresponding button.

Aim: The aim of this project is to show how buttons and relays can be connected to the Raspberry Pi, as well as show the way such a setup can be used by amateur radio operators to switch ON/OFF their equipment easily.

Block Diagram: Figure 7.1 shows the block diagram of the project.

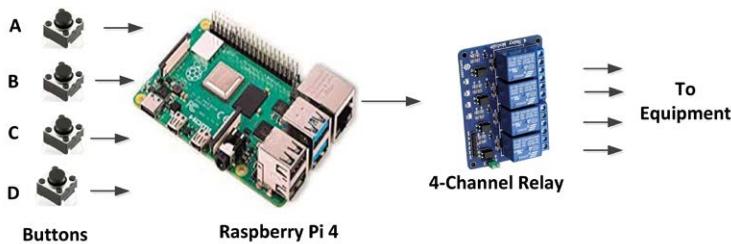


Figure 7.1: Block diagram of the project.

Circuit Diagram: In this project a 4-channel relay board (see Figure 7.2) from Elegoo (www.elegoo.com) is used. This is an opto-coupled relay board having four inputs, one for each channel. The relay inputs are at the bottom right-hand side of the board while the relay outputs are located at the top side of the board. The middle position of each relay is the common point, the connection to its left is the normally closed (NC) contact, while the connection to the right is the normally open (NO) contact. The relay contacts support 250V_{AC} at 10 A and 30 V_{DC} 10 A. IN1, IN2, IN3 and IN4 are the **active LOW** inputs, which means that a relay is activated when a logic LOW signal is applied to its input pin. Relay contacts are normally closed (NC). Activating the relay changes the active contacts such that the common pin and NC pin become the two relay contacts and at the same time the LED at the input circuit of the relay board corresponding to the activated relay is turned ON. The V_{CC} can be connected to either +3.3 V or to +5 V. Jumper JD is used to select the voltage for the relay. **Because the current drawn by a relay can be in excess of 80 mA, you must remove this jumper and connect an external power supply (e.g. +5 V) to pin JD-VCC.**

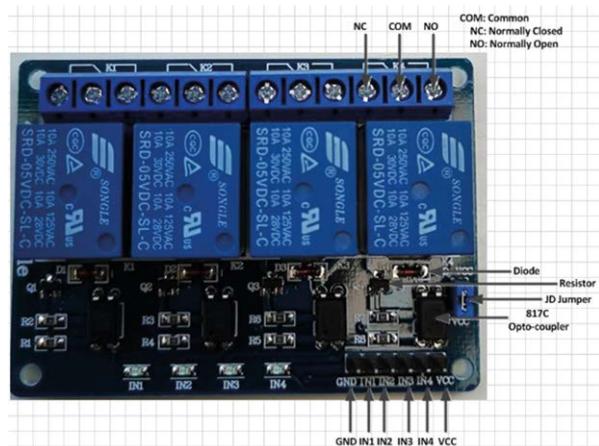


Figure 7.2: 4-channel relay board.

The circuit diagram of the project is shown in Figure 7.3. The buttons are named as A, B, C, and D and they are connected to Raspberry Pi port pins GPIO4, GPIO17, GPIO27, and GPIO22 respectively. The button outputs are at logic 1 and go to logic 0 when the button is pressed. Pins IN1, IN2, IN3, and IN4 of the relay module are connected to port pins GPIO21, GPIO20, GPIO16, and GPIO12 respectively. The relay contacts should be connected to the equipment to be switched ON/OFF through the mains supply (**WARNING: care should be taken when working with live mains voltages and you should seek professional advice if you are not sure about connections to the mains supply**)

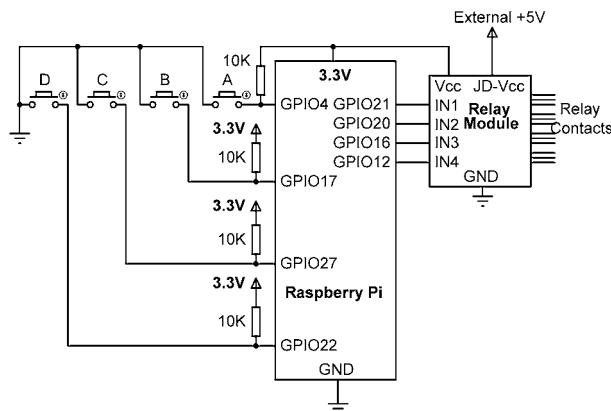


Figure 7.3: Circuit diagram of the project.

Program Listing: Figure 7.4 shows the program listing (program: **relays.py**). At the beginning of the program the button and relay connection to the Raspberry Pi are defined. Port pins where the relay input pins IN1-IN4 are connected to are configured as outputs. Port pins where the button pins BUTTONA, BUTTONB, BUTTONC, and BUTTOND are connected to are configured as inputs. All four relays are then turned OFF at the beginning of the program. The remainder of the program is executed inside the main program loop

where a **while** loop is used. Here, function **TestButton** is used to check the state of a button and if the button is pressed then the state of the corresponding relay is toggled.

Functions are useful programming tools that are used when parts of a program is to be repeated several times. A function declaration in Python starts with the keyword **def**, followed by a bracket. Inside the bracket the arguments to be passed to the function are specified (if there are any). Notice that the arguments are local to the function. Function **TestButton** is shown here:

```
def TestButton(Button, IN):
    if GPIO.input(Button) == 0:
        s = GPIO.input(IN)
        if s == ON:
            GPIO.output(IN, OFF)
        else:
            GPIO.output(IN, ON)
        while GPIO.input(Button) == 0;
            pass
```

The second line of the function tests if Button is equal to 0, i.e. if the Button is pressed. Remember that normally when the Button is not pressed its state is at logic 1. If the Button is pressed, then the state of the corresponding relay pin is read. If the relay is ON, then it is set to OFF by sending OFF to the corresponding IN pin. If on the other hand (else) the relay is OFF, then ON is sent to its corresponding IN pin turn it ON. The function then waits until the Button is released.

```
#-----
#                      STATION MAINS ON-OFF CONTROL
#-----
# In this project 4 buttons and 4 relays are connected to Raspberry
# Pi. The status of the relays are toggled each time its corresonding
# button is pressed. So, if the relay is ON, it is turned OFF, if it
# is OFF it is turned ON etc.
#
# Author: Dogan Ibrahim
# File : relays.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)

#
# Button connections
#
BUTTONA = 4                                     # Button A
```

```
BUTTONB = 17                                # Button B
BUTTONC = 27                                # Button C
BUTTOND = 22                                # Button D

OFF = 1
ON = 0

#
# Relay connections
#
IN1 = 21                                     # IN1 pin
IN2 = 20                                     # IN2 pin
IN3 = 16                                     # IN3 pin
IN4 = 12                                     # IN4 pin

GPIO.setmode(GPIO.BCM)                         # Set BCM mode

#
# IN1-IN4 are Outputs
#
GPIO.setup(IN1, GPIO.OUT)                      # IN1 is output
GPIO.setup(IN2, GPIO.OUT)                      # IN2 is output
GPIO.setup(IN3, GPIO.OUT)                      # IN3 is output
GPIO.setup(IN4, GPIO.OUT)                      # IN4 is output

#
# BUTTONA-BUTTOND are inputs
#
GPIO.setup(BUTTONA, GPIO.IN)                  # Button A is input
GPIO.setup(BUTTONB, GPIO.IN)                  # Button B is input
GPIO.setup(BUTTONC, GPIO.IN)                  # Button C is input
GPIO.setup(BUTTOND, GPIO.IN)                  # Button D is input

#
# Turn OFF all relays at the beginning
#
GPIO.output(IN1, OFF)                         # IN1 OFF
GPIO.output(IN2, OFF)                         # IN2 OFF
GPIO.output(IN3, OFF)                         # IN3 OFF
GPIO.output(IN4, OFF)                         # IN4 OFF

#
# This function checks if Button is pressed and toggles pin IN
# of the relay If the relay os ON, it is tured OFF, if it is OFF,
# it is turned ON
#
def TestButton(Button, IN):
```

```
if GPIO.input(Button) == 0:  
    s = GPIO.input(IN)  
    if s == ON:  
        GPIO.output(IN, OFF)  
    else:  
        GPIO.output(IN, ON)  
    while GPIO.input(Button) == 0:  
        pass  
  
try:  
  
    while True:  
        # Do forever  
        TestButton(BUTTONA, IN1)          # Test Button A  
        TestButton(BUTTONB, IN2)          # Test Button B  
        TestButton(BUTTONC, IN3)          # Test Button C  
        TestButton(BUTTOND, IN4)          # Test Button D  
        time.sleep(0.1)  
  
    except KeyboardInterrupt:           # Ctrl+C detected  
        GPIO.cleanup()                 # Clean GPIO  
        print("End of program")         # End of program
```

Figure 7.4: Program: *relays.py*.

Figure 7.5 shows the circuit constructed on a breadboard. An external +5 V power supply was used to provide external voltage to the relay module.

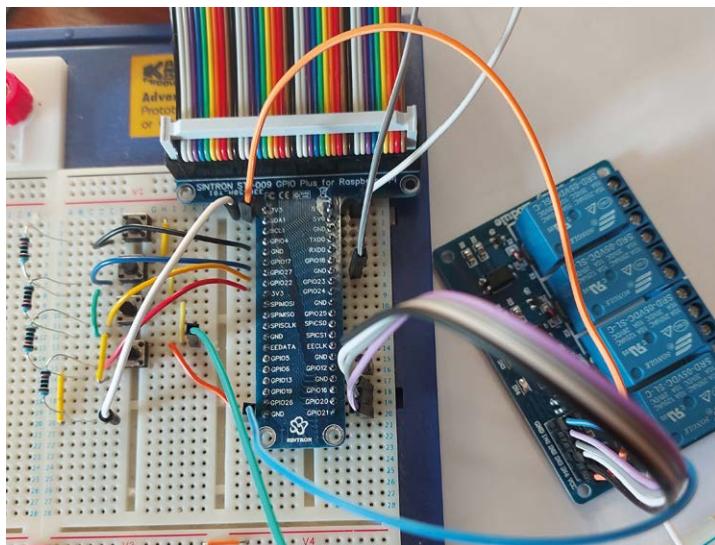


Figure 7.5: Project constructed on a breadboard.

CHAPTER 8 • Station Clock

8.1 Project

Description: In this project we will develop an LCD clock to display the current date and time. When Raspberry Pi boots, it receives the current data and time from the Internet automatically (if connected to the Internet). The date and time are displayed every second on a two-row LCD, where the date is displayed at the top row and the time is displayed at the bottom row.

Aim: The aim of this project is to show how an LCD can be interfaced to a Raspberry Pi, and how the current date and time can be received and displayed on the LCD.

Block Diagram: Figure 8.1 shows the block diagram of the project. In this project an I²C type LCD is used

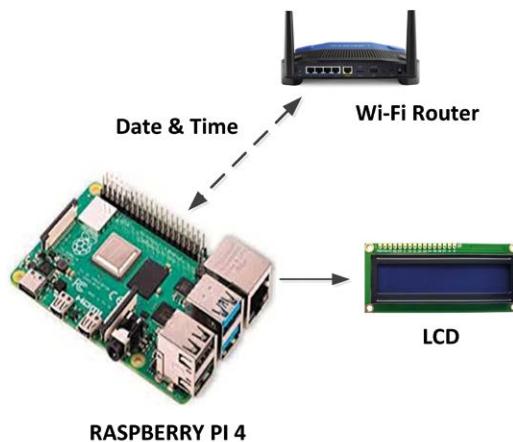


Figure 8.1: Block diagram of the project.

LCDs are used as display devices in many microcontroller-based projects. Basically, there are two types of LCD: parallel and I²C-based. Parallel LCDs are usually controlled with four data lines and two control lines. The main advantage of these types of LCDs is their low cost. Parallel LCDs have the disadvantage that an external potentiometer is required to adjust the contrast of their screens. I²C-based LCDs are basically parallel LCDs with the addition of a small board at the back of the LCD which translates the I²C signals to parallel signals. The advantage of the I²C based LCDs is that their control requires only four wires, and the contrast adjusting potentiometer is located on the I²C board. They are however more expensive than the standard parallel LCDs.

Circuit Diagram: The I²C LCD has four pins: GND, +V, SDA, and SCL. SDA is connected to pin GPIO2 and SCL is connected to pin GPIO3 of the Raspberry Pi respectively. Notice that there is no problem mixing the +3.3 V GPIO pins of the Raspberry Pi with the +5 V of the I²C LCD. This is because the Raspberry Pi is the I²C master device and the SDA and SCL lines are pulled up to +3.3 V through resistors. SCL line is the clock which is always output

from the master device. The slave device (I²C LCD here) only pulls down the SDA line when it acknowledges the receipt of data and it does not send any data to the master device. Therefore, there are no voltage level problems as long as the Raspberry Pi I²C output pins can drive the I²C LCD inputs, which is the case here.

Figure 8.2 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.

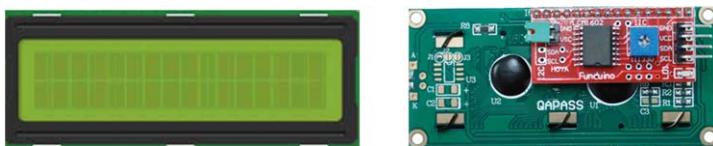


Figure 8.2: I²C based LCD (front and back views).

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.

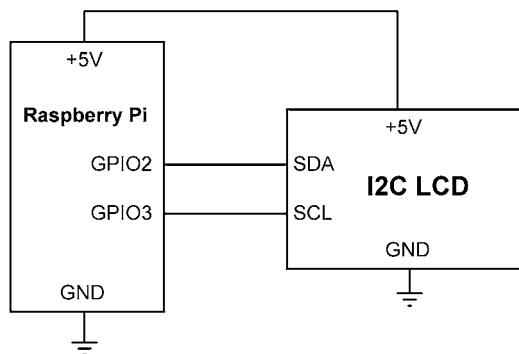


Figure 8.3: Circuit diagram of the project.

Before using an I²C device we have to enable the I²C in our Raspberry Pi configuration. The steps are (Figure 8.4):

- Start the configuration tool

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

- Move down to Interface Options and press Enter.

- Select I²C and press Enter to enable it.

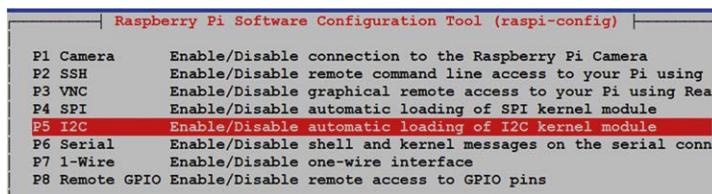


Figure 8.4: Enable I²C on your Raspberry Pi.

The I²C LCD default slave address is 0x27. Before using the I²C pins of the Raspberry Pi we have to make sure that the LCD is recognized by the Raspberry Pi I²C bus. Build your circuit and enter the following command at the command line and ensure that I²C address 27 is displayed as shown in Figure 8.5:

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
```

```
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: -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Figure 8.5: The I²C address of the LCD is 0x27.

Before using an LCD with a Raspberry Pi, we should install an I²C LCD library so that we can send commands and data to our LCD. There are many Python libraries available for the I²C type LCDs. The one chosen here is called the **RPLC**. This library is installed as follows.

- Enter the following commands:

```
pi@raspberrypi:~ $ sudo pip3 install RPLCD
pi@raspberrypi:~ $ sudo apt-get install python-smbus
```

Raspberry Pi receives the current date and time from the Internet at reboot time if it is enabled to connect to the local Wi-Fi. We have to make sure that the Time Zone is set correctly in our Raspberry Pi. The steps are:

- Start the configuration utility:

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

- Select **Localisation Options** and press Enter.
- Select **Change Timezone** (see Figure 8.6).

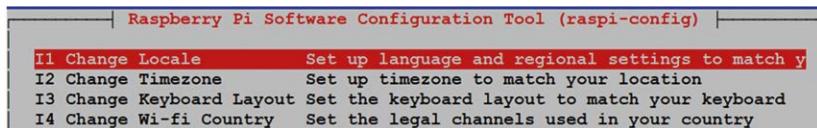


Figure 8.6: Select Change Timezone.

- Select your **Geographical area** (e.g. Europe).
- Select your nearest city (e.g. London) and press Enter.
- Click **Finish**.

Program Listing: We are now ready to write our program. Figure 8.7 shows the program listing (program: **stationtim.py**). At the beginning of the program libraries **RPI.GPIO**, **time**, **date**, and the LCD driver library **RPLCD.i2c** are all imported to the program. A new instance of the LCD is created using statement **LCD = CharLCD('PCF8574', 0x27)**. Notice that in this project we are using PCF8574 type port expander with I²C address 0x27 for the LCD. Some other compatible port expanders are: MCP23008 and MCP23017.

The parameters of the LCD can be configured if required. For example, a 20-column by 4-row LCD with a PCF8574 type port expander IC can be configured as follows (see web link: <https://readthedocs.org/projects/rplcd/downloads/pdf/latest/>):

```
LCD = CharLCD(i2c_expander='PCF8574', address=0x27, port=1, cols=20, rows=4, dot-size=8, charmap='A02', auto_linebreaks=True, backlight_enabled=True)
```

The program then clears the LCD using function **lcd.clear()**. The remainder of the program runs in an endless loop formed using a while statement. Inside this loop the current date is read from the Raspberry Pi and is formatted as **dd-mm-yyyy**, and gets displayed at the top row of the LCD. Then the current time is read from the Raspberry Pi and is formatted as **HH:MM:SS**, and is displayed at the bottom row of the LCD. Notice that the cursor is positioned using function **cursor_pos = (row, column)** where the row and column numbers start from 0, i.e. the first column at the top row is (0, 0), the first column at the bottom row is (1, 0). String data is displayed on the LCD using function **lcd.write_string()**. This process is repeated every second until terminated by the user after entering **Cntrl+C**.

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

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

```

#-----
#           STATION DATE AND TIME
#-----  

# In this project an I2C LCD is connected to the Raspberry Pi.  

# The program gets the current date and time and displays on  

# the LCD  

#  

# Author: Dogan Ibrahim  

# File  : stationtim.py  

# Date   : July, 2020  

#-----  

import RPi.GPIO as GPIO
from RPLCD.i2c import CharLCD
import time
from datetime import date

LCD = CharLCD('PCF8574',0x27)
GPIO.setwarnings(False)

#  

# Get current date and time. Display date at first row and time  

# in the second row of the LCD  

#  

try:

    while True:
        LCD.clear()
        today = date.today()
        LCD.cursor_pos=(0,0)
        LCD.write_string(date.today().strftime("%d-%m-%Y"))
        LCD.cursor_pos=(1,0)
        LCD.write_string(time.strftime("%H:%M:%S"))
        time.sleep(1)

except KeyboardInterrupt:                      # Cntrl+C detected
    GPIO.cleanup()                           # Clean GPIO
    print("End of program")                 # End of program

```

Figure 8.7: Program: stationtim.py.

Figure 8.8 shows an example display on the LCD.

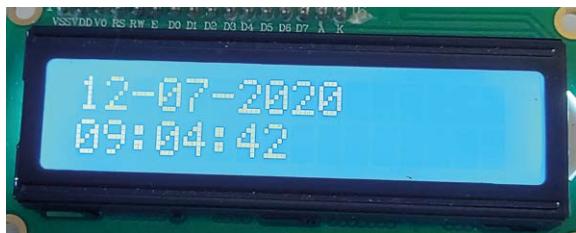


Figure 8.8: Example display on the LCD.

8.2 Real-time clock

The Raspberry Pi does have a built-in real time clock (RTC). The project we have just developed gets the date and time from the Internet. However, there are cases when the Internet may not be available. In the project in this section we will use a real-time clock module to provide clock to the Raspberry Pi as soon as it is started.

Circuit Diagram: In this project the real-time clock module shown in Figure 8.9 is used. This module, known as the **MakerHawk RPI DS1307 RTC module**, is based on the DS1307 RTC chip, operated with a CR1220 type coin battery (usually supplied with the module).

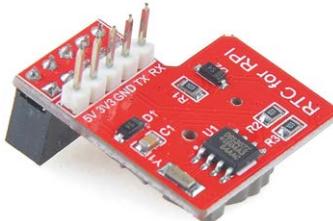


Figure 8.9: MakerHawk RPI DS1307 RTC module.

The module has a 10-way header and is plugged-in on top of Raspberry Pi, away from the USB sockets as shown in Figure 8.10.



Figure 8.10: Plug-in the RTC module to the Raspberry Pi.

The DS1307 chip operates with the I²C protocol and the connections between the RTC module and the Raspberry Pi GPIO pins are as follows (notice that this only applies to DS1307 type RTC chips):

DS1307 module pin	Raspberry Pi pin
Vin	+5V (pin 4)
SDA	GPIO2 (pin 3)
SCL	GPIO3 (pin 5)
GND	GND (pin 6)

We now have to configure the Raspberry Pi so that it takes the date and time from the RTC module. The steps are:

- Plug-in the RTC module as shown in Figure 8.10, making sure the battery is installed in the module
- pi@raspberrypi:~ \$ **sudo apt-get update**
- pi@raspberrypi:~ \$ **sudo apt-get upgrade**
- make sure that I2C is enabled in raspi-config (see the previous project)
- pi@raspberrypi:~ \$ **sudo apt-get install python-smbus i2c-tools**
- pi@raspberrypi:~ \$ **sudo i2cdetect -y 1**

You should see a display similar to the one shown in Figure 8.11, where 68 is the address of the RTC module:

```
pi@raspberrypi:~ $ 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: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
68: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Figure 8.11: 68 is the RTC module address.

- Modify the boot configuration file using the **nano** editor:
- pi@raspberrypi:~ \$ **sudo nano /boot/config.txt**
- Add the following line to the end of the file (for DS1307 type modules only):

dtoverlay=i2c-rtc,ds1307

- Enter **Cntrl+X** followed by **Y** to save the changes.
- Restart the Raspberry Pi:

```
pi@raspberrypi:~ $ sudo reboot
```

- Enter the following command. You should now see **UU** instead of 68:

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
```

- We now have to remove the fake **hwclock** package which acts as a placeholder for the RTC. Enter the following commands:

```
pi@raspberrypi:~ $ sudo apt-get -y remove fake-hwclock  
pi@raspberrypi:~ $ sudo update-rc.d -f fake-hwclock remove
```

- Edit the following file:

```
pi@raspberrypi:~ $ sudo nano /lib/udev/hwclock-set
```

Find and comment the following three lines by inserting **#** character at the beginning of the lines, i.e. change:

```
if [ -e /run/systemd/system ] ; then  
    exit 0  
fi
```

to:

```
#if [ -e /run/systemd/system ] ; then  
#    exit 0  
#fi
```

- Enter **Cntrl+X** followed by **Y** to save the changes.
- Make sure the date and time are correct on your Raspberry Pi as we will load them to the RTC chip. Use the **date** command to check:

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

- Write the date and time to the RTC module. This command will not be required again if the battery is kept on the RTC module so that the date and time are correct:

```
pi@raspberrypi:~ $ sudo hwclock -w
```

- You can read the RTC date and time with the command:

```
pi@raspberrypi:~ $ sudo hwclock -r
```

Our Raspberry Pi now reads the time automatically from the RTC chip so that the date and time will be correct even if the Internet isn't available at power-on.

CHAPTER 9 • Why Multitasking?

In the last Chapters we have developed two projects: the **Station Mains On/Off Control**, and the **Station Clock**. It is possible that we wish to run both projects at the same time. Does this mean that we need two Raspberry Pis?

The answer to the above question is NO. The solution to the above question is to have one Raspberry Pi only and run both programs at the same time in a time-sharing manner as a multitasking approach.

This idea is illustrated by a simple example with two programs named **flash1000** and **flash250**, where **flash1000** flashes an LED at GPIO2 every 1000 ms, and **flash250** flashes another LED at GPIO3 every 250 ms. The problem is that we wish to run both programs at the same time so that both LEDs flash at the required rates.

Figure 9.1 shows the listing of program **flash1000.py**. Similarly, Figure 9.2 shows the listing of program **flash250.py**.

```
-----#
#           FLASH LED EVERY SECOND
# =====
#
# This program flashes the LED at GPIO2 every second
#
# Author : Dogan Ibrahim
# File   : flash1000.py
# Date   : July 2020
#-----#
import RPi.GPIO as GPIO          # Import RPi
import time
GPIO.setwarnings(False)          # Disable warnings

LED = 2                         # LED at GPIO2
GPIO.setmode(GPIO.BCM)           # GPIO mode
GPIO.setup(LED, GPIO.OUT)         # LED is output

while True:                      # Flash the LED
    GPIO.output(LED, 1)           # LED ON
    time.sleep(1)                 # Wait 1 second
    GPIO.output(LED, 0)           # LED OFF
    time.sleep(1)                 # Wait 1 second
```

Figure 9.1: Program: flash1000.py.

```
#-----
#          FLASH LED EVERY 250ms
#          =====
#
# This program flashes the LED every 250ms
#
# Author : Dogan Ibrahim
# File   : flash250.py
# Date   : July 2020
#-----
import RPi.GPIO as GPIO      # Import RPi
import time
GPIO.setwarnings(False)      # Disable warnings

LED = 3                      # LED at GPIO3
GPIO.setmode(GPIO.BCM)        # GPIO mode
GPIO.setup(LED, GPIO.OUT)     # LED is output

while True:                  # Flash the LED
    GPIO.output(LED, 1)        # LED ON
    time.sleep(0.25)           # Wait 250ms
    GPIO.output(LED, 0)        # LED OFF
    time.sleep(0.25)           # Wait 250ms
```

Figure 9.2: Program: flash250.py.

There are several methods that we can achieve multitasking in a Python program. Here, in order to start both programs at the same time we will be using the subprocess approach. Create a program called **ledsubprocess.py** with the contents as shown in Figure 9.3 and run this program as follows:

```
pi@raspberrypi:~ $ python3 ledsubprocess.py
```

```
#-----
#          START PROGRAMS TO FLASH 2 LEDs
#          =====
#
# This program starts programs flash1000.py and flash250.py
# to flash two LEDs at the rates 1 second and 250ms
#
# Author: Dogan Ibrahim
# File  : ledsubprocess.py
# Date  : July 2020
#-----
import subprocess
print("Start flash1000.py")
```

```

subprocess.Popen(["python3", "flash1000.py"])
print("Start flash250.py")
subprocess.Popen(["python3", "flash250.py"])

```

Figure 9.3: Program: ledsubprocess.py.

Notice that the two programs **flash1000.py** and **flash250.py** will run as independent programs on the Raspberry Pi.

You can terminate the programs with the command:

```
pi@raspberrypi:~ $ sudo killall python3
```

The Station programs

We will now use the same method to run the two programs **relays.py** and **stationtim.py** developed in the last Chapters at the same time on our Raspberry Pi. Create a program called **mystation.py** with the contents as shown in Figure 9.4.

```

#-----
#           START PROGRAMS relays.py and stationtim.py
#-----=====
#
# This program starts programs relays.py and stationtim.py
# at teh same time. You should be able to control the mains
# power to your equipment and at teh same time see the current
# date and time on teh LCD
#
# Author: Dogan Ibrahim
# File  : mystation.py
# Date  : July 2020
#-----
import subprocess
print("Start relays.py")
subprocess.Popen(["python3", "relays.py"])
print("Start stationtim.py")
subprocess.Popen(["python3", "stationtim.py"])

```

Figure 9.4: Program: mystation.py.

Run the program by entering the following command:

```
pi@raspberrypi:~ $ python3 mystation.py
```

You should now be able to control the mains (US: line) power to your equipment, and at the same time see the current date and time on the LCD.

Notice that using the subprocess method is not limited to two programs only; you can run as many programs as you like.

You can terminate the programs with the command:

```
pi@raspberrypi:~ $ sudo killall python3
```

Auto startup

We can insert **mystation.py** into file **/etc/rc.local** so that both programs **relays.py** and **stationtim.py** start automatically after system startup. The steps are as follows:

- Edit file **/etc/rc.local** using the nano editor
- Go to the end of the file (before exit) and enter the following statement:

```
python3 /home/pi/mystation.py&
```

- Exit nano by entering **Cntrl+X** followed by **Y**
- Edit file **mystation.py** and modify it so that full path of files **relays.py** and **stationtim.py** are specified. The modified file (**mystation2.py**) is shown in Figure 9.5
- Restart your Raspberry Pi. You should see that the date and time are displayed every second and at the same time the relay program is working correctly.
- Don't forget to remove the above changes if you do not want the programs to start automatically after a system restart.

```
-----  
#-----  
#           START PROGRAMS relays.py and stationtim.py  
#-----  
#  
# This program starts programs relays.py and stationtim.py  
# at the same time. You should be able to control the mains  
# power to your equipment and at the same time see the current  
# date and time on the LCD  
#  
# Author: Dogan Ibrahim  
# File  : mystation2.py  
# Date  : July 2020  
#-----  
import subprocess  
print("Start relays.py")  
subprocess.Popen(["python3", "/home/pi/relays.py"])  
print("Start stationtim.py")
```

```
subprocess.Popen(["python3", "/home/pi/stationtim.py"])
```

Figure 9.5: Program: mystation2.py.

CHAPTER 10 • The Station Temperature and Humidity

10.1 Project

There are cases where we may want to know the ambient temperature and humidity to send it over the air in a suitable format. This project shows how to read the ambient temperature and humidity and display both values on an LCD.

Description: In this project we will use a sensor to read the ambient temperature and humidity and then display them on an LCD every five seconds.

Aim: The aim of this project is to show how the temperature and humidity can be read and displayed on an LCD.

Block Diagram: Figure 10.1 shows the block diagram of the project. A DHT11 type temperature and humidity sensor chip is used in this project.



Figure 10.1: Block diagram of the project.

Circuit diagram: The type DHT11 temperature and humidity sensor chip 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 the ground and the +3.3 V power supply pins of the Raspberry Pi. The Data pin must be connected to +V through a 10 kohm resistor. In this project a 3-pin DHT11 from Elektor is used with built-in 10 kohm pull-up resistor. As shown in Figure 10.2, the Data pin of the sensor is named as S and it is connected to GPIO26 of the Raspberry Pi 4.

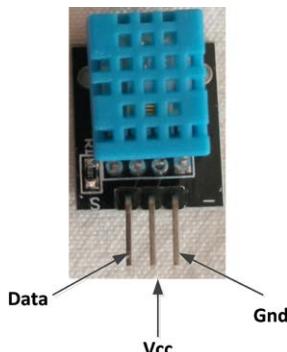


Figure 10.2: DHT11 sensor.

The DHT11 uses capacitive humidity sensor and a thermistor to measure the ambient temperature. Data output is available from the chip around every 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 °C with an accuracy of ±2 °C
- Humidity reading in the range 20-80% with 5% accuracy
- Breadboard compatible with 0.1 inch pin spacings

Figure 10.3 shows the circuit diagram of the project.

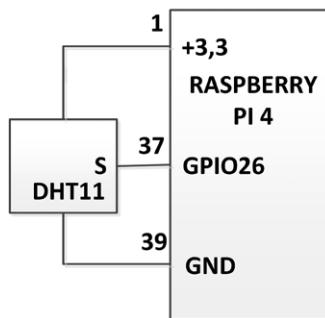


Figure 10.3: Circuit diagram of the project.

Program listing: In this program the Adafruit DHT11 library module is used. This module should be installed into Raspberry Pi before it can be used. The instructions for this are as follows:

- Go to command mode and enter:

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

- Change directory to:

```
cd Adafruit_Python_DHT
```

- Enter the following commands:

```
sudo apt-get install build-essential python-dev
sudo python setup.py install
```

Figure 10.4 shows the program listing (program: **dht11.py**). At the beginning of the program the following libraries are imported to the program:

```
RPi.GPIO  
time  
RPLCD driver  
Adafruit_DHT
```

Pin 26 is configured as the DHT11 output pin. Then the LCD is cleared, and the text **Temp and Hum** is displayed at the top row of the LCD. The remainder of the program runs in a loop formed using a **while** statement. Inside this loop, the program reads the temperature and humidity from the DHT11 chip and displays at the bottom row of the LCD every five seconds. The temperature and humidity readings are converted into string before being displayed on the LCD. The program detects the **Ctrl+C** interrupts and terminates the program cleanly.

```
#-----  
#           DISPLAY TEMPERATURE AND HUMIDITY ON LCD  
#           ======  
#  
# In this project a DHT11 type temperattrue and humidity sensor is  
# connected to the Raspberry Pi. The project displays the humidity  
# and temperature on the display every 5 seconds  
#  
# Author: Dogan Ibrahim  
# File : dht11.py  
# Date : July 2020  
#-----  
import RPi.GPIO as GPIO  
import time  
from RPLCD.i2c import CharLCD          # Import I2C LCD library  
import Adafruit_DHT                      # Adafruit DHT11 library  
LCD = CharLCD('PCF8574',0x27)  
sensor = Adafruit_DHT.DHT11  
S = 26                                     # DHT11 on GPIO 26  
GPIO.setwarnings(False)  
GPIO.setmode(GPIO.BCM)                      # GPIO mode BCM  
  
try:  
  
    while True:                            # Do forever  
        humidity,temperature = Adafruit_DHT.read_retry(sensor, S)  
        LCD.clear()  
        LCD.cursor_pos=(0,0)  
        LCD.write_string("Temp and Hum")  
        temp = str(temperature)[:4]  
        hum = str(humidity)[:4]  
        dsp = temp + "C " + hum + "%"  
        LCD.cursor_pos=(1,0)
```

```
LCD.write_string(dsp)          # Display T and H  
time.sleep(5)                  # Wait 5 seconds  
  
except KeyboardInterrupt:  
    GPIO.cleanup()  
    print("End of program")
```

Figure 10.4: dht11.py program listing.

Figure 10.5 shows an example display of the temperature and the humidity on the LCD.



Figure 10.5: Example display.

CHAPTER 11 • Station Mains On-Off Control, Station Time, and Station Weather

This is a multitasking project. In this project we combine the last three projects into one, i.e. control the mains (US: line) power to the equipment, display the current date and time on the LCD, and also display the temperature and humidity on the same LCD.

Notice that in this project two programs share the same LCD, where the date and time is displayed at every 5 seconds, and the temperature and humidity is displayed at every 12 seconds. It is important to note that in this project we cannot use subprocesses as described in Chapter 9. This is because when we use subprocesses, each process is independent of each other and they do not share any memory variables or any memory space. In this program we use process forks which is another method of using multitasking in Python programs. Using forks has the advantage that the processes are under tighter control and this is what we want in this project.

Block Diagram: Figure 11.1 shows the block diagram of the project.

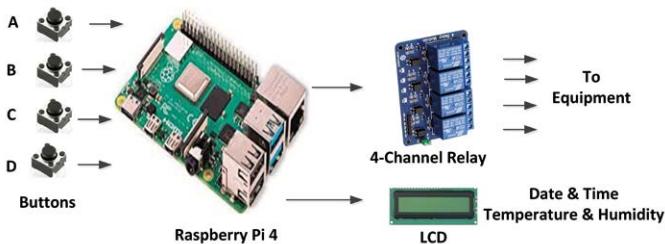


Figure 11.1: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 11.2. The SDA and SCL pins of the I²C LCD are connected to GPIO2 and GPIO3 pins respectively as in the earlier project. Pin S of the DHT11 is connected to pin GPIO26 of the Raspberry Pi.

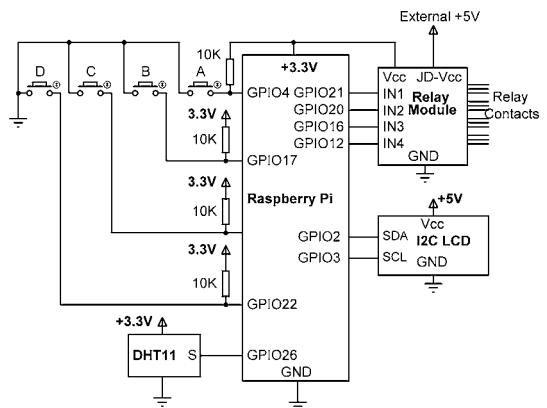


Figure 11.2: Circuit diagram of the project.

Program Listing: The program listing is shown in Figure 11.3 (program: **all.py**). This program is based on using operating system forks. A fork creates a child process which runs in a multitasking environment on the Raspberry Pi. The program is in three sections where each section is created as a function using the **def** statement. Function **relay()** implements the station mains ON-OFF control, function **tim()** displays the current station date and time, and function **dht()** reads and displays the temperature and humidity.

A *fork* process creates a *child* process by calling the operating system function **os.fork()**. The created child process has the ID of 0. As you will see in Figure 11.3, three child processes are created to implement the required operations. The current date and time are displayed at every 5 seconds. The temperature and humidity readings are displayed at every 13 seconds so that there is no race with the date and time display.

```
#-----
#     STATION RELAY, STATION TIME, AND STATION WEATHER
# -----
#
# In this project the programs relays.py, stationtim.py, and dht11.py
# are combined and they all run on the same Raspberry Pi. As a result,
# we can switch ON/OFF eh mains power to an equipment, display the
# current date and time on the LCD, and display the temperature and
# humidity on the same LCD
#
# Author: Dogan Ibrahim
# File  : all.py
# Date  : July, 2020
#-----

#
#===== STATION RELAY =====
#
import RPi.GPIO as GPIO
import time
import os
GPIO.setwarnings(False)

#
# Button connections
#
BUTTONA = 4                      # Button A
BUTTONB = 17                      # Button B
BUTTONC = 27                      # ButtonC
BUTTOND = 22                      # Button D

OFF = 1
ON = 0
```

```
#  
# Relay connections  
  
IN1 = 21 # IN1 pin  
IN2 = 20 # IN2 pin  
IN3 = 16 # IN3 pin  
IN4 = 12 # IN4 pin  
  
GPIO.setmode(GPIO.BCM) # Set BCM mode  
  
#  
# IN1-IN4 are Outputs  
  
GPIO.setup(IN1, GPIO.OUT) # IN1 is output  
GPIO.setup(IN2, GPIO.OUT) # IN2 is output  
GPIO.setup(IN3, GPIO.OUT) # IN3 is output  
GPIO.setup(IN4, GPIO.OUT) # IN4 is output  
  
#  
# BUTTONA-BUTTOND are inputs  
  
GPIO.setup(BUTTONA, GPIO.IN) # Button A is input  
GPIO.setup(BUTTONB, GPIO.IN) # Button B is input  
GPIO.setup(BUTTONC, GPIO.IN) # Button C is input  
GPIO.setup(BUTTOND, GPIO.IN) # Button D is input  
  
#  
# Turn OFF all relays at the beginning  
  
GPIO.output(IN1, OFF) # IN1 OFF  
GPIO.output(IN2, OFF) # IN2 OFF  
GPIO.output(IN3, OFF) # IN3 OFF  
GPIO.output(IN4, OFF) # IN4 OFF  
  
#  
# This function checks if Button is pressed and toggles pin IN  
# of the relay If the relay os ON, it is tured OFF, if it is OFF,  
# it is turned ON  
  
#  
def TestButton(Button, IN):  
    if GPIO.input(Button) == 0:  
        s = GPIO.input(IN)  
        if s == ON:  
            GPIO.output(IN, OFF)  
        else:  
            GPIO.output(IN, ON)
```

```

        while GPIO.input(Button) == 0:
            pass
#
# Start of program relay
#
def relay():
    pid1 = os.fork()                                # Create child
    if pid1 == 0:                                    # Child?
        while True:                                 # Do forever
            TestButton(BUTTONA, IN1)                 # Test Button A
            TestButton(BUTTONB, IN2)                 # Test Button B
            TestButton(BUTTONC, IN3)                 # Test Button C
            TestButton(BUTTOND, IN4)                 # Test Button D
            time.sleep(0.15)

#
#===== STATION TIME =====
#
from RPLCD.i2c import CharLCD
from datetime import date
LCD = CharLCD('PCF8574',0x27)

#
# Get current date and time. Display date at first row and time
# in the second row of the LCD
#
def tim():
    pid2 = os.fork()                                # Create child
    if pid2 == 0:                                    # Child?
        while True:                                 # Do forever
            LCD.clear()                            # Clear LCD
            today = date.today()                  # Get date
            LCD.cursor_pos=(0,0)
            LCD.write_string(date.today().strftime("%d-%m-%Y"))
            LCD.cursor_pos=(1,0)
            LCD.write_string(time.strftime("%H:%M:%S"))
            time.sleep(5)

#
#===== TEMPERATURE AND HUMIDITY =====
#
import Adafruit_DHT                                # Adafruit DHT11 library
sensor = Adafruit_DHT.DHT11
S = 26                                              # DHT11 on GPIO 26

```

```
def dht():
    pid3 = os.fork()                                # Create Child
    if pid3 == 0:                                    # Child?
        while True:                                 # Do forever
            humidity,temperature = Adafruit_DHT.read_retry(sensor, S)
            LCD.clear()
            LCD.cursor_pos=(0,0)
            LCD.write_string("Temp and Hum")
            temp = str(temperature)[:4]
            hum = str(humidity)[:4]
            dsp = temp + "C " + hum + "%"
            LCD.cursor_pos=(1,0)
            LCD.write_string(dsp)                      # Display T and H
            time.sleep(13)                            # Wait 15 seconds

#
# Create the fork processes
#
relay()
tim()
dht()
```

Figure 11.3: Program: all.py.

We can terminate the programs by entering the following command:

```
pi@raspberrypi:~ $ sudo killall python3
```

CHAPTER 12 • Station Geographical Coordinates

There are cases, especially when working mobile that we may want to know the geographical coordinates (e.g. latitude and longitude) of our station. In this project we use a GPS receiver to read and display the geographical coordinates of our station on an LCD.

Description: GPS receivers receive geographical data from the GPS satellites and they provide accurate information about the position of the user on Earth. These satellites circle the Earth at an altitude of about 20,000 kms and complete two full orbits every day. For a receiver to determine its position on the globe, it is necessary to 'see' i.e. communicate with, at least three satellites. Therefore, if the receiver does not have clear view of the sky, it may not be possible to determine its position on Earth. In some applications external antennas are used so that even weak signals can be received from the GPS satellites.

The data sent out from a GPS receiver is in text format and is known as the NMEA Sentences. Each NMEA sentence starts with a \$ character and the values in a sentence are separated by commas. Some of the NMEA sentences returned by a GPS receiver are given below:

\$GPGLL: This sentence returns the local geographical latitude and longitude

\$GPRMC: This sentence returns the local geographical latitude and longitude, speed, track angle, date, time, and magnetic variation.

\$GPVTG: This sentence returns the true track, magnetic track, and the ground speed.

\$GGGA: This sentence returns the local geographical latitude and longitude, time, fix quality, number of satellites being tracked, horizontal dilution of position, altitude, height of geoid, and DGPS data

\$GPGSV: There are four sentences with this heading. These sentences return the number of satellites in view, satellite number, elevation, azimuth, and SNR.

In this project the GPS Click board from Mikrolektronika (www.mikroe.com) is used. This is a small GPS receiver (see Figure 12.1) which is based on the LEA-6S type GPS. This board operates with +3.3 V and provides two types of outputs: I²C or serial output. In this project the default serial output is used which operates at 9600 baud (bits/s). An external dynamic antenna can be attached to the board in order to improve poor reception indoors, or in places not having clear view of the sky.



Figure 12.1: MikroE GPS Click board.

Figure 12.2 shows the complete list of the NMEA sentences output from the GPS Click board every second.

```
$GPGLL,5127.3917,N,00003.13141,E,10534.00,A,A*67
$GPRMC,05305.00,A,5127.35909,,0003.13148,E,0.030,,270919,,,A*7E
$GPVTG,,T,M,0030,N,0,055,K,A*20
$GGGA,105305.00,5127.35909,N,0003.13148,E,1.09,1.18,46.5,M,45.4,M,,*66
$GPSA,R,3,01,32,08,28,18,03,22,14,11,,2,12,1,18,1,76*06
$GPGSV,4,1,13,01,7,304,40,03,40,224,31,08,38,165,32,10,05,054,*77
$GPGSV,4,2,13,11,83,217,3,14,39,094,24,17,17,314,22,18,73,091,41*76
$GPGSV,4,3,13,22,63,219,33,24,1,002,,27,05,150,,28,30,284,28*7F
$GPGSV,4,4,13,32,34,063,35*4E
```

Figure 12.2: NMEA sentences output from the GPS Click board.

The MikroE GPS Click board is a 2×8 pin dual-in-line module and it has the following pin configuration (pin 1 is the top left pin of the module):

1: No connection	16: No connection
2: Reset	15: No connection
3: No connection	14: TX
4: No connection	13: RX
5: No connection	12: SCL
6: No connection	11: SDA
7: +3.3V	10: No connection
8: GND	9: GND

In serial operation only the following pins are required: +3.3 V, GND, TX. In this project an external dynamic antenna is attached to the GPS Click board as it was used indoors.

\$GPGLL is one of the commonly used NMEA sentence and this is the sentence used in this project to extract the station geographical coordinates. This sentence is output as follows:

\$GPGLL,5127.37032,N,00003.12782,E,221918.00,A,A*61

The fields in this sentence can be decoded as follows:

GLL	Geographic position, latitude and longitude
5127.37032	Latitude 51 deg, 27.3702 min. North
00003.12782	Longitude 0 deg, 3.12782 min. East
221918	Fix taken at 22L19L18 UTC
A	Data active (or V for void)
*61	checksum data

Notice that the fields are separated by commas. The validity of the data is shown by letters **A** or **V** in the data, where A shows that the data is valid, and V indicates that the data is not valid.

Block Diagram: Figure 12.3 shows the block diagram of the project. An I2C LCD is used to display the attitude and longitude of the user GPS receiver

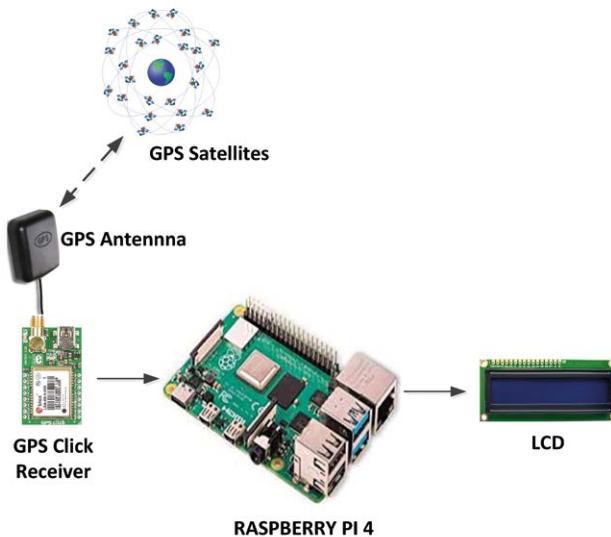


Figure 12.3: Block diagram of the project.

The Raspberry Pi 4 computers have two built-in hardware UARTs: a PL011 and a mini UART. These are implemented using different hardware blocks, so they have slightly different characteristics. Since both are 3.3 V devices, extra care must be taken when connecting to other serial communication lines. On Raspberry Pis equipped with the Wireless/Bluetooth modules (e.g. Raspberry Pi 3, Zero W, 4, etc), the PL011 UART is connected to the Bluetooth module by default, while the mini UART is the primary UART with the Linux console on it. In all other models, the PL011 is used as the primary UART. By default, `/dev/ttys0` refers to the mini UART and `/dev/ttAMA0` refers to the PL011. The Linux console uses the primary UART which depends on the Raspberry Pi model used. Also, if enabled, `/dev/`

serial0 refers to the primary UART (if enabled), and if enabled, **/dev/serial1** refers to the secondary UART.

By default, the primary UART (serial0) is assigned to the Linux console. Using the serial port for other purposes requires this default configuration to be changed. On startup, **systemd** checks the Linux kernel command line for any console entries, and will use the console defined therein. To stop this behaviour, the serial console setting needs to be removed from command line. This is easily done by using the **raspi-config** utility by selecting option **5** (Interfacing options) and then **P6** (Serial), then select **No**. Exit **raspi-config** and restart your Raspberry Pi. You should now be able to access the serial port. Don't forget to re-enable the console setting after you finish.

*On the Raspberry Pi 3 and 4 the serial port (**/dev/ttys0**) is routed to two pins GPIO14 (TXD) and GPIO15 (RXD) on the GPIO header. Models earlier than 3 uses this port for Bluetooth.*

We can easily search for the available serial ports by entering the following console command:

```
pi@raspberrypi:~ $ dmesg | grep tty
```

The last line in the output below indicates that the console is enabled on serial port **ttys0**.

```
console [ttys0] enabled
```

In this book we are using the Raspberry Pi 4 whose serial port is: **/dev/ttys0**. If you are using a model earlier than 3, use the serial port named: **/dev/ttymA0**.

Circuit Diagram: The circuit diagram of the project is shown in Figure 12.4. The UART TX pin of the GPS click board (pin 14) is connected to the RXD input (GPIO15) of the Raspberry Pi. The I²C LCD is connected as in the previous projects. i.e. the SDA and SCL pins are connected to GPIO2 and GPIO3 respectively. The GPS click board is powered from the +3.3 V supply of the Raspberry Pi.

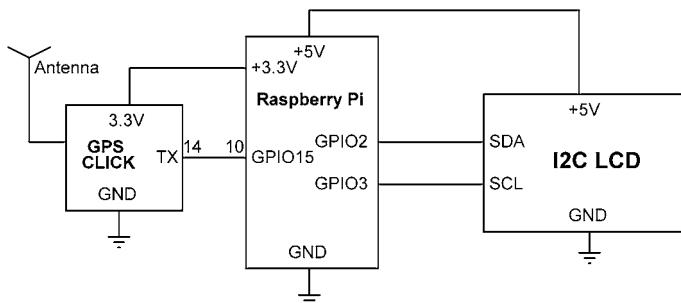


Figure 12.4: Circuit diagram of the project.

We can display all the GPS NMEA sentences sent by the GPS Click board by entering the following command. This command is useful for testing purposes since it verifies if the connection to the GPS is correct, and if the GPS is actually receiving data from the satellites:

```
pi@raspberrypi:~ $ sudo cat /dev/serial0
```

An example output is shown in Figure 12.5 (only part of the output is shown).

```
pi@raspberrypi:~ $ sudo cat /dev/serial0
$GPRMC,122152.00,A,5127.36312,N,00003.12907,E,0.133,,130720,,,A*7B
$GPVTG,,T,,M,0.133,N,0.245,K,A*21
$GPGGA,122152.00,5127.36312,N,00003.12907,E,1,09,1.44,51.4,M,45.4,M,,*6C
$GPGSA,A,3,27,20,30,21,07,08,01,10,11,,,2.21,1.44,1.68*58
$GPGSV,3,1,11,01,13,246,24,07,12,284,38,08,67,294,38,10,56,096,34*78
$GPGSV,3,2,11,11,38,269,34,20,35,056,21,21,52,069,34,26,02,167,*77
$GPGSV,3,3,11,27,76,114,38,30,11,314,34,32,00,128,*4A
$GPGLL,5127.36312,N,00003.12907,E,122152.00,A,A*64
```

Figure 12.5: Example output of NMEA sentences.

In this project, the latitude and longitude are extracted from the NMEA sentence \$GPGLL without using a library.

Program Listing: Figure 12.6 shows the program listing (program: **gps.py**). At the beginning of the program the following libraries are imported to the program:

```
RPi.GPIO
RPLCD
time
serial
```

Variable **port** is assigned to **/dev/serial0** which is the serial port name for Raspberry Pi 4. Function **Get_GPS()** receives a line of NMEA sentence and looks for string **\$GPGLL**. When this string is detected the line of sentence is broken down into parts separated by commas using built-in function **split(",")** and stored in **sdata**. If the 6th field is character **V** then it is assumed that the sentence is not valid (e.g. there is no satellite reception) and the text **NO DATA** is displayed at the top row of the LCD. Otherwise, the latitude and its direction are extracted from fields 1 and 2 and stored in variables **lat** and **latdir** respectively. The longitude and its direction are extracted from fields 3 and 4 and stored in variables **lon** and **londir** respectively.

The latitude is received in the format: **ddmm.mmmmmmD** which corresponds to **dd** degrees **mm.mmmmmm** minutes, and direction **D** which is **N** or **S**. Similarly, the longitude is received in the format: **ddmm.mmmmmmD** where **D** is **E** or **W**. The main program separates the degrees and minutes and displays them on the LCD. The latitude is displayed at the top row of the LCD in the format: **dd mm.mmmmmm D**, and the longitude is displayed as: **dd mm.mmmmmm D**.

```
#-----
#           STATION GEOGRAPHICAL COORDINATES
#-----#
# In this project a GPS receiver module (GPS CLICK) is connected
# to the serial input (GPIO15) of the Raspberry Pi. Additionally,
# an I2C LCD is connected. The program displays the latitude and
# longitude of the receiver location on the LCD
#
# Author: Dogan Ibrahim
# File  : gps.py
# Date  : July, 2020
#-----

import RPi.GPIO as GPIO          # Import RPi library
from RPLCD.i2c import CharLCD   # Import LCD library
import time                      # Import time library
import serial                     # Import serial
port = "/dev/serial0"            # Serial port
lat=latdir=lon=londir = "0"

LCD = CharLCD('PCF8574',0x27)
GPIO.setwarnings(False)

#
# This function receives and extracts the latitude and longitude
# from the NME sentence $PGLL
#
def Get_GPS(data):
    global lat,latdir,lon,londir
    dat = data.decode('utf-8')
    if dat[0:6] == "$GPGLL":
        sdata = dat.split(",")
        if sdata[6] == "V":
            LCD.clear()                # Clear LCD
            LCD.cursor_pos = (0, 0)      # At 0,0
            LCD.write_string("NO DATA")  # No data
            return
        lat = sdata[1]                  # Get latitude
        latdir = sdata[2]               # Latitude dir
        lon = sdata[3]                  # Get longitude
        londir = sdata[4]               # Longitude dir
        return

#
# Receive the GPS coordinates and display on the LCD
#
```

```

ser = serial.Serial(port,baudrate=9600,timeout=0.5)

try:

    while True:
        data = ser.readline()                      # Read a line
        Get_GPS(data)                            # Decode

        deg = lat[0:2]
        min = lat[2:]
        latitude = str(deg) + " " + str(min) + " " + str(latdir)

        deg = lon[0:3]
        min = lon[3:]
        longitude = str(deg) + " " + str(min) + " " + str(londir)

        LCD.clear()
        LCD.cursor_pos = (0, 0)
        LCD.write_string(latitude)            # Display latitude
        LCD.cursor_pos = (1, 0)
        LCD.write_string(longitude)          # Display longitude
        time.sleep(1)                        # Wait 1 seconds

    except KeyboardInterrupt:                # Ctrl+C detected
        ser.close()                         # Close serial
        print("End of program")             # End of program

```

Figure 12.6: Program: gps.py.

An example display on the LCD is shown in Figure 12.7

*Figure 12.7: Example display on the LCD.*

Using a second I²C LCD

In this and in the earlier projects using I²C LCD, the I²C device address is set to 0x27 by default. It is possible to use more than one LCD in a project as long as they have different I²C addresses. The address of the LCD can be set by soldering jumpers across the six pads marked as A0, A1, and A2 at the back of the LCD as shown in Figure 12.8:

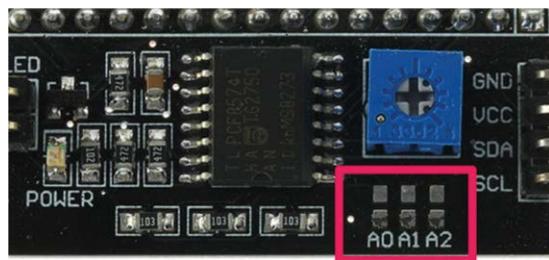


Figure 12.8: I²C LCD address jumpers

The address selection is as follows, where '1' corresponds to a jumper solder across the two specified pads:

A2	A1	A0	Address
No jumpers		0x27 (default)	
0	0	1	0x26
0	1	0	0x25
0	1	1	0x24
1	0	0	0x23
1	0	1	0x22
1	1	0	0x21
1	1	1	0x20

Finally, you should make sure that the correct device address is set in function **CharLCD**.

CHAPTER 13 • Waveform Generation — Using Software

Waveform generators are important tools for amateur radio operators. There's an abundance of commercially available equipment for generating various waveforms. In this Chapter we will be developing projects to generate various analogue waveforms such as square, sine, triangular, staircase, etc. by programming the Raspberry Pi.

Before generating an analogue waveform, it is necessary to use a Digital-to-Analog Converter chip (DAC) to convert the generated digital signals into analogue form. In this book we will be using the popular MCP4921 DAC chip from Microchip.

13.1 The MCP4921 DAC

Before using the MCP4921, it is worthwhile to look at its features and operation in some detail. MCP4921 is a 12-bit DAC that operates with the SPI bus interface. Figure 13.1 shows the pin layout of this chip. The basic features are:

- 12-bit operation
- 20 MHz clock support
- 4.5 μ s settling time
- External voltage reference input
- Unity ($1\times$) or $2\times$ gain control
- 2.7 to 5.5 V supply voltage
- -40°C to $+125^{\circ}\text{C}$ temperature range

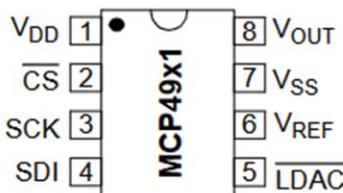


Figure 13.1: Microchip MCP4921 DAC.

The pin descriptions are:

Vdd:	supply voltage
CS:	chip select (active LOW)
SCK:	SPI clock
SDI:	SPI data in
LDAC:	Used to transfer input register data to the output (active LOW)
Vref	Reference input voltage
Vout:	analogue output
Vss:	supply ground

In this project we will be operating the MCP4921 with a gain of 1 (unity). As a result, with a reference voltage of 3.3 V and 12-bit conversion data, the LSB resolution of the DAC will be $3300 \text{ mV} / 4096 = 0.8 \text{ mV}$

The SPI Bus

Serial Peripheral Interface (SPI) bus consists of two data wires and one clock wire. Additionally, a chip enable (CE or CS) connection is used to select a slave in a multi-slave system. The wires used are:

MOSI (or SDI): Master Out Slave In. This signal is output from the master and is input to a slave.

MISO: Master In Slave Out. This signal is output from a slave and input to a master.

SCLK (or SCK): The clock, controlled by the master.

CE (or CS): Chip Enable (slave select).

There are four SPI operational modes known as mode 0 through mode 3. The mode determines the relationship between the clock pulses and the data pulses. Data can be read at the leading edge or at the trailing edge of the clock. CPOL (clock polarity) and CPHA (clock phase) determine the mode of operation. CPOL determines the polarity of the clock.

- In **mode 0**, both CPOL and CPHA are 0 and data is sampled at the leading rising edge of the clock.
- In **mode 2**, CPOL=1 and CPHA=0, data is sampled at the leading falling edge of the clock.
- In **mode 1**, CPOL=0 and CPHA=1 and the data is sampled on the trailing falling edge of the clock.
- In **mode 3**, CPOL=1 and CPHA=1 where the data is sampled on the trailing rising edge of the clock.

Mode 0,0 is used by the Raspberry Pi which is the most commonly used.

The following pins are the SPI bus pins on Raspberry Pi 4:

GPIO pin	SPI	Physical pin no.
GPIO10	MOSI	19
GPIO9	MISO	21
GPIO11	SCLK	23
GPIO8	24	CE0
GPIO7	26	CE1

The SPI bus must be enabled on the Raspberry Pi before it can be used. The steps are:

- Start the configuration tool:

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

- Select **Interfacing Options**

- Select **SPI**

- Select **Yes** to enable it

- Select **Finish**

- Select to reboot your Raspberry Pi

After the system comes up, enter the following command to confirm that the SPI bus has been enabled:

```
pi@raspberrypi:~ $ ls /dev/*spi*
```

You should get a response similar to:

```
/dev/spidev0.0 /dev/spidev0.1
```

This represents that there could be up to two SPI devices on chip enable pins 0 and 1 (we can have more SPI devices daisy-chained and sharing the same chip enable pin if we want)

The SPI bus on Raspberry Pi 4 supports the following functions:

Function	Description
open (0,0)	Open SPI bus 0 using CEO
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

13.2 Generating a squarewave signal with a peak voltage of 3.3 V

Description: As a first project we will be generating a squarewave signal. This project does not require the use of the DAC unless we want to control the output voltage level. In this project the output voltage will swing between 0 V and +3.3 V which is the logic HIGH level of a GPIO pin. In this project we will generate a 1 kHz positive only squarewave signal.

Block Diagram: Figure 13.2 shows the block diagram of the project. The output voltage was plotted using a Velleman type PCSGU250 oscilloscope & function generator. Pin GPIO26 is used as an output and is connected to the oscilloscope.



Figure 13.2: Block diagram of the project.

Program Listing: There are basically two options for generating squarewave signals using the Raspberry Pi. One option is to use the built-in PWM generator which is highly accurate. The second method, which is much easier but less accurate, is to switch ON-OFF a GPIO pin at the required rate. In this project we will be using the latter method for simplicity.

The period of a 1 kHz square wave is 1 ms. Assuming an equal ON and OFF times (i.e. 50% duty cycle), the ON and OFF times are each 0.5 ms. Figure 13.3 shows the program listing (Program: **squaresig.py**). At the beginning of the program, GPIO26 is configured as output and the period and half-period of the required waveform are stored in variables **period** and **halfperiod** respectively. The remainder of the program generates the waveform.

```
#-----
#          GENERATE SQUARE WAVEFORM
#          =====
#
# This program generates square waveform with the frequency 1kHz
#
# Author: Dogan Ibrahim
# File  : squaresig.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                      # GPIO mode BCM
Signal = 26                                  # GPIO26 is output
ON = 1
OFF = 0

GPIO.setup(Signal, GPIO.OUT)
frequency = 1000                             # Required Frequency
period = 1 / frequency                        # Period of the signal
```

```

halfperiod = period / 2           # Half of the period

try:

    while True:
        GPIO.output(Signal, ON)      # Set to 1
        time.sleep(halfperiod)
        GPIO.output(Signal, OFF)      # Set to 0
        time.sleep(halfperiod)

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 13.3: Program: squaresig.py.

Figure 13.4 shows the generated waveform. Clearly the frequency is around 850 Hz which is below the required frequency. This is because the delay function of Python is not very accurate, and it is longer than expected.

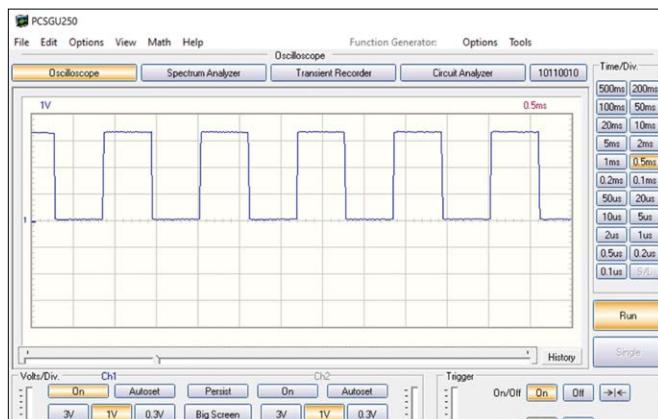


Figure 13.4: Generated waveform.

Figure 13.5 shows a program (program: **squaresig2.py**) which uses the Raspberry Pi PWM module to generate a 1000 Hz squarewave signal. As you can see from Figure 13.4, the program is very simple. The output waveform is shown in Figure 13.6 which is clearly more accurate than the one shown in Figure 13.4.

```

#-----
#          GENERATE SQUARE WAVEFORM
#          =====
#
# This program generates square waveforms using the PWM module of
# the Raspberry Pi
#

```

```

# Author: Dogan Ibrahim
# File : squaresig2.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO
import time
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM

freq = 1000                      # Initialize frequency

PWM_Port = 26                     # PWM on port 26
GPIO.setup(PWM_Port, GPIO.OUT)      # Configure as output
p = GPIO.PWM(PWM_Port, freq)       # Initialize PWM
p.start(50)                       # Start with duty cycle 50%

while True:                        # Do forever
    pass

```

Figure 13.5: Program: squaresig2.py.

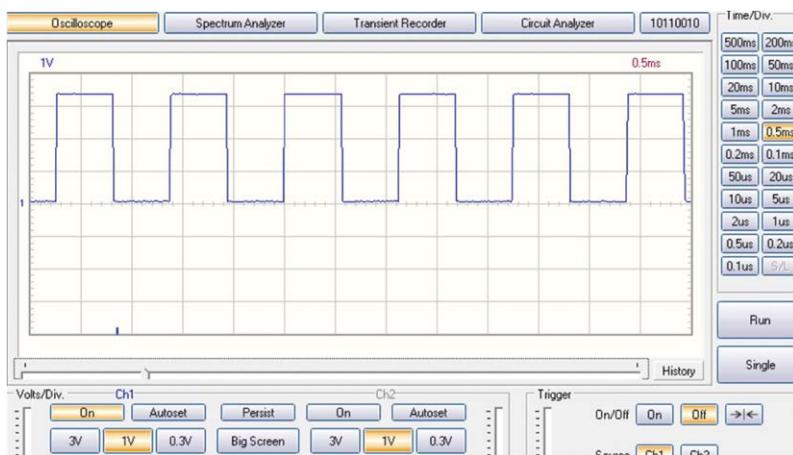


Figure 13.6: Generated waveform.

13.3 Generating a squarewave signal with any peak voltage

Description: In this project we will be using the DAC to generate a square wave signal with the frequency of 1 kHz where the output voltage will be $2 V_{peak}$.

Block Diagram: Figure 13.7 shows the block diagram of the project.



Figure 13.7: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 13.8. The output of the DAC is connected to the oscilloscope.

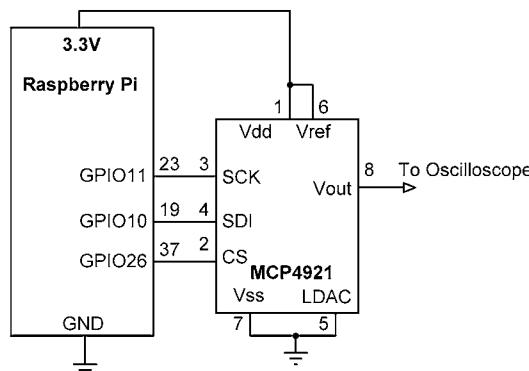


Figure 13.8: Circuit diagram of the project.

Program Listing: Data is written to the DAC in two bytes. The lower byte specifies D0:D8 of the digital input data. The upper byte consists of the following bits:

D8:D11	bits D8:D11 of the digital input data
SHDN	1: active (output available), 0: shutdown the device
GA	output gain control. 0: gain is 2×, 1: gain is 1×
BUF	0: input unbuffered, 1: input buffered
A/B	0: write to DACa, 1: Write to DACb (MCP4921 supports only DACa)

In normal operation, we will send the upper byte (D8:D11) of the 12 bit (D0:D11) input data with bits D12 and D13 set to 1 so that the device is active and the gain is set to 1×. Then we will send the low byte (D0:D7) of the data. This means that 0x30 should be added to the upper byte before sending it to the DAC.

Figure 13.9 shows the program listing (program: **squaredac.py**). At the beginning of the program library modules RPi, time, and spidev are imported to the program. GPIO26 is used as the **CS** pin and is configured as an output. Variable **frequency** set to 1000 which is the required frequency. Function **DAC** sends the 12-bit input data to the DAC. This function has two parts. In the first part the HIGH byte is sent after adding 0x30 as described above.

Function **xfer2** is used to send the data to the DAC. In the second part of the function the LOW byte is extracted and is sent to the DAC. Notice that we could have send both the high byte and the low byte using the same **xfer2** function as follows:

```
highbyte = (data >> 8) & 0x0F  
highbyte = highbyte + 0x30  
  
lowbyte = data & 0xFF  
xfer2([highbyte, lowbyte])
```

Variable **ONvalue** is set to $2000 * 4095 / 3300$ which is the digital value corresponding to 2000 mV (i.e. 2 V, remember that the DAC is 12 bits having 4095 steps, and the reference voltage is set to 3300 mV). The **OFFvalue** is set to 0V. Normally, the delay between the ON and OFF times should have been equal to **halfperiod**. However, it was found by the experiments that the DAC routine takes about 0.2 ms (0.0002 seconds) and this changes the period and consequently the frequency of the output waveform. Because of this, 2 mV is subtracted from **halfperiod** as shown in Figure 13.9.

Notice that the speed of the SPI interface is set to 3900000 which corresponds to 3.9 MHz. The table below shows the values set for the speed and the actual speed of the SPI interface:

Speed	spi.max_speed_hz value
125.0 MHz	125000000
62.5 MHz	62500000
31.2 MHz	31200000
15.6 MHz	15600000
7.8 MHz	7800000
3.9 MHz	3900000
1953 kHz	1953000
976 kHz	976000
488 kHz	488000
244 kHz	244000
122 kHz	122000
61 kHz	61000
30.5 kHz	30500
15.2 kHz	15200
7629 Hz	7629

```
#-----
#          GENERATE SQUARE WAVEFORM
#=====
#
# This program generates square waveform with the frequency 1kHz.
# In this program the MC4921 DAC chip is used to set the output
# peak voltage to 2V
#
# Author: Dogan Ibrahim
# File : squaredac.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi
import time                     # Import time
import spidev                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                  # Bus=0, device=0
spi.max_speed_hz = 3900000
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM
CS = 26                         # GPIO26 is CS output

GPIO.setup(CS, GPIO.OUT)          # CS is output
GPIO.output(CS, 1)                # Disable CS

frequency = 1000                 # Required Frequency
period = 1 / frequency           # Period of the signal
halfperiod = period / 2          # Half period

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
    GPIO.output(CS, 0)             # Enable CS
    #
    # Send HIGH byte
    #
    temp = (data >> 8) & 0x0F      # Get upper byte
    temp = temp + 0x30              # OR with 0x30
    spi.xfer2([temp])              # Send to DAC
    #
    # Send LOW byte
    #
    temp = data & 0xFF             # Get lower byte
```

```

spi.xfer2([temp])                      # Send to DAC

GPIO.output(CS, 1)                      # Disable CS

try:

    ONvalue = int(2000*4095/3300)        # 2V output
    OFFvalue = 0

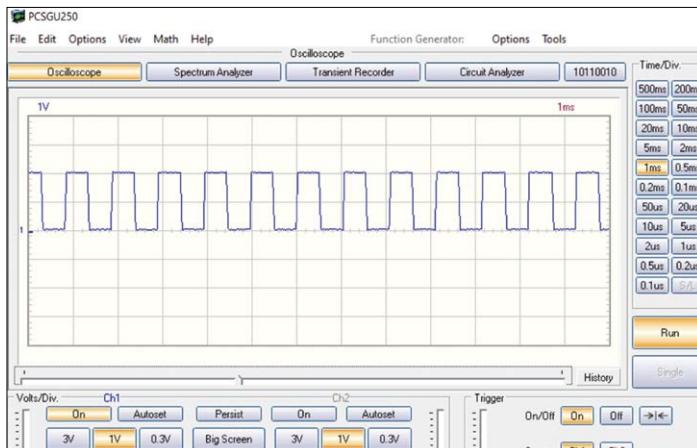
    while True:
        DAC(ONvalue)                   # Send to DAC
        time.sleep(halfperiod - 0.0002)  # Wait
        DAC(OFFvalue)                 # Send to DAC
        time.sleep(halfperiod - 0.0002)  # Wait

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 13.9: Program: squaredac.py.

Figure 13.10 shows the output waveform generated by the program. Notice that the peak output voltage is 2 V as expected.

*Figure 13.10: Output waveform.*

13.4 Generating a sawtooth-wave signal

Description: In this project we will be using the DAC to generate a sawtooth wave signal with the following specifications:

Peak voltage:	3.3 V
Step width:	1 ms
Number of steps:	6

Circuit Diagram: The circuit diagram of the project is as shown in Figure 13.8.

Program Listing: Figure 13.11 shows the program listing (program: **sawtooth.py**). The program is very similar to the one given in Figure 13.9.

```

#-----
#          GENERATE SAWTOOTH WAVEFORM
#-----=-----

#
# This program generates sawtooth waveform with 6 steps where each
# step has a width of 1ms
#
# Author: Dogan Ibrahim
# File  : sawtooth.py
# Date  : July, 2020
#-----

import RPi.GPIO as GPIO          # Import RPi
import time                      # Import time
import spidev                     # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                   # Bus=0, device=0
spi.max_speed_hz = 3900000

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM
CS = 26                          # GPIO26 is CS output

GPIO.setup(CS, GPIO.OUT)          # CS is output
GPIO.output(CS, 1)                # Disable CS

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
    GPIO.output(CS, 0)            # Enable CS
    #
    # Send HIGH byte
    #
    temp = (data >> 8) & 0x0F      # Get upper byte
    temp = temp + 0x30              # OR with 0x30
    spi.xfer2([temp])              # Send to DAC
    #
    # Send LOW byte
    #

```

```

temp = data & 0xFF           # Get lower byte
spi.xfer2([temp])            # Send to DAC

GPIO.output(CS, 1)             # Disable CS

try:

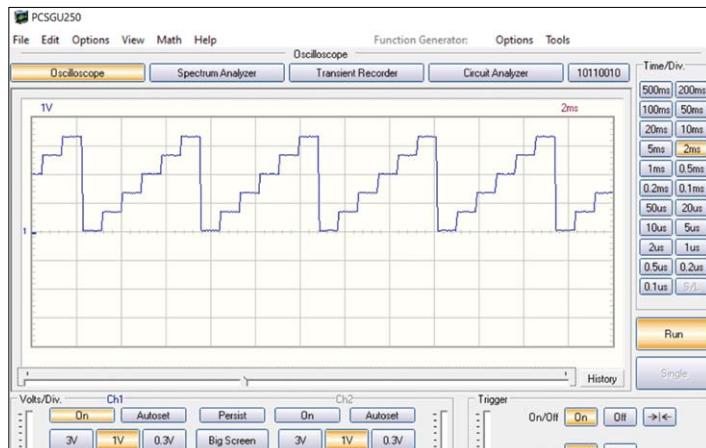
    while True:                # Do forever
        i = 0
        while i < 1.1:
            DACValue = int(i*4095)      # Value to send
            DAC(DACValue)              # Send to DAC
            time.sleep(0.0007)          # Wait
            i = i + 0.2

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 13.11: Program: sawtooth.py.

An example output waveform taken from the oscilloscope is shown in Figure 13.12. Notice that the time delay had to be adjusted experimentally to give the correct timing.

*Figure 13.12: Example output waveform.*

13.5 Generating a triangular-wave signal

Description: In this project we will be using the DAC to generate a triangular wave signal.

Circuit Diagram: The circuit diagram of the project is given in Figure 13.8.

Program Listing: Figure 13.13 shows the program listing (program: **triangle.py**). The program is very similar to the one given in Figure 13.11.

```
#-----
#          GENERATE TRIANGLE WAVEFORM
#=====
#
# This program generates triangle waveform
#
# Author: Dogan Ibrahim
# File : triangle.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi
import time                     # Import time
import spidev                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                  # Bus=0, device=0
spi.max_speed_hz = 3900000

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM
CS = 26                         # GPIO26 is CS output
sample = 0
Inc = 0.05

GPIO.setup(CS, GPIO.OUT)          # CS is output
GPIO.output(CS, 1)               # Disable CS

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
    GPIO.output(CS, 0)             # Enable CS
    #
    # Send HIGH byte
    #
    temp = (data >> 8) & 0x0F      # Get upper byte
    temp = temp + 0x30              # OR with 0x30
    spi.xfer2([temp])              # Send to DAC
    #
    # Send LOW byte
    #
    temp = data & 0xFF            # Get lower byte
    spi.xfer2([temp])              # Send to DAC

    GPIO.output(CS, 1)             # Disable CS
```

try:

```

while True:
    DACValue = int(sample*4095)          # Value to send
    DAC(DACValue)                      # Send to DAC
    time.sleep(0.0001)                  # Wait
    sample = sample + Inc              # Next sample
    if sample > 1.0 or sample < 0:
        Inc = -Inc
        sample = sample + Inc

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 13.13: Program: triangle.py.

An example output waveform taken from the oscilloscope is shown in Figure 13.14.

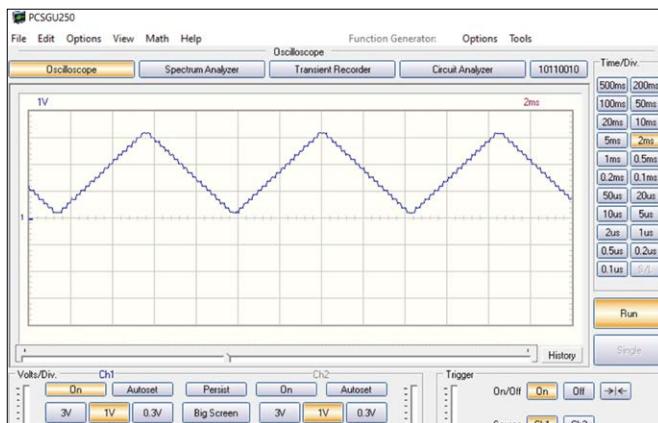


Figure 13.14: Example output waveform.

13.6 Generating an arbitrary-wave signal

Description: In this project we will be using the DAC to generate an arbitrary waveform. One period of the shape of the waveform will be sketched, and values of the waveform at different points will be extracted and loaded into a look-up table. The program will output the data points at the appropriate times to generate the required waveform.

The shape of one period of the waveform to be generated is shown in Figure 13.15. Notice that the waveform has a period of 20 ms.

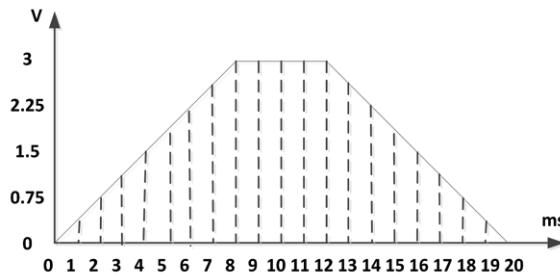


Figure 13.15: Waveform to be generated.

The waveform takes the following values:

Time (ms)	Amplitude (V)	Time (ms)	Amplitude (V)
0	0	11	3.00
1	0.375	12	3.00
2	0.75	13	2.625
3	1.125	14	2.25
4	1.50	15	1.875
5	1.875	16	1.50
6	2.25	17	1.125
7	2.625	18	0.75
8	3.00	19	0.375
9	3.00	20	0
10	3.00		

Circuit Diagram: The circuit diagram of the project is as shown in Figure 13.8

Program Listing: Figure 13.16 shows the program listing (program: **arbit.py**). The sample points of the waveform are stored in a list called **wave**. Variable **sample** indexes this list and sends the sample values to the DAC. The time of each sample was specified to be 1ms. It was found by experiment that 0.8 ms delay gave the correct results because of the delay in the DAC routine.

```

#-----
#          GENERATE ARBITRARY WAVEFORM
#  =====
#
# This program generates an arbitrary waveform whose sample points
# are defined in the program
#
# Author: Dogan Ibrahim
# File  : arbit.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO                      # Import RPi

```

```
import time                                # Import time
import spidev                               # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                             # Bus=0, device=0
spi.max_speed_hz=3900000
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                      # GPIO mode BCM
CS = 26                                    # GPIO26 is CS output
sample = 0

GPIO.setup(CS, GPIO.OUT)                    # CS is output
GPIO.output(CS, 1)                          # Disable CS
#
# Waveform sample points
#
wave = [0,0.375,0.75,1.125,1.5,1.875,2.25,2.625,3,3,3,3,3,3,\n
        2.625,2.25,1.875,1.5,1.125,0.75,0.375,0]

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
    GPIO.output(CS, 0)                      # Enable CS
    #
    # Send HIGH byte
    #
    temp = (data >> 8) & 0x0F            # Get upper byte
    temp = temp + 0x30                   # OR with 0x30
    spi.xfer2([temp])                  # Send to DAC

    temp = data & 0xFF
    spi.xfer2([temp])

    GPIO.output(CS, 1)                  # Disable CS

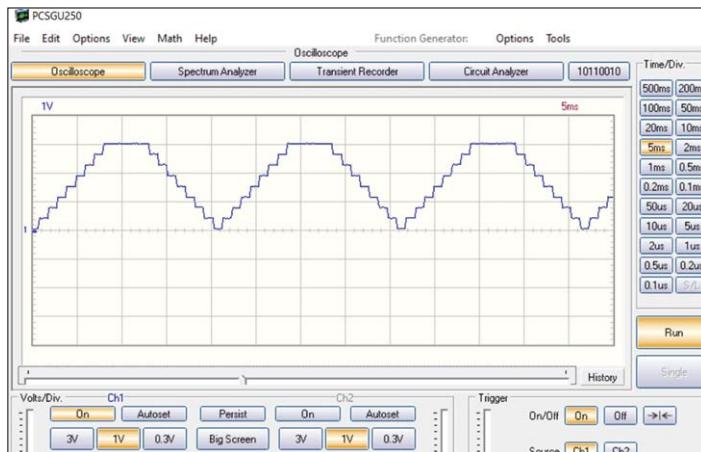
try:

    while True:
        DACValue = int(wave[sample]*4095/3.3)      # Value to send
        DAC(DACValue)                            # Send to DAC
        sample = sample + 1                      # Inc sample index
        time.sleep(0.0008)                       # Wait
        if sample == 20:                         # If 20 samples
            sample = 0
```

```
except KeyboardInterrupt:  
    GPIO.cleanup()
```

Figure 13.16: Program: arbit.py.

An example output waveform taken from the oscilloscope is shown in Figure 13.17.

*Figure 13.17: Example output waveform.*

13.7 Generating a sinewave signal

Description: In this project we will be using the DAC to generate a low frequency sinewave using the built-in trigonometric **sin** function. The generated sine wave will have amplitude of 1.5 V, a frequency of 100 Hz (period = 10 ms), and offset of 1.5 V.

Circuit Diagram: The circuit diagram of the project is as shown in Figure 13.8.

Program Listing: The frequency of the sinewave to be generated is 100 Hz. This wave has a period of 10 ms, or 10,000 μ s. If we assume that the sinewave will consist of 100 samples, then each sample should be output at $10,000/100 = 100 \mu$ s intervals. The sample values will be calculated using the trigonometric **sin** function of Python.

The **sin** function will have the format:

$$\sin\left(\frac{2\pi xcount}{T}\right)$$

where T is the period of the waveform and is equal to 100 samples. Thus, the sinewave is divided into 100 samples and each sample is output at 100 μ s. The above formula can be rewritten as:

$$\sin(0.0628 xcount)$$

It is required that the amplitude of the waveform should be 1.5 V. With a reference voltage of +3.3 V and a 12-bit DAC converter (0 to 4095 quantization levels), 1.5 V is equal to $1.5 \times 4095 / 3.3$, which is equal to 1861.3 (i.e. the amplitude). Thus, we will multiply our sine function with the amplitude at each sample to give:

$$1861.3 \times \sin(0.0628 \times \text{count})$$

The D/A converter used in this project is unipolar and cannot output negative values. Therefore, an offset is added to the sine wave to shift it so that it is always positive. The offset should be larger than the absolute value of the maximum negative value of the sinewave, which is 1861.3 when the **sin** function above is equal to 1.5. In this project we are adding 1.5 V offset which corresponds to a decimal value of 1861.3 (i.e. the offset) at the DAC output. Thus, at each sample we will calculate and output the following value to the DAC:

$$1861.3 + 1861.2 \times \sin(0.0628 \times \text{count})$$

The sine waveform values for a period are obtained outside the program loop using the following statement. List **sins** contains all the 100 sine values of the waveform. The reason for calculating these values outside the program loop is to minimize the time to calculate the **sin** function:

```
for i in range(100):
    sins[i] = int(offset + amplitude * sin(R*i))
```

where R is set to 0.0628.

Figure 13.18 shows the program listing (program: **sine.py**). Most parts of the program are similar to the other waveform generation programs. Inside the program loop samples of the sinewave are sent to the DAC at each sample time.

```
#-----
#          GENERATE SINE WAVEFORM
#          =====
#
# This program generates sine waveform with a period of 10ms. Both
# the amplitude and the offset of the waveform are set to 1.5V
#
# Author: Dogan Ibrahim
# File  : sine.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi
import time                      # Import time
import spidev                     # Import SPI
import math                       # Import math
```

```
spi = spidev.SpiDev()
spi.open(0, 0)                                     # Bus=0, device=0
spi.max_speed_hz = 3900000

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)                            # GPIO mode BCM
CS = 26                                         # GPIO26 is CS output

sample = 0
T = 100
R = 0.0628
amplitude = 1861.3
offset = 1861.3
sins = [None]*101

GPIO.setup(CS, GPIO.OUT)                          # CS is output
GPIO.output(CS, 1)                               # Disable CS

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
    GPIO.output(CS, 0)                           # Enable CS
    #
    # Send HIGH byte
    #
    temp = (data >> 8) & 0x0F                 # Get upper byte
    temp = temp + 0x30                         # OR with 0x30
    spi.xfer2([temp])                          # Send to DAC
    #
    # Send LOW byte
    #
    temp = data & 0xFF                        # Get lower byte
    spi.xfer2([temp])                          # Send to DAC

    GPIO.output(CS, 1)                           # Disable CS

#
# Generate the 100 sine wave samples and store in list sins
#
for i in range(100):
    sins[i] = int(offset + amplitude*math.sin(R*i))

try:
```

```

while True:
    DACValue = sins[sample]                      # Value to send
    DAC(DACValue)                               # Send to DAC
    time.sleep(0.0001)                           # Wait
    sample = sample + 1                          # Next sample
    if sample == 100:                            # 100 samples?
        sample = 0

```

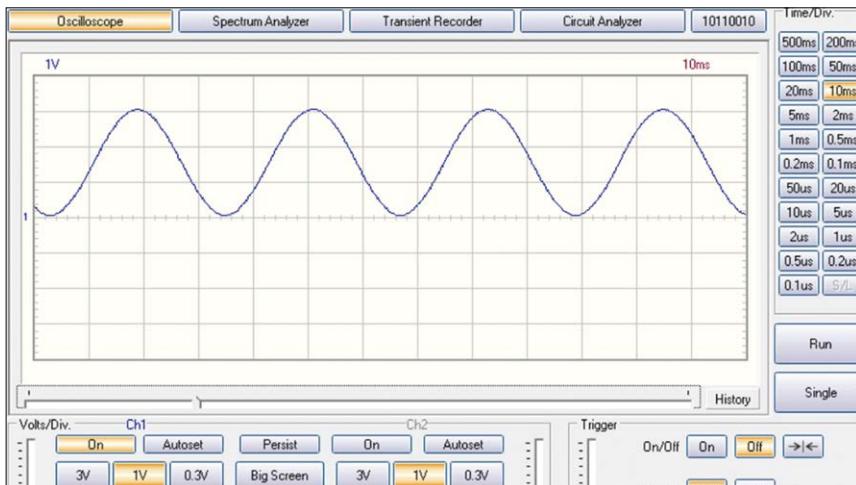
```

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 13.18: Program: sine.py.

An example output waveform taken from the oscilloscope is shown in Figure 13.19. Notice that the frequency of the waveform is not very accurate because the delay function of Python is not accurate.

*Figure 13.19: Example output waveform.*

CHAPTER 14 • Waveform Generation – Using Hardware

In this Chapter we will be using a programmable signal generator module to generate accurate sinewave and squarewave signals.

14.1 Project: Fixed-Frequency Waveform Generator

We will be using the popular AD9850 signal generator module together with the Raspberry Pi. The type AD9850 module (see Figure 14.1) is a dual sine and square waveform generator with the following features:

- 0 – 40 MHz sine wave output
- 0 – 1 MHz square wave output
- 3.3 V or 5 V operation
- 125 MHz on-board timing crystal
- On-chip 10-bit DAC
- Serial or parallel programming data
- 60 mA operating current

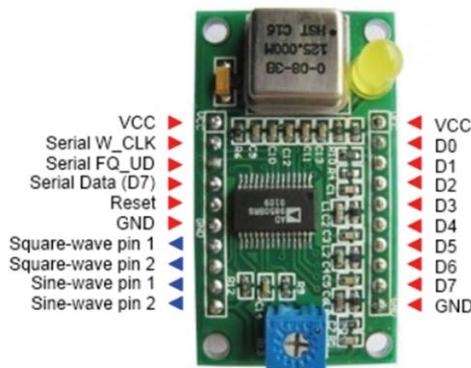


Figure 14.1: The AD9850 module.

The AD9850's circuit architecture allows the generation of output frequencies of up to one-half the reference clock frequency (or 62.5 MHz). The device also provides five bits of digitally controlled phase modulation, which enables phase shifting of its output in increments of 180°, 90°, 45°, 22.5°, 11.25°, and any combination thereof.

In this project we will be generating a 1-kHz sinewave as an example.

Before using the AD9850 in a project, we need to know how to program it, and this is described briefly in the following sections (more information on the AD9850 can be obtained from the manufacturers' data sheet at: <https://www.analog.com/media/en/technical-documentation/data-sheets/ad9850.pdf>)

Figure 14.2 shows the AD9850 pin layout.

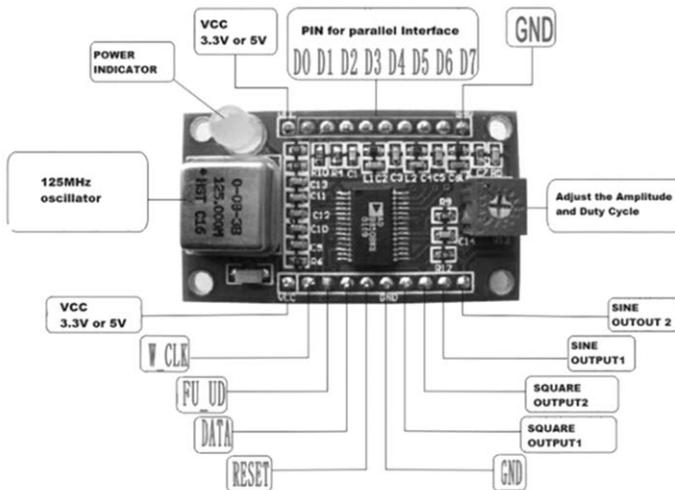


Figure 14.2: AD9850 module pins layout.

The pin definitions are:

- **VCC**: supply voltage (3.3 V or 5 V)
- **GND**: supply ground
- **W_CLK**: load word clock (used to load parallel or serial frequency/phase words)
- **FQ_UD**: frequency update (the frequency or phase is updated on rising edge of this pin)
- **DATA**: serial load pin
- **RESET**: master reset pin (HIGH to reset to clear all registers)
- **D0-D7**: parallel data input (for loading 32-bit frequency and 8-bit phase/control word)
- **Square Wave Output1**: square wave output 1 (comparator output)
- **Square Wave Output2**: square wave output 2 (comparator complement output)
- **Sine Wave Output1**: analogue sinewave output 1 (DAC output)
- **Sine Wave Output2**: analogue sinewave output 2 (DAC complement output)

The on-board potentiometer is used to set the duty cycle of the square waveform.

The AD9850 can be 'loaded' either in parallel or in serial form. In this project we will be using the **serial mode**.

Block Diagram: Figure 14.3 shows the block diagram of the project. The sine output1 of the module is connected to the oscilloscope.

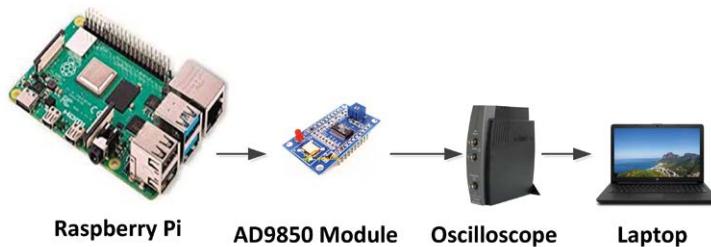


Figure 14.3: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 14.4. The following connections are made between the Raspberry Pi and the AD9850 module:

AD9850 module	Raspberry Pi
FQ_UD	GPIO26
W_CLK	GPIO19
RESET	GPIO13
DATA	GPIO6

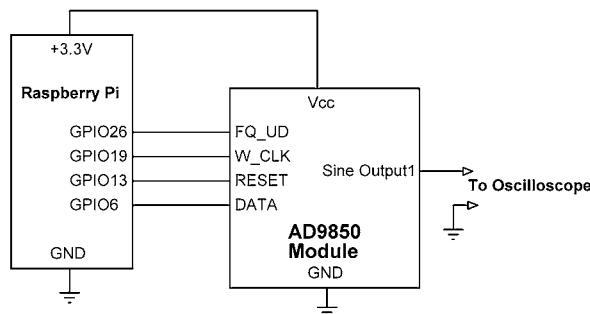


Figure 14.4: Circuit diagram of the project.

Program Listing: The relationship between the output frequency, reference clock, and tuning word of the AD9850 is determined by the following formula:

$$f_{\text{out}} = \Delta\text{Phase} \times \text{REFCLOCK} / 2^{32}$$

$$\text{or, } \Delta\text{Phase} = f_{\text{out}} \times 2^{32} / \text{REFCLOCK}$$

where

f_{out} is the output frequency in MHz, ΔPhase is the value of the 32-bit tuning word, and **REFCLOCK** is the reference clock in MHz. The output sine wave is an analogue signal being output by a 10-bit DAC. Notice that $2^{32} = 4,294,967,296$.

The AD9850 contains a 40-bit register, consisting of the 32-bit frequency control word, 5-bit phase modulation word, and the power-down function. In this project, this register is loaded serially. In this mode, rising edge of **W_CLK** shifts the 1-bit data on pin D7 (DATA) through the 40-bits of programming information. After shifting 40-bits, an **FQ_UD** pulse is required to update the output frequency (or phase).

The serial mode is enabled by the following sequence (see Datasheet page 12, Figure 10):

- Pulse RESET
- Pulse W_CLK
- Pulse FQ_UD

The 40-bit serial data gets loaded as follows:

- Send bit 0 — frequency (LSB)
- Send bit 1 — frequency
- Send bit 2 — frequency
-
-
- Send bit 31 — frequency (MSB)
- Send bit 32 — Control (set to 0)
- Send bit 33 — Control (set to 0)
- Send bit 34 — power down
- Send bit 35 — phase (LSB)
- Send bit 36 — phase
- Send bit 37 — phase
- Send bit 38 — phase
- Send bit 39 — phase
- Send bit 39 — phase (MSB)

Figure 14.5 shows the program listing (Program: **freqgen.py**). At the beginning of the program the connections of the AD9850 pins **FQ_UD**, **W_CLK**, **RESET**, and **DATA** are set to 26, 19, 13, and 6 respectively which are the GPIO pin names. These pins are configured as outputs and all are cleared to 0 at the beginning of the program.

The program consists of several functions as described below:

SendPulse: This function sends a HIGH-LOW logic signals to the pin specified in its argument **GPIOpin**.

SendByte: This function receives a byte as its argument. Then a loop is formed to send the 8 bits of this byte to serial input DATA. A pulse is sent to W_CLK after sending a bit.

SetSerialMode: This function puts the AD9850 into serial mode by pulsing pins RESET, W_CLK, and FQ_UD.

LoadFrequency: This function receives the required frequency as its argument. Then the tuning word is calculated using the formula described earlier in this section and is stored in variable **f**. Then a for loop is formed to iterate four times. The LOW byte of **f** is then sent to function **SendByte** so that its eight bits are sent to the serial input. Variable **f** is then shifted right 8 times so that the next higher byte is serialized and sent to function **SendByte**. This process is repeated for the four bytes, making a total of 32-bits. Then, the remaining eight bits are sent as zeroes so that the two Control words are 0, as are the phase bits.

The main program puts the AD9850 into serial mode, sets the required frequency as Hz (1000 Hz = 1 kHz) and loads the frequency into AD9850 by calling function **LoadFrequency**.

The program is terminated when **Cntrl+C** is entered from the PC keyboard. The AD9850 is stopped and the GPIO is cleaned just before terminating the program.

```
#-----
#          FREQUENCY GENERATOR
#=====
#
# In this program an AD9850 module is connected to the Raspberry
# Pi. The program generates a sine wave with a frequency of 1kHz.
# The on-board crystal of the AD9850 module is 125MHZ
#
# Author: Dogan Ibrahim
# File  : freqgen.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi

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

#
# AD9850 module connections
#
FQ_UD = 26                      # FQ_UD pin
W_CLK = 19                        # W_CLK pin
RESET = 13                         # RESET pin
```

```
DATA = 6                                # DATA pin
                                              # GPIO26 is CS output

#
# Configure pins as outputs
#
GPIO.setup(FQ_UD, GPIO.OUT)                # FQ_UD is output
GPIO.setup(W_CLK, GPIO.OUT)                 # W_CLK is output
GPIO.setup(RESET, GPIO.OUT)                 # RESET is output
GPIO.setup(DATA, GPIO.OUT)                  # DATA is output

#
# Set all outputs to 0 at the beginning
#
GPIO.output(FQ_UD, 0)                      # FQ_UD = 0
GPIO.output(W_CLK, 0)                      # W_CLK = 0
GPIO.output(RESET, 0)                      # RESET = 0
GPIO.output(DATA, 0)                       # DATA = 0

#
# This function sends a pulse to the pin specified in the argument
#
def SendPulse(GPIOpin):
    GPIO.output(GPIOpin, 1)                  # Send 1
    GPIO.output(GPIOpin, 0)                  # Send 0
    return

#
# This function sends bits of a byte of data to the AD9850 module
#
def SendByte(DataByte):
    for k in range(0, 8):                  # Do 8 times
        p = DataByte & 0x01               # Get bit 0
        GPIO.output(DATA, p)              # Output it
        SendPulse(W_CLK)                # Send clock
        DataByte = DataByte >> 1       # Get next bit
    return

#
# This function puts AD9850 module into serial mode
#
def SetSerialMode():
    SendPulse(RESET)                   # Pulse RESET
    SendPulse(W_CLK)                  # Pulse W_CLK
    SendPulse(FQ_UD)                 # Pulse FQ_UD
    return
```

```

#
# This function loads the 40-bit data to the AD9850. 32-bit
# frequency data is sent, then the power down bit is sent as 0,
# two Control bits are sent as 0, and then the 5 phase bits are
# sent as 0
#
def LoadFrequency(frequency):
    f = int(frequency * 4294967296 / 125000000)           # See book
    for p in range(0, 4):                                # Do 4 times
        SendByte(f & 0xFF)                             # Send Low byte
        f = f >> 8                                    # Get next byte
    SendByte(0x00)                                      # Send remaining bits
    SendPulse(FQ_UD)                                    # Terminate serial
    return

try:

    SetSerialMode()                                     # Select serial mode
    RequiredFrequency = 1000                          # required 1000Hz
    LoadFrequency(RequiredFrequency)                  # Load register

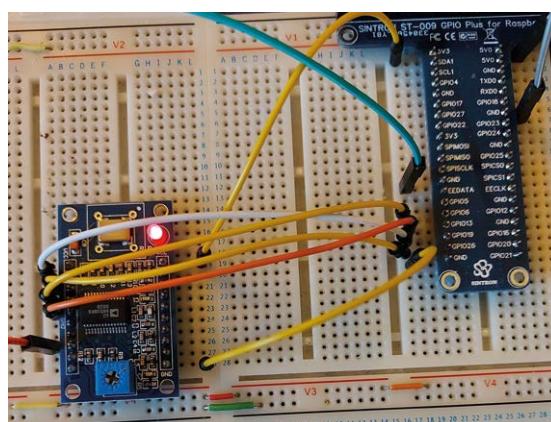
    while True:                                       # Wait forever...
        pass                                         # ...until stopped

except KeyboardInterrupt:                           # Keyboard Cntrl+C
    SendPulse(RESET)                               # Stop AD9850
    GPIO0.cleanup()                                 # Clean GPIO

```

Figure 14.5: Program: freqgen.py.

Figure 14.6 shows the project built on a breadboard.

*Figure 14.6: Project built on a breadboard.*

An example output waveform from the program is shown in Figure 14.7. As it can be seen from this figure the frequency of the waveform is exactly 1 kHz as required. The output is measured from pin **Sineoutput1**. The peak-to-peak amplitude of the waveform was measured to be exactly 1 V.

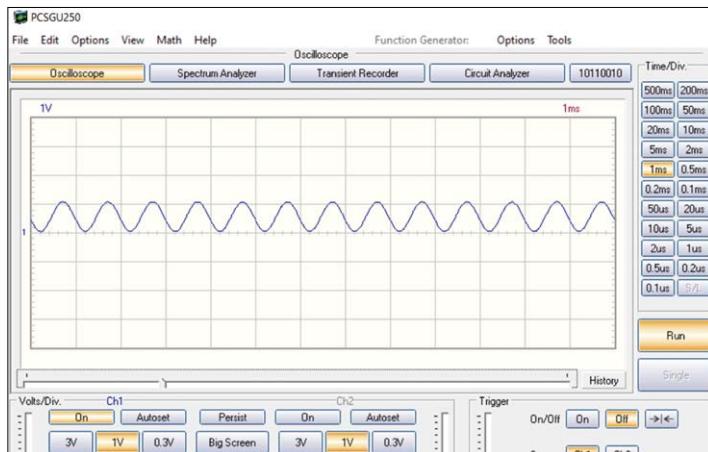


Figure 14.7: Example sinusoidal waveform display from the program.

Figure 14.8 shows the 'square' (actually: rectangle) waveform output of the program, measured from pin **Squareoutput2**. The duty cycle of the waveform can be changed by the potentiometer. The amplitude of the waveform was measured to be 3.3 V.



Figure 14.8: Example square waveform display from the program.

14.2 Project: Keypad Frequency Entry, LCD Readout, Waveform Generator

This project is similar to the earlier one but here a keypad and an I²C LCD are used to set the frequency of the waveform. This makes the project autonomous so that the Raspberry Pi can be used on its own without having to use a PC to set the frequency of the waveform.

Block Diagram: Figure 14.9 shows the block diagram of the project.

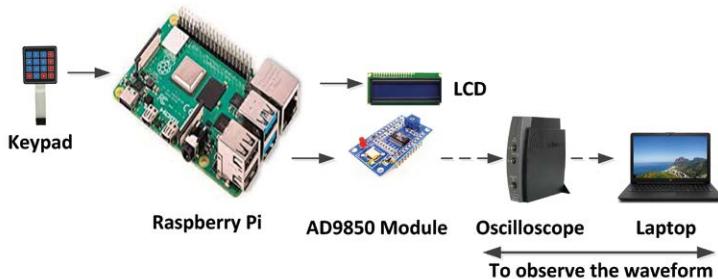


Figure 14.9: Block diagram of the project.

The Keypad: There are several types of keypads that can be used in microcontroller-based projects. In this project a 4×4 keypad is used (see Figure 14.10). This keypad has keys for numbers 0 through 9 and letters A, B, C, D, *, and #. The keypad is interfaced to the processor with eight wires with the names R1 through R4 and C1 through C4, representing the rows and columns respectively of the keypad (see Figure 14.11).



Figure 14.10: 4×4 keypad.

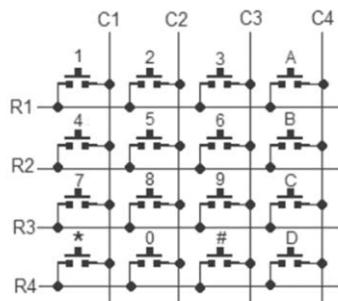


Figure 14.11: Circuit diagram of the 4×4 keypad.

The operation of the keypad is very simple: the columns are configured as outputs and the rows as inputs. The pressed key is identified by using column scanning. Here, a column is forced low while the other columns are held logic high. Then the state of each row is scanned, and if a row is found to be low, then the key at the intersection of the row which is low, and this column is the key pressed. This process is repeated for all the rows.

Circuit Diagram: Figure 14.12 shows the circuit diagram of the project. The I²C LCD is connected to the Raspberry Pi as in the previous projects using the LCD, where GPIO 2 and GPIO 3 are used as the SDA and SCL pins, respectively. The AD9850 module is connected as in the previous project. The 4×4 keypad is connected to the following GPIO pins of the Raspberry Pi. The row pins are held high using 10-kohm pullup resistors to +3.3 V. Notice that Raspberry Pi has internal pullup resistors when a pin is used as an input, but the use of these pullup resistors is not reliable.

Keypad pin	Raspberry Pi pin
R1	GPIO 14
R2	GPIO 15
R3	GPIO 12
R4	GPIO 23
C1	GPIO 24
C2	GPIO 25
C3	GPIO 8
C4	GPIO 7

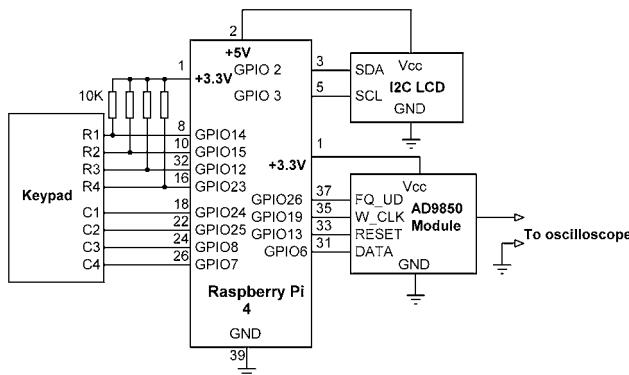


Figure 14.12: Circuit diagram of the project.

Figure 14.13 shows the pin configuration of the 4×4 keypad used in the project.



Figure 14.13: Pin configuration of the 4x4 keypad.

Test program

Before writing the project program we will start with developing the code to read keys from the keypad just to verify that the keypad hardware and software are working properly.

The basic steps to read a key are as follows:

```

Configure all columns as outputs
Configure all rows as inputs
Set all columns to 1
DO for all columns
    Set a column to 0
    DO for all rows
        IF a row is 0 THEN
            Return the key at this column and row position
        ENDIF
    ENDDO
ENDDO

```

Figure 14.14 shows the test program (program: **keypad.py**). At the beginning of the program the keypad keys are defined after importing the required modules to the program. The keypad row and columns connections are defined using lists **ROWS** and **COLS** respectively. Columns are then configured as outputs and are set to 1. Similarly, the rows are configured as inputs. Function **Get_Key** reads the pressed key and returns it to the calling program. Two **for** loops are used in the function: the first loop selects the columns and sets them to 0 one after the other one. The second loop scans the rows and checks if a row is at 0. The main program calls the function and displays the pressed key on the screen. You should test the keypad hardware and software by pressing various keys on the keypad and verifying that the correct key is displayed on the PC screen.

```
#-----
#          KEYPAD TEST PROGRAM
#=====
#
# This program shows how the a keypad can be used to display
# the pressed keys
#
# Author: Dogan Ibrahim
# File  : keypad.py
# Date  : July 2020
#-----
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

KEYPAD = [                                         # Keypad keys
    [1,2,3,"A"], 
    [4,5,6,"B"],
    [7,8,9,"C"],
    ["*",0,"#", "D"]]

ROWS = [14,15,12,23]                                # Row pins
COLS = [24,25,8,7]                                  # Column pins

for i in range(4):                                    # Conf columns
    GPIO.setup(COLS[i], GPIO.OUT)
    GPIO.output(COLS[i], 1)

for j in range(4):                                    # Conf rows
    GPIO.setup(ROWS[j], GPIO.IN)

#
# This function reads a key from the keypad
#
def Get_Key():
    while True:
        for j in range(4):
            GPIO.output(COLS[j], 0)                  # Set col j to 0
            for i in range(4):                      # For all rows
                if GPIO.input(ROWS[i]) == 0:          # Row is 0?
                    return (KEYPAD[i][j])             # Return key
                while GPIO.input(ROWS[i]) == 0:
                    pass
            GPIO.output(COLS[j], 1)                  # Col back to 1
```

```

        time.sleep(0.05)                                # Wait 0.05s

try:
    while True:
        key = Get_Key()                            # Get a key
        print(key)                                 # Display the key
        time.sleep(0.5)

except KeyboardInterrupt:
    GPIO.cleanup()

```

Figure 14.14: Program: keypad.py.

Program Listing: We are now ready to develop the program of this project. Figure 14.15 shows the program listing (program: **keypadsig.py**). In this program, key **D** is assumed to be the **ENTER** key where all the inputs to the keypad must be terminated by pressing the **ENTER** key.

At the beginning of the program modules RPi, time, and RPLCD are imported to the program. Then the keypad layout is defined. The columns of the keypad are configured as outputs and its rows are configured as inputs. Function **Get_Key** receives a key from the keypad and returns it to the main program.

The AD9850 module part of the program is same as in the previous project.

Inside the main program loop function, **Get_Key** is called to get the key numbers entered by the user. Only keys 0 to 9 and key **D** are accepted by the program. Pressing any other key is ignored by the program. When key **D** is pressed, the program stores the entered frequency in variable **Total** which is then copied to variable **frequency**. After pressing the **D** key, the LCD displays **OK** to confirm that the entered frequency value is accepted by the program. After a delay of five seconds, the second row of the LCD is cleared so that it is ready to accept a new frequency value.

As in the previous project, the AD9850 is put into serial mode and the user entered frequency is loaded so that the required waveform is generated.

```

#-----
#          FREQUENCY GENERATOR WITH KEYPAD AND LCD
#          =====
#
# In this program the required frequency is entered from a keypad
# in Hz. An LCD is used to display the entered frequency
#
# Author: Dogan Ibrahim
# File : keypadsig.py
# Date : July 2020

```

```
#-----
import RPi.GPIO as GPIO          # RPI library
import time                      # time
from RPLCD.i2c import CharLCD   # LCD

LCD = CharLCD('PCF8574', 0x27)      # LCD address
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

KEYPAD = [                           # Keypad keys
    [1,2,3,"A"], 
    [4,5,6,"B"], 
    [7,8,9,"C"], 
    ["*",0,"#", "D"] ]                # D is ENTER key

ROWS = [14,15,12,23]                 # Row pins
COLS = [24,25,8,7]                  # Column pins

for i in range(4):                   # Conf columns
    GPIO.setup(COLS[i], GPIO.OUT)
    GPIO.output(COLS[i], 1)

for j in range(4):                   # Conf rows
    GPIO.setup(ROWS[j], GPIO.IN)

#
# This function reads a key from the keypad
#
def Get_Key():
    while True:
        for j in range(4):
            GPIO.output(COLS[j], 0)           # Set col j to 0
            for i in range(4):               # For all rows
                if GPIO.input(ROWS[i]) == 0:   # Row is 0?
                    return (KEYPAD[i][j])       # Return key
                while GPIO.input(ROWS[i]) == 0:
                    pass
            GPIO.output(COLS[j], 1)           # Col back to 1
            time.sleep(0.05)                # Wait 0.05s

#
# ====== AD9850 functions ======
#



#
```

```
# AD9850 module connections
#
FQ_UD = 26                                # FQ_UD pin
W_CLK = 19                                 # W_CLK pin
RESET = 13                                 # RESET pin
DATA = 6                                   # DATA pin
# GPIO026 is CS output

#
# Configure pins as outputs
#
GPIO0.setup(FQ_UD, GPIO0.OUT)                # FQ_UD is output
GPIO0.setup(W_CLK, GPIO0.OUT)                  # W_CLK is output
GPIO0.setup(RESET, GPIO0.OUT)                  # RESET is output
GPIO0.setup(DATA, GPIO0.OUT)                  # DATA is output

#
# Set all outputs to 0 at the beginning
#
GPIO0.output(FQ_UD, 0)                      # FQ_UD = 0
GPIO0.output(W_CLK, 0)                      # W_CLK = 0
GPIO0.output(RESET, 0)                      # RESET = 0
GPIO0.output(DATA, 0)                       # DATA = 0

#
# This function sends a pulse to the pin specified in the argument
#
def SendPulse(GPIOpin):
    GPIO0.output(GPIOpin, 1)                  # Send 1
    GPIO0.output(GPIOpin, 0)                  # Send 0
    return

#
# This function sends bits of a byte of data to the AD9850 module
#
def SendByte(DataByte):
    for k in range(0, 8):                   # Do 8 times
        p = DataByte & 0x01                 # Get bit 0
        GPIO0.output(DATA, p)                 # Output it
        SendPulse(W_CLK)                   # Send clock
        DataByte = DataByte >> 1           # Get next bit
    return

#
# This function puts AD9850 module into serial mode
#
def SetSerialMode():
```

```
SendPulse(RESET)                                # Pulse RESET
SendPulse(W_CLK)                                 # Pulse W_CLK
SendPulse(FQ_UD)                                 # Pulse FQ_UD
return

#
# This function loads the 40-bit data to the AD9850. 32-bit
# frequency data is sent, then the power down bit is sent as 0,
# two Control bits are sent as 0, and then the 5 phase bits are
# sent as 0
#
def LoadFrequency(frequency):
    f = int(frequency * 4294967296 / 1250000000)      # See book
    for p in range(0, 4):                            # Do 4 times
        SendByte(f & 0xFF)                          # Send Low byte
        f = f >> 8                               # Get next byte
    SendByte(0x00)                                  # Send remaining bits
    SendPulse(FQ_UD)                               # Terminate serial
    return

Total = 0
LCD.clear()                                     # Clear LCD
LCD.cursor_pos = (0, 0)                         # To (0,0)
LCD.write_string("Frequency (Hz):")             # Heading
LCD.cursor_pos = (1, 0)                         # To (1,0)

try:

    while True:
        LCD.cursor_pos = (1, 0)
        flag = 0
        while flag == 0:
            key = Get_Key()                        # Get a key
            if key != "D":                         # Is it ENTER?
                if str(key).isnumeric():           # IS it numeric?
                    LCD.write_string(str(key))     # Display key
                    N = int(key)
                    Total = 10*Total + N          # Total so far
                else:                           # ENTER detected
                    frequency = Total          # Get frequency
                    Total = 0
                    flag = 1
                    LCD.write_string(" - OK")   # Display OK
                    time.sleep(0.2)              # Wait 0.2 sec

    SetSerialMode()                                # Select serial mode
```

```

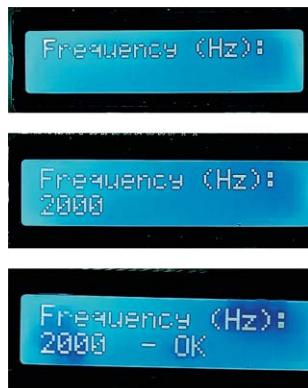
LoadFrequency(frequency)           # Load register
time.sleep(5)
LCD.cursor_pos = (1, 0)
LCD.write_string("")

except KeyboardInterrupt:          # Keyboard Cntrl+C
    SendPulse(RESET)             # Stop AD9850
    GPIO0.cleanup()               # Clean GPIO

```

Figure 14.15: Program: keypadsig.py.

Figure 14.16 shows the LCD before entering the frequency, after entering frequency and after pressing the **D** (ENTER) key, respectively.

*Figure 14.16: Before entering the frequency.*

The project was constructed on a breadboard as shown in Figure 14.17, and connections were made to the Raspberry Pi and other components using jumper wires. Interested readers can design a PCB for the project.

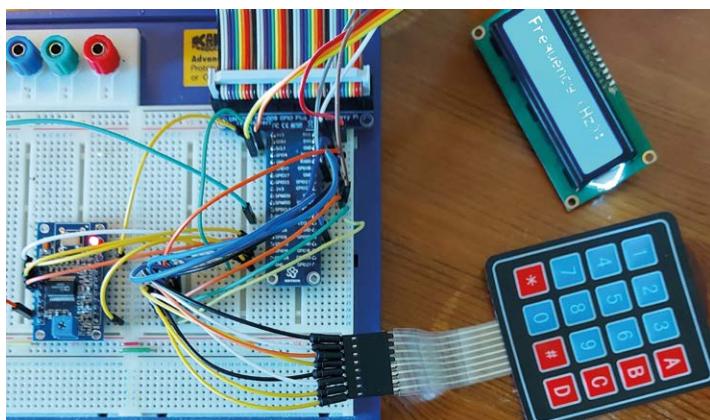
*Figure 14.17: Project constructed on a breadboard.*

Figure 14.18 shows the generated waveform on the oscilloscope. It's obvious that the frequency is exactly 2000 Hz.

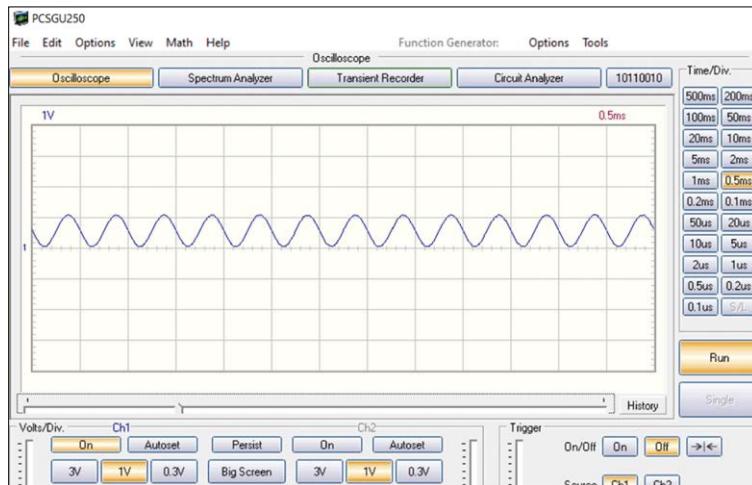


Figure 14.18: Display on the oscilloscope.

Notice that the program can be inserted into **/etc/rc.local** so that it starts automatically after restarting the Raspberry Pi so that the project is completely independent of the PC used.

CHAPTER 15 • Designing a Single Stage Common-Emitter Bipolar Transistor Amplifier Circuit

15.1 Project

In this project we will design a single common-emitter transistor amplifier circuit with a given supply voltage and a required voltage gain.

Background Information: The circuit diagram of a commonly used single stage common-emitter amplifier circuit is shown in Figure 15.1. R₁ and R₂ are the biasing resistors, R_E is the emitter resistor, R_C is the collector resistor, R_L is the load resistor, R_s is the source internal resistor, C_E is the emitter decoupling capacitor, C₁ and C₂ are the coupling capacitors.

The low frequency small signal AC parameters of this amplifier circuit can be calculated using the following formula (see the Hybrid Pi model):

$$G = \frac{V_{out}}{V_{in}} = G_1 \times G_2 = \frac{V_{out}}{V_b} \times \frac{V_b}{V_L}$$

$$G_1 = \frac{V_{out}}{V_b} = \frac{-R_c \vee R_L}{r_e}$$

$$\Re = \frac{25}{I_E}$$

$$R_B = \frac{R_1 R_2}{R_1 + R_2}$$

$$C_E = \frac{1}{2\pi f X_C} \text{ where } X_C = \frac{R_E}{10} \text{ at the required -3 dB frequency}$$

$$Z_i = \frac{R_B \beta r_e}{R_B + \beta r_e}$$

$$G_2 = \frac{V_b}{V_L} = \frac{Z_i}{R_s + Z_i}$$

$$C_2 = \frac{1}{2\pi f X_C} \text{ where } X_C = \frac{Z_o}{10} \text{ at the required -3 dB frequency}$$

$$C_1 = \frac{1}{2\pi f X_C} \text{ where } X_C = \frac{Z_L}{10} \text{ at the required -3 dB frequency}$$

The design of a single stage common-emitter amplifier normally starts from the specification of the required voltage gain, or the input voltage and the required output voltage.

The supply voltage is usually given. We first have to find the resistor values for biasing the transistor and also providing the required voltage gain.

Briefly, the steps are as follows:

- Given the input and required output voltages, calculate the required voltage gain.
- Choose a collector current of around 2 mA.
- Assume V_{CE} to be $V_{cc} / 2$.
- Choose a value for R_L . Thinking that the gain without the source and load resistors is given by $R_L \times I_E / 25$ at room temperature, choose R_L to give 30% higher gain than the required value.
- Choose a value for R_E (assuming that $I_E = I_c$) from equation $V_{cc} = I_c R_L + V_{CE} + I_c R_E$.
- Calculate base current from I_B / β .
- Calculate the base voltage from $V_B = 0.7 + I_c R_E$.
- Assume that the current in R_1 and R_2 will be 10 times the base current and calculate suitable values for R_1 and R_2 .
- Calculate the input and output impedances.
- Check that the overall gain is as required, if not go back and adjust I_c , R_L , or R_E .
- Calculate suitable values for C_1 , C_2 and C_E .

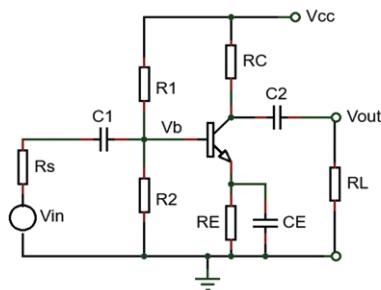


Figure 15.1: Common-emitter amplifier circuit.

Program Listing: Figure 15.2 shows the program listing (program: **amplifier.py**). At the beginning of the program a heading is displayed. The program then prompts the user to enter the supply voltage, input and required output voltage, source and load resistances, low frequency band, and the transistor current ratio.

```
#=====
#      Single Stage Common Emitter Amplifier Design
# =====
#
# This program designs a common emitter amplifier
#
#
# Author: Dogan Ibrahim
# File  : amplifier.py
# Date  : July 2020
#=====
import math

print("Single Stage Common Emitter Amplifier Design")
print("=====")
#
# Read the component values
#
Vcc = float(input("Enter supply voltage (Volts): "))
Vin = float(input("Enter Vin (Volts): "))
Vout = float(input("Enter Vout (Volts): "))
RS = float(input("Enter Rs (ohms): "))
RL = float(input("Enter RL (Ohms): "))
f = float(input("Enter lowest frequency to amplify (Hz): "))
B = float(input("Enter current ratio (B): "))

#
# Calculations
#
Gr = Vout / Vin                      # Gain
IC = 2                                # IC=2mA
Ge = Gr * (1.3)                        # Estimate gain 30% higher
RC = Ge * 25 / IC                      # Collector resistance
RCRL = (RC * RL) / (RC + RL)
re = 25.0 / IC
Rinbase = B * re                        # Base resistance
G1 = RCRL / re                         # G1
VCE = Vcc / 2                           # VCE
IC = IC / 1000.0
RE = (Vcc - RC * IC - VCE) / IC       # RE
VB = 0.7 + IC * RE
IB = IC / B
IR1R2 = 10.0 * IB                      # 10 times base current
R2 = VB / IR1R2
R1 = (Vcc - VB) / IR1R2
RB = (R1 * R2) / (R1 + R2)
```

```
Zi = (RB * Rinbase) / (RB + Rinbase)
G2 = Zi / (RS + Zi)
G = G1 * G2
XC = Zi / 10.0
C1 = 1000000.0 / (2 * math.pi * f * XC)
Zo = RCRL
XC = Zo / 10.0
C2 = 1000000.0 / (2 * math.pi * f * XC)
XC = RE / 10.0
CE = 1000000.0 / (2 * math.pi * f * XC)

#
# Display the results
#
print("\nRESULTS:")
print("====")
print("Gain=%7.3f\nIC=%7.3f mA\nIB=%7.3f mA" %(G, IC*1000.0, IB*1000.0))
print("\nVcc=%7.3f V\nVB=%7.3f V\nVCE=%7.3f V" %(Vcc, VB, VCE))
print("\nR1=%7.3f Ohm\nR2=%7.3f Ohm\nRC=%7.3f Ohm\nRE=%7.3f Ohm" %(R1, R2, RC, RE))
print("\nC1=%7.3f uF\nC2=%7.3f uF\nCE=%7.3f uF" %(C1, C2, CE))
print("\nZi=%7.3f Ohm\nZo=%7.3f Ohm" %(Zi, Zo))
```

Figure 15.2: Program listing.

An example run of the program is shown in Figure 15.3. In this example the following specifications were assumed:

$V_{in} = 0.010 \text{ V}$
 $V_{out} = 0.4 \text{ V}$
 $R_s = 100 \Omega$
 $R_L = 15 \text{ k}\Omega$
 $f_i = 40 \text{ Hz}$
 $V_{cc} = 12 \text{ V}$
 $\beta = 100$

It is clear that the required voltage gain of the transistor (without the source and load resistances) works out at $0.4\text{V} / 0.010\text{V} = 40$.

```
Single Stage Common Emitter Amplifier Design
=====
Enter supply voltage (Volts): 12
Enter Vin (Volts): 0.010
Enter Vout (Volts): 0.4
Enter Rs (ohms): 120
Enter RL (Ohms): 15000
Enter lowest frequency to amplify (Hz): 40
Enter current ratio (B): 100

RESULTS:
=====
Gain= 45.142
IC= 2.000 mA
IB= 0.020 mA

Vcc= 12.000 V
VB= 5.400 V
VCE= 6.000 V

R1=33000.000 Ohm
R2=27000.000 Ohm
RC=650.000 Ohm
RE=2350.000 Ohm

C1= 34.510 uF
C2= 63.866 uF
CE= 16.931 uF

Zi=1152.950 Ohm
Zo=623.003 Ohm
```

Figure 15.3: Example run of the program.

CHAPTER 16 • Active Low-Pass Filter Design

16.1 Project

In this project we will design second-order Butterworth type active low-pass active filters. Users enter the cut-off frequency of the filter, and the program will calculate and display the component values.

Background Information: There are several forms of second-order active low-pass filters. The one used in this project is the well-known popular Sallen-Key type filter whose circuit diagram is shown in Figure 16.1. The filter consists of two capacitors and two resistors. This type of filter has unity gain ($1\times$) in the pass-band.

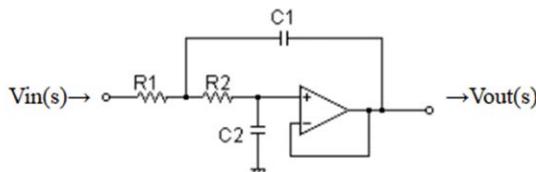


Figure 16.1: Second order low-pass filter.

It is possible to design Sallen-Key type filters with higher gains, but care should be taken since the high-gain filters can easily become unstable and oscillate, giving high overshoot in their time responses and high gains at their cut-off frequencies. The Q factor of a filter defines its quality, i.e. the damping of the response. For low-pass filters Q should have a value of 0.707. Higher values of Q may cause instability and overshoots in the time and frequency responses. Another parameter used in filter design is the damping factor, denoted with ξ . This is related to Q with the following equation:

$$\xi = \frac{1}{2Q}$$

The ideal value for ξ is also 0.707. Lower values of ξ give rise to oscillations and instability in the designed circuit. To maintain stability, the gain of an active filter must not be greater than 3. The relationship between the gain and Q factor is:

$$G = 3 - \frac{1}{Q}$$

Figure 16.2 shows the filter frequency response for various values of ξ . It is clear from this figure that $\xi = 0.707$ gives a flat response.

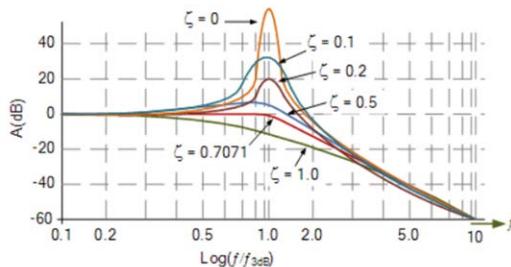


Figure 16.2: Filter frequency response for different values of ξ .

The cut-off frequency of the filter in Figure 16.1 is given by:

$$f = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}}$$

Where the frequency is in hertz, resistors in ohms, and capacitors in farads. In order to make sure that $Q = 0.707$, the following conditions must be satisfied:

Let $R_1 = R_2 = R$, and $C_1 / C_2 = n$,

then we have to make sure that

$$C_1 / C_2 \geq 4 Q^2, \text{ i.e. } C_1 / C_2 \geq 2, \text{ or } C_2 / C_1 \leq 0.5$$

The filter design equation then becomes:

$$f = \frac{1}{2\pi R C_2 \sqrt{n}}$$

or,

$$R = \frac{1}{2\pi f C_2 \sqrt{n}}$$

The design steps are then as follows:

- Read the filter cut-off frequency.
- Select a value for C_2 .
- Select a value for C_1 , making sure that $C_1 / C_2 \geq 2$. For $Q = 0.707$, select $C_1 / C_2 = 2$.
- Calculate the required R .

Program Listing: Figure 16.3 shows the program listing (program: **lowpass.py**). At the beginning of the program a heading is displayed and the user is required to enter the filter cut-off frequency, and C_1 and C_2 in microfarads. The program calculates and displays the required resistors, making sure that the above inequality is satisfied. The user is then asked to enter the actual physical resistor values to be used and displays the resulting real-life cut-off frequency. If the user is not happy with the calculated cut-off frequency, then the program is restarted allowing him/her to select new values for the capacitors.

```
#=====
#      Second Order Low-Pass Active Filter Design
# =====
#
# This program designs a second order low-pass filter
#
# Author: Dogan Ibrahim
# File  : lowpass.py
# Date  : July 2020
#=====
import math

yn = 'n'
while yn == 'n':
    print("Second Order Low-Pass Active Filter Design")
    print("===== ")
    #
    # Read the cut-off frequency and C1, C2
    #
    C1 = 1
    C2 = 1
    f = float(input("Enter the cut-off frequency (Hz): "))

    while C1/C2 < 2:
        C2 = float(input("Enter C2 (microfards): "))
        C = 2 * C2
        C1 = float(input("Enter C1 (microfarads) C1 >= %d : " %(C)))

        C1F = C1 / 1000000.0
        C2F = C2 / 1000000.0
    #
    # Calculations
    #
    n = C1F / C2F
    r = 2 * math.pi * f * C2F * math.sqrt(n)
    R = 1 / r

    #
```

```

# Display the results
#
print("\nRESULTS:")
print("====")
print("C1=%7.3f uF\nC2=%7.3f uF\nR1=%7.3f Ohm\nR2=%7.3f Ohm" %(C1,C2,R,R))

print("\nEnter the actual physical resistors to be used:")
Rnew = float(input("Enter R1, R2 (Ohms): "))
fn = 2 * math.pi * Rnew * C2F * math.sqrt(n)
fnew = 1 / fn
print("Actual cut-off frequency = %7.3f Hz" %(fnew))
yn = input("Are you happy with the actual cut-off frequency (yn)? ")
yn = yn.lower()

```

Figure 16.3: Program listing.

An example run of the program is shown in Figure 16.4. In this example, the following specifications were used:

Cut-off frequency = 1000 Hz
 $C_2 = 6 \mu F$
 $C_1 = 12 \mu F$

```

pi@raspberrypi:~ $ python3 lowpass.py
Second Order Low-Pass Active Filter Design
=====
Enter the cut-off frequency (Hz): 1000
Enter C2 (microfarads): 6
Enter C1 (microfarads) C1 >= 12 : 12

RESULTS:
=====
C1= 12.000 uF
C2= 6.000 uF
R1= 18.757 Ohm
R2= 18.757 Ohm

Enter the actual physical resistors to be used:
Enter R1, R2 (Ohms): 18
Actual cut-off frequency = 1042.033 Hz
Are you happy with the actual cut-off frequency (yn)? ■

```

Figure 16.4: Example run of the program.

The resistor values are calculated to be $R_1 = R_2 = 18.759 \Omega$. Selecting the nearest physical value as 18Ω gives the cut-off frequency as 1042.033 Hz. We accept this value as close enough to 1,000 Hz and respond with **y** to terminate the program.

The application given in the following link can be used to display the frequency response, **Q** factor etc of a low-pass Sallen-Key type filter:

<http://sim.okawa-denshi.jp/en/OPstool.php>

Using this application, the transfer function, Q factor, damping factor, the frequency and phase responses of the designed filter with the new resistors is shown in Figure 16.5 and Figure 16.6.

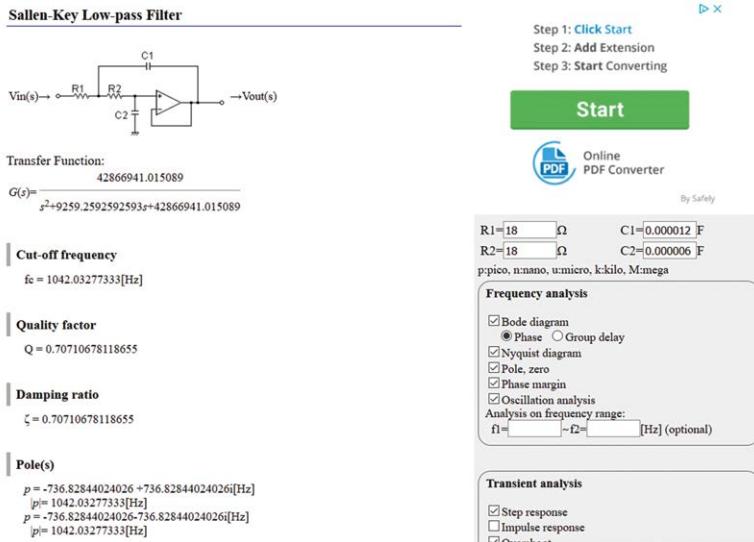


Figure 16.5: Designed filter specifications.

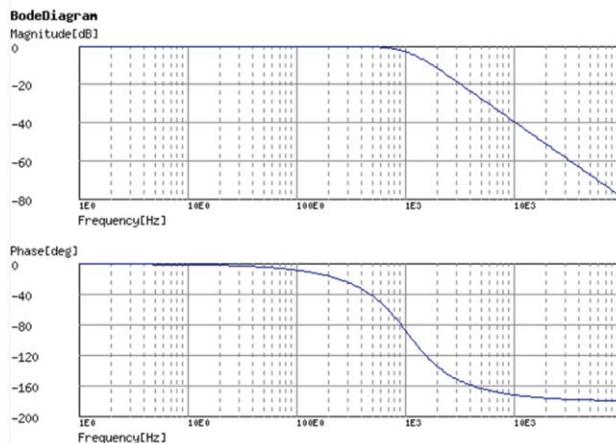


Figure 16.6: Frequency and phase responses of the designed filter.

The above application also shows the step response of the designed filter. This is shown in Figure 16.7.

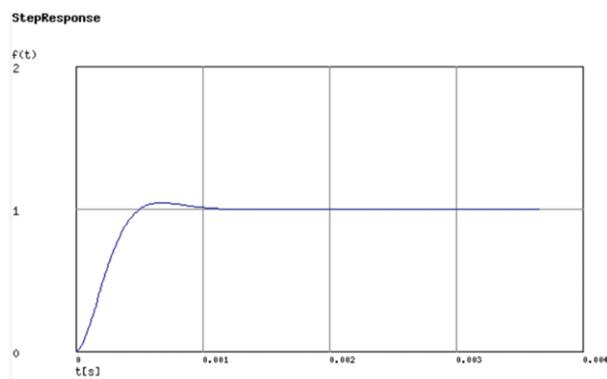


Figure 16.7: Step-time response of the designed filter.

CHAPTER 17 • Morse Code Exerciser

17.1 Project: MCE with User-Entered Characters

In this project we will design a Morse Code Exerciser (MCE) where the Morse code of a text of letters entered by the user are sent to a buzzer connected to the Raspberry Pi. The timing of the code can be set by the user via the PC keyboard.

Background: The information given here may not be new to some of the readers, but it is given for completeness.

The timing in Morse code is as follows:

- a *dit*: 1 unit
- a *dah*: 3 units
- character spacing between *dit* and *dah* of a character: 1 unit
- character spacing between the characters of a word: 3 units
- word spacing: 7 units

The speed of a Morse code is specified by how many words per minute a person can send or receive. In most Amateur radio exams candidates are expected to send and receive at least 12 words per minute.

The word **PARIS** is used as the standard word which consists of 50 units of time:

P:	.--.	1 1 3 1 3 1 1 (3)	14 units
A:	.-	1 1 3 (3)	8 units
R:	.-.	1 1 3 1 1 (3)	10 units
I:	..	1 1 1 (3)	6 units
S:	...	1 1 1 1 1 [7]	12 units

Where () is the inter-character time, and [] is the inter-word time. Notice that at the end of a character we do not insert a dot time, but instead an inter-character time (3 units). Similarly, at the end of a word, we do not insert a dot time but an inter-word time (7 units).

Total units = 50

The formula for the words per minute can be calculated as follows:

Let **wpm** = words per minute.

Then seconds per word is given by: **spw** = 60 / wpm.

If we take *dits* per word to be 50 as the standard, then

Seconds per *dit*, **spd** = 60 / (50 wpm), or **spd** = 1.2 / wpm.

We can specify this in milliseconds, giving:

$$\text{Milliseconds per } dit, \text{ mpd} = 60000 / (50 \text{ wpm}) = 1200 / \text{wpm}$$

Therefore, given the required words per minute, we can calculate the time of each *dit* in milliseconds as:

$$\text{mpd} = 1200 / \text{wpm}$$

For example, if the required words per minute is 12, then

$$\text{mpd} = 1200 / 12 = 100 \text{ milliseconds}$$

i.e. we have to allow 100 ms for the basic unit of timing.

Similarly, if the required words per minute is 20 then the bit timing should be $1200 / 20 = 60$ ms.

Block Diagram: Figure 17.1 shows the block diagram of the project.

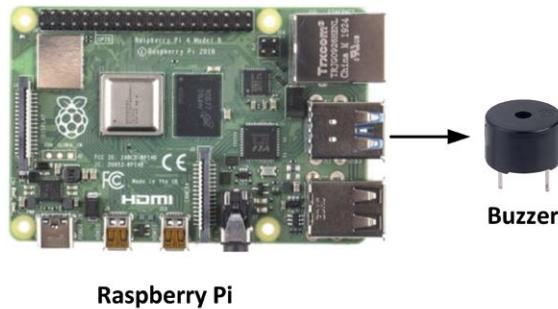


Figure 17.1: Block diagram of the project.

Circuit Diagram: There are essentially two types of buzzers available: passive and active. Passive buzzers can give sounds at different frequencies, and input to such buzzers are usually signals at the required frequencies. Active buzzers, on the other hand, generate a single tone of sound when a logic 1 is applied across their terminals. In this project an active buzzer is used for simplicity.

The buzzer is connected to pin GPIO26 of the Raspberry Pi through a transistor switch (any NPN transistor). The transistor switch is used here to increase the loudness of the buzzer.

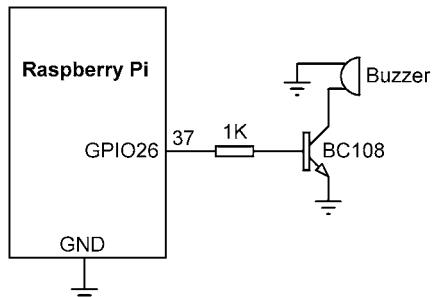


Figure 17.2: Circuit diagram of the project.

Program Listing: The program listing is shown in Figure 17.3 (Program: **Morse.py**). At the beginning of the program, the RPi's GPIO26 is configured as an output. Then, the Morse code of letters, numbers, and some symbols are defined as a dictionary, stored in **Morse-Code**. Functions **dot** and **dash** send out a dot and a dash at the correct timings. Function **dot** activates the buzzer for one unit time, while function **dash** activated the buzzer for three unit times. The user is then prompted to enter the required words per minute, and this is stored in variable **wpm**. Then, the dot time, dash time, inter-character time, and inter-word time are defined.

Inside the program loop, the user is prompted to enter the text whose Morse code is to be sent to the buzzer, and this gets stored in variable **data**. The characters of **data** are then read, converted into upper case, equivalent Morse code is read from the dictionary, and if it is a dot then function **dot** is called. If it is a dash, function **dash** is called, if it is a space then inter-word space is assumed, and a delay is introduced. The characters are displayed on the screen as they are converted into Morse code and sent to the buzzer.

Notice that, as described earlier, at the end of a character we do not insert a dot time, but we insert an inter-character time (three units). Similarly, at the end of a word, we do not insert a dot time, but we insert an inter-word time (seven units). If a dot is the last symbol in a character and the next symbol is another character, and we define dot to be buzzer ON, then one unit time delay, then buzzer OFF and one unit time delay, then we have to set the inter-character time to two units and not to three. Similarly, if a dash is the last symbol in a character and the next symbol is another character, and we define dash to be buzzer ON, then a 3-unit time delay, then buzzer OFF and one unit time delay, then we have to set the inter-word time to two units and not to three. If the next symbol is a space character (next word) then the delay must be set to six units, not seven.

```
#-----#
#          MORSE CODE EXERCISER
#          =====
#
# This is a Morse code exercise program. The user enters the
# speed (words per minute) and then a text. The text is converted
# into Morse code and is sent to a buzzer (or LED)
```

```
#  
# Author: Dogan Ibrahim  
# File : Morse.py  
# Date : July, 2020  
#-----  
import RPi.GPIO as GPIO          # Import RPi  
import time                     # Import time  
  
GPIO.setwarnings(False)  
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM  
Buzzer = 26                      # Buzzer pin  
  
GPIO.setup(Buzzer, GPIO.OUT)      # Buzzer is output  
GPIO.output(Buzzer, 0)            # Disable Buzzer  
  
#  
# Morse code table as a dictionary  
#  
MorseCode = {' ': ' ',  
             '0': '-----',  
             '1': '.----',  
             '2': '..---',  
             '3': '...--',  
             '4': '....-',  
             '5': '.....',  
             '6': '-....',  
             '7': '--...',  
             '8': '---..',  
             '9': '----.',  
             ':': '---... ',  
             ';': '-.-.-.',  
             '?': '..---..',  
             '/': '-...-.',  
             '.': '.-.-.-',  
             ',': '--..--',  
             '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': '--..'}

#
# Output a dot
#
def Dot():
    GPIO.output(Buzzer, 1)
    time.sleep(DotTime)
    GPIO.output(Buzzer, 0)
    time.sleep(DotTime)

#
# Output a dash
#
def Dash():
    GPIO.output(Buzzer, 1)
    time.sleep(DashTime)
    GPIO.output(Buzzer, 0)
    time.sleep(DotTime)

try:

#
# Get the required words per minute, calculate timings
#
    wpm = int(input("Required words per minute: "))
    UnitTime = 1.2/wpm
    DotTime = UnitTime
    InterCharTime = 2 * UnitTime
    DashTime = 3 * UnitTime
    WordTime = 6* UnitTime
```

```

while True:
    data = input("Enter the text to send: ")
    print("                                     ",end="",flush=True)
    for letters in data:
        print(letters,end="",flush=True)
        for code in MorseCode[letters.upper()]:
            if code == '-':
                Dash()
            elif code == '.':
                Dot()
            elif code == ' ':
                time.sleep(WordTime)
            time.sleep(InterCharTime)
        print("")
        print("")

except KeyboardInterrupt:
    print("")
    GPIO.cleanup()

```

Figure 17.3: Program: Morse.py.

Figure 17.4 shows an example run of the program. Here, the words per minute is set to 12, and the 'sentence' **Hello from Raspberry Pi** is converted into Morse code. Notice that the characters of the entered text are displayed under the text as they are converted into Morse code and sent to the buzzer.

```

pi@raspberrypi:~ $ python3 Morse.py
Required words per minute: 12
Enter the text to send: Hello from Raspberry Pi
Hello f■

```

Figure 17.4: Example run of the program.

17.2 Project: MCE sending randomly generated characters

In this project, random characters will be generated from the Morse code dictionary and they will be converted into Morse code and sent to the buzzer. The characters will be displayed on the PC screen as they are converted.

The block diagram and circuit diagram of the project are as in the previous project.

Program Listing: Figure 17.5 shows the program listing (Program: **Morse2.py**). For the most part, this program is similar to the one given in Figure 17.3. As before, the program prompts the user to enter the required words per minute (wpm). A random integer number is then generated between 1 and 42 (which is the size of the MorseCode dictionary). This number is used to index the dictionary and get a random character and its Morse code. This code is then sent to the buzzer. Notice that two lists are formed with the names **values** and **keys** from the MorseCode dictionary, and these lists are indexed by the random number. Characters sent to the buzzer are also displayed on the PC screen.

```
#-----
#                      MORSE CODE EXERCISER
#                      =====
#
# This is a Morse code exercise program. The program generates
# random characters which are converted into Morse code and sent
# to the buzzer
#
# Author: Dogan Ibrahim
# File  : Morse2.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi
import time                     # Import time
import random                   # Import random

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM
Buzzer = 26                      # Buzzer pin

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

#
# Morse code table as a dictionary
#
MorseCode = {' ': ' ', '0': '-----',
              '1': '.----', '2': '..---',
              '3': '...--', '4': '....-',
              '5': '.....', '6': '-....',
              '7': '--...', '8': '---..',
              '9': '----.', ':': '---..',
              ';': '-.-.-.', '?': '...--..',
              '/': '-...-.', '.': '.-.-.-',
              ',': '--...-',
              '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': '--..'}

#
# Output a dot
#
def Dot():
    GPIO.output(Buzzer, 1)
    time.sleep(DotTime)
    GPIO.output(Buzzer, 0)
    time.sleep(DotTime)

#
# Output a dash
#
def Dash():
    GPIO.output(Buzzer, 1)
    time.sleep(DashTime)
    GPIO.output(Buzzer, 0)
    time.sleep(DotTime)

try:
    #
    # Get the required words per minute, calculate timings
```

```
#  
wpm = int(input("Required words per minute: "))  
UnitTime = 1.2/wpm  
DotTime = UnitTime  
InterCharTime = 2 * UnitTime  
DashTime = 3 * UnitTime  
WordTime = 6* UnitTime  
  
values = MorseCode.values()  
values_list = list(values)  
keys = MorseCode.keys()  
keys_list = list(keys)  
  
while True:                                # Do forever  
    r = random.randint(1, 42)                 # Random number  
    c = values_list[r]                       # Get a value  
    d = keys_list[r]                         # Get its key  
    print(d, end="", flush=True)              # Display key  
  
    for code in c:                           # Do for code  
        if code == '-':                      # If dash  
            Dash()  
        elif code == '.':                    # if dot  
            Dot()  
    time.sleep(InterCharTime)  
  
except KeyboardInterrupt:  
    print("")  
    GPIO.cleanup()
```

Figure 17.5: Program: Morse2.py.

An example run of the program is shown in Figure 17.6.

```
pi@raspberrypi:~ $ python3 Morse2.py  
Required words per minute: 12  
O;G./JBX6F2J?K4F4V.U,QRP?OK5GYV9LI,.E;:8V7H4EUAN7█
```

Figure 17.6: Example run of the program.

17.3 Project: MCE with Rotary-Encoder WPM Setting and CD readout

In this project we will use an LCD and a rotary encoder to set the speed of the generated Morse code. Turning the knob of the rotary encoder increments the wpm, while turning it the other way decrements it. Random characters will be generated as in the previous project, and these characters will be converted into their corresponding Morse codes and then sent to the buzzer. The generated characters will be displayed at the bottom row of the LCD.

The program can be included at the end of file **/etc/rc.local** so that it starts automatically after reboot, thus making it autonomous. i.e. not requiring you to log in via a PC to start the program.

Block Diagram: Figure 17.7 shows the block diagram of the project.



Figure 17.7: Block diagram of the project.

A rotary encoder is a device that looks like a potentiometer. It senses the rotation and direction of its knob. The device has two internal contacts that make and break a circuit as the knob is turned. As the knob is turned, a click is felt that indicates that the knob has been rotated by one position. With a simple logic we can determine the direction of rotation.

A rotary encoder has the following pins:

GND: power supply ground.

Vcc (+): power supply.

CLK: This is an output pin used to determine the amount of rotation. Each time the knob is rotated by one click in either direction, the CLK output goes to HIGH and then LOW.

DT: This is an output similar to CLK pin, but it lags the CLK by 90 degrees. This output is used to determine the direction of rotation.

SW: This is an active-LOW pushbutton. When the knob is pushed, the voltage goes LOW.

In our project, each rotation (i.e. click) of the knob will increment (or decrement) the words per minute count by 1. Turning the knob in one direction will increment the count by 1, while turning it in the other direction will decrement it by 1. When the required value is reached, the user has to push the knob so that the program starts generating random characters of Morse code on the buzzer.

Circuit Diagram: Figure 17.8 shows the circuit diagram of the project. The I²C LCD is connected to Raspberry Pi as in the previous LCD based projects, where the SDA and SCL pins are connected to GPIO2 and GPIO3 respectively. The buzzer is connected to pin GPIO26

through a transistor switch as in the previous project. The rotary encoder is connected as follows:

Rotary encoder	GPIO pin
CLK	GPIO19
DT	GPIO13
SW	GPIO6
GND	GND
Vcc(+)	3.3V

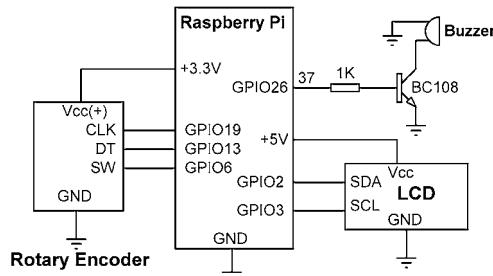


Figure 17.8: Circuit diagram of the project.

Program Listing: Figure 17.9 shows the program listing (Program: **Morse3.py**). At the beginning of the program the LCD library is imported, and the type of LCD is defined. Then the interface between the rotary encoder and the Raspberry Pi are defined and pins CLK, DT, and SW of the rotary encoder are configured as inputs.

The words per minute value increments or decrements as the user turns the knob of the rotary encoder clockwise or anticlockwise respectively, where the minimum value is 1. The LCD displays the wpm at any time. Pushing in the knob of the rotary encoder starts the program loop where random characters are generated and their Morse codes are sent to the buzzer. The LCD displays the characters in real time as they are generated randomly by the program.

You can start the program from the command line as:

```
pi@raspberrypi:~ $ python3 Morse3.py
```

```
#-----
#          MORSE CODE EXERCISER
#          =====
#
# This is a Morse code exercise program. The program generates
# random characters which are converted into Morse code and sent
# to the buzzer. In this program the speed is set using a rotary
# encoder and an I2C LCD. Turning the rotary encoder know increments
# or decrements the wpm. Pushing in the knob start generating the
```

```
# Morse codes. The generated characters are displayed at the bottom
# row of the LCD
#
# Author: Dogan Ibrahim
# File : Morse3.py
# Date : July, 2020
#-----

import RPi.GPIO as GPIO          # Import RPi
import time                      # Import time
import random                    # Import random
from RPLCD.i2c import CharLCD

LCD = CharLCD('PCF8574', 0x27)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM
Buzzer = 26                      # Buzzer pin

#
# Rotary encoder connections
#
CLK = 19                         # CLK pin
DT = 13                           # DT pin
SW = 6                            # SW pin

GPIO.setup(Buzzer, GPIO.OUT)        # Buzzer is output
GPIO.setup(CLK, GPIO.IN)           # CLK is input
GPIO.setup(DT, GPIO.IN)
GPIO.setup(SW,GPIO.IN, pull_up_down=GPIO.PUD_UP)

GPIO.output(Buzzer, 0)              # Disable Buzzer

#
# Morse code table as a dictionary
#
MorseCode = {' ': ' ', '0': '-----', '1': '.----', '2': '..---', '3': '...--', '4': '....-', '5': '.....', '6': '-....', '7': '--...', '8': '---..', '9': '----.', ':': '---..', ',' : '---...'}

---


```

```
';': '-.-..',
'?': '..---',
'/': '-...-',
'.': '.-.-.-',
',': '--....',
'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': '--..'}  
  
#
# Output a dot
#
def Dot():
    GPIO.output(Buzzer, 1)
    time.sleep(DotTime)
    GPIO.output(Buzzer, 0)
    time.sleep(DotTime)  
  
#
# Output a dash
#
def Dash():
    GPIO.output(Buzzer, 1)
```

```
time.sleep(DashTime)
GPIO.output(Buzzer, 0)
time.sleep(DotTime)

wpm = 1                                # Default value
ClkOldState = GPIO.input(CLK)             # Get CLK state
flag = 1
LCD.clear()                               # Clear LCD
LCD.cursor_pos = (0, 0)                  # At (0,0)
LCD.write_string("Enter WPM:")           # Heading
LCD.cursor_pos = (1, 0)
LCD.write_string("1")

try:

#
# Get the required words per minute. User turns the rotary
# encoder know to set the wpm. Also, calculate timings
#
while flag == 1:
    ClkState = GPIO.input(CLK)
    DTState = GPIO.input(DT)
    if ClkState != ClkOldState and ClkState == 1:
        if DTState != ClkState:
            wpm = wpm + 1
        else:
            wpm = wpm - 1
        if wpm == 0:
            wpm = 1
    LCD.cursor_pos = (1, 0)
    LCD.write_string("      ")
    LCD.cursor_pos = (1, 0)
    LCD.write_string(str(wpm))
    ClkOldState = ClkState

    SWState = GPIO.input(SW)                # Get state of SW
    if SWState == 0:                      # Knob pushed in?
        flag = 0                          # Clear flag

#
# We come here when the user pushes in the rotary encode knob
#
    UnitTime = 1.2/wpm
    DotTime = UnitTime
    InterCharTime = 2 * UnitTime
    DashTime = 3 * UnitTime
```

```
WordTime = 6* UnitTime

values = MorseCode.values()
values_list = list(values)
keys = MorseCode.keys()
keys_list = list(keys)

LCD.clear()                                # Clear LCD
LCD.cursor_pos = (0, 0)                      # At (0, 0)
s = "Working: " + str(wpm) + " wpm"        # Heading
LCD.write_string(s)
LCD.cursor_pos = (1, 0)                      # At (1, 0)
col = 0

#
# The Morse codes are generated in the loop below
#
while True:                                  # Do forever
    r = random.randint(1, 42)                  # Random number
    c = values_list[r]                        # Get a value
    d = keys_list[r]                          # Get its key
    #   print(d, end="", flush=True)          # Display key
    LCD.write_string(d)                      # Display chars
    col = col + 1
    if col > 16:
        col = 0
        LCD.cursor_pos = (1, 0)
        LCD.write_string("")                 # Clear LCD
        LCD.cursor_pos = (1,0)
    for code in c:                            # Do for code
        if code == '-':                      # If dash
            Dash()
        elif code == '.':                    # if dot
            Dot()
        time.sleep(InterCharTime)

except KeyboardInterrupt:
    print("")
    GPIO.cleanup()
```

Figure 17.9: Program: Morse3.py.

Figure 17.10 shows example displays on the LCD.



Figure 17.10: Example displays.

CHAPTER 18 • Voltmeter – Ammeter – Ohmmeter – Capacitance Meter

18.1 Project: Voltmeter

In this project we design a voltmeter to measure analogue voltages and display on an LCD.

Block Diagram: Figure 18.1 shows the block diagram of the project. An analogue-to-digital converter (ADC) chip is used to convert the analogue input voltage to digital, which is then displayed on the LCD in millivolts.

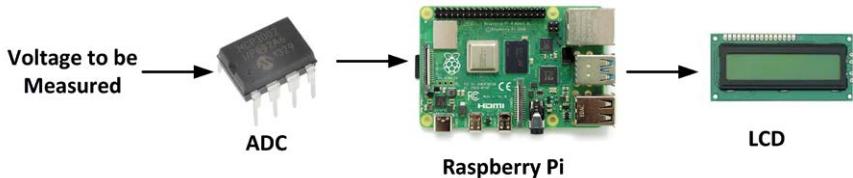


Figure 18.1: Block diagram of the project.

Circuit Diagram: In this project the MCP3002 ADC chip is used. This chip has the following basic 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 μ A active current

The MCP3002 is an *SPI bus compatible, 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 18.2 shows the pin configuration of the MCP3002.

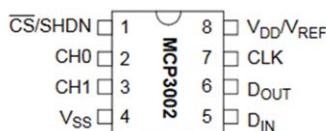


Figure 18.2: Pin configuration of the MCP3002.

The pin definitions are as follows:

Vdd/Vref:	Power supply and reference voltage input
CH0:	Channel 0 analogue input
CH1:	Channel 1 analogue 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, digital output code = $310.30 \times V_{in}$

each quantization level corresponds to $3300 \text{ mV} / 1024 = 3.22 \text{ mV}$ and this is the smallest voltage we can measure. 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 we 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.

The circuit diagram of the project is shown in Figure 18.3.

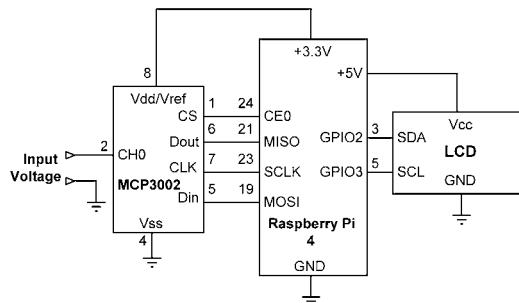


Figure 18.3: Circuit diagram of the voltmeter project.

Program Listing: Figure 18.4 shows the program listing (Program: **volts.py**). At the beginning of the program the SPI bus initialized, and its clock is configured. Function **GetAD-CData** reads data from analogue channel 0 and returns the data as a 10-bit result. the channel number (**channel_no**) is specified in the function argument as 0 or 1. Notice that we 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 must send eight 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 we must extract the correct 10-bit ADC data from this 24-bit data. The 24-bit data is in the following format ('X' is don't care bit):

XXXX XXXX XXXX DDDD DDDD DDXX

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

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

Thus, we 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 we get the 10-bit converted ADC data as:

DD DDDD DDDD

Inside the main program loop the analogue data is displayed on the bottom row of the LCD.

```
#-----  
#                    VOLTMETER  
#              ======  
#  
# This is a voltmeter program. Analog voltage to be measured is  
# applied to the ADC converter and is then displayed on an LCD  
#
```

```

# Author: Dogan Ibrahim
# File  : volts.py
# Date  : July, 2020
#-----
import time                      # Import time
import spidev                     # Import SPI
from RPLCD .i2c import CharLCD   # Import RPLCD

LCD = CharLCD('PCF8574', 0x27)
spi = spidev.SpiDev()
spi.open(0, 0)                    # Bus=0, device=0
spi.max_speed_hz=1953000          # SPI bus freq

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

LCD.cursor_pos = (0, 0)           # Cursor at (0,0)
LCD.write_string("Measured (mV):") # Heading

try:

#
# Read analog data every second and display on the LCD
#
    while True:
        adc = GetADCData(0)          # Read channel 0
        mV = adc * 3300.0 / 1023.0   # To mV
        mVstr = str(mV)[:7]          # To string
        LCD.cursor_pos = (1, 0)       # Cursor at (1,0)
        LCD.write_string("          ") # Clear
        LCD.cursor_pos = (1, 0)       # Cursor at (1,0)
        LCD.write_string(mVstr)       # Display mv
        time.sleep(1)                # wait 1 second

except KeyboardInterrupt:
    exit(0)

```

Figure 18.4: Program: volts.py.

An example display from the program is shown in Figure 18.5.



Figure 18.5: Example display produced by the voltmeter.

It is important to notice that the maximum input voltage must not exceed +3.3 V. If it is required to measure higher voltages then a resistive potential divider circuit should be used to lower the input voltage as shown in Figure 18.6, where two resistors, 10 kΩ and 100 kΩ are used. Using this circuit, the input voltage will be attenuated by a factor of 0.909. It is therefore necessary to multiply the result by 11 as shown below:

$$mV = adc * 3300.0 * 11.0 / 1023.0$$

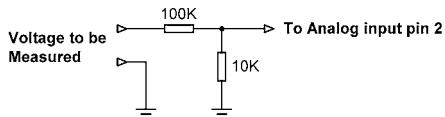


Figure 18.6: Resistive potential divider circuit.

18.2 Project: Ammeter

The design of an ammeter is easy after we have designed a voltmeter. The easiest setup is to use a known fixed resistor in series with the circuit whose current will be measured. By measuring the voltage across this resistor, we can calculate the current through the circuit by dividing the voltage by the resistance.

Assuming that the maximum current we want to measure at any time is 1 A, we can choose a 10-Ω fixed resistor and measure the voltage across this resistor. The maximum voltage will be 10 V. We have to make sure that the resistor has a power rating of at least 20 W (e.g. a wire-wound resistor).

18.3 Project: Ohmmeter

Designing an ohmmeter is easy. All we need is a fixed resistor in series with the unknown resistor. By measuring the voltage across the unknown resistor, we can calculate its value easily. The value of the unknown resistor is displayed on the LCD.

Circuit Diagram: The circuit diagram is shown in Figure 18.7. The voltage across the unknown resistor R_x is given by:

$$V_o = V_{in} \times R_x / (R_1 + R_x)$$

Or, the value of the unknown resistor R_x is given by:

$$R_x = V_o \times R_1 / (V_{in} - V_o)$$

But, V_{in} is 3.3 V. If we choose R_1 to be 3.3 kΩ then the unknown resistor R_x is given as:

$$R_x = V_o \times 3300 / (3300 \text{ mV} - V_o)$$

Where R_x is in ohms and V_o is in millivolts

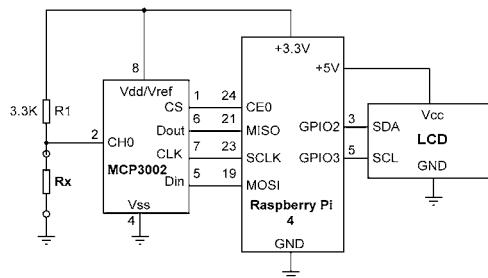


Figure 18.7: Circuit diagram of the project.

Program Listing: Figure 18.8 shows the program listing (Program: **resistor.py**). At the beginning of the program the ADC and LCD parameters are initialized as in the previous project. The program then calculates and displays R_x on the LCD using the above formula.

```
#-----
#          OHMMETER
#      =====
#
# This is an ohmmeter program. The unknown resistor is connected
# in series with a known resistor as a potential divider circuit
# and its value is calculated and displayed on the screen
#
# Author: Dogan Ibrahim
# File : resistor.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO
import time                                # Import time
import spidev                                # Import SPI
from RPLCD.i2c import CharLCD                # Import RPLCD
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

LCD = CharLCD('PCF8574', 0x27)              # LCD address
spi = spidev.SpiDev()
spi.open(0, 0)                               # Bus=0, device=0
spi.max_speed_hz=1953000                      # SPI bus freq
```

```
R1 = 3300                                # 3.3K resistor

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

LCD.clear()
LCD.cursor_pos = (0, 0)
LCD.write_string("Resistance (0hm):")

try:

    while True:
        m = GetADCData(0)
        Vo = m * 3300 / 1023                  # Vo
        Rx = Vo * R1 / (3300 - Vo)            # Calculate Rx
        Rx = str(Rx)[:6]
        LCD.cursor_pos = (1, 0)                # At (1, 0)
        LCD.write_string(Rx)                  # Display Rx
        time.sleep(1)
        LCD.cursor_pos = (1, 0)
        LCD.write_string("          ")
        LCD.cursor_pos = (1, 0)

except KeyboardInterrupt:
    GPIO.cleanup()
```

Figure 18.8: Program: resistor.py.

18.4 Project: Capacitance Meter

This is a simple capacitance meter project which can be used to measure capacitances fairly accurately in the microfarads range. The circuit is based on charging a capacitor through a fixed resistor and the value of the capacitor is found by finding the time constant of the circuit. The capacitor value is displayed on the LCD.

Block Diagram: Figure 18.9 shows the block diagram of the circuit.

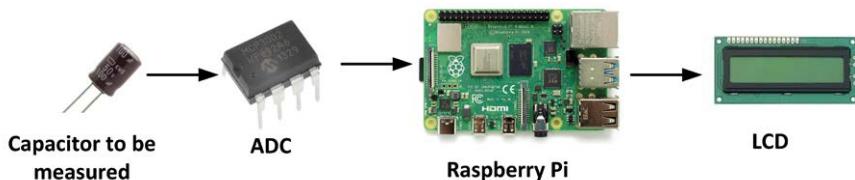


Figure 18.9: Block diagram of the project.

Circuit Diagram: The circuit diagram of the circuit is shown in Figure 18.10. A $10\text{ k}\Omega$ resistor in series with the unknown capacitor are connected in series to GPIO26 of the Raspberry Pi. The ADC and the LCD are connected as in the previous project.

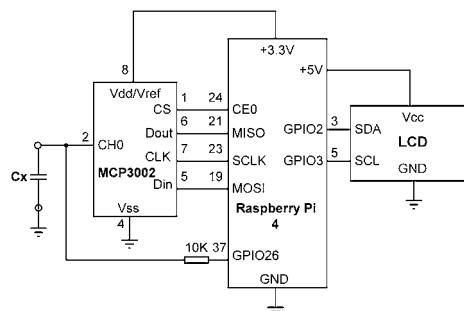


Figure 18.10: Circuit diagram of the capacitance meter project.

Program Listing: A $+3.3\text{ V}$ potential is applied to the resistor to charge the capacitor. The time constant of the circuit is given by $t = R C$. At the time constant, the voltage across the capacitor reaches 63.2% of its final value. By calculating the time constant and dividing it to the value of the resistor we can find the value of the unknown resistor.

Figure 18.11 shows the program listing (Program: **capacitance.py**). At the beginning of the program, the LCD and ADC parameters are initialized as in the previous program. Inside the program loop a logic 1 ($+3.3\text{ V}$) is applied to pin GPIO26 to charge the unknown capacitor through the $10\text{ k}\Omega$ resistor. The program then reads the internal clock time and stores it in variable **StartTime**, while continually reading the voltage across the capacitor until it becomes 63.2% of $+3.3\text{ V}$. Since the ADC is 10 bits, $+3.3\text{ V}$ corresponds to digital value 1023; 63.2% of 1023 equals 646 (as an integer number), and this number is used while reading the analogue input. When the voltage across the capacitor reaches 63.2% of its final value, the time is read and stored in variable **EndTime**. The difference between the **EndTime** and **StartTime** is the time taken to charge the capacitor up to its time constant. Dividing this time by the value of the resistor gives the value of the unknown capacitor. The voltage across the capacitor is then removed and the capacitor value is displayed on the LCD. The program waits until the capacitor discharges.

```
#-----
#          CAPACITANCE METER
#          =====
#
# This is a capacitance meter program. The unknown capacitor is
# charged through a fixed resistor and the time constant is measured.
# Then the value of the capacitor is calculated and displayed on LCD
#
# Author: Dogan Ibrahim
# File : capacitance.py
# Date : July, 2020
#-----
import RPi.GPIO as GPIO
import time                      # Import time
import spidev                     # Import SPI
from RPLCD.i2c import CharLCD    # Import RPLCD
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

LCD = CharLCD('PCF8574', 0x27)      # LCD address
spi = spidev.SpiDev()                # Bus=0, device=0
spi.open(0, 0)                      # SPI bus freq
spi.max_speed_hz=1953000

charge = 26                          # Charge resistor
resistor = 10000                     # 10K resistor

GPIO.setup(charge, GPIO.OUT)
GPIO.output(charge, 0)

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

#
# Read analog data every second and display on the LCD
#
#
while GetADCData(0) > 0:
    pass

while True:
```

```

LCD.cursor_pos = (0, 0)
LCD.write_string("Measured (uF):")      # Heading
GPIO.output(charge, 1)                  # Start charge
StartTime = time.time()                # Get start time
while GetADCData(0) < 646:             # Time constant?
    pass

EndTime = time.time()                  # Get end time
ElapsedTime = EndTime - StartTime     # Time constant
uf = (ElapsedTime / resistor) * 1000000 # Calculate C
mf = str(uf)[:7]                      # As string
LCD.cursor_pos = (1, 0)                # At (1, 0)
LCD.write_string(mf)                  # Display C
GPIO.output(charge, 0)

while GetADCData(0) > 0:               # Wait until discharged
    pass
time.sleep(1)
LCD.clear()                           # Clear LCD

```

Figure 18.11: Program: capacitance.py.

Figure 18.12 shows an example display.



Figure 18.12: Example display as output by the capacitance meter.

With a 10 k Ω charging resistor, the time constants are as follows for different values of capacitors:

Capacitor	Time constant
0.001 μ F	0.01 ms
0.01 μ F	0.1 ms
0.1 μ F	1 ms
1 μ F	10 ms
10 μ F	100 ms
100 μ F	1 s
1000 μ F	10 s

It is feasible to get more accurate results with small capacitors by increasing the resistor value. This is because the Raspberry Pi timing is not very accurate at microseconds level. Increasing the resistor value will result in long times to charge the capacitor. For example, with a 1 M Ω resistor the time constants are as given below. As you can see, it will take

nearly 2 minutes to measure a 100 μF capacitor:

Capacitor	Time constant
0.001 μF	1 ms
0.01 μF	10 ms
0.1 μF	100 ms
1 μF	1 s
10 μF	10 s
100 μF	100 s

Readers may be interested to know there are professional quality low-cost LCR meters available in the market. One such example is the Atlas LCR40 (Figure 18.13) resistance-capacitance-inductance meter from Peak Electronic Design Ltd. This device has the following measuring ranges features:

- Resistance: 1 Ω to 2 M Ω
- Capacitance: 0.5 pF to 10,000 μF
- Inductance: 1 μH to 10 H



Figure 18.13: Peak Electronic LCR meter type Atlas LCR40.

CHAPTER 19 • Frequency Counter

There are many cases where we want to know the frequency of a signal. Frequency counters are one of the important instruments used by all radio amateurs. Without a frequency counter the only other option to find out the frequency of a signal is to use an oscilloscope.

19.1 Project: Frequency Counter

In this project we will be using a Raspberry Pi to design a frequency counter circuit. The frequency of the measured signal will be displayed on an I²C LCD.

The signal whose frequency is to be measured is initially converted into square wave so that it is easier to interface to a processor. There are basically three methods to measure the frequency of a signal as described below.

Measuring the period: In this method the signal is applied to one of the digital pins of the processor. The processor measures the period of the signal by using an internal timer. Here, the timer is started on the rising edge of the signal, and is stopped on the next rising edge. Knowing the period, we then take its inverse to calculate the frequency. Although this method can give accurate results it requires very accurate timer. Raspberry Pi is not an MCU as it has an operating system and its internal timer is not suited to measuring the frequency using this method.

Measuring the pulses (internal timer): In this method an internal timer is used as a gate. The processor starts a timer with a known duration (e.g. one second) and counts the number of pulses received during this time period. The frequency is then calculated from the knowledge of this count. For example, if the gate time is one second, then the number of counts is directly proportional to the frequency of the signal. This method requires an internal processor counter which can be updated by an external signal, and is not well suited to Raspberry Pi.

Measuring the pulses (external timer): Here, an external counter chip is used to count the number of pulses in a given time duration. This count is then read by the processor and the frequency is easily calculated. In this project we will be using this third method.

Block Diagram: Figure 19.1 shows the block diagram of the project.

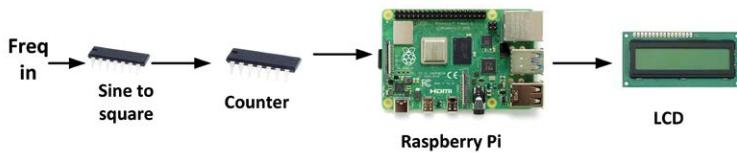


Figure 19.1: Block diagram of the project.

Circuit Diagram: In this project an SN74LV8154 type counter chip is used. The basic features of this chip are:

- dual 16-bit counter
- can be chained as a 32-bit counter
- 2 V to 5.5 V operating voltage
- 'Clear' input

The nice thing about this timer is that it can be chained as a 32-bit counter for more accuracy. Additionally, it operates with 3.3 V, thus making it compatible with the Raspberry Pi GPIO.

Figure 19.2 shows the pin configuration of the chip.

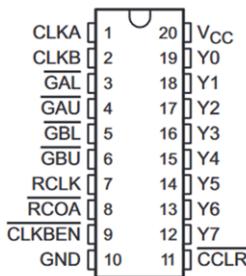


Figure 19.2: Pin configuration of the SN74LV8154.

The pin descriptions are (A and B are the two identical halves of the counter):

CCLR: Clear input

CLKA, CLKB: rising edge clock inputs

CLKBEN: clock B enable (active-LOW)

GAL, GAU: gate A lower and upper bytes (active-LOW puts A data on Y bus)

GBL, GBU: gate B lower and upper bytes (active-LOW puts B data on Y bus)

GND: power supply ground

RCLK: register clock (rising edge stores data internally)

RCOA: active-LOW when counter A is full, and ready to overflow to B

Vcc: power supply

Y0-Y7: Data outputs (Y0 is LSB)

The circuit diagram of the project is shown in Figure 19.3. The circuit requires a 1-Hz timing pulse for its RCLK input. On the rising edge of RCLK, the value of the 32-bit counter is stored internally so that it can be read through its Y0 - Y7 outputs in 4 stages. In this project the 1-Hz timing pulses are obtained using a GPS receiver (the *GPS Click* from www.mikroe.com). It is also possible to use a 32,768 Hz crystal with a 15-stage binary counter circuit as shown in Figure 19.4. This circuit is made up of a 14-bit counter and a D-type flip-flop, driven with a 32,768 Hz rectangular waveform.

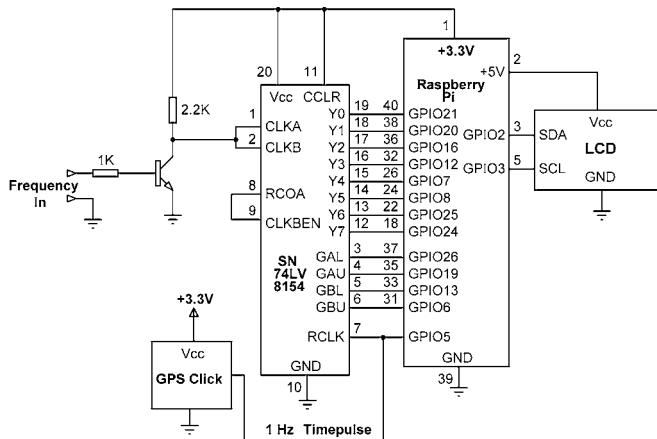


Figure 19.3: Circuit diagram of the project.

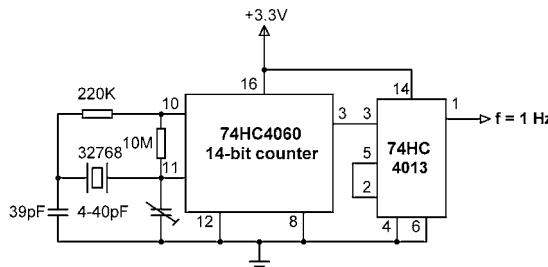


Figure 19.4: 15-stage binary counter with 32768 Hz crystal

Program Listing: Figure 19.5 shows the program listing (Program: **freq.py**). The operation of the project is as follows: RCLK input is pulsed every second by the time pulses received from the GPS receiver (these time pulses have about 1ms ON time). As soon the rising edge of this pulse is detected, the 32-bit value is read and stored in variable **Reading1**. The next reading is obtained on the arrival of the next pulse and this is stored in variable **Reading2**. The time difference between the two adjacent pulses is exactly one second. Therefore, the difference between the two readings (i.e. variable **difference**) is the number of counts in a second, i.e. the frequency of the waveform. By dividing this value by 1,000 we can find the frequency in kHz.

Function **GetByte** reads eight bits (a 'byte') of data from the output Y0 – Y7 of the counter and returns this value to the main program. Function **GetData** reads the 32-bit data stored inside the 74LV8154 counter chip by enabling inputs GAL, GAU, GBL, and GBU. The data is returned to the main program. reading the data four times.

```
#-----#
#          FREQUENCY COUNTER
#-----#
#
# This is a frequency counter software. The frequency is displayed
# on an LCD
#
# Author: Dogan Ibrahim
# File  : freq.py
# Date  : July, 2020
#-----
import RPi.GPIO as GPIO          # Import RPi
import time                      # Import time
from RPLCD.i2c import CharLCD

LCD = CharLCD('PCF8574', 0x27)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)           # GPIO mode BCM

ypins = [21, 20, 16, 12, 7, 8, 25, 24]      # Y0-Y7 pins
gpins = [26, 19, 13, 6]                     # GA/GB pins
data = [0, 0, 0, 0]

RCLK = 5                                     # RCLK pin
disable = 1
enable = 0
pwrof32 = 2**32

for j in range(4):
    GPIO.setup(gpins[j], GPIO.OUT)             # Set as outputs

for j in range(8):
    GPIO.setup(ypins[j], GPIO.IN)              # Set as inputs

GPIO.setup(RCLK, GPIO.IN)

for j in range(4):                            # Disable all
    GPIO.output(gpins[j], disable)

#
# Read a byte of data from specified 8 port bits
#
def GetByte():
    total = 0
    k = 0
    for j in range(8):
```

```

        data = GPIO.input(ypins[j])
        data = data << k
        total = total | data
        k = k + 1
    return total

#
# Get 32-bit data from the counter
#
def GetData():
    for j in range(4):
        GPIO.output(gpins[j], enable)
        data[j] = GetByte()
        GPIO.output(gpins[j], disable)
    count = data[0] | (data[1] << 8) | (data[2] << 16) | (data[3] << 24)
    return count

while GPIO.input(RCLK) == 1:                      # Wait if 1
    pass
while GPIO.input(RCLK) == 0:                      # Wait if 0
    pass

Reading1 = GetData()                            # Read Reading1
while GPIO.input(RCLK) == 1:
    pass

LCD.clear()
LCD.cursor_pos = (0, 0)
LCD.write_string("Frequency (kHz):")          # Heading

try:

    while True:                                # Do forever
        while GPIO.input(RCLK) == 0:              # While 0
            pass

        Reading2 = GetData()                    # Read Reading2
        while GPIO.input(RCLK) == 1:              # While 1
            pass

        difference = Reading2 - Reading1      # Calculate diff
        if difference < 0:
            difference = difference + pwrof32
        frequency = difference / 1000           # Frequency (kHz)
        frequency = str(frequency)[:12]         # As string
        LCD.cursor_pos = (1, 0)

```

```
LCD.write_string("          ")
LCD.cursor_pos = (1, 0)
LCD.write_string(frequency)      # Display
Reading1 = Reading2

except KeyboardInterrupt:
    GPIO.cleanup()
```

Figure 19.5: Program 'freq.py' for the frequency counter.

A resistive potential divider circuit can be used at the input for large inputs. Also, binary divider circuits can be used at the input of the circuit to increase the frequency range of the meter. For example, by using a 4-bit binary dividing circuit, the useful range of the meter can be extended by a factor of 16. Notice that have to make sure that the components you use can work at the chosen higher frequencies.

Figure 19.6 shows an example output of the program on the LCD. In this example, the input frequency was 500 kHz.



Figure 19.6: Example output from the RPi-driven frequency counter.

The project was constructed on a breadboard as shown in Figure 19.7.

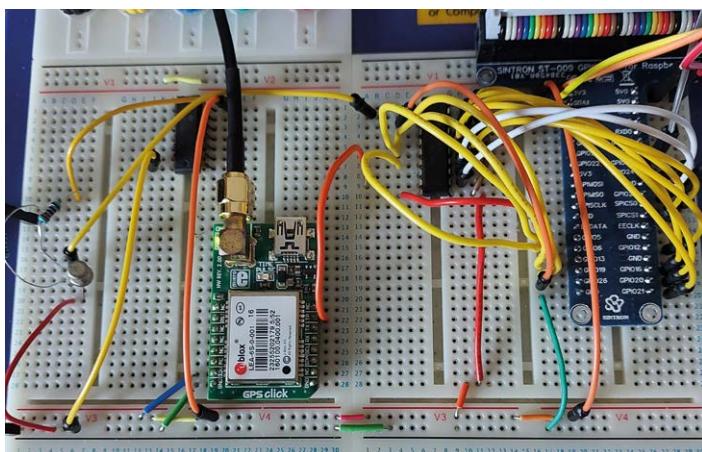


Figure 19.7: Frequency counter project constructed on a breadboard.

CHAPTER 20 • Raspberry Pi 4 Audio Input & Portable Power Supply

20.1 Raspberry Pi audio outputs

Raspberry Pi 4 has no audio inputs but boasts two audio outputs: **HDMI** and **headphone jack**. Users can switch between the two easily at any time.

If you have connected to your Raspberry Pi via the HDMI cable and if your monitor has speakers, then you can play audio over the HDMI cable, which is the default mode

The audio output can be selected in one of three ways: in Desktop volume control, in command line, and using **raspi-config**

In Desktop: Right-click the volume icon at the top right corner of the Desktop to see the audio output selector (Figure 20.1). As shown in Figure 20.1, the author's Raspberry Pi had the analogue output (headphone audio jack on the board) and HDMI. With the selector you can select between the internal audio outputs and any external audio device, such as USB sound card, Bluetooth audio device, etc. The selected device is highlighted in green. You can change the volume control by left-clicking the mouse while in this menu.

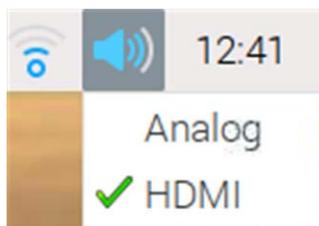


Figure 20.1: Audio selector.

In Command Line: The audio output can be switched to HDMI by entering the following command:

```
pi@raspberrypi:~ $ amixer cset numid=3 2
```

Output 2 is the HDMI; 1 is the analogue headphone jack. The default setting '0' is automatic.

You can display the **amixer** controls with the command:

```
pi@raspberrypi:~ $ amixer scontrols
Simple mixer control 'PCM', 0
```

To set the sound level to 70%, use the command:

```
pi@raspberrypi:~ $ amixer sset 'PCM' 70%
```

Using raspi-config: The steps are:

- Start raspi-config

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

- Select **Advanced options**
- Select **Audio**
- Select **Auto, or Force 3.5mm jack, or Force HDMI** (see Figure 20.2)
- Select OK and then Finish
- Restart your Raspberry pi for the changes to take effect

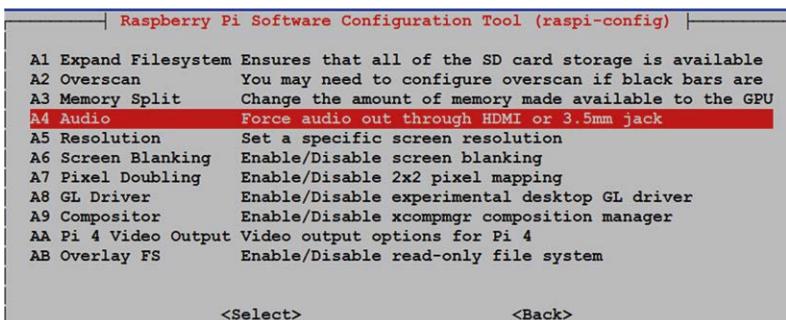


Figure 20.2: Select an audio output device.

20.1.1 Testing

After configuring the audio device, you may want to verify you are getting sound output. Connect a speaker to your headphone jack (or HDMI cable if you are using the HDMI audio output). Enter the command **speaker-test** from command line and you should hear white noise.

A better test is to use one of the sound files already installed on your Raspberry Pi. Connect your speaker and enter the following command. You should hear sound on your speaker:

```
pi@raspberrypi:~ $ aplay /usr/share/scratch/Media/Sounds/Vocals/  
Singer2.wav
```

You can enter the option **-v** to get information about the sound file being played, as shown in Figure 20.3 (only part of the display is shown).

```
pi@raspberrypi:~ $ aplay /usr/share/scratch/Media/Sounds/Vocals/Singer2.wav -v
Playing WAVE '/usr/share/scratch/Media/Sounds/Vocals/Singer2.wav' : Signed 16 bit
little Endian, Rate 11025 Hz, Mono
Plug PCM: Hardware PCM card 0 'bcm2835 ALSA' device 0 subdevice 0
Its setup is:
  stream      : PLAYBACK
  access       : RW_INTERLEAVED
  format       : S16_LE
  subformat    : STD
  channels     : 1
  rate         : 11025
  exact rate   : 11025 (11025/1)
  msbits       : 16
  buffer_size  : 5513
  period_size  : 1376
  period_time  : 124807
  tstamp_mode  : NONE
  tstamp_type  : MONOTONIC
  period_step  : 1
  avail_min    : 1376
```

Figure 20.3: Using the `-v` option.

20.2 Using an external USB audio input-output device

In many amateur radio applications we may need to use microphone input. Additionally, the sound output quality of the Raspberry Pi is not very good since it is based on PWM.

The author recommends the use of a USB sound adapter, such as the UGREEN (see Figure 20.4), or one with a built-in amplifier and volume control. Having a hardware volume control has the advantage that the volume can be set easily. The author uses the UGREEN adapter which has two sockets, one for the microphone input, and one for the speaker output.



Figure 20.4: UGREEN audio adapter.

The steps to configure the UGREEN adapter are:

- Connect the adapter to one of the USB ports of your Raspberry Pi.
- If you are in Desktop, right click the volume control as described in this Chapter.
- Select USB Audio Device (Figure 20.5).

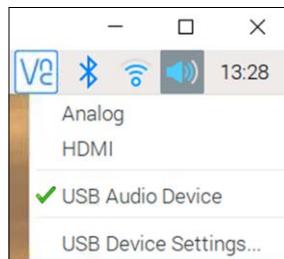


Figure 20.5: Select USB Audio Device.

You can configure the UGREEN adapter by selecting the USB Device Settings. You should enable the Speaker and the Microphone as shown in Figure 20.6 and make it the default.



Figure 20.6: Configuring the audio adapter.

20.2.1 Testing (1-2-3)

Enter the following command to list the audio devices on your system:

```
pi@raspberrypi:~ $ aplay -l
```

Notice that as shown in Figure 20.7, there are two devices listed as **card 0:** and **card 1:** where **card 1:** is our USB audio device.

- Test your audio output connection by entering the following command:


```
pi@raspberrypi:~ $ aplay /usr/share/scratch/Media/Sounds/Vocals/Singer2.wav -D sysdefault:CARD=1
```
- It might be worthwhile to make USB adapter to be your default audio device by entering the following commands:

```
pi@raspberrypi:~ $ sudo nano /usr/share/alsa/alsa.conf
```

- Scroll and find the following two lines:

```
defaults.ctl.card 0
defaults.pcm.card 0
```

- Change the 0 to a 1 to match the card number of the USB device:

```
defaults.ctl.card 1
defaults.pcm.card 1
```

- Exit from **nano** by entering **Cntrl+X** followed by **Y**.
- Restart your system for the changes to take effect.
- Enter the following command to test your connection:

```
pi@raspberrypi:~ $ speaker-test -c2 -t sine -f 500
```

- Test by playing the sound file:

```
pi@raspberrypi:~ $ aplay /usr/share/scratch/Media/Sounds/
Vocals/Singer2.wav -D plughw:1,0
```

- To test the microphone input, connect a microphone to the adapter, enter the following command and record some audio.

```
pi@raspberrypi:~ $ arecord -format=S16_LE -duration=5 -
rate=16000 -Dplug:default -file-type=raw test.raw
```

- Now playback the recording to verify that the microphone input is working.

```
pi@raspberrypi:~ $ aplay -format=S16_LE -rate=16000 test.raw -D
sysdefault:CARD=1
```

20.3 Powering the Raspberry Pi 4

Raspberry Pi 4 is normally powered through its USB-C port using a mains adapter. In mobile applications the user has the option of using mobile phone power packs to provide power to the Raspberry Pi 4. Mobile phone power packs are nowadays available in very high capacities and they are sources of +5 V supply voltage. When working with the Raspberry Pi 4 it will be necessary to purchase a USB-C adapter (see Figure 20.7) to connect to the power input port of the board.



Figure 20.7: USB-C adapter.

When powering a Raspberry Pi in a vehicle, the easiest option is to use a phone charger and connect it to the cigarette lighter of the car. You should however make sure that the lighter socket can supply at least 2 amps of current. You can also use a step-down DC-DC converter module to provide power to your Raspberry Pi. One such module is the Recom R-78B5.0-2.0 which can have 12 V input voltage and provides 5 V at up to 2 A output. It is recommended to use a current limiting fuse (say 3 A) as a means of protection since the car batteries can supply very large currents.

Another option to power your Raspberry Pi 4 when working remotely is by using the Power over Ethernet (PoE) feature of the Raspberry Pi 4. PoE is only supported by Raspberry Pi 4 Model B and Raspberry Pi 3 Model B+. There are many distributors supplying PoE modules (called PoE HAT) for the Raspberry Pi 4. The PoE HAT allows you to power your Raspberry Pi using power over Ethernet-enabled networks. One requirement is that the network where we are connected to needs to have power-sourcing equipment installed. There is no need to make any modifications to your Raspberry Pi. The HAT is plugged on top of the Raspberry Pi connector as shown in Figure 20.8. PoE HAT is controlled by the Raspberry Pi via an I²C interface. A small fan on the HAT ensures that the temperature of the board is within safe limits.



Figure 20.8: Raspberry Pi PoE HAT.

The basic feature of the PoE HAT shown in Figure 20.8 are (see <https://www.raspberrypi.org/products/poe-hat/>):

- fully isolated switched-mode power supply
- 37–57 V DC, Class 2 device
- 5 V/2.5 A DC output
- small fan for processor cooling
- fan control (fan is only ON when necessary)

CHAPTER 21 • Raspberry Pi FM Transmitter

21.1 Project: Raspberry Pi 4 VHF FM Transmitter

In this project we will see how a Raspberry Pi 4 can be turned into short-distance simple FM transmitter that can operate over the VHF FM broadcast band. ***It is important to mention at this point that although all amateur radio operators know it, transmitting over any broadcast band and/or setting up a pirate radio station are not legal. This project is provided for educational purposes only and the author or the publishers are not responsible for the illegal use of this project or part of it. Even if you are a licensed amateur radio operator, using this project to transmit radio waves on ham frequencies without any filtering between the Raspberry Pi and an antenna is most probably illegal because the generated squarewave carrier is very rich in harmonics, so the bandwidth requirements are likely not met.***

With a proper antenna, the transmitter described in this project can cover an area with a radius of about 50-100 meters. i.e. it can be used to transmit in your local area, or for example in a school or campus environment.

There are several Raspberry Pi compatible programs for transmitting. The one used in this project is called **PiFmAdv**. The program is based on the FM transmitter created by Oliver Mattos and Oskar Weigl, and later adapted to using DMA by Richard Hirst. Christophe Jacquet adapted it and added the RDS data generator and modulator. The transmitter uses the Raspberry Pi's PWM generator to produce VHF signals. This program generates frequency modulation (FM), with RDS (Radio Data System) data generated in real time. It can include monophonic or stereophonic audio. The program modulates the PLLC instead of the clock divider for better signal purity, which means that the signal is also less noisy. This has a great impact on stereo as its reception is way better.

The steps to use the program are as follows (see web link: <https://github.com/miegl/PiFmAdv>):

- pi@raspberrypi:~ \$ **sudo apt-get install libsndfile1-dev**
- pi@raspberrypi:~ \$ **git clone https://github.com/Miegl/PiFmAdv.git**
- pi@raspberrypi:~ \$ **cd PiFmAdv/src**
- pi@raspberrypi:~ /PiFmAdv/src \$ **make clean**
- pi@raspberrypi:~ /PiFmAdv/src \$ **make**
- pi@raspberrypi:~ /PiFmAdv/src

As an example to play the music file **mymusic.wav** at frequency 100.3 MHz, enter the following command. Notice that by default GPIO4 is used as the RF output, but there seems to be a problem using GPIO4 on Raspberry Pi 4 with this software. It is recommended to use, for example, GPIO20. Connect a short antenna to pin GPIO20 and enter (***note: it is not legal to broadcast over the VHF FM radio band***):

```
pi@raspberrypi:~ /PiFmAdv/src $ sudo ./pi_fm_adv --freq 100.3 --gpio 20  
--audio mymusic.wav
```

PiFmAdv supports large number of options (see the web link). Some of the commonly used options are:

- freq specifies the carrier frequency in MHz
- audio specifies the audio file to play (.wav)
- ps specifies the station name of the RDS broadcast (up to 8 characters)
- power specifies the gpio drive current. 0 = 2mA, 7 = 16mA (default is 5)
- cutoff specifies the cut-off frequency (Hz) of the internal low-pass filter (max: 15000)

Using a microphone

Connect the microphone to your audio adapter (see Chapter 20). Enter the following command (frequency = 100.3 MHz, GPIO20 is used):

```
pi@raspberrypi:~ /PiFmAdv/src $ sudo arecord -fS16_LE -r 44100
-Dplughw:1,0 -c 2 | sudo ./pi_fm_adv - -freq 100.3 - -gpio 20 --audio -
```

Note: The carrier signal generated by the Raspberry Pi isn't clean and can have up to the 5th harmonic, causing considerable interference and peril to other radio users. Thus, you should not connect it to an antenna before filtering out the harmonics even if you are a licensed user.

21.2 Project: RadioStation Click board

This is another small power programmable FM transmitter project. In this project we will be using a small board known as the **RadioStation Click**, manufactured by mikroElectronika (www.mikroe.com). Figure 21.1 shows the board which has a microphone socket, an antenna pin, and control pins.

RadioStation Click is a unique Click-series board that can be used to broadcast music via the FM radio broadcast band, operated by licensed users. It features the type Si4713-B30 chip from Silicon Labs, the best in class integrated FM broadcast stereo transmitter, which operates in the frequency range of 76 MHz to 108 MHz. It can also broadcast RDS/RDBS data. The Click board can be equipped with a small FM antenna, which is used to extend the broadcasting range. One of the advantages of this versatile FM broadcast stereo transmitter IC is that it needs almost no external components, making it easy to work with it. It's an ideal solution for use it in cellphones, MP3 players, portable media players, wireless speakers, personal computers and any other applications, where short-range radio broadcasting is wanted.

The Si4713-B30 IC performs the frequency modulation in the digital domain, in order to achieve high fidelity and optimal performance. The onboard DSP provides modulation adjustment and audio dynamic range control of the signal, for the best listening experience. Audio signal is processed to have the optimal dynamic qualities. Also, the Si4713 has programmable low-audio and high-audio level indicators that allow enabling and disabling of the carrier signal, based on the presence of audio content. The integrated DSP also takes care of the stereo MPX encoding and FM modulation of the signal. The low-level digital

intermediate frequency (IF) signal is then filtered out and sent to the output stage mixer, where it is converted to a radio frequency signal (RF). RF harmonics and noise get suppressed additionally to get a signal which is compliant with local regulations on RF transmission (FCC, ETSI, ARIB...). The RF signal is broadcast via the small antenna. The device supports all kinds of closed loop, dipole and monopole FM antennas.



Figure 21.1: RadioStation Click board.

The board operates with +3.3 V and has the following pin layout (NC = No Connection):

Pin no	Function	Description
1	NC	
2	RST	Reset (LOW to reset)
3	SEN	Set I ² C address
4	NC	
5	NC	
6	NC	
7	+3.3V	
8	GND	
9	GND	
10	NC	
11	SDA	I ² C data
12	SCL	I ² C clock
13	NC	
14	NC	
15	INT	Interrupt pin
16	NC	

The RadioStation Click board communicates with the host computer through the I²C interface. The SEN pin is used for selecting the I²C address. When CS is HIGH, the 7-bit I²C address is 0x63. When SEN is LOW, the 7-bit address is 0x11.

The Si4713-B30 IC has the capability of received signal measurement and evaluation. The antenna which is used to broadcast the signal can also be used to accept the incoming signal, sent by the receiving device. Although it can be used both to receive and transmit

signal, the antenna can't operate in both modes simultaneously. This feature can be useful when calibrating the transmission power of the Click board

Before using the Si4713-B30 chip, it's worthwhile to look at briefly the operation of this chip (see datasheets: <https://download.mikroe.com/documents/datasheets/Si4712-13-B30.pdf> and <https://www.silabs.com/documents/public/application-notes/AN332.pdf>)

The Si4713-B30 chip has the following basic features:

- 76-108 MHz band support
- Digital stereo modulator
- Programmable pre-emphasis
- Analogue/digital audio input
- RDS/RBDS encoder
- Programmable transmit level
- Audio level control

The Si4713 supports the standard European RDS and the US RBDS, including symbol decoding, block synchronization, and error correction. Using this feature you can transmit text (e.g. name of an artist playing a song, etc.) to an RDS/RBDS compatible FM receiver.

The TXO output of the chip connects directly to an antenna with one external inductor to provide harmonic filtering. Both digital and analogue audio inputs are supported. The analogue stereo audio input level is programmable and can be configured by software. The default peak input audio level is set to 636 mV.

A digital audio dynamic range control with programmable gain, threshold, attack rate, and release rate, is provided. This feature can be used to reduce the dynamic range of the audio signal, thus improving the listening on the FM receiver. A digital audio limiter prevents over-modulation of the transmitter output by dynamically attenuating peaks in the audio input signal that exceed a programmable threshold value.

Programmable pre-emphasis filter can be applied to the transmitted signal to increase the high audio frequencies and improve the signal to noise ratio (the receivers de-emphasize to attenuate high frequencies and so restore a flat frequency response). The standard European pre-emphasis time constant is 50 µs (75 µs in the US).

In addition to being a basic transmitter, the Si4713 provides three general-purpose GPIO pins to interface LEDs, etc. to the chip (note: these pins are not accessible from the RadioStation Click board).

Table 21.1 shows a list of the available Si4713-B30 commands (see the datasheet for further information). In addition to these basic commands, a large number of **Property Commands** are provided for programming features such as the pre-emphasis, line input level, audio deviation, RDS deviation, reference clock, audio threshold levels, audio dynamic range control, etc.:

Cmd	Name	Description
0x01	POWER_UP	Power up device and mode selection. Modes include FM transmit and analog/digital audio interface configuration.
0x10	GET_REV	Returns revision information on the device.
0x11	POWER_DOWN	Power down device.
0x12	SET_PROPERTY	Sets the value of a property.
0x13	GET_PROPERTY	Retrieves a property's value.
0x14	GET_INT_STATUS	Read interrupt status bits.
0x15	PATCH_ARGS	Reserved command used for patch file downloads.
0x16	PATCH_DATA	Reserved command used for patch file downloads.
0x30	TX_TUNE_FREQ	Tunes to given transmit frequency.
0x31	TX_TUNE_POWER	Sets the output power level and tunes the antenna capacitor
0x32	TX_TUNE_MEASURE	Measure the received noise level at the specified frequency.
0x33	TX_TUNE_STATUS	Queries the status of a previously sent TX Tune Freq, TX Tune Power, or TX Tune Measure command.
0x34	TX_ASQ_STATUS	Queries the TX status and input audio signal metrics.
0x35	TX_RDS_BUFF	Si4713 Only. Queries the status of the RDS Group Buffer and loads new data into buffer.
0x36	TX_RDS_PS	Si4713 Only. Set up default PS strings.
0x80	GPO_CTRL	Configures GPO3 as output or Hi-Z.
0x81	GPO_SET	Sets GPO3 output level (low or high).

Table 21.1: Si4713-B30 commands.

In this project, our interest is limited to the following commands:

0x30 TX_TUNE_FREQ tune to a given transmit frequency

0x31 TX_TUNE_POWER set the output power level and tune the antenna capacitor

Block Diagram: Figure 21.2 shows the block diagram of the project.

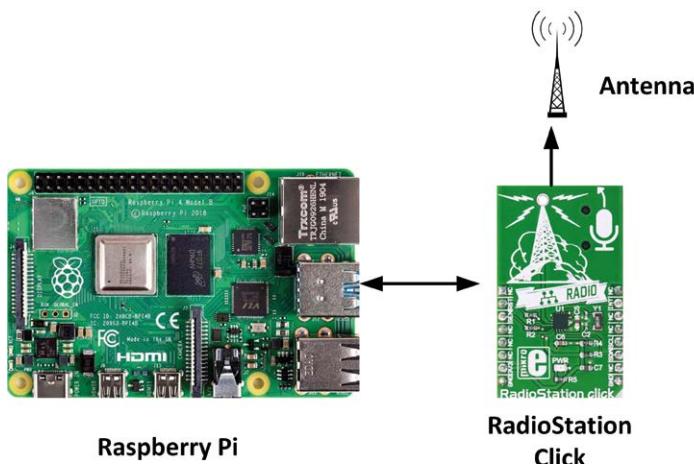


Figure 21.2: Block diagram of the project.

Circuit Diagram: Figure 21.3 shows the circuit diagram of the RadioStation Click board. The audio input and the antenna connector are at the left of the figure. The 16-pin mikro-BUS connector is located at the top right of the figure.

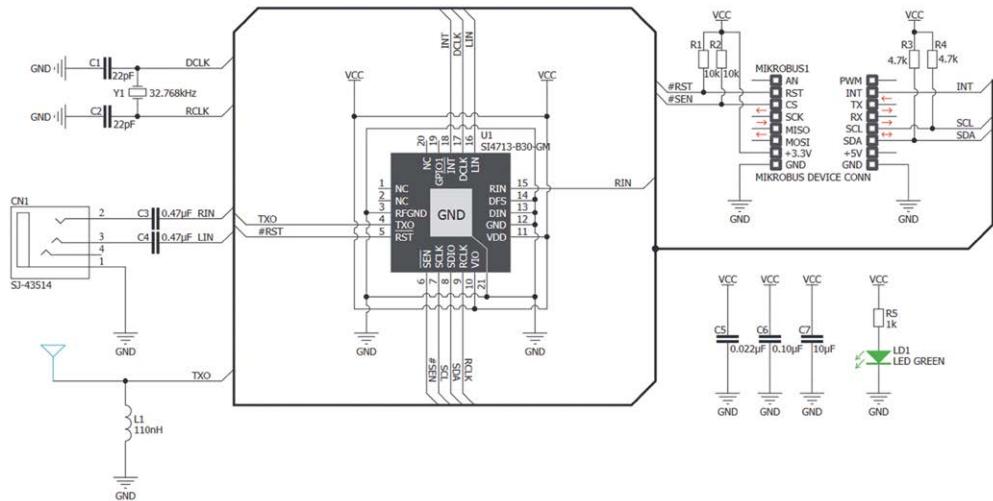


Figure 21.3: RadioStation Click board (www.mikroe.com).

The circuit diagram of the project is shown in Figure 21.4. The I²C address is set to 0x11 by connecting the CS pin to GND. I²C pins SDA and SCL are connected to Raspberry Pi pins SDA (GPIO2) and SCL (GPIO3) respectively.

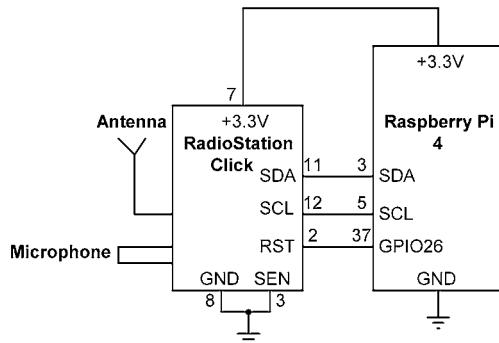


Figure 21.4: Circuit diagram of the project.

Operation of the Program

The operation of the program is shown in Figure 21.5 as a PDL (Program Description Language) which is a free format description of the algorithm of a program.

```

MAIN:BEGIN
    Display "RADIO STATION"
    Call Reset to reset the chip
    Call SendCommand to set to operate analog with crystal reference clock
    Call SetProperty to set pre-emphasis to 50µs
    Read required operating frequency from user
  
```

```
    Read required TX power from user
    Call SendCommand to set required operating frequency
    Call SendCommand to set required TX power
    Get operating frequency
    Display operating frequency
    Exit if Ctrl+C detected
```

MAINEND

Reset:BEGIN

```
    Ser RST HIGH
    Wait 0.5 second
    Set RST LOW
    Wait 0.5 second
    Set RST HIGH
    Wait 0.5 second
```

Reset:END

SendCommand:BEGIN

```
    Get the command type (first byte)
    Get the command arguments (remaining bytes)
    Send command with arguments to the chip
    Get response from the chip
    IF error in response THEN
        Display error message
        Exit program
    ENDIF
```

SendCommand:END

SetProperty:BEGIN

```
    Extract upper and lower bytes of the property
    Extract the upper and lower bytes of the data
    Call SendCommand to send the property to the chip
```

SetProperty:END

RcvTuneStatus:BEGIN

```
    Send command to get the Tune Status
    Return the upper and lower frequency bytes
```

RcvTuneStatus:END

PowerOff:BEGIN

```
    Clear RST
```

PowerOff:END

Figure 21.5: PDL of the program.

Program Listing: Figure 21.6 shows the program listing (Program: **station.py**). At the beginning of the program the RPi, time, and smbus modules are imported to the program. Pin RST is assigned to GPIO26 and this pin is configured as an output. Then, the commands and properties used in the program are defined.

The main program loop starts by displaying the heading RADIO STATION. Then the chip is reset by calling function Reset, and pre-emphasis is set to 50 μ s. The user is prompted to enter the required operating frequency. The frequency must be entered between 76 and 108 MHz, without a colon and with the kHz part specified. It must be in 10 kHz units and steps of 50 kHz. For example, 7605 (i.e. 76.05 MHz) is valid, but 7601 (i.e. 76.01 MHz) is not, as it is not in 50 kHz step. Some examples are given below:

97 MHz	enter 9700
97.5 MHz	enter 9750
106 MHz	enter as 10600
106.5 MHz	enter as 10650

The user is then prompted to enter the required TX power as dB μ V from 88 to 115. Transmission starts after the frequency and the power are set. The program then receives the required operating frequency and displays it as a confirmation.

The transmission is terminated by entering **Cntrl+C**. Please note that stopping the program by other means does not terminate the transmission.

```

#-----
#                               RADIO STATION
#                               =====
#
# This program uses the RadioStation Click board to design a
# radio station system (note: it is illegal to transmit over the
# FM radio broadcast frequencies)
#
# Author: Dogan Ibrahim
# File  : station.py
# Date  : July, 2020
#-----

import RPi.GPIO as GPIO          # Import RPi
import time
import smbus
bus = smbus.SMBus(1)
address = 0x11                  # Address

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

RST = 26                         # RST pin

```

```
GPIO.setup(RST, GPIO.OUT)                                # RST is output

#
# Chip parameters used in the program
#
POWER_UP = 0x01
SET_PROPERTY = 0x12
TX_TUNE_FREQ = 0x30
TX_TUNE_POWER = 0x31
TX_TUNE_STATUS = 0x33
TX_PREEMPHASIS = 0x2106

#
# Reset the chip before starting to use it
#
def Reset():
    GPIO.output(RST, GPIO.HIGH)                         # RST=1
    time.sleep(0.5)                                     # Wait 0.5s
    GPIO.output(RST, GPIO.LOW)                           # RST=0
    time.sleep(0.5)                                     # Wait 0.5s
    GPIO.output(RST, GPIO.HIGH)                           # RST=1

#
# This function sends a command to the chip. The command is the
# first byte, the arguments are the remaining bytes
#
def SendCommand(BytesInput, ByteResponse):
    command = BytesInput[0]                            # Cmd is 1st byte
    args = BytesInput[1:len(BytesInput)]              # Arguments
    bus.write_i2c_block_data(address, command, args)  # Send to chip
    time.sleep(0.2)                                    # Wait 0.2s
    res = bus.read_i2c_block_data(address, 0, 1)      # Get response

    if res[0] != ByteResponse:                         # Error?
        print("Error in command response:")           # Display error
        print("Response code = %d" %hex(res[0]))       # Display code
        exit(0)                                         # Exit

#
# This function sets the required Property. The upper and lower
# parts of the property and data are extracted and sent to the chip
#
def SetProperty(ReqProperty, Data):
    p1 = ReqProperty >> 8                            # Uper byte
    p2 = ReqProperty & 0xFF                          # Lower byte
    p3 = Data >> 8                                  # Upper byte
```

```
p4 = Data & 0xFF                      # Lower byte
SendCommand([SET_PROPERTY,0, p1, p2, p3, p4], 0x80)

#
# This function returns the TUNE status to the program. The upper
# and lower values of the operating frequency are returned to the
# main program. The CTS bit is set when it is safe to send the
# next command
#
def RcvTuneStatus():
    bus.write_i2c_block_data(address, TX_TUNE_STATUS, [0x01])
    res = bus.read_i2c_block_data(address, 0x0, 8)
    return(res[2], res[3])

#
# This function turns OFF the station and stops transmission
#
def PowerOff():
    GPIO.output(RST, GPIO.LOW)

#
# Start of the MAIN program
#
try:

    print("")
    print("RADIO STATION")                  # Display Heading
    print("=====")

    Reset()                                # Reset chip
    SendCommand([POWER_UP, 0x12, 0x50], 0x80) # Use crystal, set analog
    time.sleep(1)                            # Wait 1s
    SetProperty(TX_PREEMPHASIS, 1)           # Standard 50us
    freq = int(input("Enter frequency (7600-10800): "))
    TXPower = int(input("Enter TX power (88dBuV-115dBuV): "))
    print("Wait...")
    print("")

    SendCommand([TX_TUNE_FREQ, 0, freq >>8, freq & 0xFF], 0x80)
    time.sleep(5)
    fh,fl=RcvTuneStatus()
    time.sleep(5)
    SendCommand([TX_TUNE_POWER, 0, 0, TXPower, 0], 0x80)
    time.sleep(2)

    print("End of setup...")
```

```
fh, fl = RcvTuneStatus()
setfreq = fh*256 + fl
print("")
print("Transmitting on: %d" %(setfreq)) # Display freq
print("")
print("Enter Cntrl+C to STOP transmission")

while True:
    pass

except KeyboardInterrupt:                      # Cnrl+C detected
    PowerOff()                               # Stop transmission
    GPIO0.cleanup()                          # Cleanup RPi
    print("")
    print("END OF TRANSMISSION")
```

Figure 21.6: Program: station.py.

Figure 21.7 shows an example run of the program where the operating frequency and the TX power were selected as 106 MHZ and 95 dB_uV respectively.

```
pi@raspberrypi:~ $ python3 station.py
RADIO STATION
=====
Enter frequency (7600-10800): 10600
Enter TX power (86dBuV-115dBuV): 95
Wait...

End of setup...

Transmitting on: 10600

Enter Cntrl+C to STOP transmission
^C
END OF TRANSMISSION
pi@raspberrypi:~ $ █
```

Figure 21.7: Example run of the program.

The project was constructed on a breadboard and connections were made to the Raspberry Pi using a T-connector.

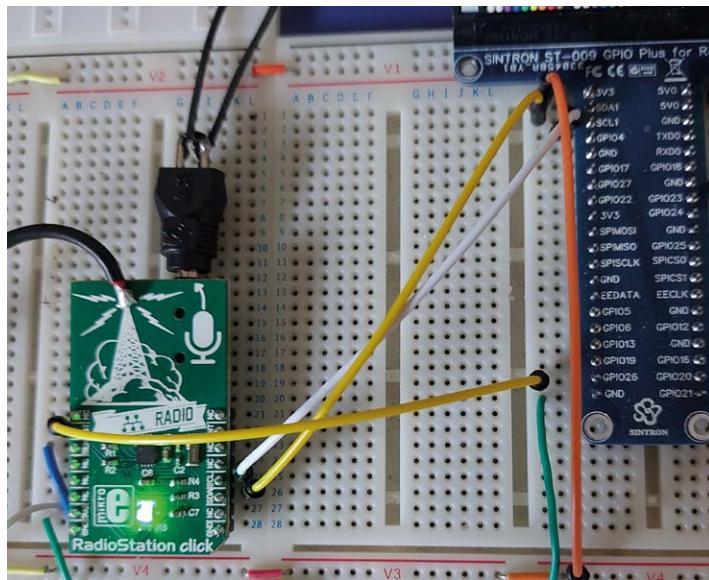


Figure 21.8: Project constructed on a breadboard.

You may want to experiment to include the RDS/RBDS in the program.

As with the previous project, transmitting over any broadcast band and/or setting up a pirate radio station are illegal unless you have a valid license. This project is provided for educational purposes only and the author or the publishers are not responsible for the illegal use of this project or part of it.

CHAPTER 22 • RF Power Meter

22.1 Project: RF Power Meter

This project is a programmable RF power level meter, using the RF Meter Click board (www.mikroe.com), that can be used to measure RF power in the range 1 MHz to 8 GHz over a 60 dB range (approximately). The input power range is –60 dBm to about 0 dBm. It is therefore possible to measure power in the milliwatts (mW) range using this meter. It will be necessary to use RF power attenuator modules to measure higher powers.

An antenna socket is provided to connect the transmitter, whose output power is to be measured. The signal is processed by the AD8318 logarithmic detector chip from Analog Devices. The resulting voltage is fed into an MCP3201 ADC which displays the power in dBm. In this project the power is displayed on an LCD.

The RF Meter Click (Figure 22.1) operates with three signals: CS, Data, and Clock.

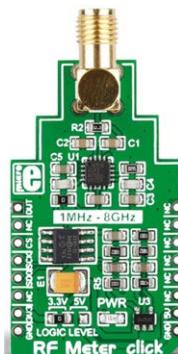


Figure 22.1: The RF Meter Click board from mikroe.com

Figure 22.2 shows the circuit diagram of the RF Meter Click. The antenna connector is located at the top left of the figure, next to it is the AD8313 logarithmic detector chip, and the MCP3201 ADC. The following pins are used on the mikroBUS connector shown at the top right hand side of the figure (NC=No Connection):

RF Meter pin	Function
1	Analogue output
2	NC
3	CS
4	Clock
5	Data
6	NC
7	+3.3V
8	GND
9	GND
10	+5V
11	NC

12	NC
13	NC
14	NC
15	NC
16	NC

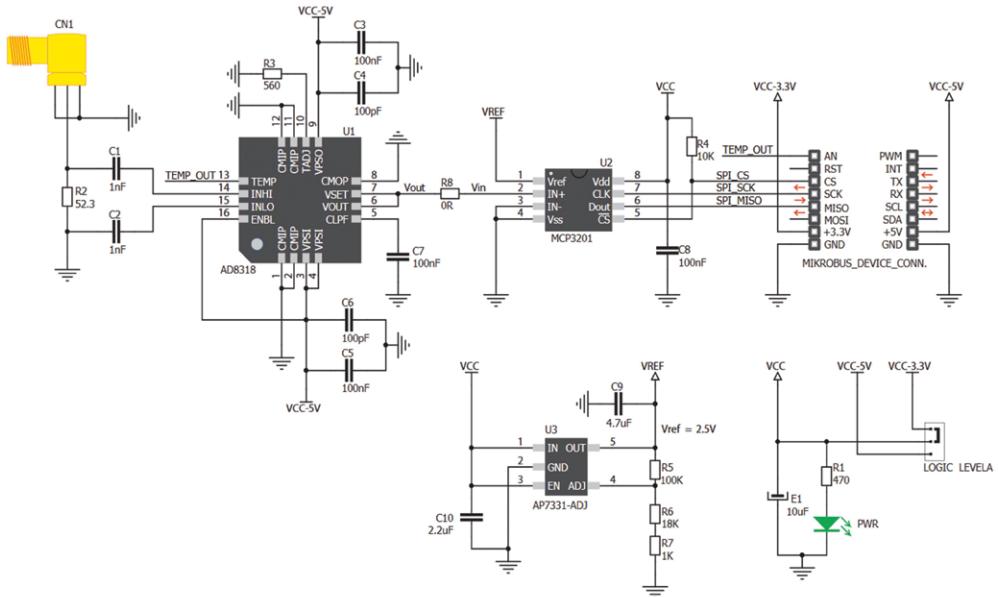


Figure 22.2 RF Meter Click circuit diagram.

Block Diagram: Figure 22.3 shows the block diagram of the project.

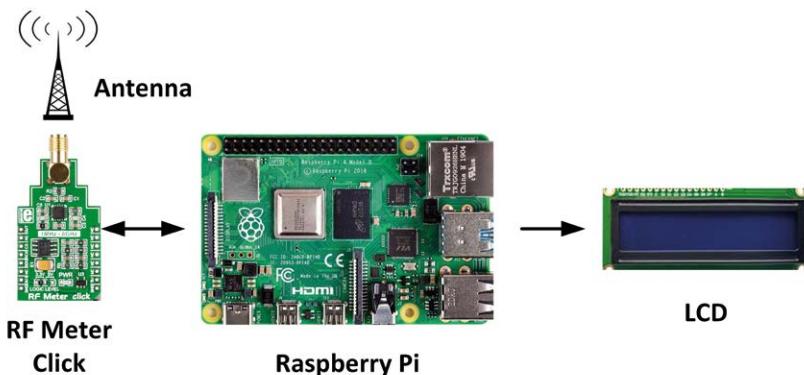


Figure 22.3: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 22.4. The Clock and Data pins of the RF Meter are connected to GPIO11 and GPIO9 of the Raspberry Pi. The CS pin is connected to GPIO19. The I2C LCD is connected as in the previous LCD based

projects where its SDA and SCL pins are connected to Raspberry Pi SDA (GPIO2) and SCL (GPIO3) respectively.

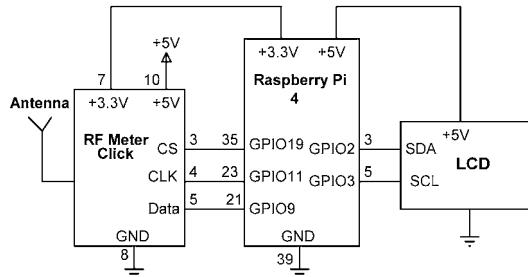


Figure 22.4: Circuit diagram of the project.

Before going into the details of the program, it is worthwhile to look at the operation of the AD8318 and MCP3201 chips briefly.

AD8318

The AD8318 is a logarithmic demodulating amplifier that can convert an RF input signal to a corresponding decibel-watt scaled output voltage. The chip provides accurate output for signals of 1 MHz to 6 GHz, and can be used for operation to 8 GHz. The logarithmic slope of the chip is nominally -25 mV/dB , but can be adjusted if required. The basic features of the AD8318 are:

- 1 MHz to 8 GHz operation
- 5 V operation
- 60 dB dynamic range (-60 dB to 0 dB)
- Low noise output
- Integrated temperature sensor
- $\pm 1.0 \text{ dB}$ accuracy over 55 dB range (for $f < 5.8 \text{ GHz}$)

Figure 22.5 shows the logarithmic response of the AD8318 (for further information, refer to the Data Sheet at:

<https://www.analog.com/media/en/technical-documentation/data-sheets/AD8318.pdf>).

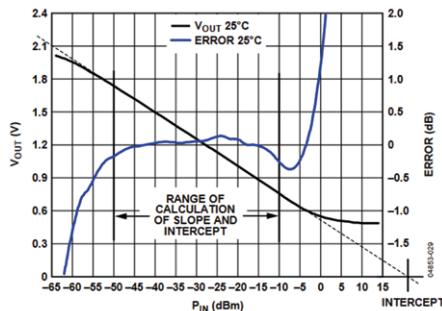


Figure 22.5: Logarithmic response of the AD8318 (Analog Devices).

It can be seen from Figure 22.5 that the slope of the curve is 25 mV/dB and the intercept of the P_{in} axis is at 20 dB. Therefore, the input-output relationship is given by (to 50 Ω):

$$P_{in} = 20 - 40 V_o$$

where,

P_{in} is the input power in dBm, and V_o is the output voltage in volts. We will be using the above equation to calculate the RF power from a measurement of the voltage at the output of AD8318. The voltage will be measured using the on-board MCP3201 ADC.

Because the slope and the intercept vary slightly from device to device, calibration of the chip is recommended for high accuracy. The chip can be calibrated by performing two unknown signal levels to the chip's input and measuring the corresponding output voltages. The slope and the intercept are then found by:

$$\text{slope} = (V_{out1} - V_{out2}) / (P_{in1} - P_{in2})$$

and

$$\text{intercept} = P_{in1} - V_{out1} / \text{slope}$$

Once the slope and the intercept are found, the relationship between the input and output can be written as:

$$P_{in} = \text{intercept} + V_{out} / \text{slope}$$

An external 52.3 Ω shunt resistor is connected at the input circuitry to provide adequate broadband 50 Ω matching, and therefore it is not necessary to reactively match the input circuitry.

MCP3201

This is a 12-bit ADC having the following basic features:

- 2.7 to 5.5 V operating voltage
- SPI interface to host
- on-chip sample and hold
- 100 ksps sampling rate (at 5 V)
- 50 ksps sampling rate (at 2.7 V)
- 400 μA active current (at 5 V)

The reference voltage of the ADC is set to +2.5 V on the RF Meter Click board. The ADC is connected to the output of the AD8318 and reads and converts the analogue signal into digital. The output voltage is read by the Raspberry Pi, converted into dBm (decibel-milliwatts) and displayed on the LCD.

Program Listing: Figure 22.6 shows the program listing (program: **rfmeter.py**). At the beginning of the program the libraries RPi, time, RPLCD, and spidev are imported to the program. The connections between the Raspberry Pi and the RF Meter Click board are defined. The clock and CS are configured as outputs, while data is configured as an input. The main program loop calls function **GetADCData** to read the analogue output voltage of the RF Meter Click board. Since the reference voltage of the ADC is 2.5 V and it is a 12-bit converter, the digital reading of the ADC is multiplied by 2.5 / 4096 to convert it to volts. The power in dBm is then calculated and displayed on the LCD every second.

```
#-----
#          RF METER
# =====
#
# This is a RF meter program which displays te RF power on an LCD
#
# Author: Dogan Ibrahim
# File  : rfmeter.py
# Date  : July, 2020
#-----
import time                      # Import time
import spidev                     # Import SPI
import RPi.GPIO as GPIO           # Import RPLCD
from RPLCD.i2c import CharLCD      # Import RPLCD
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

LCD = CharLCD('PCF8574', 0x27)

CS = 19                           # CS pin
DOUT = 9                            # Data pin
CLK = 11                           # CLK pin

GPIO.setup(CS, GPIO.OUT)            # CS is output
GPIO.setup(DOUT, GPIO.IN)           # Data is input
GPIO.setup(CLK, GPIO.OUT)           # CLK is output

#
# This function returns the ADC data read from the MCP3201
#
def GetADCData():
    Data = 0
    GPIO.output(CS, GPIO.HIGH)        # CS=1
    GPIO.output(CLK, GPIO.HIGH)        # CLK=1
    GPIO.output(CS, GPIO.LOW)          # CS=0

    m = 14
```

```

while m >= 0:
    GPIO.output(CLK, GPIO.LOW)
    Dat = GPIO.input(DOUT)
    GPIO.output(CLK, GPIO.HIGH)
    Dat = Dat << m
    Data |= Dat
    m = m - 1
    GPIO.output(CS, GPIO.HIGH)
    Data &= 0xFFFF
    return Data

LCD.cursor_pos = (0, 0)                      # Cursor at (0,0)
LCD.write_string("RF Power (dBm):")          # Heading

try:

#
# Read analog data, calculate the power and display on the LCD
#
    while True:
        adc = GetADCData()                  # Read ADC
        V = adc * 2.5 / 4096.0              # To Volts
        power = 20.0 - 40.0 * V             # Power
        powerstr = str(power)[:7]           # To string
        LCD.cursor_pos = (1, 0)              # Cursor at (1,0)
        LCD.write_string("      ")          # Clear
        LCD.cursor_pos = (1, 0)              # Cursor at (1,0)
        LCD.write_string(powerstr)          # Display power
        time.sleep(1)                      # wait 1 second

except KeyboardInterrupt:
    exit(0)

```

Figure 22.6: Program: rfmet.py.

Figure 22.7 shows an example output on the LCD. The project was constructed on a breadboard and is shown in Figure 22.8.

*Figure 22.7: Example output.*

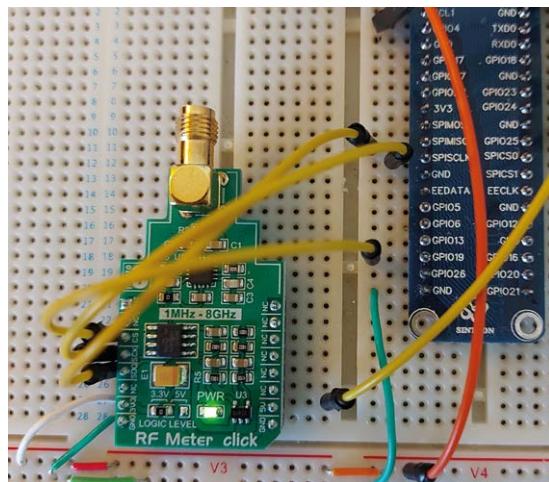


Figure 22.8: Project constructed on a breadboard.

22.2 RF attenuator

The RF meter project described in this project can measure from -65 dBm to up to a few dBm . It will therefore most definitely be required to use RF attenuator modules at the input of the meter in order to increase the useful range. RF attenuators can be home made, or preferably purchased from RF equipment sellers or from the Amazon. Figure 22.9 shows some typical RF 30-dB and 40-dB attenuator modules.



Figure 22.9: RF attenuator modules.

22.3 dB, dBm, and watt?

dB

Most readers may be confused with the terms dB, dBm, and watts. In this section we will attempt to clarify these terms.

dB (decibel) is a logarithmic scale, used to measure the ratio of two physical quantities. For example, it is used to express the voltage gain or the power gain of a transistor amplifier. The voltage gain is the ratio of the output voltage to the input voltage. Similarly, the power gain is the ratio of the output power to the input power.

When expressing the voltage gain, we use the formula:

$$\text{Voltage gain} = 20 \log_{10} (V_o / V_i)$$

Where V_o and V_i are the output and input voltages of the amplifier, respectively. For example, if $V_o / V_i = 100$, this is equal to 40 dB since $\log_{10} (100) = 2$. Similarly, if the voltage gain is expressed as 30 dB, then the ratio of $V_o / V_i = 31.6$ since:

$$\begin{aligned} 30 &= 20 \log_{10} (x) \\ 30 / 20 &= \log_{10} (x) \\ 1.5 &= \log_{10} (x) \\ \text{or, } x &= 10^{1.5} = 31.6 \end{aligned}$$

When expressing the power gain, we use the formula:

$$\mathbf{Power\ gain = 10 \ log_{10} (P_o / P_i)}$$

Where P_o and P_i are the output and input powers of the amplifier, respectively. For example, if $P_o / P_i = 80$, this is equal to 19 dB since $\log_{10} (80) = 1.90$.

dBm

The unit 'dBm' is used to express the power level (dB) with reference to one milliwatt (mW) into a defined impedance, usually 50Ω . It is used frequently in RF field as a measure of absolute power since it is referenced to the watt (W). The dB is a dimensionless unit since it is the ratio of two quantities. Although dBm is also dimensionless, it is an absolute measurement since it compares to a fixed reference.

In audio work dBm is typically referenced to a 600Ω load, while in RF work it is referenced to 50Ω load. In TV work dBm is referenced to 75Ω load.

The relationship between dBm and power in mW is given by:

$$\mathbf{dBm = 10 \ log_{10} (P / 1\ mW)}$$

where P is in milliwatts. For example, if $P = 1000$ mW, its equivalent dBm value is found as:

$$\text{dBm} = 10 \log_{10} (1000) = 30$$

Given the dBm, we can calculate the power as:

$$\mathbf{P = 10^{x/10}}$$

Where x is dBm and P is in milliwatts (mW).

Figure 22.10 shows a voltage generator and a load. The relationship between the voltage at the load and its dBm value depends on the value of the load as follows (assuming sinewave input). For the notations used here, accept that $W = P$, $V = U$.

For a 50Ω load (what we're interested in):

$$W = V^2 / R \quad (W \text{ in watts, } V \text{ in volts, } R \text{ in ohms})$$

or $V^2 = R \times W$

Taking W in milliwatts, we can write

$$V^2 = R \times 10^{x/10} \times 10^{-3}$$

or

$$V = \sqrt{50 \times 10^{\left(\frac{x-30}{10}\right)}}$$

where, x is in dBm, and V in volts (rms).

As an example,

10 dBm into a 50Ω load requires a voltage of:

$$V = \sqrt{50 \times 10^{\left(\frac{10-30}{10}\right)}} = 0.707 \text{ V}_{\text{rms}}$$

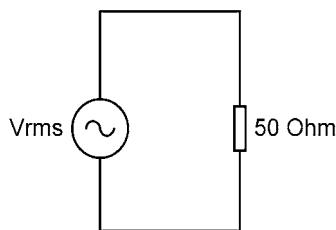


Figure 22.10: Voltage into a load.

Table 22.1 gives the dBm, its equivalent watts (or mW), and voltage (rms; root-mean-square) for some commonly used values.

dBm	W	Vrms
+60	1000	223.6
+57	501.2	158.3
+53	199.5	99.88
+50	100	70.71
+47	50.12	50.06
+44	25.12	35.44
+43	19.95	31.59
+40	10	22.36

+37	5.01	15.83
+33	2	10
+30	1	7.07
+25	316 mW	3.97
+20	100 mW	2.23
+15	31.6 mW	1.25
+10	10 mW	707.1 mV
+5	3.16 mW	397.6 mV
0	1 mW	223.6 mV
-10	0.1 mW	70.71 mV
-20	0.01 mW	22.36 mV
-30	0.001 mW	7.07 mV
-40	1 μ W	2.23 mV
-50	0.01 μ W	707 μ V
-60	0.001 μ W	223 μ V

Table 22.1: Overview of dBm, watt, and V_{rms} .

CHAPTER 23 • Raspberry Pi – Smartphone Projects

In this chapter we will be developing a project using the Raspberry Pi together with an Android smartphone. In the project we will control four relays effectively connected to the Raspberry Pi from the smartphone.

The project in this Chapter is developed using the **MIT App Inventor**. MIT App Inventor is a visual graphics-based tool for developing application programs (apps) on the Android smartphones. Perhaps it is worthwhile to introduce the MIT App Inventor for those readers who may not be familiar with it (interested readers can refer to author's book entitled *MIT APP Inventor Projects* published by Elektor).

23.1 The MIT App Inventor

MIT App Inventor is GUI-based and is similar to tools like Scratch and StarLogo, where developers drag and drop and join visual blocks to create an application. Many blocks are offered in the MIT App Inventor that enable users to create projects using components such as textboxes, labels, buttons, sliders, checkboxes, switches, notifiers, camcorders, cameras, text to speech components, speech recognizer, drawing and animation components, web tools, sensors, maps, storage components, Bluetooth connectivity, and so on. These components are organized under the heading **Palette** and are placed at the left-hand side of the MIT App Inventor startup screen as shown in Figure 23.1.

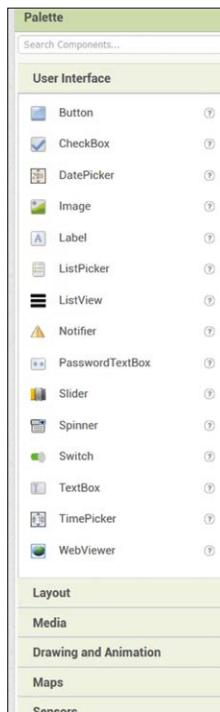


Figure 23.1: MIT App Inventor components.

When a new application is started, a mobile phone image is shown in the middle part of the screen. The development of a project is in two stages: **Designer** and **Blocks**. A project starts in the Designer stage where the user places the required components onto the mobile phone image to build the view of the final application. Some components are hidden and only shown outside at the bottom of the phone image. After designing the screen layout, the user clicks the Blocks menu where the second stage of the development starts. Here, the block program is constructed by clicking, dragging, dropping, and joining the required blocks on the mobile phone image.

When the design is complete it is required to test the project. Here, the user has the option of either using a built-in Emulator, to connect to the mobile phone using a USB cable, or to upload the developed application to the smartphone using wireless Wi-Fi link. Emulator option is useful if the user has no mobile phone at the time of the development, or if an Android compatible smartphone is not available. The second option is useful if there is no Wi-Fi connection where the developed application is uploaded to the smartphone via a USB cable. The third option is the preferred option where the developed visual program is uploaded to the smartphone using the Wi-Fi link. In this Chapter we will be using this third option to upload our program.

23.2 Setting up the MIT App Inventor

Although there are several ways of setting up the App Inventor, here we will be using an Android device with Wi-Fi connectivity. This is the recommended option where the software is developed on the PC and then uploaded (installed) to the Android device using the Wi-Fi link for testing (see Figure 23.2).

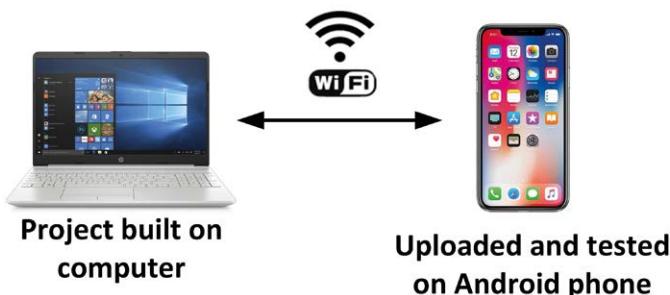


Figure 23.2: Using the Android device with Wi-Fi link.

This option requires the apps **MIT AI2 Companion** to be installed from the Play Store to your Android device as shown in Figure 23.3.

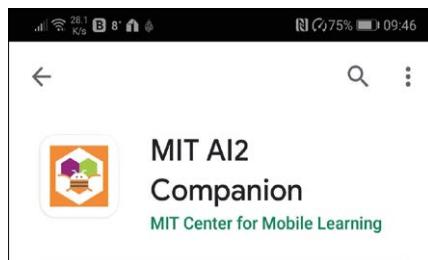


Figure 23.3: Install the MIT AI2 Companion to your Android device.

After you create your project, the next step is to upload (install) it to your Android device for testing. The steps to upload your project to the Android device are as follows (these steps will become clearer when we look at the steps to create our projects):

- After the project is complete, click **Connect** and then **AI Companion** as shown in Figure 23.4, or choose **Build** and then **App (provide QR code for .apk)** to install the project permanently on your Android device.

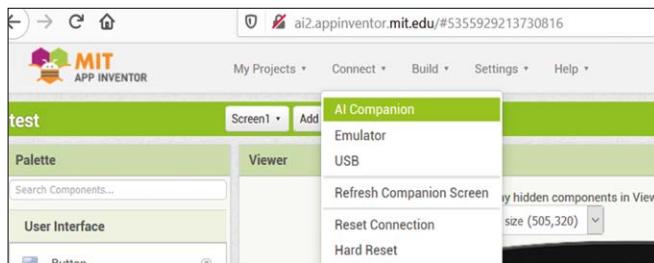


Figure 23.4: Click Connect and then AI Companion.

- Wait until the project is compiled.
- A dialogue with a QR (quick response) code will be displayed on your PC screen as shown in Figure 23.5.

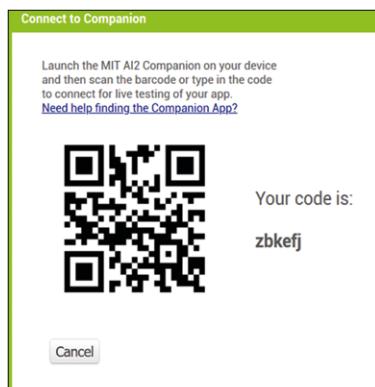


Figure 23.5: The QR code.

- Start the app **MIT AI2 Companion** on your Android device and click the **scan QR code** and hold your device to scan the displayed QR code.
- After a few seconds, the project will be uploaded to your device. Follow the instructions to install the project.
- You can now test your project on the Android device

Alternatively, you can enter the 6-character code displayed next to the QR code to your Android device to upload the project.

23.3 Project: Web Server to Control Multiple Relays

In this project a 4-channel relay module is connected to the Raspberry Pi. The relay channels are controlled individually by clicking buttons on the Android mobile phone. The relays can easily be used to control various electrical equipment in our station.

In this project the 4-channel relay board (see Figure 23.6) from Elegoo (www.elegoo.com) is used. This is an optocoupled relay board having four inputs, one for each channel. The relay inputs are at the bottom right-hand side of the board while the relay outputs are located at the top side of the board. The middle position of each relay is the common point, the connection to its left is the normally closed (NC) contact, while the connection to the right is the normally open (NO) contact. The relay contacts support 250 VAC at 10 A and 30 VDC, 10 A. IN1, IN2, IN3 and IN4 are the **active-LOW** inputs, i.e. a relay is activated when a logic LOW signal is applied to the input pin. Relay contacts are normally closed (NC). Activating the relay changes the active contacts such that the common pin and NC pin become the two relay contacts and at the same time the LED at the input circuit of the relay board, corresponding to the activated relay is turned ON. The V_{CC} can be connected to either +3.3 V or to +5 V. Jumper JD is used to select the voltage for the relay. **Because the current drawn by a relay can be in excess of 80 mA, you must remove this jumper and connect an external power supply (e.g. +5 V) to pin JD-VCC.**

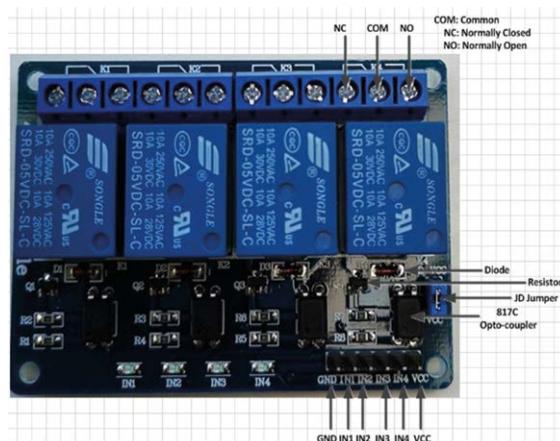


Figure 23.6: 4-channel relay board.

Block Diagram: Figure 23.7 shows the block diagram of the project.

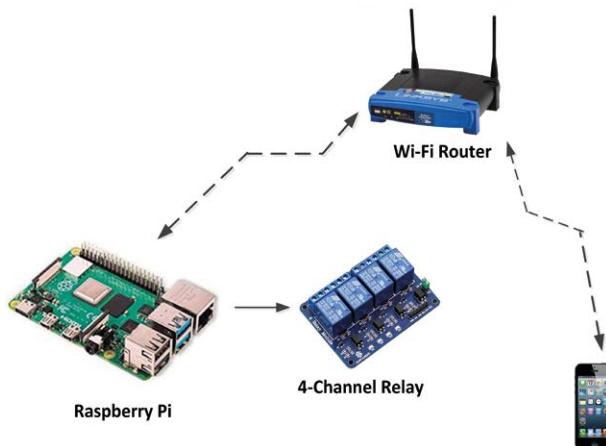


Figure 23.7: Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 23.8. IN1, IN2, IN3 and IN4 inputs of the relay board are connected to Raspberry Pi GPIO pins 2, 3, 4, and 17 respectively. Also, GND and +3.3 V pins of the development board are connected to GND and VCC pins of the relay board. You must make sure that jumper JD is removed from the board. Connect an external +5 V power supply to the JD-VCC pin of the relay board.

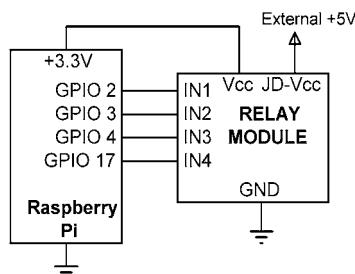


Figure 23.8: Circuit diagram of the project.

MIT App Inventor Application: This project uses the component Web to communicate with the Raspberry Pi.

The steps are (see Figure 23.9). Although parts of some of the Buttons seem to be missing in this figure, they are correctly displayed on the actual mobile phone):

- Create a new project and name it as **Relay**.
- Insert a **HorizontalArrangement** and insert a **Label** on it with its **Text** set to **RELAY CONTROLLER**.

- Insert a **TableArrangement** with four columns and four rows, **Height** set to 40 percent, and **Width** set to Fill parent.
- Insert eight **Buttons** on the **TableArrangement** with the following configuration:

Button Name	BackgroundColor	FontBold	FontSize	Text	TextColor
Relay1ON	Yellow	ticked	20	RELAY1 ON	Default
Relay1OFF	Yellow	ticked	20	RELAY1 OFF	Default
Relay2ON	Yellow	ticked	20	RELAY2 ON	Default
Relay2OFF	Yellow	ticked	20	RELAY2 OFF	Default
Relay3ON	Yellow	ticked	20	RELAY3 ON	Default
Relay3OFF	Yellow	ticked	20	RELAY3 OFF	Default
Relay4ON	Yellow	ticked	20	RELAY4 ON	Default
Relay4OFF	Yellow	ticked	20	RELAY4 OFF	Default

Click tab **Connectivity** and insert the hidden **Web** component on the Viewer.

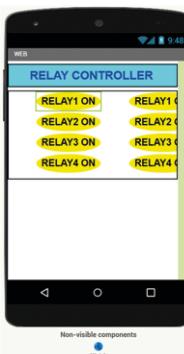


Figure 23.9: Design of the project.

Figure 23.10 shows the block program of the project. The relay channels are controlled by sending the following HTTP requests to the Raspberry Pi web server:

To turn ON channel 1	http://192.168.1.202/RELAY/on1
To turn OFF channel 1	http://192.168.1.202/RELAY/off1
To turn ON channel 2	http://192.168.1.202/RELAY/on2
To turn OFF channel 2	http://192.168.1.202/RELAY/off2
To turn ON channel 3	http://192.168.1.202/RELAY/on3
To turn OFF channel 3	http://192.168.1.202/RELAY/off3
To turn ON channel 4	http://192.168.1.202/RELAY/on4
To turn OFF channel 4	http://192.168.1.202/RELAY/off4

The steps are:

- Initialize a variable called **RaspberryPi** and enter the IP address of your Raspberry Pi.
- There are eight buttons, four to turn the relays ON, and four to turn the relays OFF. In the remainder of this project one of group of block will be described. As it can be seen from Figure 23.10, the groups of blocks are similar.
- Click **Relay1ON** and select **when Relay1ON.Click do**. This block will be activated when the user clicks button **RELAY1 ON**. We want to turn ON channel 1 of the relay, and at the same time to change the text colour of button **RELAY1 ON** to red, and the colour of **RELAY1 OFF** to black.
- Click **Web1** and select **set Web1.Url to** and insert a **Join** block. Insert he IP address block and a **Text** block with the text set to **/RELAY/on1**. The command that will be sent to Raspberry Pi is: <http://192.168.1.202/RELAY/on1>, where 192.168.1.202 is the IP address of author's Raspberry Pi.
- Set the **Text** colour of button **Relay1ON** to red to indicate that the channel 1 of the relay has been activated. At the same time set the colour of button **Relay1 OFF** to black to indicate that this button is not active (it may already be a black colour).
- Click **Web1** and select **call Web1.Get** to send a request to the web server to activate channel 1 of the relay module.
- Repeat the blocks for the other buttons similarly as shown in Figures 23.10a and 23.10b.

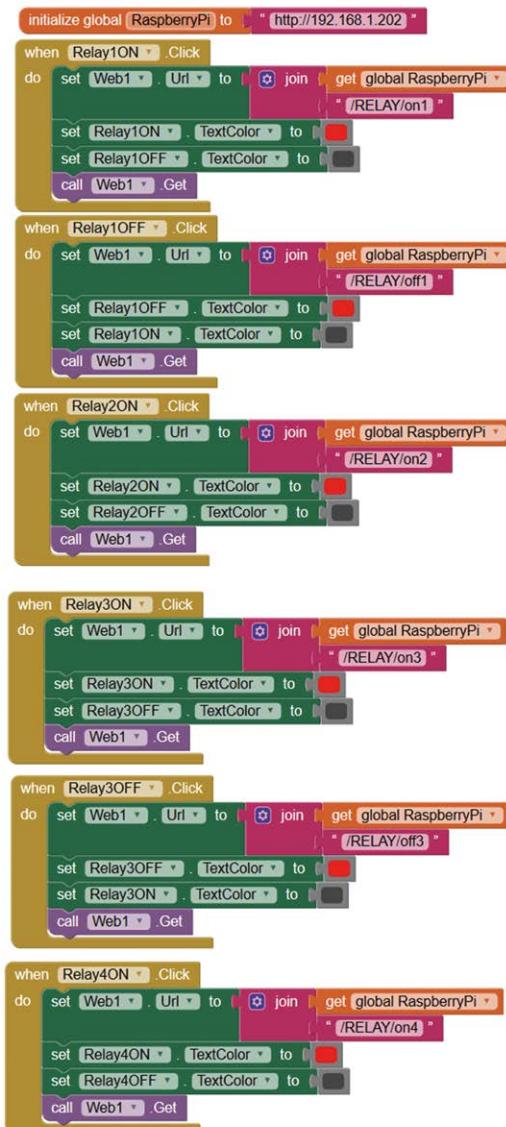


Figure 23.10a: Block program of the project.

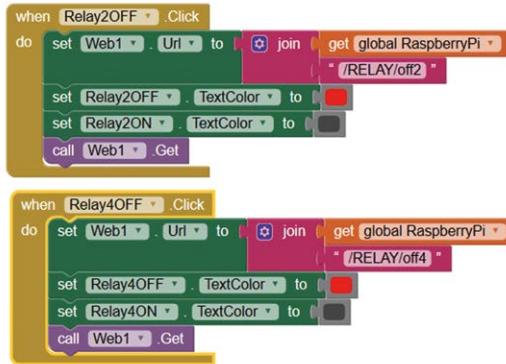


Figure 23.10b: ... continued ...

The QR code of the project is shown in Figure 23.11.



<http://ai2.appinventor.mit.edu/b/f6a3>

Figure 23.11: QR code of the project.

Python Program: The Python program (**relay4.py**) is shown in Figure 23.12. At the beginning of the program the libraries used in the program are imported, relay control pins are assigned to GPIO ports 2, 3, 4, and 17. Additionally, these port pins are configured as outputs and they are set to 1 so that all the relay channels are deactivated at the beginning of the program. Notice that a relay channel is activated if a logic 0 is sent to that channel, and is deactivated if a logic 1 is sent. The commands received from the Android mobile phone are then checked and the relays are controlled as follows:

RELAY1	on1	activate channel 1 of the relay
	off1	deactivate channel 1 of the relay
RELAY2	on2	activate channel 2 of the relay
	off2	deactivate channel 2 of the relay
RELAY3	on3	activate channel 3 of the relay
	off3	deactivate channel 3 of the relay
RELAY4	on4	activate channel 4 of the relay
	off4	deactivate channel 4 of the relay

A relay channel is activated by the statement: **GPIO.output(actuator, GPIO.LOW)**, where actuator is the selected relay. Similarly, a relay channel is deactivated by the statement: **GPIO.output(actuator, GPIO.HIGH)**

```
-----
#           WEB SERVER RELAY CONTROL
# =====
#
# In this project a 4 channel Relay module is connected to
# Raspberry Pi. The Relay channels are controlled using a
# web server
#
# File:    relay4.py
# Date:    August, 2020
# Author:  Dogan Ibrahim
-----

from flask import Flask,render_template
import RPi.GPIO as GPIO

app=Flask(__name__)

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
#
# Assign Relay ports
#
RELAY1 = 2                      # Relay IN1 at GPIO 2
RELAY2 = 3                      # Relay IN2 at GPIO 3
RELAY3 = 4                      # Relay IN3 at GPIO 3
RELAY4 = 17                     # RELay IN4 at GPIO 17

#
# Configure Relay ports as outputs
#
GPIO.setup(RELAY1, GPIO.OUT)       # Conf RELAY1 as output
GPIO.setup(RELAY2, GPIO.OUT)       # Conf RELAY2 as output
GPIO.setup(RELAY3, GPIO.OUT)       # Conf RELAY3 as output
GPIO.setup(RELAY4, GPIO.OUT)       # Conf RELAY4 as output

#
# De-activate all Relay channels at the beginning
#
GPIO.output(RELAY1, 1)            # Relay chan 1 OFF
GPIO.output(RELAY2, 1)            # Relay chan 2 OFF
GPIO.output(RELAY3, 1)            # Relay chan 3 OFF
GPIO.output(RELAY4, 1)            # Relay chan 4 OFF
```

```
@app.route("/<device>/<action>")
def action(device, action):
    actuator = RELAY1
    if action == "on1":
        GPIO.output(actuator, GPIO.LOW)                      # Relay 1 ON
    if action == "off1":
        GPIO.output(actuator, GPIO.HIGH)                     # Relay 1 OFF

    actuator = RELAY2
    if action == "on2":
        GPIO.output(actuator, GPIO.LOW)                      # Relay 2 ON
    if action == "off2":
        GPIO.output(actuator, GPIO.HIGH)                     # Relay 2 OFF

    actuator = RELAY3
    if action == "on3":
        GPIO.output(actuator, GPIO.LOW)                      # Relay 3 ON
    if action == "off3":
        GPIO.output(actuator, GPIO.HIGH)                     # Relay 3 OFF

    actuator = RELAY4
    if action == "on4":
        GPIO.output(actuator, GPIO.LOW)                      # Relay 4 ON
    if action == "off4":
        GPIO.output(actuator, GPIO.HIGH)                     # Relay 4 OFF
    return ""

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

Figure 23.12: Python program (relay4.py) of the project.

To test the project, start the Python program either from Thonny or from the command line as follows:

```
pi@raspberry:~ $ sudo python3 relay4.py
```

Then, start the application on your Android mobile phone. Click a button to activate the required relay channel. You should hear the relay clicking and at the same time the relay LED turns ON. Figure 23.13 shows an example run on the Android mobile phone. The circuit built on a breadboard is shown in Figure 23.14.



Figure 23.13: Example run on the Android mobile phone.

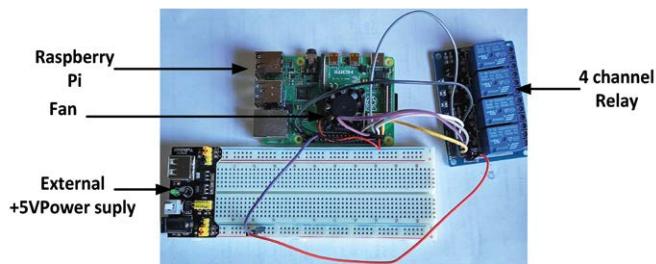


Figure 23.14: Circuit built on a breadboard.

CHAPTER 24 • RTL-SDR and Raspberry Pi

24.1 Overview

In this Chapter we will be looking at ways to use the Raspberry Pi 4 with RTL-SDR devices. DVB-T dongles in the market that are based on the RTL2832U chip have recently become popular as they can be used as cheap SDRs. There are many different types of RTL-SDR dongles in the market, but nearly all are based on the RTL2832U chip. Figure 24.1 shows such a dongle with the name DVB-T+DAB+FM (priced around £12 while writing this book). The author uses the Nooelec Nesdr Smart RTL-SDR dongle with an upconverter which work very nicely at all frequencies of interest.



Figure 24.1: DVB-T dongle.

Back in 2012 an undocumented feature of the RTL2832U chip was discovered which enabled it to be used as a general-purpose SDR. With the development of hardware and software, these cheap dongles can now be used as sophisticated SDR receivers. For example, an RTL-SDR dongle can be connected to a Raspberry Pi and many interesting software packages can be downloaded to the Raspberry Pi.

Without these dongles, such receivers would have costed hundreds or thousands of dollars having similar features. Some of the application areas of RTL-SDR dongles are:

- listening to air traffic control conversations
- tracking aircraft positions
- receiving meteorological transmissions
- listening to amateur radio bands
- listening to shortwave and FM radio
- listening to DAB broadcast transmissions
- receiving and decoding GPS signals
- decoding ham radio APRS packets
- developing a radio scanner
- watching analogue broadcast TV
- using it as a spectrum analyser
- using it as a receiver server
- listening to satellites

- scanning cordless phones and baby monitors
- etc., etc.

The minimum requirements to use an RTL-SDR dongle are:

- an RTL-SDR dongle device
- a suitable antenna (some are shipped with antennas)
- a powerful computer (e.g Raspberry Pi 4)
- RTL-SDR driver and application software (there are many and most are free)

Depending on your requirements and applications, you may also need to use filters to improve the signal-to-noise ratio (S/N).

RTL-SDR devices can work from about 24 MHz to over 1.7 GHz. The lower frequency range can be extended by using an **upconverter** device or by **direct sampling mode**. The upconverter is connected to the antenna before the RTL-SDR device. Some upconverters operate at as low as several kHz. If the upconverter oscillator frequency is 125 MHz and we want to tune to 5 MHz, then we have to tune our receiver (or e.g. GQRX) to 130 MHz. Direct sampling mode requires a small change to be made to the RTL-SDR hardware and the software, where a connection is made inside the hardware. Some RTL-SDR devices have a small hole on them so that a connection can be made by inserting a jumper wire, thus there is no need to open the device and make soldering. The software should be configured so that the sampling mode is set to Direct Sampling.

The RTL-SDR has 3.2 MHz bandwidth, 8-bit analogue-to-digital converters (ADC), less than 4.5 dB noise figure, and $75\ \Omega$ input impedance (not $50\ \Omega$ which is the impedance used in amateur radio). In general, the mismatch between the $75\ \Omega$ and $50\ \Omega$ is less than 0.2 dB). Because the RTL-SDR devices are cheap, they use 28.8 MHz crystal oscillators and their clock accuracy may drift several kHz. Most popular RTL-SDR software packages have options in the form of ppm drift values for calibrating this drift in software. There are also spike noises in the form of harmonics at multiples of 28.8 MHz. These spikes can usually be observed in waterfall displays.

For good reception and low noise, the RTL-SDR should be placed close to the antenna so that the lossy coaxial cable is replaced with non-lossy USB cable. You should take care however that the length of the USB cable is not more than 5 metres for USB2.0, and 3 metres for USB3.0. If longer length USB cables are required, then it is recommended to use USB hubs or USB repeater devices.

The RTL-SDR dongles should be placed in metal enclosures in order to minimize external interferences., and they should not be near power lines, motors, TVs, appliances, or other equipment that may generate electromagnetic noise.

Having a good antenna is also very important while using your RTL-SDR. The cheap antenna delivered with the device is about 12-15 cm long and not generally suitable for amateur work. You should place this antenna over a metal surface which has a radius of around 12-

15 cm to make a quarter-wave antenna and improve the reception characteristics.

In this Chapter we are only interested in using an RTL-SDR device with the Raspberry Pi. The author has used the latest Raspberry Pi 4 while writing this book. The theory and operation of the RTL-SDR devices are beyond the scope of this book. Interested readers can get tons of information on RTL-SDR from the Internet and from many books available on this topic.

24.2 Installing the RTL-SDR software on Raspberry Pi 4

When an SDR-RTL dongle is plugged in to one of the USB ports of the Raspberry Pi, it is recognized as a DVB-T device. This is no surprise and is expected because most people who purchase such dongles use them to watch TV.

The result of connecting an RTL-SDR dongle to the Raspberry Pi 4 are described by the following steps. ***Notice that the RTL-SDR drivers described in this section must be loaded for most of the RTL-SDR based programs to work.***

- Connect the dongle to one of the USB ports of your Raspberry Pi 4.
- Enter the following command to list the recognized USB devices. Figure 24.2 shows the results before and after plugging in the dongle. Notice that the dongle is recognized as: **RealTek Semiconductor Corp. RTL2838 DVB-T**

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

```
pi@raspberrypi:~ $ lsusb
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
pi@raspberrypi:~ $ lsusb
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 0bda:2838 Realtek Semiconductor Corp. RTL2838 DVB-T
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
```

Figure 24.2: List of recognized devices.

- Although the dongle has been detected, it has been recognized as a DVB-T television dongle.
- Check to see what modules are loaded into the Kernel for the dongle. Enter (see Figure 24.3):

```
pi@raspberrypi:~ $ lsmod | egrep 'sdr|dvb'
```

```
pi@raspberrypi:~ $ lsmod | egrep 'sdr|dvb'
dvb_usb rtl28xxu      32768  0
dvb_usb_v2            24576  1 dvb_usb_rtl28xxu
dvb_core             110592  2 dvb_usb_v2,rtl2832
pi@raspberrypi:~ $ █
```

Figure 24.3: Kernel-loaded modules.

- As you can see from Figure 24.3, we have to disable module **dvb_usb_rtl28xxu** so that the dongle is not recognized as a TV device. The easiest option is to blacklist the dongle device so that it is not recognized as a TV dongle. Create the following file using the **nano** editor:

```
pi@raspberrypi:~ $ sudo nano /etc/modprobe.d/blacklist-dvb.conf
```

Enter the following line of code inside the new file. Enter **Cntrl+X** followed by **Y** to save the file:

```
blacklist dvb_usb_rtl28xxu
```

- Confirm that the correct code is entered into the file:

```
pi@raspberrypi: ~ $ cat /etc/modprobe.d/blacklist-dvb.conf
```

- Reboot the Raspberry Pi.
- Enter the following command to check the modules loaded for the sdr/dvb. There should be no output displayed:

```
pi@raspberrypi:~ $ lsmod | egrep 'sdr|dvb'
```

Installing from command line

We should now install the **rtl-sdr** driver software to our Raspberry Pi. Remove the dongle from your computer and enter the following commands:

```
pi@raspberrypi: ~ $ sudo apt-get update
pi@raspberrypi: ~ $ sudo apt-get install rtl-sdr
```

We should now test our installation. Plug in your dongle, reboot your Raspberry Pi, and enter the following command (see Figure 24.4):

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

```
pi@raspberrypi:~ $ rtl_test
Found 1 device(s):
  0: Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Fitipower FC0012 tuner
Supported gain values (5): -9.9 -4.0 7.1 17.9 19.2
Sampling at 2048000 S/s.

Info: This tool will continuously read from the device, and report if
samples get lost. If you observe no further output, everything is fine.

Reading samples in async mode...
```

Figure 24.4: Testing the RTL software.

Assuming that everything appears okay, you should enter **Cntrl+C** to exit from the test program. If you get an error while running the test program, it is recommended to update your system by entering the following commands (this may take very long time to complete):

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt full-upgrade
pi@raspberrypi:~ $ sudo apt clean
```

Installing from Desktop

The rtl-sdr software can be installed from the Desktop as follows:

- Choose Preferences → Add/Remove Software
- Enter rtl-sdr in the search box and press Enter
- Wait until a list of software are listed. Select **Software defined radio receiver for RealtekRTL2832U (tools)** as shown in Figure 24.5

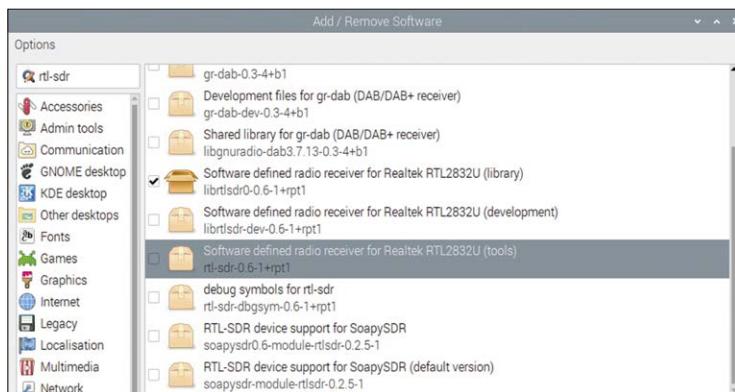


Figure 24.5: Select SDR software.

- Click to install the software.
- Reboot your Raspberry Pi.

- Plug in the dongle and run the **rtl_test** program.

We are now ready to install RTL-SDR applications software. There are large number of free programs that can be used with the RTL-SDR. Some of the popular ones compatible with the Raspberry Pi are:

- GQRX
- Linrad
- CubicSDR
- QtRAdio
- Sdrangelove
- Natpos
- OpenWebRX
- WebRadio
- SigDigger

Tuning to a frequency manually

We can use the RTL-SDR software to tune to a frequency by entering command at the command line. As an example, the following command tunes to frequency 97.3 MHz which happens to be the LBC radio station broadcasting on FM in London:

```
pi@raspberrypi:~ $ rtl_fm -M wbfm -f 97300000 | aplay -r 32000 -f S16_
LE -c 1
```

You should be able to hear the radio broadcast on the 97.3 MHz. Figure 24.6 shows what is displayed by Raspberry Pi when the above command is entered. Enter **Cntrl+C** to terminate the command.

```
pi@raspberrypi:~ $ rtl_fm -M wbfm -f 97300000 | aplay -r 32000 -f S16_LE -c 1
Found 1 device(s):
  0: Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Tuner gain set to automatic.
Tuned to 97571000 Hz.
Oversampling input by: 6x.
Oversampling output by: 1x.
Buffer size: 8.03ms
Exact sample rate is: 1020000.026345 Hz
Allocating 15 zero-copy buffers
Sampling at 1020000 S/s.
Output at 170000 Hz.
Playing raw data 'stdin' : Signed 16 bit Little Endian, Rate 32000 Hz, Mono
underrun!!! (at least 126.369 ms long)
```

Figure 24.6: Tuning to a frequency manually.

In the remaining sections of this Chapter we will be looking at some of the popular ones and learn how we can install and use them.

24.3 The GQRX

The GQRX is an excellent software package created by Alexander Csete (OZ9AEC), and it uses GNU Radio libraries. A large number of RTL-SDR dongles are supported by GQRX.

The installation instruction (see link: <http://www.kk5jy.net/gqrx-build/>) given below install the GQRX from its sources. As an aside, the author had the problems of not getting sound output when the software was installed from the Raspbian repository with the **Add/Remove Software** menu option.

```
pi@raspberrypi:~ $ sudo apt update
pi@raspberrypi:~ $ sudo apt install gnuradio
pi@raspberrypi:~ $ sudo apt install gnuradio-dev
pi@raspberrypi:~ $ sudo apt install gr-osmosdr
pi@raspberrypi:~ $ sudo apt install libgnuradio-osmosdr0.1.4
pi@raspberrypi:~ $ sudo apt install libosmosdr-dev
pi@raspberrypi:~ $ sudo apt install libqt5svg5-dev
pi@raspberrypi:~ $ sudo apt install librtlsdr-dev
pi@raspberrypi:~ $ sudo apt install osmo-sdr
pi@raspberrypi:~ $ sudo apt install portaudio19-dev
pi@raspberrypi:~ $ sudo apt install qt5-default
```

Create a directory called **GQRX**:

```
pi@raspberrypi:~ $ mkdir GQRX
```

Copy the following file from web site: <http://gqrx.dk/download> (click **Gqrx source code – SourceForge** as shown in Figure 24.7) to directory GQRX

gqrx-sdr-2.11.5-src.tar.xz

Download Gqrx SDR

Gqrx is distributed as source code as well as binary packages. The supported platform extent Mac OS X.

A recent version of gqrx is probably already available through the official software ch Linux distributions and it is recommended to investigate that first. If those packages hardware support you can try some of these alternate options:

- [Gqrx source code](#) ([SourceForge](#))
- [Generic Gqrx binary package for 64 bit linux](#) ([SourceForge](#))

Figure 24.7: Get the source code.

Enter the following commands (the **make** command will take a long time to complete):

```
pi@raspberrypi:~ $ cd GQRX
pi@raspberrypi:~ /GQRX $ tar -Jxf gqrx-sdr-2.11.5-src.tar.xz
pi@raspberrypi:~ /GQRX $ mkdir build-gqrx
pi@raspberrypi:~ /GQRX $ cd build-gqrx
pi@raspberrypi:~/GQRX/build-gqrx $ cmake -DLINUX_AUDIO_BACK-
```

END=Portaudio ..//gqrx-sdr-

```
pi@raspberrypi:~/GQRX/build-gqrx $ make  
pi@raspberrypi:~/GQRX/build-gqrx $ sudo make install
```

This completes the installation of GQRX. Enter the following command to optimize the Raspberry Pi for DSP work:

```
pi@raspberrypi:~/GQRX/build-gqrx $ cd ~  
pi@raspberrypi:~ $ sudo volk_profile
```

Reboot the Raspberry Pi:

```
pi@raspberrypi:~ $ sudo reboot
```

Using the GQRX

Before using the GQRX make sure that your RTL-SDR dongle and the USB sound card are connected to the USB ports of your Raspberry Pi. This is important since the dongle and the sound card are checked at the beginning of the program and the program may not recognize them if they are connected after the program starts.

The steps are:

- Start your Desktop.
- Start a terminal session and type:

```
pi@raspberrypi:~ $ sudo gqrx
```

Figure 24.8 shows the GQRX main screen tuned to 97.3 MHz. You will see the following menu items at the upper part of the screen:

- File
- Tools
- View
- Help

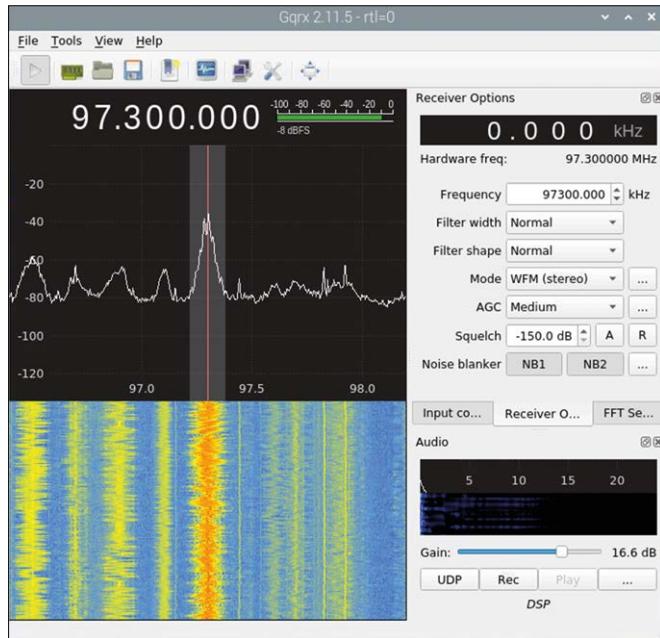


Figure 24.8: GQRX main screen.

By selecting the File menu, you can configure the I/O devices, load saved settings, save the current settings, save the waterfall, or quit the program. The I/O devices menu (Figure 24.9) is one of the important menus as it allows you to select items such as the type of device you are using, input rate, audio device, audio sample rate. In this example, the **Realtek RTL2838UHIDII** is selected as the device. The **Input rate** is set to 1800000, but you can experiment different values. The **Audio output** is set to **USB Audio Device** (the author had the UGREEN audio adapter connected to one of the USB ports of the Raspberry Pi).

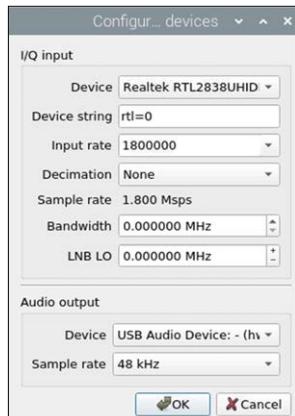


Figure 24.9: The I/O device menu.

Near the bottom right of the screen you will see three tabs with the names: **Input Controls**, **Receiver Options**, and **FFT Settings**.

At the top of the **Input Controls** (Figure 24.10) is the **LNB LO** when using local oscillator (LO) frequency in front of your SDR dongle and it is set to 0 in most cases.

The **Hardware AGC** should be ticked as it enables automatic gain control (if supported by your hardware).

Swap I/Q is used to swap I and Q channels, and it should normally not be ticked.

No limits should not be ticked as it may enable to use the SDR beyond its normal frequencies (you may have to enable it to receive HF frequencies).

DC remove removes the DC bias and it should be ticked.

IQ balance should not be ticked unless there seems to be ghost images in the spectrum.

Freq Correction is used to correct for the drift in the SDR internal oscillator. This is adjusted by tuning the dongle to a very well-known stable frequency (e.g. using a frequency generator) and then adjusting this correction until the displayed frequency matches the frequency of the generator. In most cases you can leave this set to 0.0. After making a correction you should save the settings.

Antenna should be kept at **Rx** as there is usually no other options with most dongles.

You should tick the **Reset frequency controller digits**

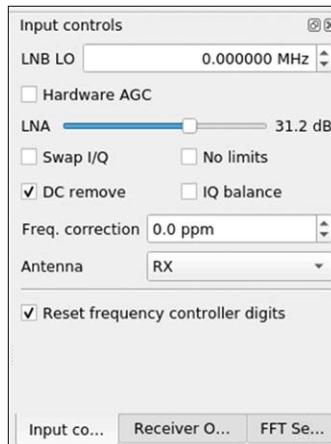


Figure 24.10: Input Controls.

At the top of the **Receiver Options** (Figure 24.11) you can offset the selected frequency from the main frequency entered in the left window. This is normally kept at 0.000.

Filter width and **Filter shape** can be selected as required. They are usually set to **Normal**.

Mode is used to select the require modulation mode and it can be:

Demod Off turns off signal processing (the real time spectrum can still be viewed)

Raw I/Q is selected to pass raw I/Q data without any demodulation

Narrow FM used to select narrow FM

WFM (mono or stereo) is used to select broadcasting stations in the FM band

USB or **LSB** are the upper and lower side bands, selected for side band amplitude modulation

CW-L or **CW-U** are selected for Morse code modulation

Select the **AGC** as required

Squelch value can be set manually, or you can wait until there is silence in the band and then click on the button **A** to its right to automatically adjust the squelch level to the current signal or noise level. Clicking **R** resets the squelch level to its default value.

Noise blanker **NB1** and **NB2** attenuate noise.

The **Audio** section at the bottom of the **Receiver Options** controls the audio signal level by using the slider. Set the audio level to several dB. The audio signal can be recorded by clicking the **Recording** tab. The **UDP** tab allows you to stream the raw audio over UDP connection.

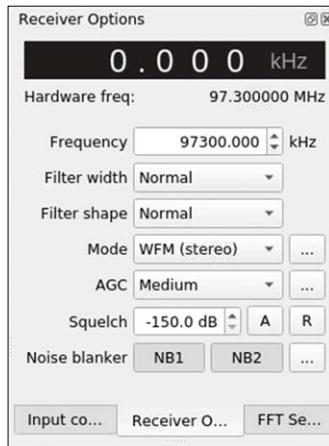


Figure 24.11: Receiver Options.

The **FFT Settings** window is shown in Figure 24.12. **FFT Size** is the number of points used in the FFT calculation, and it determines the resolution of the plot. For example, if the bandwidth is 200 kHz and FFT size is 8192, then the FFT resolution is $200000 / 8192 = 24$ Hz / FFT which is related to the screen pixels.

FFT rate is the frames per second and it can be selected to up to 60 fps.

Window is the type of windowing used in the DSP and by default it is set to Hann.

FFT Averaging is the averaging gain used to reduce the noise level. The averaging affects the FFT plot and not the **waterfall**.

The **Pandapter** slider modifies the display by changing the amount of space allocated to the spectrum and the **waterfall**.

The **Peak Detect** button will show little circles at the peaks of the spectrum display.

The **Peak Hold** button displays the spectrum faintly.

The **Pan dB** slider changes the **pandapter** gain in dB.

The **Wf.dB** sets the **waterfall** gain in dB.

Freq zoom sets the frequency axis scale.

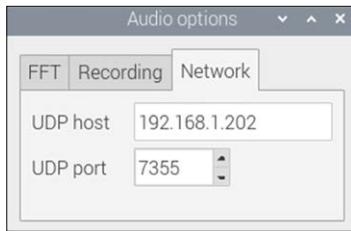


Figure 24.12: FFT Settings frame.

Streaming audio to the PC

GQRX has the option of streaming audio data over the network using the UDP protocol. This is useful, for example, if you are having problems outputting audio directly to your speakers with the GQRX. Before streaming the data, we have to configure the GQRX network settings. Open the network window by clicking the audio settings in the Audio window. Click the network tab as shown in Figure 24.13, where 192.168.1.202 is the IP address of the Raspberry Pi (use command **ifconfig** to find the IP address it if you do know what it is). Click the UDP tab under the Audio tab to start outputting the audio from the GQRX.

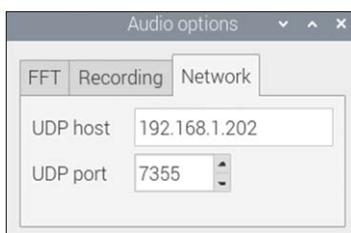


Figure 24.13: Open the network tab.

Then enter the following command (see Figure 24.14) at the command line to stream the audio on your Raspberry Pi speakers. Enter **Cntrl+C** to terminate the streaming:

```
pi@raspberrypi:~ $ nc -l -u 7355 | aplay -r 48000 -f S16_LE -t raw -c 1
```

```
pi@raspberrypi:~ $ nc -l -u 7355 | aplay -r 48000 -f S16_LE -t raw -c 1
Playing raw data 'stdin' : Signed 16 bit Little Endian, Rate 48000 Hz, Mono
underrun!!! (at least 119.522 ms long)
```

Figure 24.14: Start the audio streaming.

You could get a better audio quality by sending the output to the **VLC** media play program as shown below (Figure 24.15). Enter **Cntrl+C** to terminate the streaming:

```
pi@raspberrypi:~ $ vlc --demux=rawaud --rawaud-channels=1
--rawaud-samplerate=48000 udp://:@7355
```

```
pi@raspberrypi:~ $ vlc --demux=rawaud --rawaud-channels=1 --rawaud-samplerate=48
000 udp://:@7355
VLC media player 3.0.11 Vetinari (revision 3.0.11-0-gdc0c5ced72)
```

Figure 24.15: Sending the output to VLC.

24.4 The CubicSDR

The **CubicSDR** is probably one of the best SDR software for the beginners, and yet it includes almost everything that a user may want. **CubicSDR** supports most RTL-SDRs.

The **CubicSDR** runs nicely on the Raspberry Pi 4. The installation steps are as follows:

- Start the Desktop on your Raspberry Pi.
- Click **Preferences -> Add/Remove Software**.
- Enter **CubicSDR** in the search box and press **Enter**.
- Click to select the software and click **Apply**.
- Enter your Raspberry Pi password and wait until the installation is finished.
- To run the software, start a terminal session by clicking the Terminal menu in Desktop.
- Enter **CubicSDR** and press Enter. You should see the **CubicSDR** software starting.
- Select the type of device you are using and click Start.
- Select the mode e.g. FM, enter the required frequency e.g. 93.7 MHz (this is the LBC radio broadcasting frequency in London).
- **You should click your mouse anywhere on the waterfall to start the Modem and select your audio device.**

Figure 24.16 shows the **CubicSDR** running at 93.7 MHz FM. Part of the screen is shown in the Figure. Further details on the **CubicSDR** can be obtained from the website: <https://cubicsdr.readthedocs.io/en/latest/application-window.html>

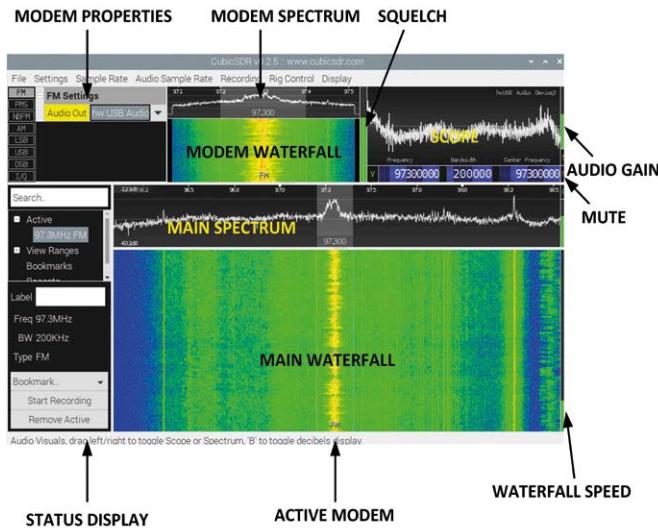


Figure 24.16: CubicSDR running.

The main spectrum and waterfall displays can be zoomed using the arrow keys or mouse, and the mouse wheel. **CubicSDR** uses a fixed resolution FFT. The Visual Gain can be adjusted by right clicking and dragging on the Main Spectrum or the Waterfall.

The centre frequency is set by dragging left or right on the main spectrum, or by using the left and right arrow keys, or pointing the mouse on a number and using the mouse wheel to change the number.

The modulation can be selected as: AM, FM, FMS (stereo), NBFM, LSB, USB, DSB, or I/Q Raw.

The squelch can be set by dragging the slider at the right side of the Modem Waterfall. Right clicking the squelch will set it just above the current signal level.

The Waterfall speed can be adjusted from 1 to 1024 lines per second by dragging the meter to the right of the main waterfall (or by using the mouse wheel).

Most numeric controls can be entered directly using the keyboard. Just hover over the desired value and press SPACE to open the input dialog, or just start typing the numbers.

When entering the frequency values, if the value is greater than 3000 then Hz will be assumed automatically. Direct input accepts suffixes such as Hz, MHz, or GHz.

24.5 RTL-SDR server

We can configure our RTL-SDR dongle to send received data over a network using the TCP protocol. This feature can be very useful for remote access of the dongle. The steps are as follows:

- Plug in the dongle to one of the USB ports of your Raspberry Pi.
- Enter the following command:

```
pi@raspberrypi:~ $ rtl_tcp -a 192.168.1.202
```

where 192.168.1.202 is the IP address of the Raspberry Pi (use command **ifconfig** if you want to find the IP address). You should see a message as in Figure 24.17.

```
pi@raspberrypi:~ $ rtl_tcp -a 192.168.1.202
Found 1 device(s):
  0: Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
[R82XX] PLL not locked!
Tuned to 100000000 Hz.
listening...
Use the device argument 'rtl_tcp=192.168.1.202:1234' in OsmoSDR (gr-osmosdr) source
to receive samples in GRC and control rtl_tcp parameters (frequency, gain, ...)
```

Figure 24.17: Start the server.

- Start **SDR Sharp** on your laptop (assuming you have installed the **SDR Sharp** on your laptop) and select **RTL-SDR (TCP)** under **Sources** (Figure 24.18)

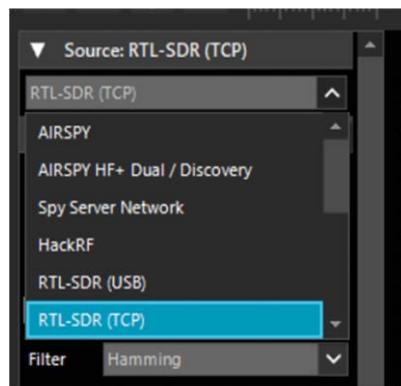


Figure 24.18: Select RTL-SDR (TCP).

- Click the **Configure Source** (Cog wheel icon) and enter the IP address of your Raspberry Pi in field **Host** and set the RF gain (Figure 24.19).

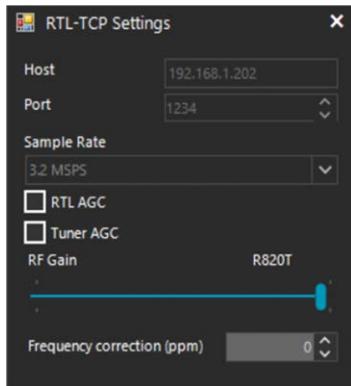


Figure 24.19: Enter the IP address.

- Tune the **SDR Sharp** to the frequency you wish to listen to (e.g. set to a local FM broadcast frequency for easy checking).

You may find out that the audio quality is not as good as expected. A better response can be obtained if your Raspberry Pi is connected to your network directly (i.e. through an Ethernet cable) rather than over the Wi-Fi.

24.6 SimpleFM

This is a few-lines-of-Python program developed by the author which enables the user to tune to a broadcast frequency in the FM band and then listen to the broadcast.

The program is based on the RTL_FM and Figure 24.20 shows the program listing (Program: **simpleFM.py**) which consists of only a few lines of code. The program prompts the user to enter the required frequency in kHz and then uses RTL_FM to tune to the entered frequency. Copy the program to your Raspberry Pi, plug-in your RTL-SDR and turn on the microphone and enter the following command to start the program:

```
pi@raspberrypi:~ $ python3 simpleFM.py
```

```
#=====
#          TUNE TO FM FREQUENCY
#=====
#
# This program tunes to an FM frequency (WBFM) using the RTL_FM
#
# Author: Dogan Ibrahim (G7SCU)
# File  : simpleFM.py
# Date  : August, 2020
#=====
import os
print("")
print("SimpleFM - Tune to a FM frequency")
```

```

print("====")
print("")

freq = 1000.0*float(input("Enter the FM frequency (kHz): "))
frequency = str(freq)
rtl = "rtl_fm -M wbmf -f" + frequency + " | aplay -r 32000 -fS16_LE -c 1"
os.system(rtl)

```

Figure 24.20: Program: simpleFM.py.

Figure 24.21 shows an example run of the program where the required frequency was 97.3 MHz (frequency of the LBC radio broadcast in London). Enter **Cntrl+C** to terminate the program.

```

pi@raspberrypi:~ $ python3 simpleFM.py
SimpleFM - Tune to a FM frequency
=====
Enter the FM frequency (kHz): 97300
Found 1 device(s):
 0: Realtek, RTL2838UHIDIR, SN: 00000001
Using device 0: Generic RTL2832U OEM

```

Figure 24.21: Example run of the program.

Creating a shell script

An alternative to creating a Python program is to create a shell script and run RTL_FM from this script. The steps are:

- Use the **nano** editor and create a file called **fm** with the contents as follows:

```

#!/bin/sh
rtl_fm -M wbmf -f$1 | aplay -r 32000 -fS16_LE -c 1

```

- Make the script executable:

```
pi@raspberrypi:~ $ sudo chmod +x fm
```

- Run the script file and specify the frequency. For example, if the frequency is 97.3 MHz, enter:

```
pi@raspberrypi:~ $ ./fm 97.3M
```

or,

```
pi@raspberrypi:~ $ sh fm 97.3M
```

Notice that **\$1** in the script corresponds to the entered frequency value.

24.7 ShinySDR

ShinySDR is an excellent SDR software that is accessible over a LAN. The software supports many versions of SDR-based hardware, including the RTL-SDR, HackRF, USRP, and many others. The software can be used to receive AM, FM (narrowband and broadcast), SSB, and CW. Multiple frequencies can be monitored at once. Packages such as ADS-B, APRS, WSPR, RTTY are supported (telemetry support requires the installation of an additional open source software). When position data is available it is displayed on a map with animated positions.

The receiver can be listened to and remotely controlled over a LAN or the Internet. The software has been developed using the Python language. ShinySDR is copyrighted by Kevin Reid and is available under the GNU General Public License.

This package is highly recommended as it is configurable and provides access to the receiver from anywhere.

The instructions to install the ShinySDR on the Raspberry Pi are as follows:

- Obtain a copy of the ShinySDR source, which will create a directory named **shinysdr**:

```
pi@raspberrypi:~ $ git clone https://github.com/kpreid/shinysdr/
```

Move to directory shinysdr and start the build:

```
pi@raspberrypi:~ $ cd shinysdr  
pi@raspberrypi:~/shinysdr $ python setup.py build  
pi@raspberrypi:~/shinysdr $ sudo python setup.py install
```

- Create a configuration directory in the default home directory called **shiny**. This will be the new configuration directory:

```
pi@raspberrypi:~/ shinysdr shinysdr -create /home/pi/shiny/
```

- Edit file **config.py** in the configuration directory, using **nano**:

```
pi@raspberrypi:~/shinysdr cd ..
```

```
pi@raspberrypi:~ $ nano config.py
```

change the following line:

```
config.devices.add((u'osmo', OsmoSDRDevice(' '))
```

to:

```
config.devices.add((u'osmo', OsmoSDRDevice('rtl=0')))
```

- Exit from **nano** by entering **Cntr+X** followed by **Y**.

We are now ready to start the server software on the Raspberry Pi. Enter the following command (see Figure 24.22):

```
pi@raspberrypi:~ $ cd shinysdr
pi@raspberrypi:~/shinysdr shinysdr /home/pi/shiny/
```

```
INFO:shinysdr:Starting factory <txws.WebSocketFactory instance at 0x9a7f8170>
INFO:shinysdr:ShinySDR is ready.
INFO:shinysdr:Visit http://localhost:8100/4hZ9fNHldkS1FZZvEWBx8g/
INFO:shinysdr:Starting RFC 6455 conversation
INFO:shinysdr:Stream connection to /4hZ9fNHldkS1FZZvEWBx8g/audio-stream?rate=480
00
INFO:shinysdr:Flow graph: Rebuilding connections because: added audio callback
gr::log :INFO: audio source - Audio source arch: alsa
INFO:shinysdr:Flow graph: ...done reconnecting (13.8690471649 ms).
gr::log :INFO: audio source - Audio source arch: alsa
INFO:shinysdr:Starting RFC 6455 conversation
INFO:shinysdr:Stream connection to /4hZ9fNHldkS1FZZvEWBx8g/radio
■
```

Figure 24.22: Starting the server software (only end of the display is shown).

The last line displays:

INFO:shinysdr:Visit http://localhost:8100/4hZ9fNHldkS1FZZvEWBx8g/

Now, start the Google Chrome Browser on your PC and enter the address at the top of the display and press Enter:

<http://192.168.1.202:8100/4hZ9fNHldkS1FZZvEWBx8g/>

Notice that 192.168.1.202 is the IP address of the Raspberry Pi, 8100 is the port number, and the remainder is the security key.

You should see the shinySDR started. Click to set the RF source at the top right of the display to **OsmoSDR rtl=0**. Tune to a frequency (e.g. a local FM broadcast frequency). In Figure 24.23 the author tuned to 97.3 MHz which is the frequency of the LBC local radio in London.

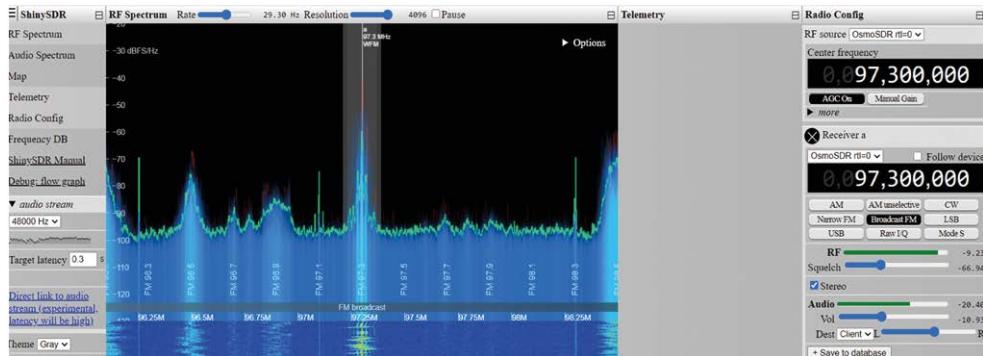


Figure 24.23: The shinySDR, tuned to 97.3 MHz.

If you do not hear any sound, click **audio stream** at the left and then click **Direct link to audio**, then click the 'play' button to start the audio.

In addition to displaying the RF spectrum and the waterfall, you can display various other useful items, such as the audio spectrum, telemetry, map, etc.

Figure 24.24 shows the **Radio Config** display where you can enter the required frequency.

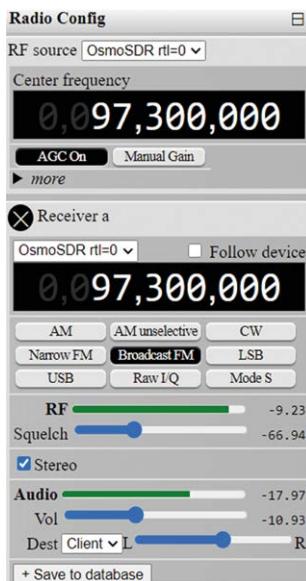


Figure 24.24: Radio Config display.

Enter **Cntrl+C** at the Raspberry Pi terminal to terminate the server. For further information on installing, configuring, and using the shinySDR browse the web site: <https://shinysdr.switchb.org/manual/configuration#rx>

Notice that if you do not want to enter the secret key, you should edit the configuration file **config.py** in directory **shiny** and comment the line starting with **root_cap**, i.e. change the line to:

```
# Set to None to not use any secret.  
#     root_cap='4hZ9fNHIdkS1FZZvEWBx8g',  
# Page title / station name
```

You should then enter the following address in your Google Chrome browser to start the client software on your PC:

<http://192.168.1.202:8100/public/>

24.8 Other SDR-RTL software

There's a lot of other SDR-RTL software around that can be downloaded from the Internet. Examples include:

- Linrad
- WebRadio
- QtRAdio
- Natpos
- Zeus Radio
- PLSDR
- etc

24.9 The SDR – The big brother of RTL-SDR?

So far, we have been using the RTL-SDR type dongles, costing around £12 and using the Realtek RTL2832U controller and tuner chip. Both the hardware and software of the RTL-SDRs have been changed dramatically in the last decade. Prior to 2012, SDRs were complex devices and costed several hundreds of pounds.

In addition to the RTL-SDR dongles, you can purchase SDR devices costing several hundreds of pounds. The choice whether or not to purchase an SDR device depends on what you want to listen to, and what software you want to run, as well as the cost of course. For most general purpose applications, such as listening to the broadcast stations, police, ambulance, emergency services, etc the cheap RTL-SDR dongles are great because the signal levels are usually high and indoor aerials can be used. SDR devices will certainly give better weak signal performance, especially in the presence of stronger local signals. SDR devices also give better dynamic ranges, wider bandwidth, and better filters.

A simple comparison of a typical RTL-SDR dongle and an SDR (e.g. SDRPlay RSP1) is shown in Table 24.1.

	Cost	Freq range	ADC resolution	Bandwidth	Preselector
RTL_SDR	£12	24 MHz - 1.7GHz	8 bits	3.2 MHz	On-chip filters
SDRPlay RSP1	£62	10 kHz - 2GHz	12 bits	10 MHz	8 switched filters

Table 24.1: Comparison of a typical RTL-SDR and SDR.

Some SDRs are both RX and TX devices (e.g.HackRF One, RedPitaya, RS-HFIQ, Xiegu G90, RS-918, Hermes Lite-2 etc). One of the problems of using SDR devices with the Raspberry Pi is the requirement of high CPU speed.

Transceiver software (e.g Quisk, linHPSDR etc) is available for TX/RX type SDR hardware and can be installed on a Raspberry Pi.

24.9.1 The HackRF One

This SDR transceiver (web link: <https://greatscottgadgets.com/hackrf/one/>) from Great Scott Gadgets costs around £320, and it has the following features (Figure 24.25).

- 1 MHz to 6 GHz operating frequency
- 20 million samples/s
- Half-duplex TX/RX
- Compatible with popular software (SDR#, GNU radio etc)
- USB 2.0 interface
- 8-bit resolution
- Open source hardware
- Protected by moulded plastic enclosure



Figure 24.25: The HackRF One SDR.

The transmit power is up to 10 dBm for the frequency range up to 4 GHz. An external RF amplifier is recommended to increase the power if required. The maximum receive power is -5 dBm.

24.9.2 The NooElec NESDR Smart HF bundle

One of the problems of most RTL-SDR dongles is that their frequency range starts from around 24 MHz, which seems to rule out HF-band work.

The NooElec NESDR Smart HF is a 100 kHz – 1.7 GHz SDR bundle (Figure 24.26) for HF/UHF/VHF, including RTL-SDR, upconverter, balun, all necessary adapters, desktop indoor antennas, and cables, costing £103. The bundle is a good starting kit for those who wish to enter into the world of SDR. Using the upconverter, the RTL-SDR can operate at as low as 100 kHz, thus enabling the amateurs to use the HF.



Figure 24.26: The NooElec SDR.

The instructions to use the SDR in the HF band is:

- connect the antenna to the Ham It Up unit
- connect the SDR to the IF output using the supplied connector
- make sure the toggle switch is in the enable position
- connect the SDR to your Raspberry Pi using a USB cable
- plug the Ham It Up USB-B USB jack to a USB power source
- start using the device. Tune to 125 MHz + (\pm the tuning offset from the tuning procedure) + your desired frequency

24.9.3 The AirSpy HF+

This is a high performance SDR developed jointly between Airspy, Itead Studio, and ST Microelectronics (see web site: <https://airspy.com/airspy-hf-plus/>).

The basic technical specifications of this device are:

- HF coverage starting from 9 kHz
- 60 to 260 MHz VHF coverage
- 22-bit resolution
- 0.5 ppm high precision low noise clock
- 1 PPB frequency adjustment
- Excellent noise reduction (claimed to be the best one in the market)
- High dynamic range (ADC up to 36 MSPS)
- Wideband RF filter bank
- Tracking RF filters

- Inputs matched to 50 Ohms
- 4 x GPIO
- Smart AGC with real time optimization
- Supported by well-known software, such as SDR Sharp, SDR-Console, GQRX, HDSR, Krypto1000 etc
- Supported by operating systems, such as Windows, Linux, BSD, OSX
- Supported by hardware, such as Windows PPC, Raspberry Pi, Odroid, etc

24.9.4 The Quisk

Quisk is an SDR **transceiver software** developed in Python by James Ahlstrom (N2ADR). The software runs on higher-end SDR devices such as the SoftRock, Hermes Lite-2, Red Pitaya, Odyssey, SdrMicron, RadioBerry-2, etc.

Quisk can be installed on the Raspberry Pi in Desktop mode:

- Start the Desktop
- Click **Preferences** → **Add/Remove Software**.
- Enter **quisk** in the search box and press Enter.
- Click next to quisk and click Apply.
- Enter your Raspberry Pi password.
- Wait until the software is installed.

To run Quisk, open a terminal session in Desktop and enter the command **quisk**:

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

Figure 24.27 shows the quisk startup screen. Click **Config** button and then click **Radios** and select your radio as shown in Figure 24.28.

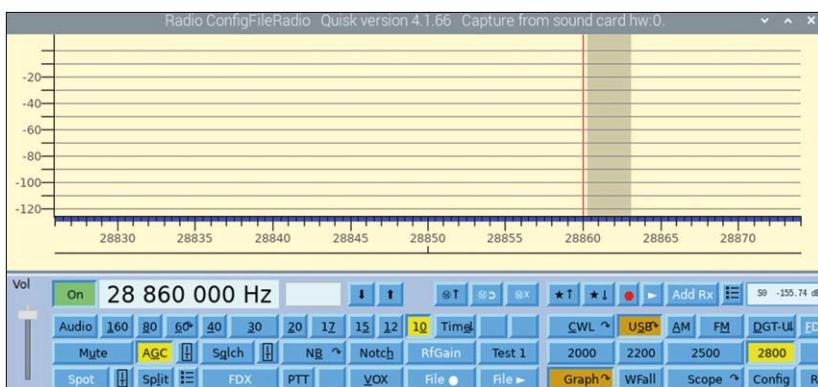


Figure 24.27: Quisk startup screen.



Figure 24.28: Select your radio.

Give a name to your radio (at the right-hand side) and click **Add** (you may have to move the slider to the right).

24.10 Receiving Weather Fax (WEFAX)

WEFAX is transmitted at 60, 90, 100, 120, 180 or 240 LPM (lines per minute) speeds with modes IOC 576 or IOC 288. Most weather forecasts are sent at 120 LPM in IOC 576. In this format, the duration of the fax is as follows:

Start tone:	5 seconds
Phasing signal:	30 seconds
Image:	10 minutes (at 120 LPM)
Stop tone:	5 seconds
Black:	10 seconds

At 120 LPM, fax is transmitted line by line at a rate of 120 lines per minute (or half a second per line). The pixels in the image are converted into certain audio tones. For example, 1500 Hz represents black, 2300 Hz represents white, and frequencies in between represent various shades of grey.

Weather faxes are transmitted by many countries around the world. For example, Northwood in UK transmits weather fax messages at specified times (see link: <http://www.yachtcom.co.uk/comms/weather/>) at the following frequencies:

Nominal frequency	GQRX freq	Times
2618.5 kHz	2616.6 kHz	2000-0600 UTC
4610.0 kHz	4608.1 kHz	0000-2400 UTC
8040.0 kHz	8038.1 kHz	0000-2400 UTC
11086.5 kHz	11084.6 kHz	0600-2400 UTC

You can set your RTL-SDR for example with the GQRX to receive the weather fax transmissions. The transmission is at the AM Upper band, where the frequency should be set 1.9 kHz below the nominal frequency as shown above. Therefore, set the **Mode** to AM with **Filter** set to Normal and select one of the above frequencies. Set the recording directory of the GQRX to where the received data will be stored. If you do not have a transceiver unit, connect a suitable antenna to your RTL-SDR device. You will need an upconverter to listen to the HF band with the RTL-SDR device.

Wait until you get the tone of the transmitter and press the recording button on the GQRX to start recording. The message will take around 11 minutes and you should press the button again to stop recording.

You can now use the **hamfax** software (by Christof Schmitt, DH1CS) to see the image. First, install the software:

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

Open a terminal session in Desktop and start the program by entering (see Figure 24.29):

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

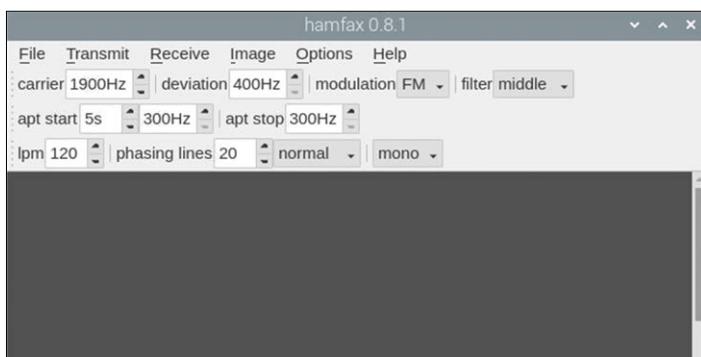


Figure 24.29: Main menu.

Click **Receive** at the top and select **Receive from file** and browse to select the recorded file.

A sample fax sound file is available at the link:

https://sourceforge.net/projects/hamfax/files/hamfax/hamfax-example-2002-06/2001-06-21-09-00-GYA-11086.au/download?use_mirror=master&download=

An example image file built on the screen is shown in Figure 24.30. Notice that the slant of the image can be corrected by selecting **image** and then **slant correction**.

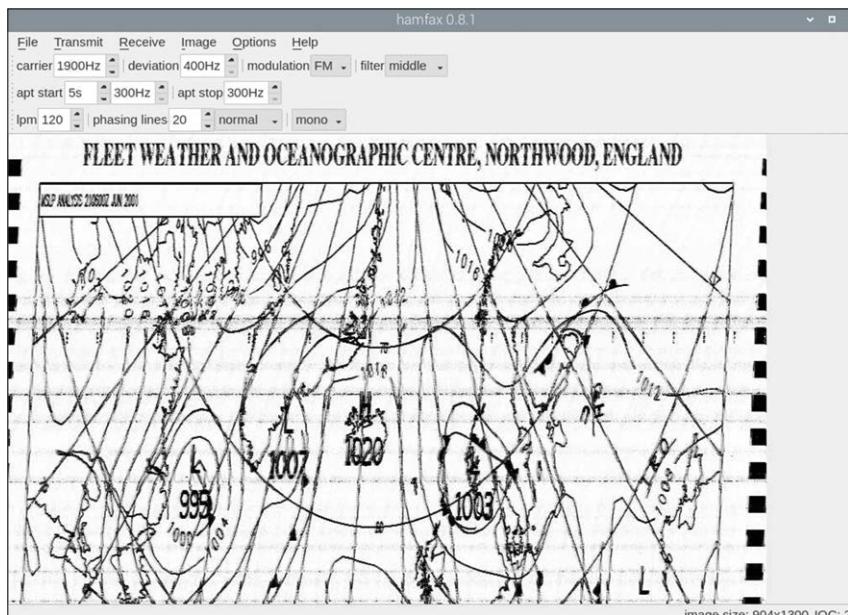


Figure 24.30: Example fax image.

The following weather fax related web sites can be of interest to readers:

- www.bbrc.info/articles/receiving-weatherfax-maps
- www.blackcatsystems.com/radio/rfax.html
- www.blackcatsystems.com/software/multimode/fax.html#GFA
- www.blackcatsystems.com/software/multimode/fax.html#CFH
- www.blackcatsystems.com/software/multimode/fax.html#NMC

CHAPTER 25 • Using Some of the Popular Radio Applications

This is the last Chapter in the book. In it we will be learning how to install and use some of the popular amateur radio applications onto our Raspberry Pi.

In this Chapter we will be installing most of the software from the Desktop via the Raspbian repository. It is therefore important that the readers follow the steps given in section 25.1 to install the software.

25.1 TW CLOCK

This program (by WA0EIR) displays the local time and GMT in hundreds of major cities around the world. The program also has a timer which can be set so that the station ID is output as audio Morse code at the specified time intervals. The sound output can be connected to the Audio-In pin of your rig's accessory jack to have it transmitted automatically at the set intervals.

Installation

The program can easily be installed from the Raspbian repository in desktop. The steps are as follows (notice that the installation process described here is very easy if the application program is already in the Raspbian repository. But, the problem is that the software may not be the latest version. Installing from the sources will guarantee that the latest version of the software is installed):

- Start the Desktop.
- Click **Preferences → Add / Remove Software** in the top left **Applications Menu**
- Enter **twclock** in the search path and press **Enter**.
- Click to select the software (Figure 25.1) and click **Apply**.



Figure 25.1: Select the software.

- Enter your Raspberry Pi password and wait until the software is installed. Click **OK** to exit.

Using the program

To start the program, open a terminal session in Desktop and enter (or click to open **Accessories** in the top left **Applications Menu** and click **twclock**):

```
pi@raspberrypi@~ $ twclock
```

Hold at the bottom right-hand side of the display to enlarge it.

To see the World Time, click the right button of your mouse and select **Others**. Select **Region** (e.g. Europe), select **City** (e.g. Turkey) as in Figure 25.2 click **OK**. You should see the time in Turkey displayed as shown in Figure 25.3.

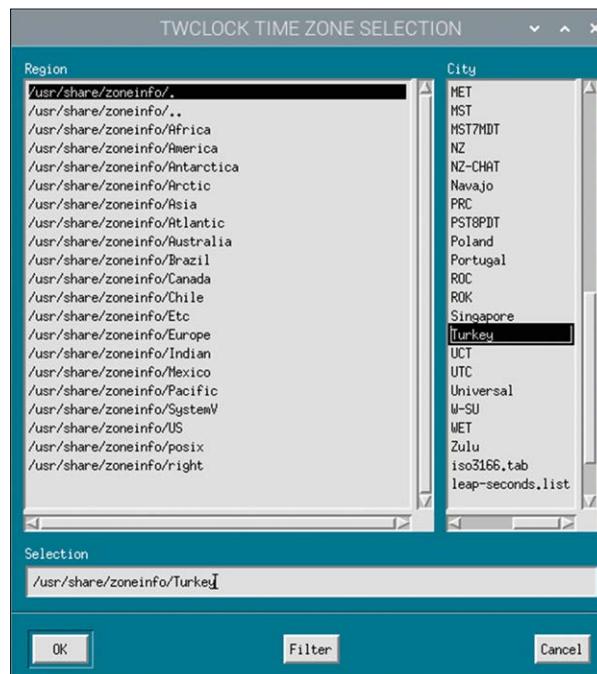


Figure 25.2: Select a city.

Figure 25.3: Displaying time in the selected city.

twclock is installed by default to the following directory:

```
/etc/X11/app-defaults
```

Before using the ID and the timer, we have to specify our station ID (call sign) and the Morse code to be transmitted at the set timer intervals. This is done as follows:

- Use the **nano** editor and edit file:

```
pi@raspberrypi:~ $ sudo nano /etc/X11/app-defaults/Twclock
```

- Specify your station ID by editing the following statement (e.g. assuming the ID is G7SCU):

Twclock.form.call_toggleB.labelXString: **G7SCU**

- Specify the Morse code to be transmitted at the set time intervals (you can also change the CW speed, tone frequency etc):

Twclock.cwStr: **de G7SCU**

- You may like to look at the other settings and change them if you wish.
- You should now restart twclock for the changes to take effect.

Clicking **ID Now** in the main menu plays your station ID as an audible Morse code. The time interval to play the station ID can be set by clicking **Set Timer** in the main menu (see Figure 25.4) and entering the time in minutes and seconds, followed by **OK**. You should click **CW ID** to enable playing the CW. Setting **Auto Reset** will repeat the audio output at the specified intervals. The CW speed and the tone frequency can also be set from this menu.

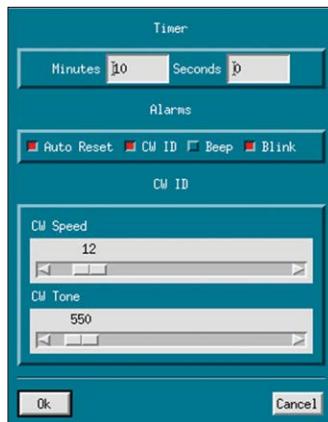


Figure 25.4: Set Timer selected.

A help option is provided in the menu for displaying information on various features of **twclock**.

25.2 Klog

This program (by Jaime Robles, EA4TV) has been developed to replace the radio amateur's paper logbook. The program is available in many platforms (e.g. Windows, MAC etc) and in many languages. Klog provides QSO management, QSL management, DXCC management, club log integration, and much more.

Installation

The program should be installed as described in section 25.1 but enter **klog** for the program name.

Using the program

To start the program, open a terminal session in Desktop and enter:

```
pi@raspberrypi@~ $ klog
```

When the program is run first-time you will be asked to accept a license condition and click to download country data. Then, a **Config Dialog** screen will be displayed to enter configuration data e.g. personal data, station data, band details, modes of operation etc. Figure 25.5 shows the main screen of Klog. The program is very comprehensive and includes many menus and options. At top left we have the entry box, top right the output box, and at the bottom part we have the Log, DX-Cluster, and DXCC. Perhaps the best way to learn to use this program is to download and play with it.

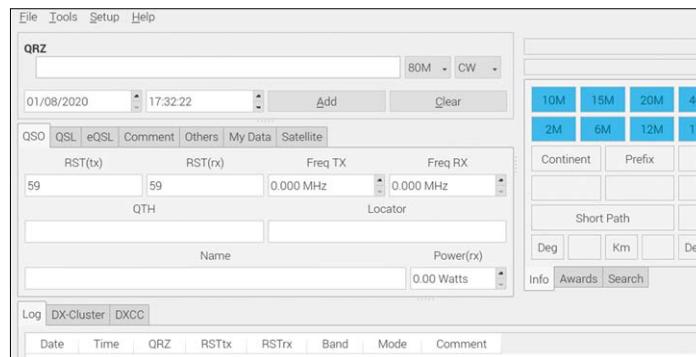


Figure 25.5: Klog main screen.

Using Klog, the operator can:

- add/edit/search/remove QSOs
- manage QSLs
- import data from formats such as ADIF, TLF, Cabrillo
- support DXCC and WAZ local awards
- perform eQSL management
- perform ClubLog integration
- have support with satellite logging

25.3 Gpredict

Gpredict is a real-time satellite tracking and orbit prediction application. The program can track satellites and display their position on a map and other data in lists and tables. Gpredict can predict the time of future passes and detailed information for a satellite (website for further information: <https://www.pe0sat.vgnet.nl/satellite/sat-information/tracking/>).

Installation

The program should be installed as described in section 25.1 but enter **Gpredict** for the program name.

Using the program

To start the program, select **Internet** in the Desktop **Applications Menu** and then click **Gpredict**. Figure 25.6 shows the screen when the program is run.

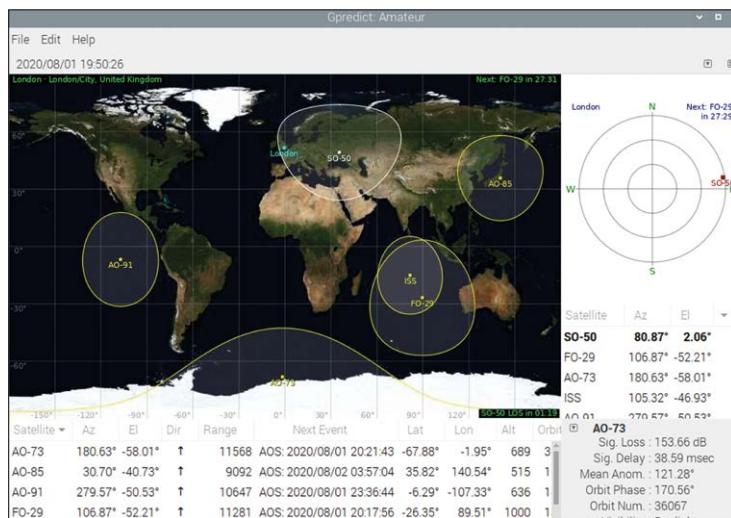


Figure 25.6: The Gpredict screen.

The program has three menus: **File**, **Edit**, and **Help**. The **Edit** menu is of interest for us. By selecting this option, we can update the TLE and the transponder data from the network. Option **Preferences** under **Edit** has some interesting options. The important tabs here are:

General: here we can change the time to UTC or local time, change the time format, change the geographical coordinate display, and select metric or imperial units.

Modules: here we can select the display layout, refresh rates, list view, map view, polar view, and single satellite view.

Some important features of **Gpredict** are:

- No limit on the number of satellites or ground stations.
- Visual presentation of the satellite data on maps. Users can configure and customise the look and feel of the data.
- Prediction of future satellite passes where the prediction parameters can be fine-tuned.
- Automatic updates of the Keplerian Elements from the web or from local files.
- Radio and antenna rotator control for autonomous tracking.

25.4 FLDIGI

FLDigi is a popular Digital Mode data modem software (by David Freese, W1HKJ) operating in conjunction with a conventional HF SSB transceiver, using audio frequency signals over an audio adapter. Fldigi includes most of the popular digital modes, such as MFSK16, PSK31, Contestia, DominoEX, CW, FSQ, RTTY, Thor, Olivia, etc.

The following YouTube video gives a good introduction to FLDIGI:

<https://www.youtube.com/watch?v=jvOJFFkYIAs>

Figure 25.7 shows the typical FLDIGI based system setup.

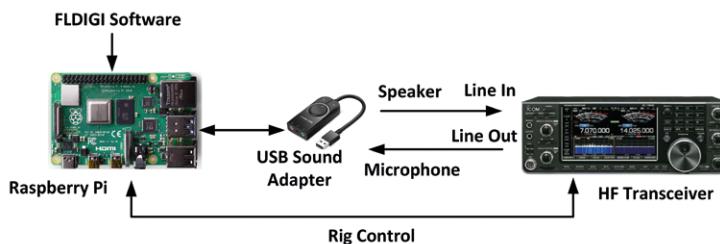


Figure 25.7: FLDIGI based system setup.

Installation

In this section we will be installing the **fldigi** from Raspbian repository as in section 25.1 (a version of the software which is not in the repository may be available, and you should check it if you wish to install the latest version).

Install the program as described in section 25.1 but enter **fldigi** for the program name.

Using the program

To start the program, select **Internet** in the Desktop **Applications Menu** and then click **Fldigi**. When the program is run the first time you will be presented with a configuration wizard as shown in Figure 25.8.



Figure 25.8: The Fldigi screen.

Click **Next** and enter the details such as the station name, operator callsign, station QTH etc. Click **Next** to configure the audio. Click **Devices** and then **PortAudio** and select **USB Audio Device** (assuming you have the USB audio adapter connected to the USB port) as shown in Figure 25.9. You should also configure your rig either during the first startup or by clicking the **Configure** menu and selecting **Rig Control → Rig → Hamlib** after the program is up and running.

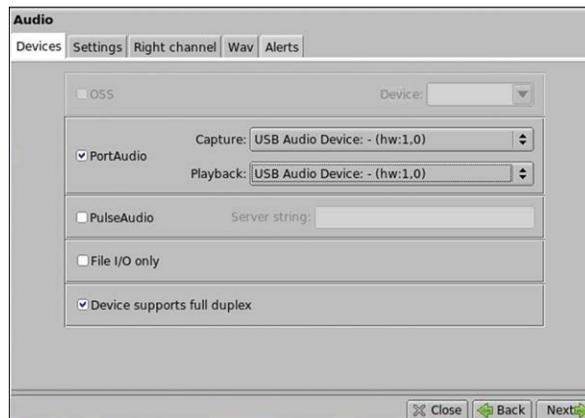


Figure 25.9: Configure the audio.

Figure 25.30 shows the FLDIGI operating screen. The top part (yellow) is where the received text is displayed. The middle part (blue) is where you type the text to be transmitted. Under the middle part are some buttons where you can click to activate various controls. The operation mode is selected by clicking menu item **Op Mode** at the top left part of the screen.

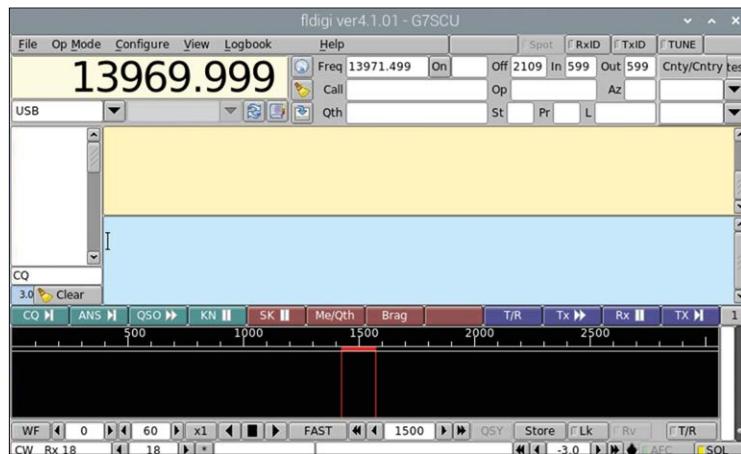


Figure 25.30: FLDIGI operating screen.

Testing

Perhaps the easiest way to test your installation is to connect the audio output of your PC to the microphone input of the Raspberry Pi through a USB adapter (e.g. UGREEN). Then run the FLDIGI program and select **CW** as the **OP Mode**. There are many sample Morse code audio files on the Internet (e.g. see website: https://commons.wikimedia.org/wiki/Category:Audio_files_of_Morse_code). Play one of the files (e.g. **A through Z in Morse code**) and select in the waterfall. You should see the text displayed on your FLDIGI screen.

25.5 Direwolf

Direwolf is a software packet modem (by John Langer, WB2OSZ) that's used for packet communication. In the early days of Amateur Packet Radio, specialized hardware called Terminal Node Controller (TNC) was used for packet communication (Figure 25.31).

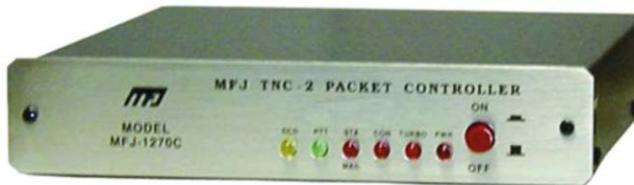


Figure 25.31: A popular TNC (Universal Radio Inc.).

A typical setup using the **TNC is shown in Figure 25.32**. The interface between the computer and the TNC was RS232 type serial communication. All old computers used to have hardware RS232 ports which were either 25-way or 9-way D-type connectors mounted at the back of the computer. These ports were also used to connect the computer to printers and plotters. The TNC device was connected to the transceiver equipment (audio and PTT) so that packets can be sent and received and displayed on the computer screen.



Figure 25.32: Typical TNC setup.

Nowadays, there is no need to use TNC equipment. We can get much better results at lower costs by connecting our radio to the soundcard interface of a computer (e.g. Raspberry Pi) and using software (e.g. Direwolf) to receive and decode the packet radio signals.

In this section we will be installing the Direwolf software package onto our Raspberry Pi.

Installation

Enter the following steps to install the Direwolf on your Raspberry Pi:

- pi@raspberrypi:~ \$ **sudo apt install libasound2-dev libudev-dev**
- pi@raspberrypi:~ \$ **git clone https://www.github.com/wb2osz/direwolf**
- pi@raspberrypi:~ \$ **cd direwolf**
- pi@raspberrypi:~/direwolf \$ **make**
- pi@raspberrypi:~/direwolf **sudo make install**
- pi@raspberrypi:~/direwolf **make install-conf**
- pi@raspberrypi:~/direwolf **make install-rpi**

Using the program

Before using the program, we have to edit the configuration file **direwolf.conf** using the **nano** editor. The steps are (setting as an **IGATE** Internet gateway):

- Enter:

```
pi@raspberrypi:~/direwolf sudo nano direwolf.conf
```

- Scroll down to section **FIRST AUDIO DEVICE PROPERTIES** and remove the comment # from the line **#ADEVICE plughw:1, 0**.
- Scroll down to section **CHANNEL 0 PROPERTIES** and to line **MYCALL NOCALL**, and replace **NOCALL** with your callsign, e.g. **MYCALL G7SCU**.
- Notice that the modem is set to 1200 by default, but can be changed if you want.
- Scroll down to DIGIPEATER PROPERTIES and remove # from line starting with DIGIPEATER.
- Scroll down to section **INTERNET GATEWAY**, line IGSERVER and set to **euro.aprs2.net** (for Europe). i.e. **IGSERVER euro.aprs2.net**.

- Remove # from line **IGLOGIN** and enter your callsign and passcode (use a passcode generator to get a passcode. e.g. see <https://apps.magicbug.co.uk/passcode>).
- Enter **Cntrl+X** followed by **Y** to save the changes.

Make sure the speaker and microphone of your rig are connected to the Raspberry Pi USB audio sockets. Start **Direwolf**, open a terminal session in Desktop and enter the command:

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

Alternatively, click the **direwolf** icon in the Desktop. You should see a screen similar to the one shown in Figure 25.33, displaying that the **Direwolf** is ready to accept AGW client on port 8000 and KISS TCP client on port 8001. You should see messages on the screen when packet transmissions are received.

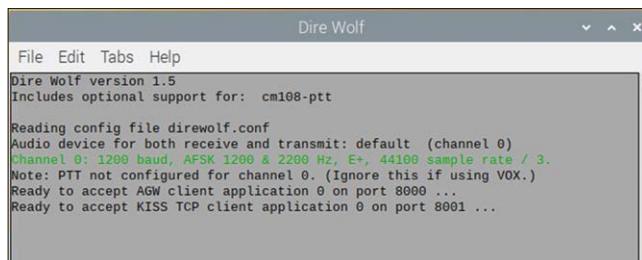


Figure 25.33: *Direwolf* screen.

To test your Direwolf installation, tune your radio to the APRS frequency (144.8 MHz in Europe). Listen via speaker to confirm traffic at the frequency and check for packet receipts.

Readers can get detailed information on **Direwolf** from the following directory on the Raspberry Pi:

```
/usr/local/share/doc/direwolf
```

RTL-SDR output to Direwolf (Receive only)

We can direct the RTL-SDR output to **Direwolf** by entering the following **rtl_fm** command (see Figure 25.34):

```
pi@raspberrypi:~ $ rtl_fm -f 144.80M - | direwolf -r 24000 -D 1 -
```

```
pi@raspberrypi:~ $ rtl_fm -f 144.80M | direwolf -r 24000 -D 1 -
Dire Wolf version 1.5
Includes optional support for: cm108-ptt

Reading config file direwolf.conf
Audio input device for receive: stdin (channel 0)
Audio out device for transmit: default (channel 0)
Found 1 device(s):
  0: Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Tuner gain set to automatic.
Tuned to 145052000 Hz.
Oversampling input by: 42x.
Oversampling output by: 1x.
Buffer size: 8.13ms
Exact sample rate is: 1008000.009613 Hz
Allocating 15 zero-copy buffers
Sampling at 1008000 S/s.
Output at 24000 Hz.
Channel 0: 1200 baud, AFSK 1200 & 2200 Hz, E+, 24000 sample rate.
Note: PTT not configured for channel 0. (Ignore this if using VOX.)
Ready to accept KISS TCP client application 0 on port 8001 ...
Ready to accept AGW client application 0 on port 8000 ...
```

Figure 25.34: RTL-SDR with Direwolf.

25.6 xcwcp

xcwcp is a graphical Morse code training program. The user can choose from a number of options for practising, including sending random characters, random words, CW words, and many other groups of word combinations. The generated Morse code text is displayed on the screen while at the same time it is played on the speaker. The speed, tone, and volume of the generated code can all be adjusted from the main menu.

Installation

In this section we will be installing the **xcwcp** from Raspbian repository.

Install the program as described in section 25.1 but enter **xcwcp** for the program name.

Using the program

To start the program, open a terminal session in Desktop and enter the following command:

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

The command accepts options (see website: <http://manpages.ubuntu.com/manpages/trusty/man1/xcwcp.1.html>) . Some of the commonly used options are (you can also enter the command as **xcwcp - --help** to see a list of all the options):

- s change the sound device (pulseaudio is the default)
- w set initial wpm (can also be set from the main menu)
- t set initial tone (default = 800 Hz, range 0 - 4000 Hz)
- v set initial sound volume (0 to 100)

Figure 25.35 shows the screen of the **xcwcp**, where the program was set to generate letters at 12 wpm with a tone of 800 Hz. The volume was set up at 70%. The text font and text colour can be changed from the **Settings** menu.

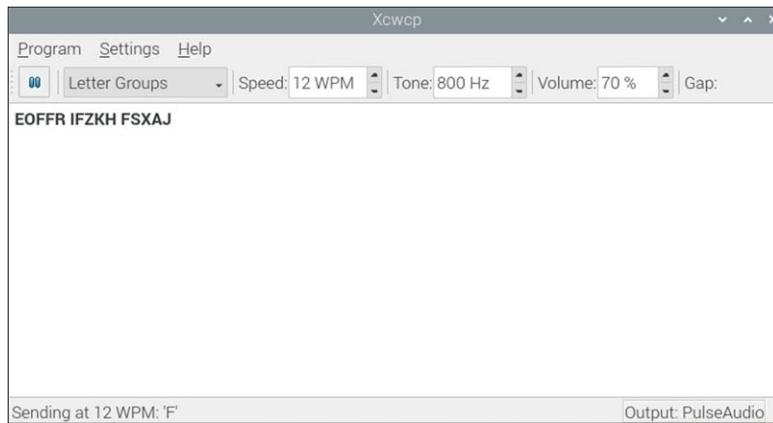


Figure 25.35: xcwcp screen.

25.7 QSSTV

SSTV is a means of sharing images among amateurs. In a typical application, the computer (e.g. Raspberry Pi) audio adapter is connected to the audio interface of the rig. Then a program (e.g. qsstv) runs on the computer to receive the images. There are several STV frequencies in the amateur bands including 2 metres (144.550 MHz, 145.500 MHz, 145.600 MHz) and 20 metres (14.230 MHz and 14.233 MHz).

This is the Slow Scan TV application (by Johan Maes, ON4QZ). Version 9.4.x is the latest official version. Earlier version (9.2.6) was on the Raspbian repository at the time of writing this book.

Installation

The steps to install the latest version are given below (notice that the library name is **lib-v4l-dev** where **I** is the lower-case **L**, not number 1):

- pi@raspberrypi:~ \$ **sudo apt-get install g++ libfftw3-dev qt5-default lib-pulse-dev**
- pi@raspberrypi:~ \$ **sudo apt-get install libhamlib-dev libasound2-dev lib-v4l-dev**
- pi@raspberrypi:~ \$ **sudo apt-get install libopenjp2-7 libopenjp2-7-dev**
- Navigate to QSSTV website <http://users.telenet.be/on4qz/> and download the latest version (version 9.4.4, filename: **qsstv_9.4.4.tar.gz** at the time of writing this book).
- Move to directory **Downloads** and check that the file is there:

```
pi@raspberrypi:~ $ cd Downloads
pi@raspberrypi:~/ Downloads $ ls
```

```
qsstv_9.4.4.tar.gz  
pi@raspberrypi:~/ Downloads $
```

- Unpack the file:

```
pi@raspberrypi:~/ Downloads $ tar -xvf qsstv_9.4.4.tar.gz
```

- Move to subdirectory **qsstv_9.4.4**

```
pi@raspberrypi:~/ Downloads $ cd qsstv_9.4.4  
pi@raspberrypi:~/ Downloads/qsstv_9.4.4 $
```

- Enter **qmake** and then **make**:

```
pi@raspberrypi:~/ Downloads/qsstv_9.4.4 $ qmake  
pi@raspberrypi:~/ Downloads/qsstv_9.4.4 $ make  
pi@raspberrypi:~/ Downloads/qsstv_9.4.4 $ sudo make install
```

Using the program

To start the program, open a terminal session in Desktop and enter the following command:

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

Alternatively, you can click the Applications Menu in Desktop and then click QSSTV.

When the program is run the first time, it should be configured. Click **Options** and then **Configuration** and enter the station details, audio details etc (Figure 25.36). Click **OK** when finished.

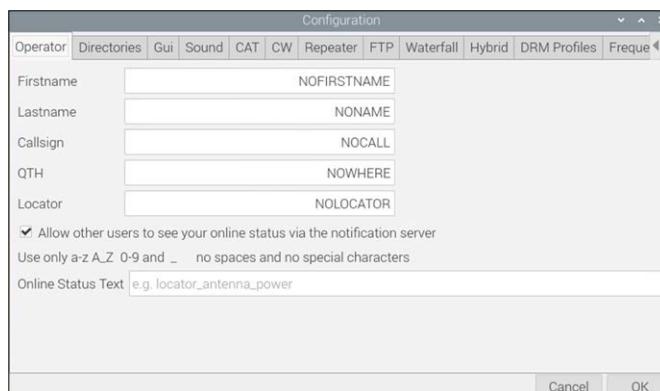


Figure 25.36: Configure station details.

Click **Options** and then **Calibrate** to calibrate your sound card (Figure 25.37). You may have to wait for several minutes until the calibration is over.

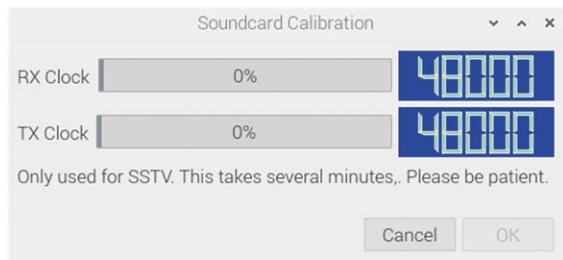


Figure 25.37: Calibrating the sound card.

Testing the program

There are many SSTV audio files available on the Internet that can be used to test the program. For example, Essex Ham (UK) provides four audio CCTV signals on their website: <https://www.essexham.co.uk/sstv-the-basics>. The steps to use one of these signals to test the software are as follows:

- Connect the speaker output of your PC to the microphone input of your Raspberry Pi. Here, the author was using a USB adapter (e.g. UGREEN).
- Configure the QSSTV software to receive data from the USB port (Figure 25.38) and click **OK**.

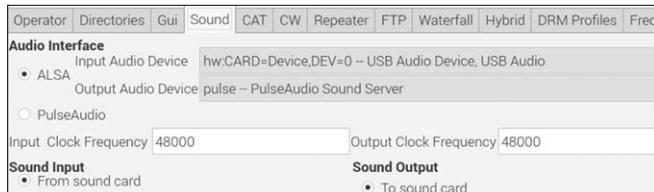


Figure 25.38: Configure the QSSTV software.

- Open the Essex Ham website where the audio files are located.
- Make sure the QSSTV software is set in **Receive** mode and click the start button (the blue play button under **Receive**).
- Click to play one of the audio files on the Essex Ham website.
- You should see the image received on the QSSTV screen as shown in Figure 25.39.

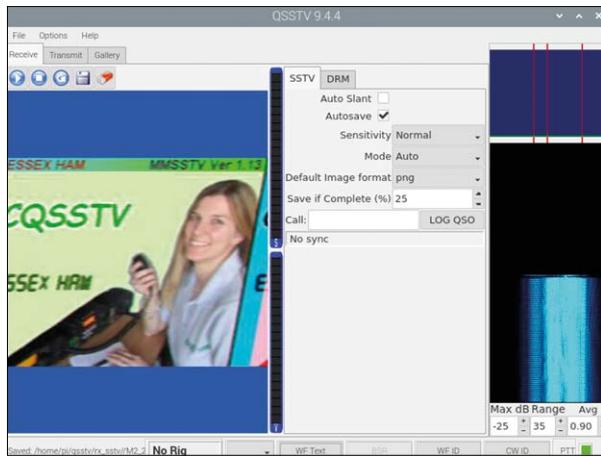


Figure 25.39: Image built on QSSTV screen.

25.8 LinPsK

LinPsK operates on digital modes, supporting BPSK, QPSK, and RTTY formats. The basic features of this program are (see website: <http://linpsk.sourceforge.net/docs/en/index-1.html#ss1.1>):

- simultaneous decoding of up to 4 channels
- mixing digital modes
- logging received data
- defining macros for larger texts
- viewing the signal as spectrum or in waterfall display

Installation

In this section we will be installing the **LinPsK** from Raspbian repository.

Install the program as described in section 25.1 but enter **linpsk** for the program name.

Using the program

To start the program, open a terminal session in Desktop and enter the following command:

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

Alternatively, select **Hamradio** in Desktop **Applications Menu** and click **Linpsk**.

When the program runs you should configure it by selecting **Settings** and then **General Settings** from the main menu. Enter your callsign, date and time format, sound device, etc.

Testing

The installation can easily be tested as follows:

- Connect the audio output of your PC to the microphone input of your Raspberry Pi using a USB audio device (e.g. UGREEN).
- There are many sample audio files on the Internet. For example, we can select to play a sample BPSK file from the following site:

<https://www.nonstopsystems.com/radio/radio-sounds-orig.html#PSK>

- Click **RxParams** and set the mode to **BPSK31**.
- Click the large **Rx** button and start to play the BPSK file on your PC. You should see the received data displayed on the screen as shown in Figure 25.40.

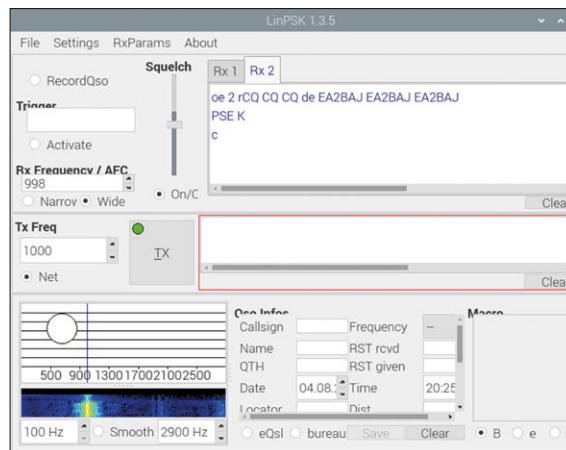


Figure 25.40: Example LinPsk display.

For interested readers, PSK apps are also available on the smartphones (e.g. DroidPSK on Android).

25.9 Ham Clock

The **Ham Clock** (by Karl-Heinz, DL1GKK) is a detailed dynamic display (see: <https://dl1gkk.com/ham-clock-raspberry-pi/>) which is updated in real time, and displays the following interesting and useful items (only some items are listed here):

- your callsign
- local (or UTC) date and time
- analogue clock
- satellite information
- XRay or Kp indx or VOACAP
- sunspots or solar flux
- solar images
- NCDXF beacons
- etc.

Installation

The steps to install the software are as follows:

- pi@raspberrypi:~ \$ **sudo apt-get update**
- pi@raspberrypi:~ \$ **sudo apt-get upgrade**
- pi@raspberrypi:~ \$ **sudo apt-get dist-upgrade**
- pi@raspberrypi:~ \$ **sudo reboot**

After the reboot, enter the following commands:

```
pi@raspberrypi:~ $ curl -o ESPHamClock.zip http://www.clearskyinstitute.com/ham/HamClock/ESPHamClock.zip
```

- pi@raspberrypi:~ \$ **unzip ESPHamClock.zip**
- pi@raspberrypi:~ \$ **cd ESPHamClock**
- pi@raspberrypi:~/ESPHamClock \$ **make -j 4 hamclock**

Running the program

- Enter the following command to run the program:

```
pi@raspberrypi:~/ESPHamClock $ ./hamclock
```

When the program runs you should do some configuration (see the User Guide at website: <https://www.clearskyinstitute.com/ham/HamClock/>). Figure 25.41 and Figure 25.42 shows two displays from the program. Notice that the Raspberry Pi was accessed remotely via VNC from a PC and a terminal session was created to run the program. Another option is to run the program with the standard 7-inch Raspberry Pi display (see the User Guide).

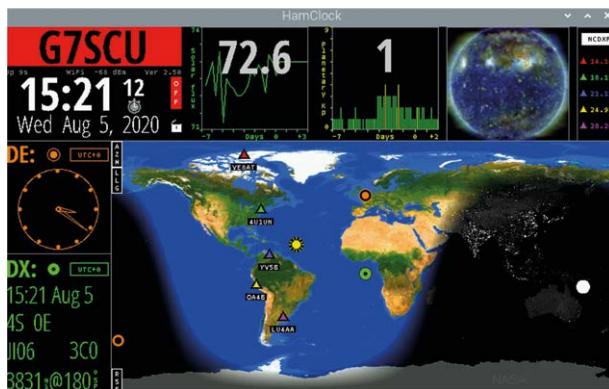


Figure 25.41: Display from the program.

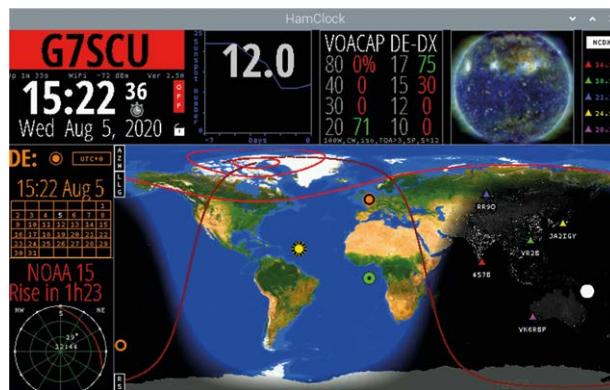


Figure 25.42: Another display from the program.

A Desktop shortcut can be created to run the program easily by clicking on this shortcut. The steps are (see the *User Guide*):

- Right click in Desktop to create an empty file on the Desktop with the name **HamClock.desktop**.
- Edit the file and add the following statements:

```
[Desktop Entry]
Name=HamClock
Comment=Open HamClock
Exec=/home/pi/ESPHamClock/hamclock
Type=Application
Encoding=UTF-8
Terminal=false
Categories=None;
```

25.9 CHIRP

CHIRP is a popular tool for programming amateur radios. CHIRP is available on several platforms including the Raspberry Pi. The program supports many manufacturers and models. It is especially popular for programming cheap Chinese radios such as Baofeng, Wouxun, etc. The CHIRP website given below provides detailed information on the supported models and how to use the software:

<https://chirp.danplanet.com/projects/chirp/wiki/Home>

also:

https://chirp.danplanet.com/projects/chirp/wiki/Beginners_Guide

Installation

Enter the following commands to install the software on the Raspberry Pi:

- pi@raspberrypi:~ \$ **sudo apt-get -y update**
- pi@raspberrypi:~ \$ **sudo apt-get -y install python-gtk2 python-serial python-libxml2**
- Locate the latest version of the software from the link (see Figure 25.43). At the time of writing this book the latest version was named: **chirp-daily-20200718.tar.gz**.

	Parent Directory	
Model_Support.html		18-Jul-2020 00:10
SHA1SUM		18-Jul-2020 00:10
Test_Report.html		18-Jul-2020 00:10
chirp-daily-20200718-installer.exe		18-Jul-2020 00:10
chirp-daily-20200718-win32.zip		18-Jul-2020 00:10
chirp-daily-20200718.app.zip		18-Jul-2020 00:10
chirp-daily-20200718.flatpak		18-Jul-2020 00:10
chirp-daily-20200718.tar.gz		18-Jul-2020 00:10
chirp-unified-daily-20200718.app.zip		18-Jul-2020 00:10

Figure 25.43: Locate the latest version of the CHIRP software.

- Enter the command to get the file:

```
pi@raspberrypi:~ $ wget --no-check-certificate https://trac.chirp.danplanet.com/chirp_daily/LATEST/chirp-daily-20200718.tar.gz
```

- Unpack the files:

```
pi@raspberrypi:~ $ tar xzf chirp-daily-20200718.tar.gz
```

```
pi@raspberrypi:~ $ cd chirp-daily-20200718  
pi@raspberrypi:/chirp-daily-20200718 $ sudo python setup.py install  
pi@raspberrypi:~/chirp-daily-20200718 $ cd ~  
pi@raspberrypi:~ $ wget https://skars.co.uk/files/chirp-setup.sh  
pi@raspberrypi:~ $ chmod +x chirp-setup.sh  
pi@raspberrypi:~ $ ./chirp-setup.sh
```

This completes the installation.

Running the program

The program can be run from the Desktop by clicking Accessories and then CHIRP as shown in Figure 25.44



Figure 25.44: Running CHIRP.

Figure 25.45 shows the startup screen when the program runs. Click Radio to download from the radio. You should connect your radio to Raspberry Pi using either serial connection or through the USB port depending on your radio and the type of cable. The USB port is recognized as **/dev/ttyUSB0** assuming that the only USB device connected to your Raspberry Pi is the radio cable. If using the serial port on your Raspberry Pi, the device name is **/dev/ttys0** (see Figure 25.46). Depending on the type of radio you have, you will either see a progress display bar, or the program will jump to the memory editor when the download is complete.



Figure 25.45: CHIRP startup screen.

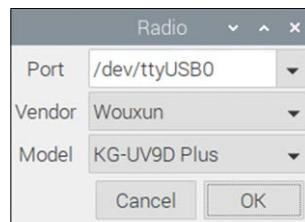


Figure 25.46: Selecting the USB port.

25.10 Xastir

Xastir is an APRS program (Automatic Packet Reporting System) that provides mapping, tracking, messaging, weather, weather alerts, and search and rescue operations over radio. APRS is a digital communication system for real-time exchange of digital information.

Installation

The steps to install the software are:

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

- pi@raspberrypi:~ \$ **sudo reboot**

Running the program

Open a terminal session in Desktop and enter the following command:

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

when the program starts first time, you should see the **Configure Station** menu (Figure 25.47) where you should enter your callsign and the LAT/LONG of your station, etc. You should then see the main screen as shown in Figure 25.48.

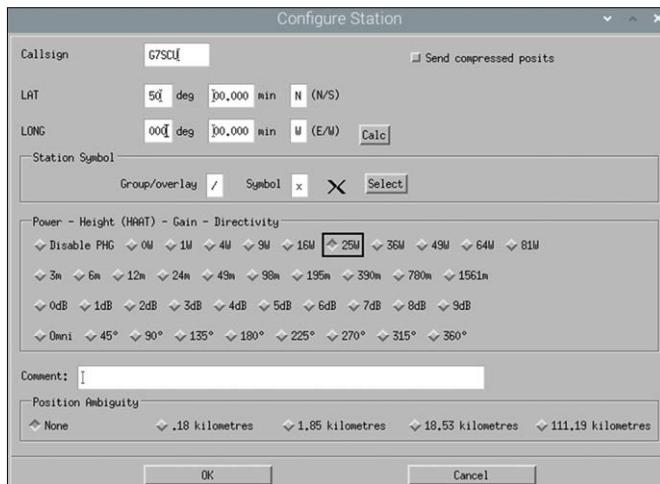


Figure 25.47: Configure Station menu.

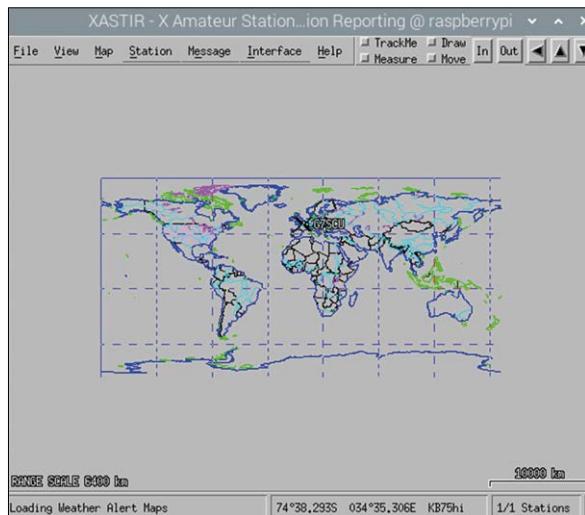


Figure 25.48: Main menu.

For details of using **xastir**, see link: https://xastir.org/index.php/Main_Page

25.11 CQRLOG

CQRLOG is one of the popular amateur radio logging programs (by Petr, OK7AN and Martin, OK1RR), based on the MySQL database. The program is installed as follows.

Installation

Install the program from the Raspbian repository as described in Section 25.1 but enter **qrlog** for the search path.

Running the program

Run the program in Desktop by selecting **CQRLOG** from menu item **Hamradio** in the **Applications Menu**. When the program is run the first time it creates the MySQL database and this may take several minutes. Figure 25.49 shows the main screen of CQRLOG

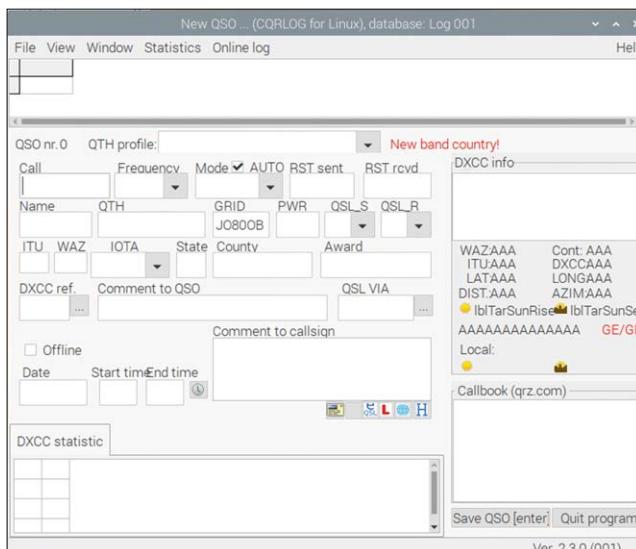


Figure 25.49: Main screen of CQRLOG.

For information on using the software, look at website: <https://www.cqrlog.com/>. Online help is also available from the program Help menu.

25.12 What next?

The Raspberry Pi is a £/€35 computer, but it has all the power to run almost all of the software that is available on a standard PC. Because of its low cost, small size, and portability, Raspberry Pi is currently used by many ham radio operators.

In this Chapter we have looked at some of the commonly used Ham radio applications. There are however many more such applications that can be installed and used on a Raspberry Pi computer.

The following websites could be useful to readers who may want to learn more about the Raspberry Pi computer. They also help you to find out what other applications are available for ham radio operators.

General books:

- *Learning Python with Raspberry Pi* — Elektor publication
- *The Ultimate Compendium of Raspberry Pi Projects* — Elektor publication
- *Raspberry Pi 3 – Basic to advanced projects* — Elektor publication
- *Hardware projects for Raspberry Pi* — Elektor publication

Ham radio application:

- https://groups.io/g/RaspberryPi-4-HamRadio/topic/w3djs_raspberry_pi_ham_radio/39671852?p=,,20,0,0,0::recentpostdate%2Fsticky,,,20,2,0,39671852
- <http://www.hamblog.co.uk/top-10-amateur-radio-uses-for-raspberry-pi/>
- <http://www.stargazing.net/david/RPi/hrrpi.html>
- <https://www rtl-sdr.com/big-list-rtl-sdr-supported-software/>
- <https://cromwell-intl.com/open-source/raspberry-pi/sdr-getting-started.html>
- <https://dl1gkk.com/setup-raspberry-pi-for-ham-radio/>
- <http://www.g0hwc.com/raspberry-pi-ham-radio.html>
- <https://www.essexham.co.uk/news/raspberry-pi-as-an-sdr.html>
- https://www.dxzone.com/catalog/Technical_Reference/Raspberry_Pi
- *The Hobbyist's Guide to the RTL-SDR: Really Cheap Software Defined Radio* (Kindle edition)

Index

\$GGGA	125	C	
\$GPGLL	125, 126	Calculator	65
\$GPGSV	125	Capacitance Meter	204
\$GPRMC	125	centre frequency	270
\$GPVTG	125	CHIRP	301
4×4 keypad	161	Cog wheel icon	271
32-bit counter	210	column scanning	162
		common-emitter amplifier	171
A			
AC parameters	171	Contestia	289
AD8318	234, 236	CQRLOG	305
AD9850	153	crondaemon	91
Add/Remove software	68	crontab	91
Advanced IP Scanner	33	cat	43
AGC	266	CE	134
AirSpy HF+	279	chmod	49
ambient temperature	116	cp	52
amixer	215	CPHA	134
Ammeter	202	CPOL	134
amplifier circuit	171	CubicSDR	269
analogue-to-digital converter	198	current limiting fuse	220
aplay	218	current-sinking mode	80
APRS packets	256	current-source mode	79
APRS program	303	cut-off frequency	177
arbitrary waveform	146	CW	289
ARM processors	12	CW ID	286
Atlas LCR40	208	CW-L	266
audio adapter	223, 295	CW-U	266
Audio Input	215	D	
audio jack	215	DAB broadcast	256
audio output	215	DAC	133
Audio output	264	damping factor	176
audio stream	276	dB	240
		dBm	241
B			
balun	279	DC-DC converter	220
Baofeng	301	decoupling capacitor	171
Bluetooth	70	Deleting a file	53
Boot Options	32	dhcpcd	40
BPSK	298	DHT11	116
built-in PWM generator	136	Digital-to-Analog Converter	133
Butterworth	176	direct sampling mode	257
		Direwolf	291
		Disk usage	59
		DominoEX	289

dpkg	55	GQRX	261
DroidPSK	299	GUI mode	74
DS1307	108		
DSP	223	H	
DVB-T+DAB+FM	256	HackRF One	278
DVB-T device	258	Ham Clock	299
DVB-T dongles	256	hamfax software	282
DXCC management	287	Hamlib	290
DX-Cluster	287	Hann	267
		Hardware AGC	265
E		HAT shield	21
echo	54	HDMI	215
Essex Ham	297	HDMI display	27
Etcher program	31	Head	55
eth0	41	Help	52, 67
external counter	209	Hermes Lite-2	280
		HF SSB transceiver	289
F		home directory	45
FFT Averaging	267	humidity	116
FFT plot	267		
FFT rate	267	I	
FFT Settings	265, 267	I2C based LCD	104
FFT Size	267	ID Now	286
File Manager	65	ifconfig	268
File permissions	47	inductive load	81
Filter shape	266	input power	234
Filter width	266	Input rate	264
fixed resistor	202	Interface Options	33
FLDIGI	289	internal timer	209
FM antenna	223	IQ balance	265
FM modulation	223	Itead Studio	279
FM transmitter	222		
Freq Correction	265	K	
Frequency Counter	209	Keypad	160
frequency modulation	223	Killing a process	59
Freq zoom	267	Klog	287
FSQ	289		
L			
G		latitude	125
GIMP Image Editing Benchmark	23	LCD	103
GNU Radio libraries	261	LCR meters	208
GPIO connector	78	LEA-6S	125
GPIO library	82	Libre Office Calc	64
Gpredict	288	Libre Office Writer	63
GPS	125	lines per minute	281
GPS Click board	125	Linpack Benchmark	22

LinPsK	298	output power	234
LNB LO	265		
logarithmic detector	234	P	
longitude	125	packet modem	291
Low-Pass Filter	176	Pandapter	267
ls	46	Pan dB	267
LSB resolution	133	PDL	227
		Peak Detect	267
M		Peak Hold	267
mapping	303	PiFmAdv	222
Matching a string	54	Pin numbering	82
MCP3201	236	PL011 UART	127
MCP4921	133	PoE	220
MFSK16	289	PoE HAT	220
microphone input	217	PortAudio	290
mini UART	127	potential divider	214
MIT App Inventor	244	power gain	241
MCP3002	198	Power over Ethernet	220
Microphone	218	power packs	219
MISO	134	pre-emphasis filter	225
mkdir	47, 51	ps	59
Morse code	182, 284	PSK31	289
Morse Code Exerciser	182	PTT	291
Morse code training	294	Putty	33
MOSFET	80	pwd	46
MOSI	134	PWM generator	222
MPX encoding	223	PWM module	137
Multiple Relays	247		
Multitasking	111	Q	
MySQL	305	Q factor	176
		QPSK	298
N		QSL management	287
nano	71	QSO management	287
Narrow FM	266	QSSTV	295
NB1	266	quarter-wave antenna	258
NB2	266	Quisk	280
NMEA sentence	125		
Nodepad++	33	R	
NooElect NESDR	279	RadioBerry-2	280
		Radio Config	276
		RadioStation Click	223
O			
Odyssey	280	Raspbian Buster	30
Ohmmeter	202	Raspbian repository	284
Olivia	289	raspi-config	32
On/Off Power Control	98	Raw I/Q	266
os.fork()	121	RDS/RDBS	223

real-time clock	108	sinewave	149
Recom R-78B5.0-2.0	220	slant correction	283
Redirecting the output	53	Slow Scan TV	295
RedPitaya	278	Smart HF bundle	278
reference voltage	133	Smartphone Projects	244
relays	81	SN74LV8154	209
Removing a directory	53	SoftRock	280
Renaming a file	53	Speaker	218
RF attenuator	240	speakers	215
RF harmonics	224	speaker-test	216
RF power	234	SPI bus	198
RF Power Meter	234	SPI Bus	134
RGB LED	87	SPI operational modes	134
rotary encoder	190	squarewave signal	135
RS-HFIQ	278	squelch	270
RTC	108	Squelch	266
RTL2832U	256	SSH	33
RTL-SDR	256	startx	32
rtl-sdr driver software	259	Static IP address	39
RTL-SDR server	270	Station Clock	103
rtl_test program	261	Station Weather	120
RTTY	289, 298	stereo transmitter	223
		ST Microelectronics	279
		Streaming audio	268
S		STV frequencies	295
Sallen-Key	176	sudo	55
sawtooth wave	142	Swap I/Q	265
SCL	128		
SCLK	134		
Screenshots	57	T	
scrot	57	Tail	55
SDA	128	Task Manager	67
SD Card Copier	66	T-connector	86
SDR bundle	279	Terminal	67
SdrMicron	280	Terminal Node Controller	291
SDR Sharp	271	Thonny	74
SDR transceiver	278	TightVNC	38
SDR transceiver softwar	280	TightVNC Viewer	39
serial console	128	time constant	204
Serial Peripheral Interface	134	timing pulses	210
Set Timer	286	TNC	291
shell script	273	top	58
ShinySDR	274	tracking	303
Shutdown	69	track satellites	288
Shutting down	59	transistor amplifier	171
Si4713-B30	223	triangular wave	144
SimpleFM	272	TRRS	27

Tuning to a frequency	261
TWCLOCK	284

U

UART	128
UDP protocol	268
UGREEN	217
upconverter	257, 279
USB Audio Device	264, 290
USB-C connector	21
USB-C port	219
USB sound adapter	217

V

Visual Gain	270
VLC media player	64
VNC	38
VNCViewer	74
voltage gain	240
voltmeter	198
volume icon	215

W

waterfall	267
Waveform Generator	153
Waveform generators	133
weather alerts	303
Weather faxes	281
weather forecasts	281
WEFAX	281
WFM	266
whoami	44
Wi-Fi	69
Wildcards	52
wlan0	41
Wouxun	301
wpm	190

X

Xastir	303
xcwcp	294
Xiegu G90	278
xpdf	55

Raspberry Pi for Radio Amateurs

Program and build RPi-based ham station utilities, tools, and instruments

Although much classical HF and mobile equipment is still in use by many amateurs, the use of computers and digital techniques has now become very popular among amateur radio operators. Nowadays, anyone can purchase a £40 **Raspberry Pi** computer and run almost all amateur radio software on the 'RPi' which is slightly bigger than the size of a credit card.

The **RTL-SDR** devices have become very popular among hams because of their very low cost (around £12) and rich features. A basic system may consist of a USB based RTL-SDR device (dongle) with a suitable antenna, an RPi computer, a USB-based external audio input-output adapter, and software installed on the Pi. With such a simple setup it is feasible to receive signals from around 24 MHz to over 1.7 GHz. With the addition of a low-cost upconverter device, an RTL-SDR can easily and effectively receive the HF bands.

This book is aimed at amateur radio enthusiasts, electronic engineering students, and anyone interested in learning to use the Raspberry Pi to build electronic projects. The book is suitable for the full range of beginners through old hands at ham radio. Step-by-step installation of the operating system is described with many details on the commonly used **Linux** commands. Some knowledge of the **Python** programming language is required to understand and modify the projects given in the book. Example projects developed in the book include a station clock, waveform generation, transistor amplifier design, active filter design, Morse code exerciser, frequency counter, RF meter, and more. The block diagram, circuit diagram, and complete Python program listings are given for each project, including the full description of the projects.

Besides wide coverage of RTL-SDR for amateur radio, the book also summarizes the installation and use instructions of the following ham radio programs and software tools you can run on your Raspberry Pi: **TWCLOCK**, **Klog**, **Gpredict**, **FLDIGI**, **DIRE WOLF**, **xcwcp**, **QSSTV**, **LinPsk**, **Ham Clock**, **CHIRP**, **xastir**, and **CQRLOG**.



Prof. Dr. 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.

Dogan has worked in many industrial organizations before he returned to academic life. He is the author of over 70 technical books and has published over 200 technical articles on electronics, microprocessors, microcontrollers, and related fields. He has been a fully licensed amateur radio operator (G7SCU) for several decades and has participated in many local ham radio events. Being an amateur radio operator himself, he understands well the needs of both starting and experienced radio amateurs.

Elektor International Media BV
www.elektor.com

ISBN 978-3-89576-404-2

9 783895 764042