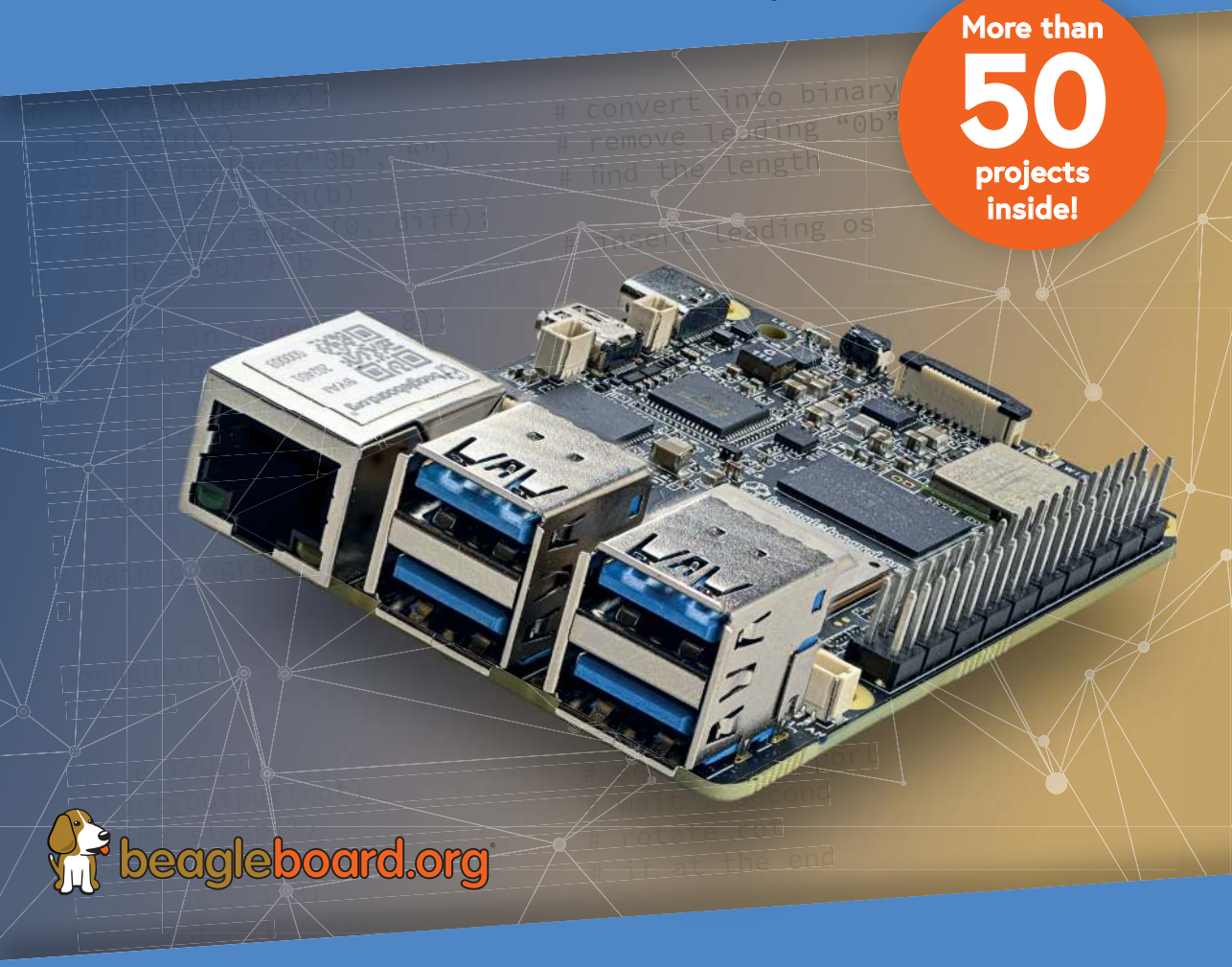
elektor books

The BeagleY-AI Handbook

A Practical Guide to AI, Python, and Hardware Projects

More than 50 projects inside!



beagleboard.org

Dogan Ibrahim
Ahmet Ibrahim

elektor
design > share > earn

The BeagleY-AI Handbook

A Practical Guide to AI, Python, and Hardware Projects

Dr. Dogan Ibrahim
Ahmet Ibrahim BSc, MSc

elektor
design › share › earn

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

Declaration

The author and publisher have made every effort to ensure the accuracy of the information contained in this book. They do not assume, or hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident, or any other cause.

ISBN 978-3-89576-656-5 Print
ISBN 978-3-89576-657-2 eBook

www.elektor.com
Editor: Glaucileine Vieira
Prepress Production: D-Vision, Julian van den Berg
Printers: Ipskamp, Enschede, The Netherlands

Contents

Chapter 1 • Introduction

1.1 The BeagleY-AI Single Board Computer (SBC)

BeagleY-AI is a low-cost, open-source, and powerful 64-bit quad-core single-board computer, equipped with a GPU, DSP, and vision/deep learning AI accelerators, designed for developers and makers. Developed by BeagleBoard.org Foundation, it is designed to meet the needs of both professional developers and educational environments. It is affordable, easy to use, and eliminating barriers to innovation. Developers can explore in-depth lessons or push practical applications to their limits without restrictions.

For more information about BeagleY-AI, including detailed specifications, documentation, and resources for getting started, visit the official website at

beagleboard.org

The board is controlled by the Debian Linux operating system, which includes a built-in development environment. This enables the seamless running of AI applications on a dedicated 4 TOPS co-processor, while simultaneously handling real-time I/O tasks with an 800 MHz microcontroller.

BeagleY-AI is based on the Texas Instruments AM67A Arm-based vision processor. It features a quad-core 64-bit Arm®Cortex®-A53 CPU subsystem at 1.4 GHz, dual general-purpose C7x DSP with Matrix Multiply Accelerator (MMA) capable of 4 TOPs each, Arm Cortex-R5 subsystem for low-latency I/O and control, a 50 GFLOP GPU, video and vision accelerators, and other specialized processing capabilities.

In this chapter, you will learn the basic features and hardware details of the BeagleY-AI board. A comparison is made with the popular Raspberry Pi 5 computer which has very similar board layout and features. In the remaining chapters of the book, you will learn how to install the operating system, how to access the BeagleY-AI board remotely, how to create Python programs to run on the board, and how to create software-only and hardware-based projects using the peripheral ports such as GPIO, SPI, UART, I^2C, and many others.

1.2 BeagleY-AI Features

The board has the following features:

Feature	Description
Processor	Texas Instruments AM67A, Quad 64-bit Arm® Cortex®-A53 @1.4 GHz, multiple cores including Arm/GPU processors, DSP, and vision/deep learning accelerators
RAM	4GB LPDDR4
Wi-Fi	Beagleboard BM3301, 802.11ax
Bluetooth	Bluetooth Low Energy 5.4 (BLE)

USB Ports	4x USB 3.0 ports (5Gbps shared) + USB 2.0 Type-C Port with Device-mode capability
Ethernet	Gigabit Ethernet, with PoE+ support (requires separate PoE HAT)
Camera/Display	2 x 4-lane MIPI camera connector (one connector muxed with DSI capability)
Display Output	1 x HDMI display, 1 x OLDI display, 1 x DSI MIPI Display
Real-time Clock (RTC)	Supports external coin-cell battery for power failure time retention
Debug UART	1 x 3-pin debug UART
Power	5 V/3 A DC power via USB-C
Power Button	On/Off included
PCIe Interface	PCI-Express® Gen3 x 1 interface for fast peripherals (requires separate M.2 HAT or other adapter)
Expansion Connector	40-pin header
Fan connector	1 x 4-pin fan connector, supports PWM control and fan speed measurement
Storage	microSD card slot with UHS-1 support
Tag Connect	1 x JTAG, 1 x External PMIC programming port

Table 1.1: BeagleY-AI features

The AM67A scalable processor family is based on the evolutionary Jacinto™ 7 architecture, targeted at Smart Vision Camera and General Compute applications. The AM67A processor family is designed for a broad set of cost-sensitive, high-performance computing applications in factory automation, building automation, human-machine interface, security systems, test and measurement, robotics, industrial PC, and other markets.

For more information about the AM67A processor, visit:

https://www.ti.com/product/AM67A

1.3 BeagleY-AI Board Component Layout

Front view

Figure 1.1 shows the components at the front of the board. Starting from the top-right-hand corner of the board and moving to the left we can see the following components:

- 4-pin External fan connector
- AM67A processor
- 40-pin expansion header
- 4 GB LPDDR4 memory
- BM3301 WiFi (802.1ax) + BLE (v5.4)
- BM3301 antenna
- PCIe port (Gen 3)
- Power On/Off button

- Bicolour LED
- Power management IC
- USB-C power and USB-2 port
- microHDMI monitor port
- 3-pin UART debug port
- 4-lane MIPI CSI connector
- 4-lane MIPI DSI/CSI connector
- Power over Ethernet port (PoE)
- Gigabit Ethernet port
- 2 x USB-3 (5 Gbps) ports
- 2 x USB-3 (5 Gbps) ports

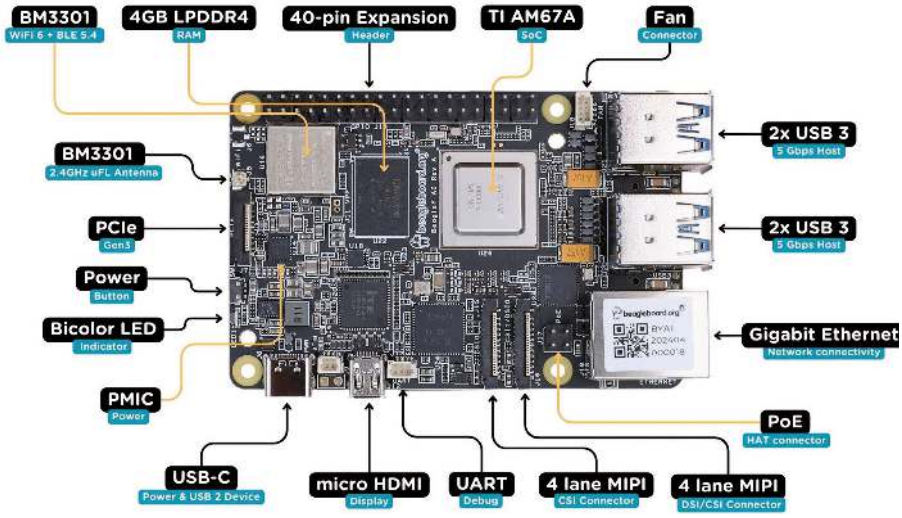


Figure 1.1 BeagleY-AI front view.

Back view

Figure 1.2 shows the components at the back of the board, which include the following:

- JTAG SoC debug connector
- JTAG PMIC debug connector
- OLDI display connector
- microSD card adapter

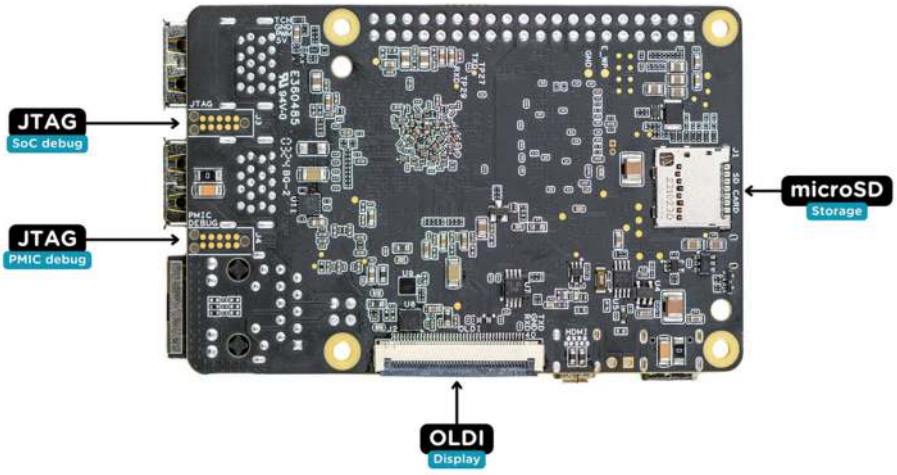


Figure 1.2 BeagleY-AI back view.

1.4 Comparison with the Raspberry Pi 5

Figure 1.3 shows the front views of the BeagleY-AI board and the Raspberry Pi 5 board. The two boards look identical in size and in most component layouts. Table 1.1 shows a comparison of the BeagleY-AI and the Raspberry Pi 5.

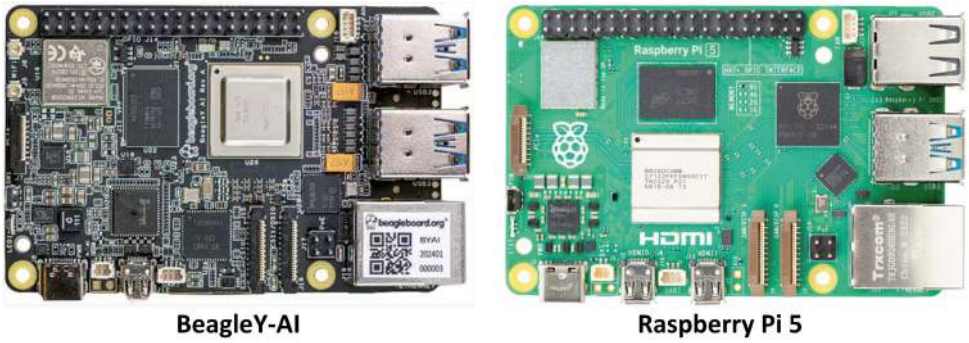


Figure 1.3 BeagleY-AI and the Raspberry Pi 5.

Feature	BeagleY-AI	Raspberry Pi 5
CPU	AM67A, Quad-core 64-bit, Cortex-A53 1.4GHz	BCM2712, Quad-core 64-bit Cortex-A76 2.4GHz
Memory	4GB	2GB, 4GB, 8GB
R5 core	YES	None
microHDMI	1	2
USB-3 ports (5Gbps)	4	2
USB-2 port (480Mbps)	1	2

Display support	3x (1x HDMI, 1x OLDI, 1x DSI)	2x HDMI
Graphics processing unit	IMG-BXS-4-64	Videocore VII
Dual C7x DSP with Matrix multiply accumulator (4 TOPS), NPU	1	
CSI/DSI ports	1	0
Video encode/decode	1	None
CSI port	2	2
Fan connector	1	1
UART connector	1	1
PCIe port	1	1
microSD card slot	1	1
40-pin GPIO header	1	1
Ethernet port (Gigabit)	1	1
Power button	1	1
WiFi + BLE	1	1

Table 1.2 Comparison of the BeagleY-AI and Raspberry Pi 5

1.5 Pros and Cons

Pros:

- **AI Performance**: The dual C7x DSPs and MMAs deliver up to 4 TOPS, making it ideal for deep learning tasks.
- **Connectivity**: With USB 3.0, Gigabit Ethernet, Wi-Fi 6, and Bluetooth 5.4, the board is well-equipped for various applications.
- **Expandability**: The PCIe Gen3 x1 connector and 40-pin GPIO header offer significant customization options.
- **Open-Source Hardware**: Users can access and modify all hardware design files, fostering innovation and adaptation.
- **Industrial-Grade Components**: The use of Texas Instruments hardware ensures reliability and long-term support, making it suitable for both development and deployment.

Cons:

- **CPU Performance**: The 1.4 GHz quad-core Cortex-A53 is underwhelming compared to newer SBCs.

- **RAM Limitations**: 4 GB of LPDDR4 RAM may not be sufficient for all applications.
- **Software Gaps**: Some AI features and tools are not fully supported, limiting the board's out-of-the-box capabilities.
- **Heat Management**: The board runs warm under load, and while it's fanless, some users may prefer active cooling.

Chapter 2 • Installing the Operating System

2.1 Overview

It is necessary to install a compatible operating system on a microSD card before the BeagleY-AI SBC board can be used. In this chapter, you will learn how to install the BeagleY-AI Debian operating system on a blank microSD card. Details on how to access the board remotely are also given in this chapter.

2.2 The Installation of the Operating System

Before installing the operating system, make sure you have the following:

- 5 V 3 A power supply
- 32 GB microSD card
- Boot image (operating system software image)

Using the bb-imager

You can use the bb-imager to install the operating system on the SD card. The steps are as follows::

- Download and install the bb-imager for your operating system from the following link:

  https://beagley-ai.beagleboard.io/bb-imager/

- Click to start the bb-imager. You should see a screen similar to the one shown in Figure 2.1.

Figure 2.1 bb-imager screen.

- Select **BeagleY-AI** as the device (Figure 2.2)

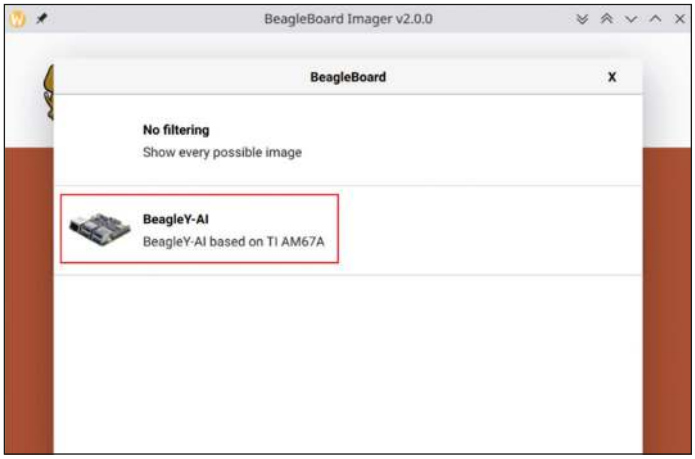


Figure 2.2 Enter the details.

- Choose the operating system as **BeagleY-AI Debian XFCE (Recommended)** as shown in Figure 2.3.

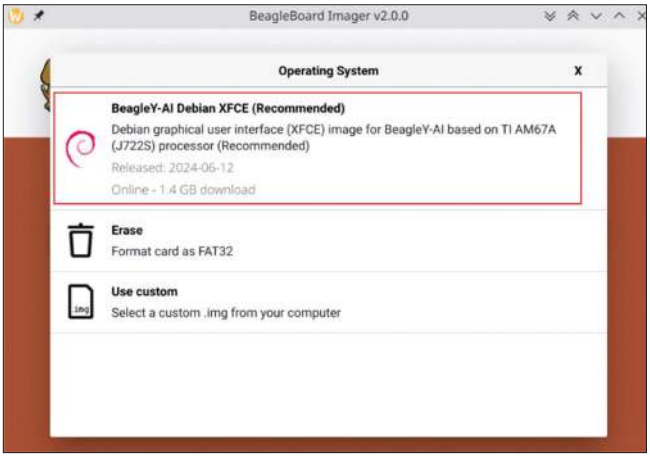


Figure 2.3 Choose the operating system.

- Choose your SD card storage and click **NEXT**

- Click **EDIT SETTINGS** and enter your chosen username, password, Wi-Fi SSID, Wi-Fi password, and time zone (Figure 2.4)

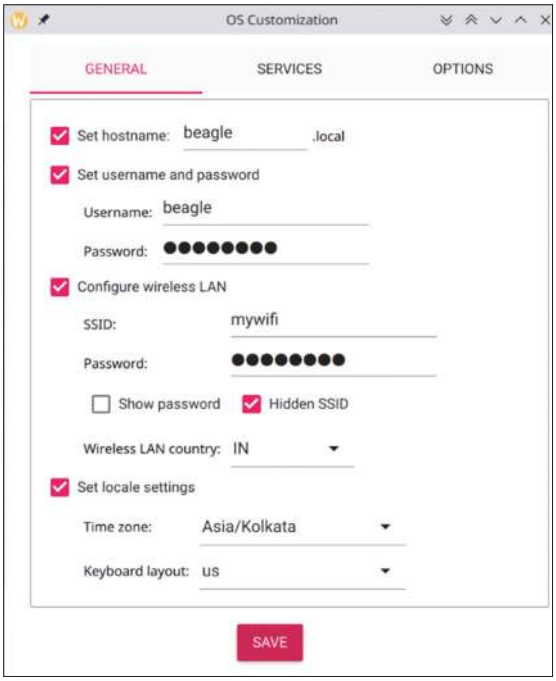


Figure 2.4 Edit the settings.

- Click **SERVICES** and make sure that the **Enable SSH** and **Use password authentication** are checked.
- Click **SAVE**, and then click **YES** on the screen **Would you like to apply OS customization settings?**
- Click **YES** to confirm that all existing data will be deleted on the SD card and to continue writing the operating system image on the SD card. Wait until the writing and the verification processes are complete.
- Remove the microSD card adapter from the PC and insert the microSD card into the slot on your BeagleY-AI as in Figure 2.5.
- Connect a monitor to the micro HDMI port of your BeagleY-AI board.
- Connect a keyboard and mouse to the USB-3 ports.
- Connect 5 V 3 A power supply to the USB-C power port of the BeagleY-AI.
- Figure 2.11 shows a typical setup with a monitor.

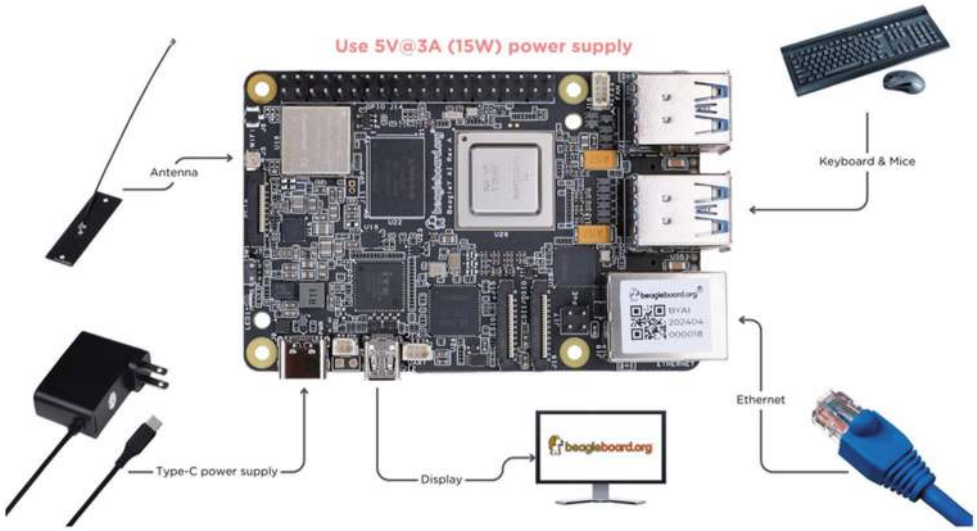


Figure 2.5 A typical setup (BeagleBoard.org).

- After a while you should see the green LED heartbeat and the GUI desktop displayed as shown in Figure 2.6. Please note, it may take several minutes.

2.3 Connection to a Wi-Fi

Follow these steps to connect to a Wi-Fi network:

- Click the wireless icon at the top right-hand side of the screen.
- A list of Wi-Fi networks will be displayed.
- Click **Connect** to connect to your network and enter your password.
- Click **Submit** (Figure 2.6).

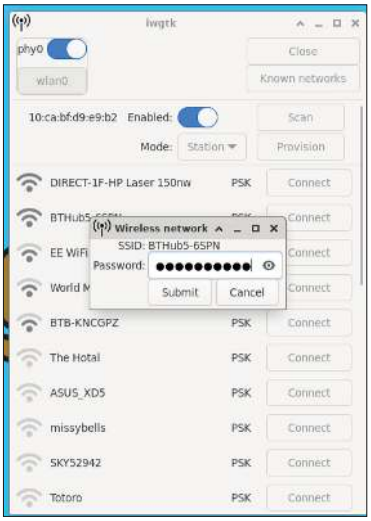


Figure 2.6 Click Submit.

- After a short wait, your BeagleY-AI will connect to your Wi-Fi. Click **Close** to exit the window. You should see the Wi-Fi icon change color to green, indicating a successful connection.

You can display the IP address of your connection as follows:

- Click **Applications**, then **Terminal Emulator.**
- In the terminal, enter the following command:

  ```
  sudo ifconfig
  ```

- You should see your IP address displayed under wlan0. In the author's setup, the IP address was 192.168.1.127 (see Figure 2.7).

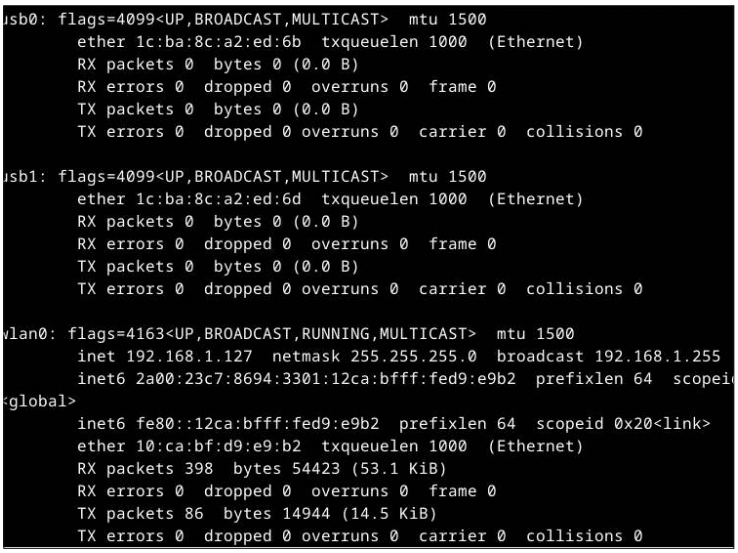


Figure 2.7 Command ifconfig (part of the display is shown).

2.4 Accessing Your BeagleY-AI Console from Your PC – The PuTTY Program

In many applications, you may want to access your BeagleY-AI from your PC over the Wi-Fi link. This can be done using a terminal emulator program on your PC. The author uses the popular PuTTY for this purpose. You can download PuTTY from the following website:

https://www.putty.org

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

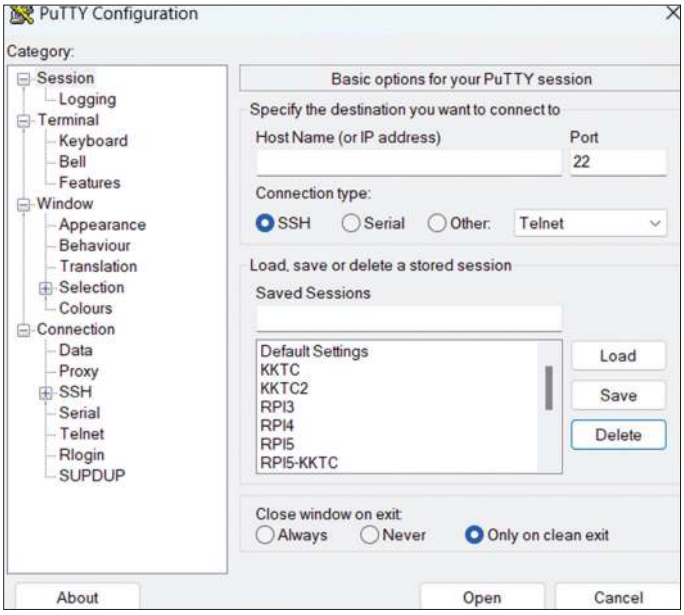


Figure 2.8 Putty startup screen.

- Make sure that the Connection type is SSH and enter the IP address of your BeagleY-AI. You can obtain the IP address by entering the command **ifconfig** as shown earlier.

- Click **Open** in PuTTY after entering the IP address and selecting **SSH.**

- The first time you run PuTTY, you may get a security message. Click **Yes** to accept this security alert.

- You will then be prompted to enter the BeagleY-AI username and password (these were entered in the **sysconf.txt** file during installation of the operating system). You can now enter all Console-based commands through your PC. Figure 2.9 shows the PuTTY screen with default screen settings.

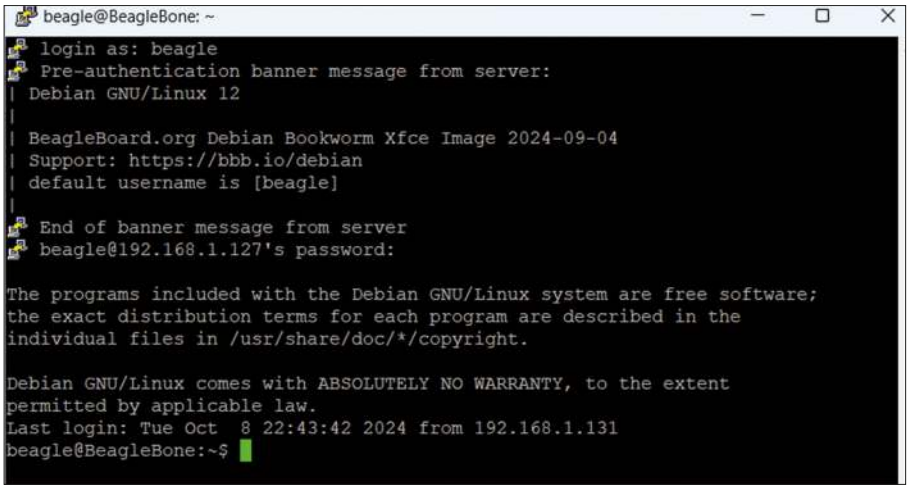


Figure 2.9 PuTTY screen with default settings.

- To change your password, enter the following command:

  passwd

- To restart the BeagleY-AI enter the following command:

  sudo reboot

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

  sudo shutdown –h now

2.4.1 Configuring PuTTY

By default, the **PuTTY** screen background is black with white foreground characters. The author prefers to have a white background with black foreground characters, and the font size set to 12 points in bold. It is recommended that you save your settings so that they are available the next time you use PuTTY. Follow these steps to configure PuTTY with the desired settings:

- Restart PuTTY.
- Select **SSH** and enter the Raspberry Pi IP address.
- Click **Colours** under **Window.**
- Set the **Default Foreground** and **Default Bold Foreground** colors to black (Red:0, Green:0, Blue:0).

- Set the **Default Background** and **Default Bold Background** to white (Red:255, Green:255, Blue:255).
- Set the **Cursor Text** and **Cursor 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**, give the session a name (e.g., MyZero), and click **Save**.
- Click **Open** to open the **PuTTY** session with the saved configuration.
- Next time you re-start the **PuTTY**, select the saved session and click **Load,** followed by **Open**, to start a session with the saved configuration.

Figure 2.10 shows the PuTTY screen with black bold characters on a white background. In this example, the PuTTY session was named as beagle.

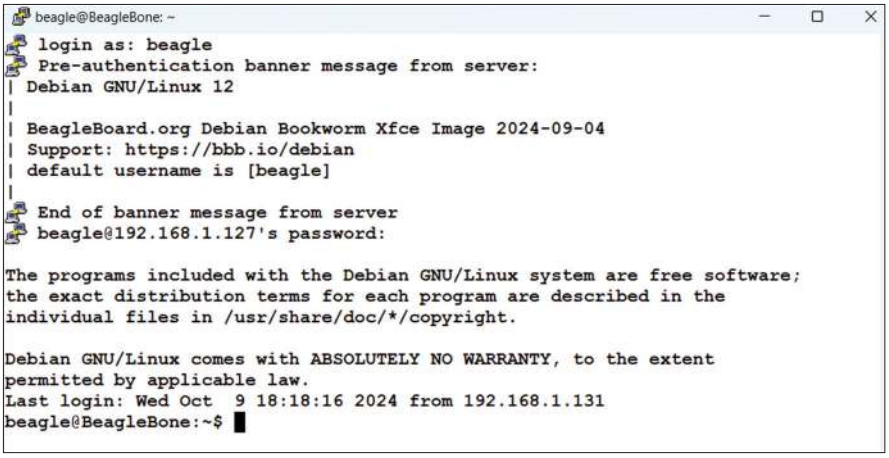


Figure 2.10 Putty screen with white background and black characters.

2.5 BeagleY-AI CPU Temperature

Without a heatsink, the Beagle-Y-AI typically heats up to about 58 - 60°C when idle. With 4 cores running in a complex operation, the CPU temperature can reach nearly 70°C. It is recommended to use a heatsink or an active cooler (such as the Raspberry Pi 5 active cooler) to help lower the device temperature, particularly during CPU-intensive tasks.

The CPU temperature can be displayed by entering the following command. As shown in the example below, the temperature is in milli-Celsius. In this case, the CPU temperature was measured shortly after the board was started, and it was 48.819ºC:

```
beagle@beagle:~ $ cat /sys/devices/virtual/thermal/thermal_zone
[0-2]/temp

47697
48146
48819
beagle@beagle:~ $
```

Chapter 3 • Using the Console Commands

3.1 Overview

BeagleY-AI is based on a version of the Linux operating system, one of the most popular operating systems in use today. Linux shares similarities with other operating systems, such as Windows and UNIX, and is an open-source system based on UNIX, 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 others.

Linux commands are text-based. In this chapter, you will be looking at some of the useful Linux commands and see how you can manage your BeagleY-AI using these commands.

The console commands can either be entered using the Putty terminal emulator, as described in the previous chapter, or they can be entered using the Terminal Emulator application in GUI Desktop.

3.2 The Command Prompt

Assuming your username is **beagle**, after you log in to BeagleY-AI, you will see the following prompt displayed where the system waits for you to enter a command:

```
beagle@beagle: ~$
```

Here, the **~** character indicates that you are currently in your default directory.

3.3 Useful Console Commands

In this section, you will be learning some of the useful Console commands, with examples provided for each command. **In this chapter, commands entered by the user are shown in bold for clarity**. Also, it is important to remind you that all commands must be terminated by the Enter key.

3.3.1 System and user information

These commands are useful as they provide information about the system. The command **cat /proc/cpuinfo** displays information about the processor (the command **cat** displays the contents of a file, and in this example, it shows the contents of the **/proc/cpuinfo** file). Figure 3.1 shows an example display, where only part of the display is shown here.

```
beagle@beagle:~$ cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 400.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 1
BogoMIPS        : 400.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 2
BogoMIPS        : 400.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4

processor       : 3
```

Figure 3.1 Command: **cat /proc/cpuinfo** *(part of the display is shown).*

The command **uname –s** displays the operating system kernel name, which is Linux. The command **uname –a** displays complete detailed information about the kernel and the operating system. An example is shown in Figure 3.2.

```
beagle@beagle:~$ uname -a
Linux beagle 6.1.83-ti-arm64-r63 #1bookworm SMP PREEMPT_DYNAMIC Wed Jul 10 23:0
:56 UTC 2024 aarch64 GNU/Linux
beagle@beagle:~$
```

Figure 3.2 Command: **uname – a**.

The command **cat /proc/meminfo** displays information about the memory on your BeagleY-AI, such as the total memory and free memory at the time the command is issued. Figure 3.3 shows an example, where only part of the display is shown here.

```
beagle@beagle:~$ cat /proc/meminfo
MemTotal:        3883876 kB
MemFree:         2566148 kB
MemAvailable:    3027204 kB
Buffers:           30508 kB
Cached:           540800 kB
SwapCached:            0 kB
Active:          1034332 kB
Inactive:          96644 kB
Active(anon):     561736 kB
Inactive(anon):        0 kB
Active(file):     472596 kB
Inactive(file):    96644 kB
Unevictable:          64 kB
Mlocked:              64 kB
SwapTotal:       4194300 kB
SwapFree:        4194300 kB
Zswap:                 0 kB
Zswapped:              0 kB
Dirty:                 8 kB
Writeback:             0 kB
AnonPages:        544692 kB
Mapped:           293488 kB
Shmem:              2060 kB
KReclaimable:      47144 kB
Slab:              93992 kB
SReclaimable:      47144 kB
SUnreclaim:        46848 kB
```

Figure 3.3 Command: **cat /proc/meminfo** *(part of the display is shown).*

The command **whoami** displays the name of the current user. In this case, **beagle** is displayed as the current user.

A new user can be added to your BeagleY-AI using the command **useradd**. In the example in Figure 3.5, a user called **Jane** is added. A password for the new user can be added using the **passwd** command followed by the username. In Figure 3.4, the password for user Jane is set to **mypassword** (not displayed for security reasons). Notice that both the **useradd** and **passwd** commands are privileged and the keyword **sudo** must be entered before these commands. Notice that the **–m** option creates a home directory for the new user.

```
beagle@beagle:~$ sudo useradd -m Jane
beagle@beagle:~$ sudo passwd Jane
New password:
Retype new password:
passwd: password updated successfully
beagle@beagle:~$
```

Figure 3.4 Commands: **useradd** *and* **passwd***.*

You can log in to the new user account by specifying the username and password. You can type the command **exit** to log out from the new account.

The command **sudo apt-get upgrade** is used to upgrade all the software packages on the system.

3.3.2 Some useful commands

To display the default home directory, enter:

```
beagle@beagle: ~$ pwd
/home/beagle
beagle@beagle: ~$
```

To display the directory structure, enter the command **ls /** (Figure 3.5):

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

Figure 3.5 Files in the directory.

To show the subdirectories and files in your working directory, enter **ls** (Figure 3.6)

```
beagle@beagle:~$ ls
Desktop    Downloads  Pictures  Templates  led.py
Documents  Music      Public    Videos
beagle@beagle:~$
```

Figure 3.6 Files in the home directory.

Notice that the subdirectories are displayed in blue and the files in black.

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

To display the subdirectories and files in a single row (Figure 3.7).

```
beagle@beagle:~$ ls -1
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
led.py
beagle@beagle:~$
```

Figure 3.7 Files in a single row.

To display the file types, enter the command **ls –F**. Note that directories have a "/" after their names, and executable files have a "*" character after their names:

To list the filenames separated by commas, enter the command **ls –m**.

You can mix the arguments, as shown in Figure 3.8.

```
beagle@beagle:~$ ls -m -F
Desktop/, Documents/, Downloads/, Music/, Pictures/, Public/, Templates/,
Videos/, led.py
beagle@beagle:~$
```

Figure 3.8 Mixing the arguments.

Subdirectories are created using the command **mkdir** followed by the name of the subdirectory.

The command **find** is used to search for a file and outputs a list of all directories that contain the file. For example, the command **find / -name myfile.txt** searches the default folder for the file **myfile.txt.**

File Permissions

One of the important arguments used with the **ls** command is **-l** (lowercase letter l) which displays the file permissions (Figure 3.9), 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. The characters at the beginning of each line indicate the permissions. i.e., who is allowed to use or modify the file or the directory.

```
beagle@beagle:~$ ls -l
total 36
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Desktop
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Documents
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Downloads
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Music
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Pictures
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Public
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Templates
drwxr-xr-x 2 beagle beagle 4096 Sep  4 08:56 Videos
-rw-r--r-- 1 beagle beagle  119 Oct  9 22:52 led.py
beagle@beagle:~$
```

Figure 3.9 Command **ls -l***.*

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.**

In the example in Figure 3.9, the first word, **beagle**, shows who the user of the file (or directory) is, and the second word, **beagle**, shows the group name that owns the file. In this example, both the user and the group names are **beagle**.

The permissions can be analyzed by breaking down the characters into four chunks for: File type, User, Group, and 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; if no permission is assigned, then a "-" character is used.

As an example, considering the **Music** directory, where 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.

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. In this example, a file with the name **led.py** was created in the home directory for demonstration purposes. The **led.py** file has user read and write permissions.

We will be changing the permissions so that the user does not have read permission on this file:

beagle@beagle: ~$ **chmod u-r led.py**
beagle@beagle: ~$ **ls –lh**

The result is shown in Figure 3.10.

```
beagle@beagle:~$ chmod u-r led.py
beagle@beagle:~$
beagle@beagle:~$ ls -lh
total 36K
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Desktop
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Documents
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Downloads
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Music
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Pictures
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Public
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Templates
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Videos
--w-r--r-- 1 beagle beagle  119 Oct  9 22:52 led.py
beagle@beagle:~$
```

Figure 3.10 File permissions of **led.py**.

Notice that if you now try to display the contents of file **led.py** using the **cat** command, you will get an error message:

beagle@beagle: ~$ **cat led.py**
cat: led.py: Permission denied
beagle@beagle: ~$

All the permissions can be removed from a file by the following command (Figure 3.11):

pi@raspberrypi: ~$ **chmod a= led.py**

```
beagle@beagle:~$ chmod a= led.py
beagle@beagle:~$ ls -lh
total 36K
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Desktop
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Documents
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Downloads
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Music
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Pictures
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Public
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Templates
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Videos
---------- 1 beagle beagle  119 Oct  9 22:52 led.py
beagle@beagle:~$
```

Figure 3.11 Remove all permissions.

In the following example, **rwx** user permissions are given to file **led.py**:

```
beagle@beagle: ~$ chmod u+rwx led.py
```

Figure 3.12 shows the new permissions of file led.py.

```
beagle@beagle:~$ chmod u+rwx led.py
beagle@beagle:~$ ls -lh
total 36K
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Desktop
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Documents
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Downloads
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Music
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Pictures
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Public
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Templates
drwxr-xr-x 2 beagle beagle 4.0K Sep  4 08:56 Videos
-rwx------ 1 beagle beagle  119 Oct  9 22:52 led.py
beagle@beagle:~$
```

Figure 3.12 New permissions of file **led.py**.

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

```
beagle@beagle: ~$ cd /home/pi/Music
beagle@beagle: ~/Music $
```

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

```
beagle@beagle: ~/Music $ cd ..
beagle: ~$
```

To change your working directory to **Music**, you can also enter the command:

```
beagle@beagle: ~$ cd ~/Music
beagle@beagle: ~/Music $
```

To go back to the default working directory, you can enter:

```
beagle@beagle: ~/Music $ cd ~
beagle@beagle: ~$
```

To find out more information about a file, you can use the **file** command. For example:

```
beagle@beagle: ~$ file led.py
led.py: Python script,  ASCII text executable
beagle@beagle: ~$
```

The **–R** argument of the command **ls** lists all the files in all the subdirectories of the current working directory.

To display information on how to use a command, you can use the command **man**. Figure 3.13 shows an example to get help on using the **mkdir** command. Press **q** to exit from the man menu.

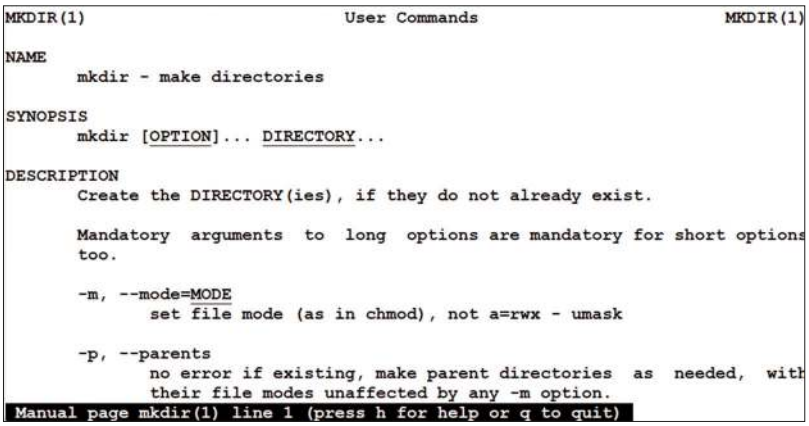


Figure 3.13 Help on mkdir command (part of the display is shown).

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

The command **less** 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:

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

Date and Time
To display the current date and time, the command **date** is used. For example:

```
beagle@beagle: ~$ date
Thu Oct 10 10:01:20 UTC 2024
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ cp led.py test.py
beagle@beagle: ~$
```

Wildcards
You 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 **.txt** are listed:

```
beagle@beagle: ~$ ls *.py
led.py   test.py
beagle@beagle: ~$
```

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**, or **t**, and with the **.py** extension:

```
beagle@beagle: ~$ ls [o-t]*.py
test.py
beagle@beagle: ~$
```

Renaming a File
You can rename a file using the **mv** command. In the example below, the name of the file **test.py** is changed to **test2.py**:

```
beagle@beagle: ~$ mv test.py test2.py
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ rm test2.py
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ rm –vi test2.py
rm: remove regular file 'test2.py'? y
removed 'test2.py'
beagle@beagle: ~$
```

Sorting a file
The command sort displays the contents of a file in ascending order. The general format of this command is:

sort <options> <filename>

Valid options are:

-u	Removes duplicates from the output
-r	Sorts the output in descending order
-o	Writes the sorted output to a file

Word count
The command **wc** <filename> displays the word count in a file.

File differences
The command **diff** <file1> <file2) displays the differences between two files, line by line.

Removing a Directory
A directory can be removed using the command **rmdir**:

```
beagle@beagle: ~$ rmdir Music
beagle@beagle: ~$
```

Re-directing 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 command **ls** to a file called **lstest.txt**:

```
beagle@beagle: ~$ ls > lstest.txt
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ cat led.py
This is a file
This is line 2
beagle@beagle: ~$
```

Using two greater-than signs **>>** adds to the end of a file.

Writing to the Screen or a File
The command **echo** 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:

```
beagle@beagle: ~$ echo $((5*6))
30
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ echo a line of text > lin.dat
beagle@beagle: ~$ cat lin.dat
a line of text
beagle@beagle: ~$
```

Matching a String
The command **grep** 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:

```
beagle@beagle: ~$ grep line lin.dat
a line of text
beagle@beagle: ~$
```

Head and Tail Commands
The command **head** can be used to display the first 10 lines of a file. The format of this command is as follows:

```
beagle@beagle: ~$ head led.py
……..
……..
beagle@beagle: ~$
```

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

```
beagle@beagle: ~$ tail led.py
……..
……..
beagle@beagle: ~$
```

The command **which** displays the location of an executable program. For example, the location of the Python program can be found as follows:

```
beagle@beagle: ~$ which python
/usr/bin/python
beagle@beagle: ~$
```

Super User Commands
Some of the commands are privileged and only authorized users can use them. Inserting the word **sudo** at the beginning of a command gives us the authority to use the command without having to log in as an authorized user.

What software is installed on my BeagleY-AI
To find out what software is installed on your system, enter the following command. You should get several pages of display (Figure 3.14).

```
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                          Version
+++-=============================================-==============================
ii  acl                                           2.3.1-3
ii  adduser                                       3.134
ii  adwaita-icon-theme                            43-1
ii  adwaita-qt:arm64                              1.4.2-3
ii  alsa-utils                                    1.2.8-1
ii  apparmor                                      3.0.8-3
ii  appstream                                     0.16.1-2
ii  apt                                           2.6.1
ii  apt-file                                      3.3
ii  apt-utils                                     2.6.1
ii  aspell                                        0.60.8-4+b1
ii  aspell-en                                     2020.12.07-0-1
ii  at-spi2-common                                2.46.0-5
ii  at-spi2-core                                  2.46.0-5
ii  avahi-daemon                                  0.8-10
lines 1-20
```

Figure 3.14 Software installed (part of the display is shown).

You can also find out if a certain software package is already installed on your system using the command **dpkg** with option **–s**.

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

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

3.3.3 Resource monitoring on BeagleY-AI

System monitoring is an important topic for managing the usage of your BeagleY-AI. One of the most useful system monitoring commands is the command **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 (only part of the display is shown, enter **q** to exit):

```
beagle@beagle: ~$ top
beagle@beagle: ~$
```

```
top - 10:16:50 up  1:24,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 188 total,   1 running, 187 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3792.8 total,   2284.5 free,    866.7 used,    797.6 buff/cache
MiB Swap:   4096.0 total,   4096.0 free,      0.0 used.   2926.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   3213 beagle    20   0    8788   4356   2468 R   0.7   0.1   0:00.27 top
   1570 beagle    20   0  416128  31936  23584 S   0.3   0.8   0:08.91 panel-8+
      1 root      20   0   21508  12248   8884 S   0.0   0.3   0:04.28 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.03 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+
      5 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 slub_fl+
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
     10 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_perc+
     11 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tas+
     12 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tas+
     13 root      20   0       0      0      0 I   0.0   0.0   0:00.00 rcu_tas+
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.07 ksoftir+
     15 root      20   0       0      0      0 I   0.0   0.0   0:00.74 rcu_pre+
```

Figure 3.15 Typical system resource display (part of the display is shown).

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

- There are a total of 188 processes in the system.
- Currently, only one process is running, 187 processes are sleeping, and 0 processes are stopped.
- The percentage CPU utilization is 0.1us for user applications (us).
- The percentage CPU utilization is 0.2 us for system applications (sy).
- There are no processes requiring more or less priority (ni).
- 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 3792 bytes, of which 866 bytes are in use, 2284 bytes are free, and 797 bytes are used by buffers/cache.
- Line 5 displays the swap space usage.

The process table gives the following information for all the processes loaded into 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: the 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 command **htop** is similar to the command **top**, except it has more features and is more user-friendly.

The command **ps** can be used to list all the processes used by the current user:

```
beagle@beagle: ~$ ps
PID     TTY     TIME          CMD
1842    pts/0   00:00:00      bash
3255    pts/0   00:00:00      ps
beagle@beagle
```

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) a process. A process can be killed by specifying its PID and using the following command:

beagle@beaglei: ~$ **kill -9 <PID>**

Disk (microSD card) usage

The disk-free command **df** can be used to display the disk usage statistics. An example is shown in Figure 3.16. Option **–h** displays in human-readable form.

```
beagle@beagle:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        25G  6.4G   18G  28% /
devtmpfs        1.5G     0  1.5G   0% /dev
tmpfs           1.9G     0  1.9G   0% /dev/shm
tmpfs           759M  1.5M  758M   1% /run
tmpfs           5.0M  8.0K  5.0M   1% /run/lock
/dev/mmcblk1p1  253M   57M  196M  23% /boot/firmware
tmpfs           380M   60K  380M   1% /run/user/1000
beagle@beagle:~$
```

Figure 3.16 Command: **df -h**

The command **free** shows how much memory is used and the amount of free memory.

3.3.4 Shutting Down

Although you can disconnect the power supply from your BeagleY-AI 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, make the file system safe, and then turn off the system safely:

beagle@beagle: ~$ **sudo halt**

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

beagle@beagle: ~$ **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:

beagle@beaglei: ~$ **shutdown –r <time> <message>**

For example, to shut down at 1:55 AM, enter:

beagle@beagle: ~$ sudo `shutdown –h 01:55`

or enter the following command to shut down immediately:

beagle@beagle: ~$ **sudo shutdown now**

The system will power off immediately!

3.3.5 Networking

Some useful networking commands are:

ifconfig: check the IP address of your Raspberry Pi.

iwconfig: check which network the BeagleY-AI is using. An example is shown in Figure 3.17. Here, the SSID of the Wi-Fi adapter used is BTHub5-6SPN.

```
beagle@beagle:~$ iwconfig
lo        no wireless extensions.

dummy0    no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:"BTHub5-6SPN"
          Mode:Managed  Frequency:2.437 GHz  Access Point: B8:BE:F4:AF:57:9C
          Bit Rate=65 Mb/s   Tx-Power=20 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off
          Link Quality=69/70  Signal level=-41 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:10  Invalid misc:16   Missed beacon:0

SoftAp0   IEEE 802.11  Mode:Master
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:off

usb0      no wireless extensions.
```

Figure 3.17 Command **iwconfig**

ping: used to test the availability of a network device. An example is shown in Figure 3.18.

```
beagle@beagle:~$ ping 192.168.1.127
PING 192.168.1.127 (192.168.1.127) 56(84) bytes of data.
64 bytes from 192.168.1.127: icmp_seq=1 ttl=64 time=0.273 ms
64 bytes from 192.168.1.127: icmp_seq=2 ttl=64 time=0.177 ms
64 bytes from 192.168.1.127: icmp_seq=3 ttl=64 time=0.179 ms
64 bytes from 192.168.1.127: icmp_seq=4 ttl=64 time=0.175 ms
64 bytes from 192.168.1.127: icmp_seq=5 ttl=64 time=0.163 ms
64 bytes from 192.168.1.127: icmp_seq=6 ttl=64 time=0.145 ms
^C
--- 192.168.1.127 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5112ms
rtt min/avg/max/mdev = 0.145/0.185/0.273/0.040 ms
```

Figure 3.18 Command **ping**

wget: this command is used to download a file from the web and saves the file in the current directory.

hostname – I: shows the IP address.

iwctl device list: lists the wireless devices attached (you should see wlan0 listed).

iwctl station wlan0 get-networks: lists of available Wi-Fi networks (Figure 3.19).

```
beagle@beagle:~$ iwctl station wlan0 get-networks
                           Available networks
--------------------------------------------------------------------
      Network name                      Security            Signal
--------------------------------------------------------------------
  >   BTHub5-6SPN                       psk                 ****
      DIRECT-1F-HP Laser 150nw          psk                 ****
      EE WiFi                           open                ****
      World Machine                     psk                 ****
      The Hotal                         psk                 ****
      BT-X5F9ZQ                         psk                 ****
      Hi Spencer                        psk                 ****
      TP-Link_46F1                      psk                 ****
      BT-HXAGHX                         psk                 ****
      ASUS_XD5                          psk                 ****

beagle@beagle:~$
```

Figure 3.19 List of available networks.

iwctl station wlan0 show: check wlan0 status

3.3.6 System information and other useful commands

The command **uname** is used to display system information. This command has the following options:

-a	Show all system information
-s	Display the kernel name
-n	Print the network node hostname

-r	Print the kernel release
-v	Print the kernel version number
-m	Print the system hardware name
-p	Print the processor type
-i	Print the hardware platform type
-o	Print the operating system type

If you have executed many commands and want to use some of them again but you cannot remember the command name, you can use the command **history**. An example is shown in Figure 3.20. To execute a command from the history, enter **!** followed by the command number.

```
beagle@beagle:~$ history
    1  nano x.y
    2  y
    3  ls
    4  python
    5  ifconfig
    6  sudo apt-get install tightvncserver
    7  tightvncserver
    8  hostname -I
    9  vncserver
   10  chmod +x home/debian/xstartup
   11  ls ~/vnc
   12  ls ~/.vnc
   13  ls ~/.vnc/xstartup
   14  chmod +x ~/.vnc/xstartup
   15  cat ~/.vnc/xstartup
   16  sudo apt-get install dbus-x11
   17  ls ~/.vnc/xstartup
   18  ls ~/.vnc/xstartup/
   19  cp ~/.vnc/xstartup ~/.vnc/xstartup2
```

Figure 3.20 The history command.

The command **clear** is also useful and it is used to clear the screen.

To install a package, use the command: **`sudo apt install <package_name>`**

The **&** operator allows you to run any command in the background so that you can use the terminal for other tasks. This operator must be added to the end of a command.

The **&&** operator allows you to run two or more commands at the same time. For example: command1 **&&** command2

Chapter 4 • GUI Desktop Applications

4.1 Overview

In this chapter, you will learn how to use some of the important GUI Desktop applications. It is assumed that you have a monitor, a keyboard, and a mouse connected to your BeagleY-AI.

4.2 The GUI Desktop

Figure 4.1 shows the GUI Desktop. At the top left corner, there is the Applications icon. On the left side of the screen, we have icons for Trash, File System, and Home. At the top right-hand side, we have the Wi-Fi icon, Speaker icon, date and time, and the computer name. At the bottom of the screen, we have several icons for quick access, such as the Terminal Emulator, the File Manager, the Web Browser, the Application Finder, and the Home directory icon.

Figure 4.1 The GUI Desktop.

4.2.1 Applications Menu

Figure 4.2 shows the items under the Applications Menu.

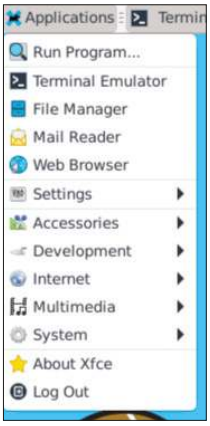


Figure 4.2 Items under the Applications Menu.

Terminal Emulator

This application is used to enter console commands to your BeagleY-AI. The window is similar to the window opened when using the Putty. Figure 4.4 shows the Terminal Emulator application window.

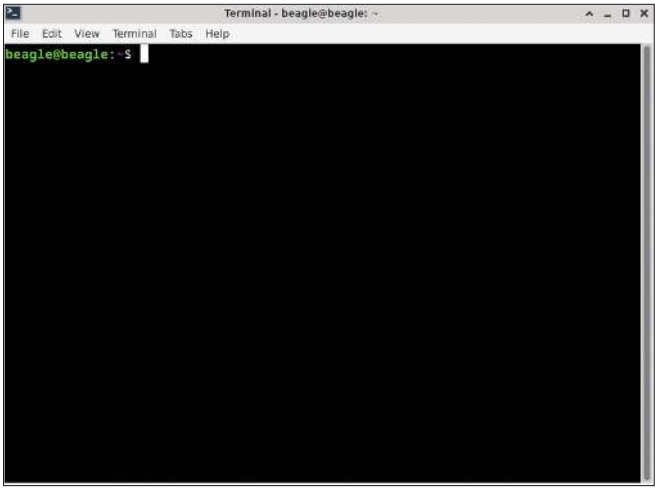


Figure 4.4 Terminal Emulator application window.

File Manager

This is a graphical file manager application (Figure 4.5). You can open files and folders, delete files, create folders, zoom in and out, and many more file processing options.

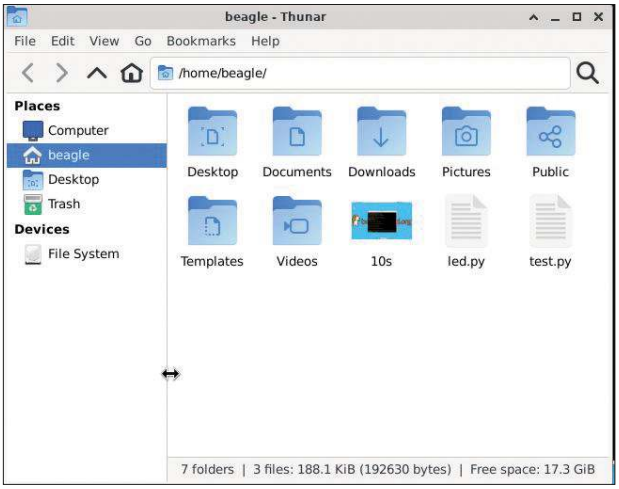


Figure 4.5 File Manager application window.

Mail Reader
This application can be used to read your mails.

Web Browser
Click on this application to start a Web Browser.

Settings
This menu item includes many applications (see Figure 4.6). With this menu, you can change the display settings, configure color profiles, configure the keyboard, configure the power manager, set up date and time, set up users and groups, and many more applications. Some important applications are described below:

Desktop: use this application to change the wallpaper, change the color of the desktop, change the icon size, etc. Figure 4.7 shows the Desktop application window.

Display: use this application to configure the display settings, such as the resolution, scale, refresh rate, rotation, etc. Figure 4.8 shows the Display application window.

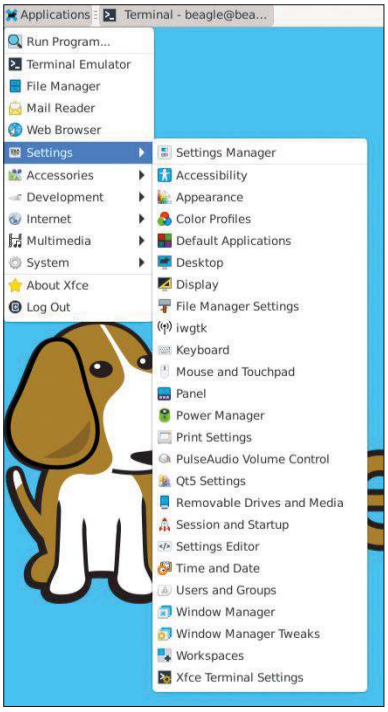


Figure 4.6 Settings menu applications.

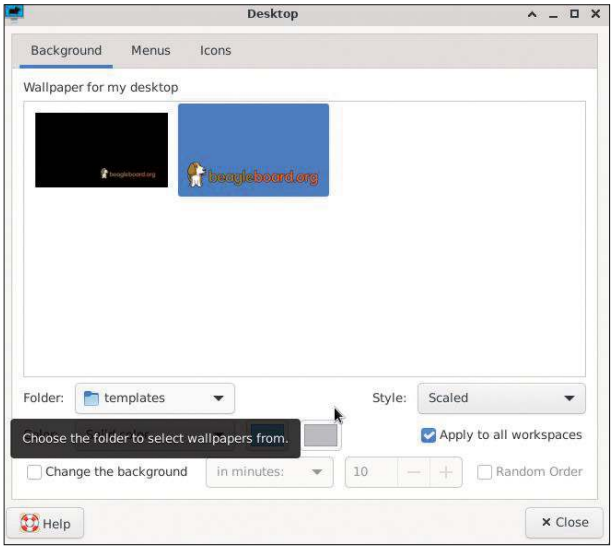


Figure 4.7 Desktop application window.

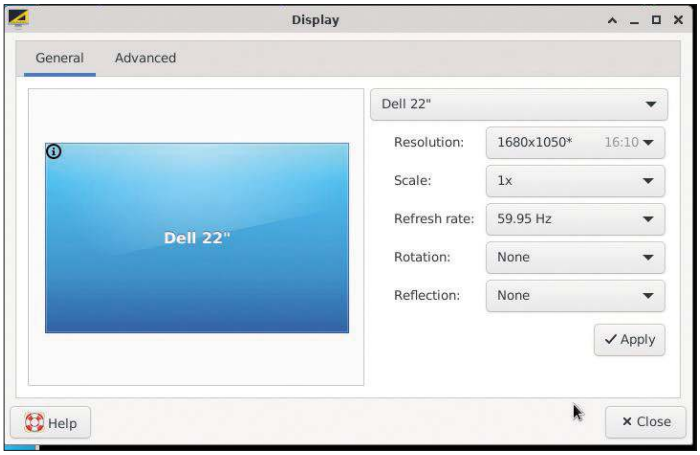


Figure 4.8 Display application window.

iwgtk: this is the Wi-Fi configuration application where all available networks are listed as shown in Figure 4.9. You can see from this figure that the author's BeagleY-AI is connected to a network called BTHub5-6SPN.

Figure 4.9 Available Wi-Fi networks.

Keyboard: this is a very useful application that enables you to change the keyboard settings, such as the key repeat speed, cursor type, define keyboard shortcuts, and keyboard layout. Figure 4.10 shows the Keyboard application window.

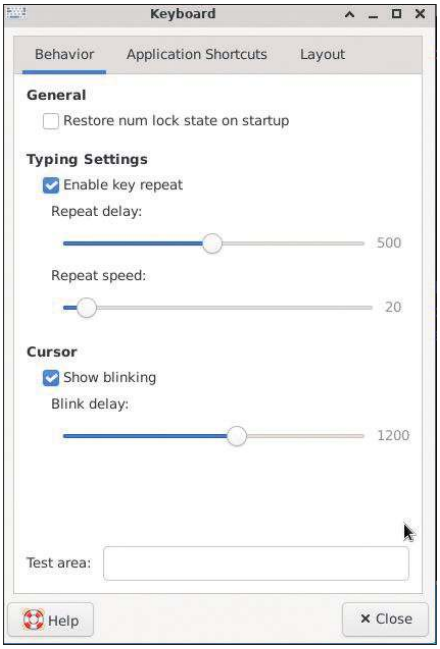


Figure 4.10 Keyboard application window.

Mouse and Touchpad: this application enables you to configure the mouse by specifying the mouse type, mouse button configuration (left-handed or right-handed), mouse pointer speed, mouse double-click time, etc.

Power Manager: with this application, you can configure the power button options, system sleep mode (suspend or hibernate), display blanking after a given time, display brightness control, and security settings. Figure 4.11 shows the Power Manager application window.

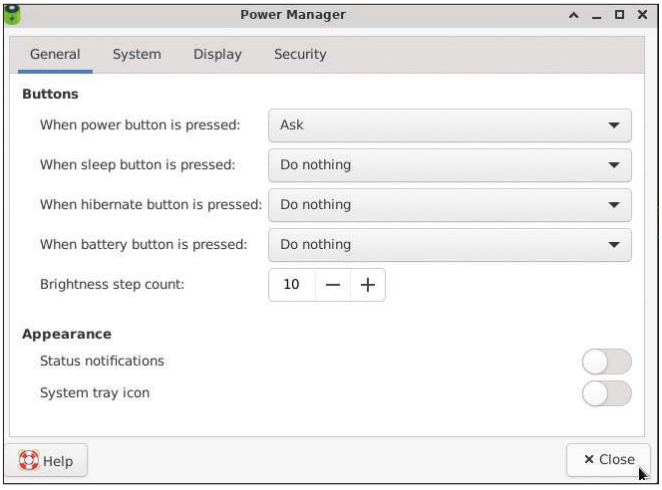


Figure 4.11 Power Manager application window.

Print Settings: with this application, you can configure a printer for your BeagleY-AI.

PulseAudio Volume Control: this application enables you to set the volume control, configure recording and playback, specify input device, etc.

Removable Drives and Media: use this application to mount removable devices, configure CDs and DVDs, play CDs and DVDs, configure digital camera inputs, and configure to run a program when a keyboard, mouse, or tablet is connected. Figure 4.12 shows the application window.

Figure 4.12 Application window.

Session and Startup: this application is used to configure login and logout sessions, auto-start application configuration, current and saved sessions, and manage remote applications, etc.

Settings Editor: use this application to configure various display and keyboard options.

Time and Date: configure the current date and time using this application. Figure 4.13 shows the application window.

Figure 4.13 Date and time application window.

Users Settings: this option allows you to add or delete users on the system. Figure 4.14 shows the application window. In this example, there are 3 users registered on the system: Jane, John, and Beagle User (beagle).

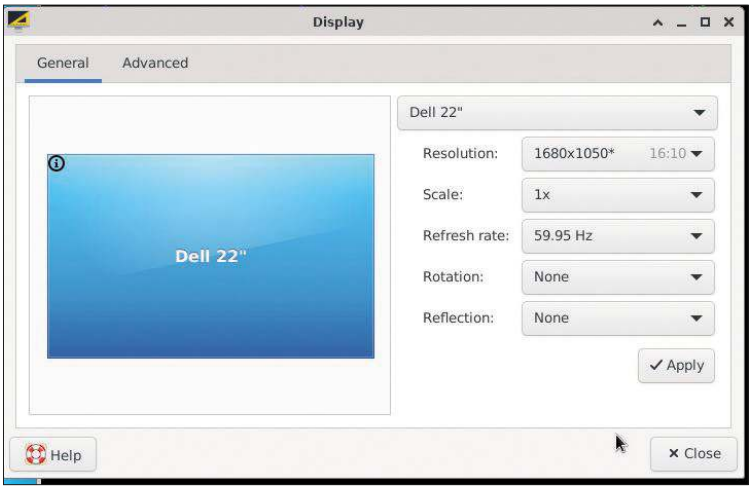


Figure 4.14 Users Settings application window.

Window Manager: use this application to configure window settings, such as the title font, title alignment, button layout, keyboard shortcuts, window focus, and various other window-based options.

Xfce Terminal Settings: with this application, you can configure the terminal preferences, display the toolbar and menubar, display borders around windows, colors, shortcut options, mouse pointer options, and many more.

Accessories

This menu item includes several applications (Figure 4.15). Some commonly used applications are described in this section.

Application Finder: use this application to locate an application in the system.

Barrier: use this application to share your mouse and keyboard between multiple computers on your desk.

Figure 4.15 Accessories menu options.

Bulk Rename: with this application, you can rename multiple files.

Screenshot: this application enables you to take screenshots. You can specify to capture either the entire screen, an active window, or a selected region. You can specify a delay before capturing the screen if required.

VIM: this is a text editor that you can use either to create program files or any other text files.

Development

Thonny: this is the Thonny Integrated Development Environment (IDE) and a text editor that can be used to develop Python programs. You can create a Python program, download it to BeagleY-AI, and then run it. We will cover how to use Thonny in a later chapter. Figure 4.16 shows the Thonny screen.

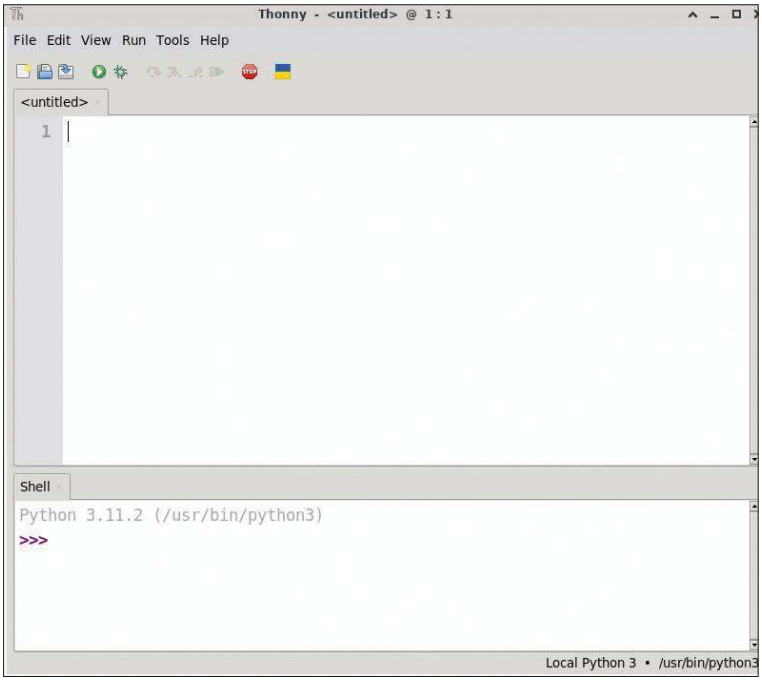


Figure 4.16 The Thonny screen.

Internet
This menu option includes the following applications. Use the Chrome or Firefox to access the internet:

- Chrome Web Browser
- Firefox ESR
- Firefox Nightly

Multimedia
PulseAudio Volume Control: as described earlier, this application allows you to configure the recording and playback of audio, as well as the audio input and output ports.

System
This menu has the applications shown in Figure 4.17. Some commonly used applications in this menu are described below.

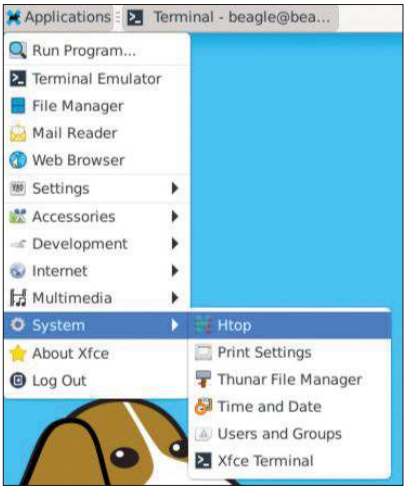


Figure 4.17 System applications window.

htop: this application displays the tasks running on the system. An example is shown in Figure 4.18.

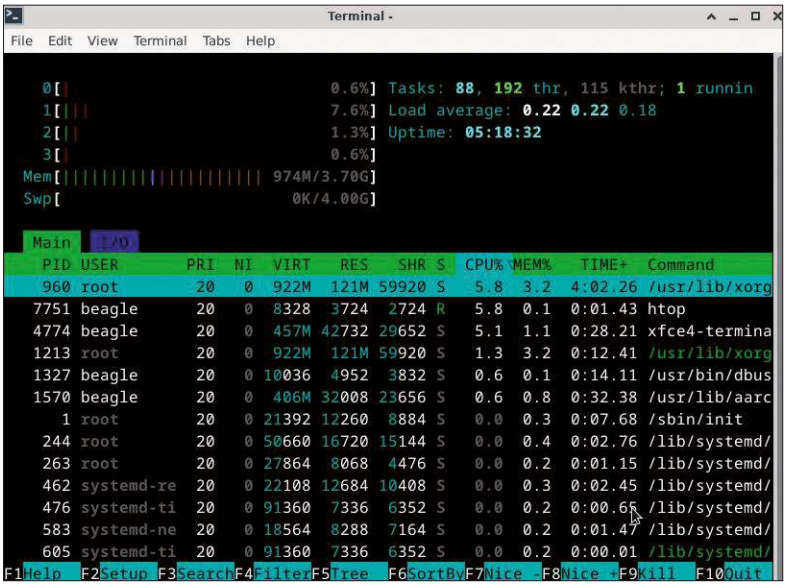


Figure 4.18 htop command display.

Print Settings: as described earlier, use this option to configure a printer for the system.

Log Out
This menu includes the following applications (Figure 4.19):

- Log Out
- Restart
- Shut Down
- Suspend
- Hibernate
- Hybrid Sleep
- Switch User

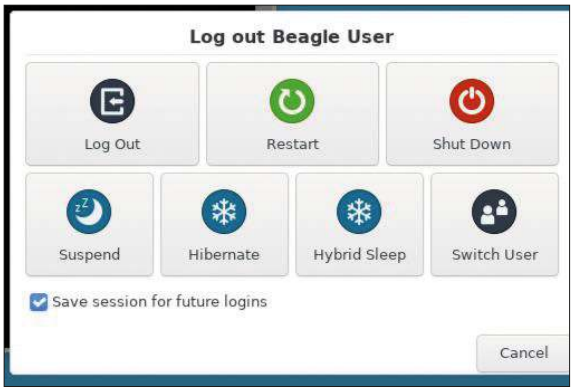


Figure 4.19 Log Out window.

Chapter 5 • Using a Text Editor in Console Mode

5.1 Overview

A text editor is used to create or modify the contents of a text file. There are many text editors available for the Linux operating system. Some popular ones are nano, vim, vi, Thonny, gedit, and many more. In this chapter, we will look at some of these text editors and show how to use them.

5.2 The nano Text Editor

Start the **nano** text editor by entering the word **nano**, followed by the filename you wish to create or modify. The example below shows how to create a new file called **first.txt**:

```
beagle@beagle: ~ $ nano first.txt
```

You should see the editor screen as in Figure 5.1. The name of the file to be edited is written at the top middle part of the screen. The message **New File** at the bottom of the screen shows that this is a newly created file. The shortcuts at the bottom of the screen are there to perform various editing functions. These shortcuts are accessed by pressing the Ctrl key together with another key. Some of the useful shortcuts are given below:

Ctrl+W: Search for a word

Ctrl+V: Move to the next page

Ctrl+Y: Move to the previous page

Ctrl+K: Cut the current line of text

Ctrl+R: Read file

Ctrl+U: Paste the text you previously cut

Ctrl+J: Justify

Ctrl+\: Search and replace text

Ctrl+C: Display the current column and row position

Ctrl+G: Get detailed help on using the nano

Ctrl+-: Go to specified line and column position

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

Ctrl+X: Exit nano

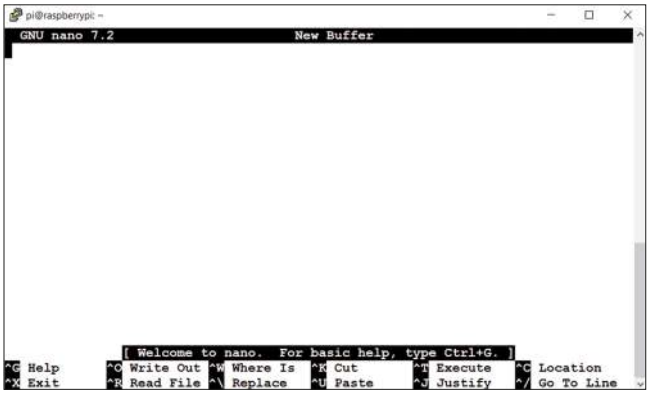


Figure 5.1 nano text editor screen.

Now, type the following text as shown in Figure 5.2:

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

Figure 5.2 Sample text.

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

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

Step 2: Look for the word **simple** by pressing **Ctrl+W** and then typing **simple** in the window opened at the bottom left-hand corner of the screen. Press the Enter key. The cursor will be positioned on the word **simple** (see Figure 5.3).

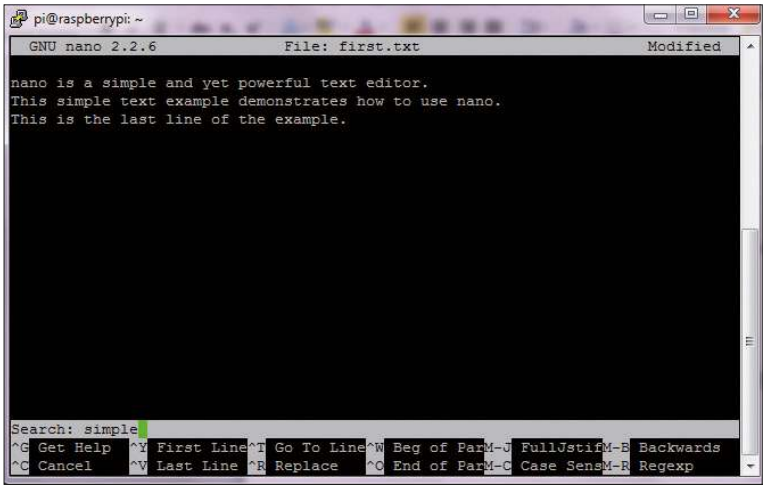


Figure 5.3 Searching word **simple.**

Step 3: Cut the first line by placing the cursor anywhere on the line and then pressing Ctrl+K. The first line will disappear as in Figure 5.4.

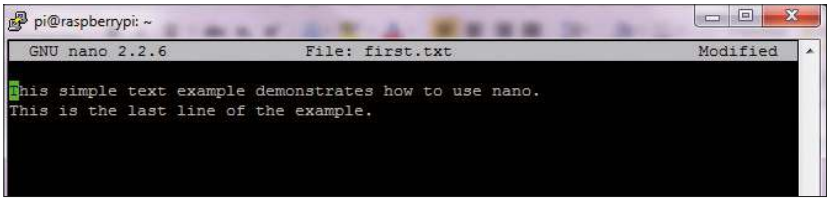


Figure 5.4 Cuttig the first line.

Step 4: Paste the cut line after the first line. Place the cursor on the second line and press **Ctrl+U** (see Figure 5.5).

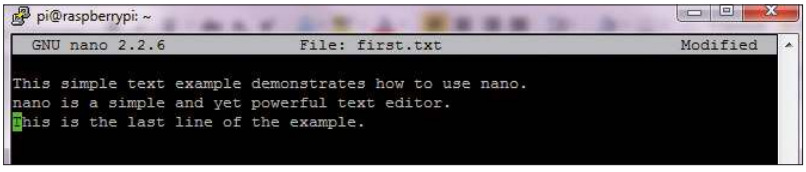


Figure 5.5 Paste the line cut previously.

Step 5: Place the cursor at the beginning of the word **simple** on the first row. Enter **Ctrl+C**. This word's row and column positions will be displayed at the bottom of the screen (see Figure 5.6).

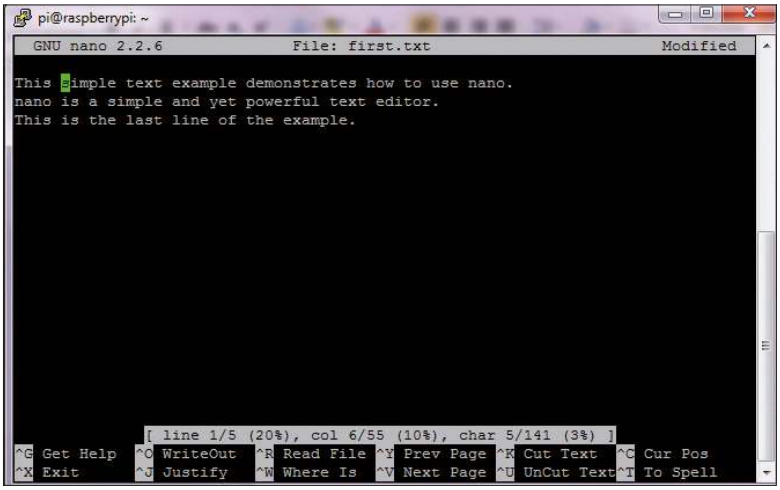


Figure 5.6 Displaying the row and column position of a word.

Step 6: Press **Ctrl+G** to display the help page as in Figure 5.7. Notice that the display is many pages long and you can jump to the next pages by pressing **Ctrl+Y** or to the previous pages by pressing **Ctrl+V**. Press **Ctrl+X** to exit the help page.

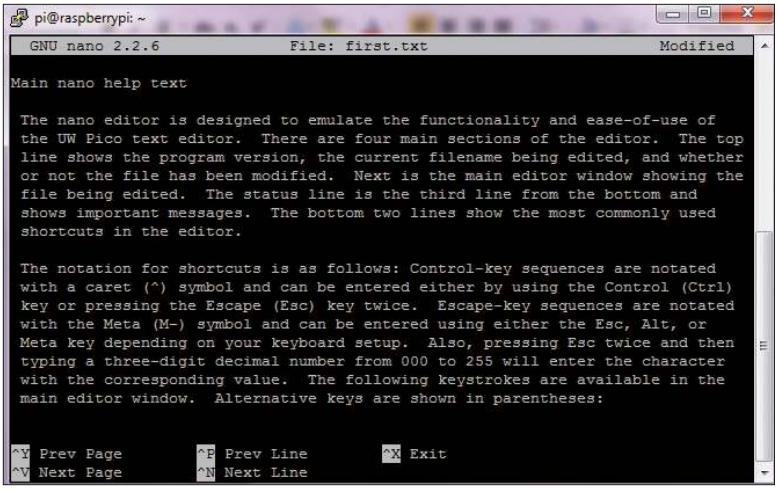


Figure 5.7 Displaying the help page.

Step 7: Press **Ctrl+-** and enter line and column numbers as 2 and 5, followed by the Enter key, to move the cursor to line 2, column 5 (see Figure 5.8).

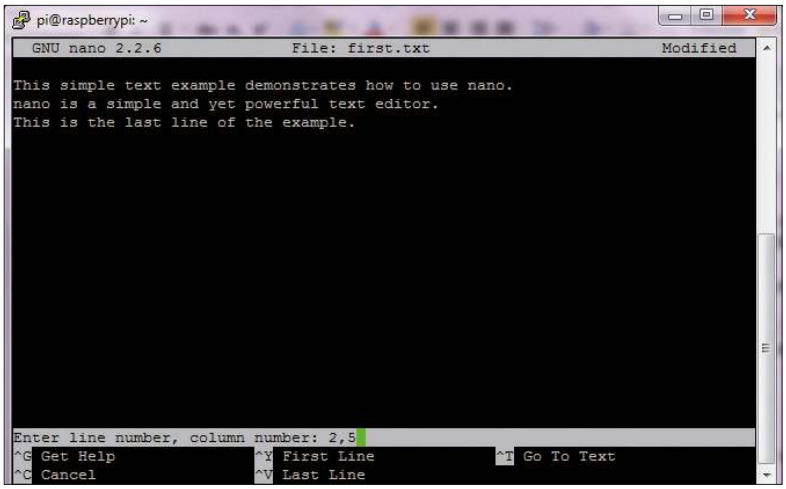


Figure 5.8 Moving to line 2, column 5.

Step 8: Replace the word **example** with the word **file**. Press **Ctrl+** and type the first word as **example** (see Figure 5.9). Press Enter and then type the replacement word as **file**. Press Enter and accept the change by typing y.

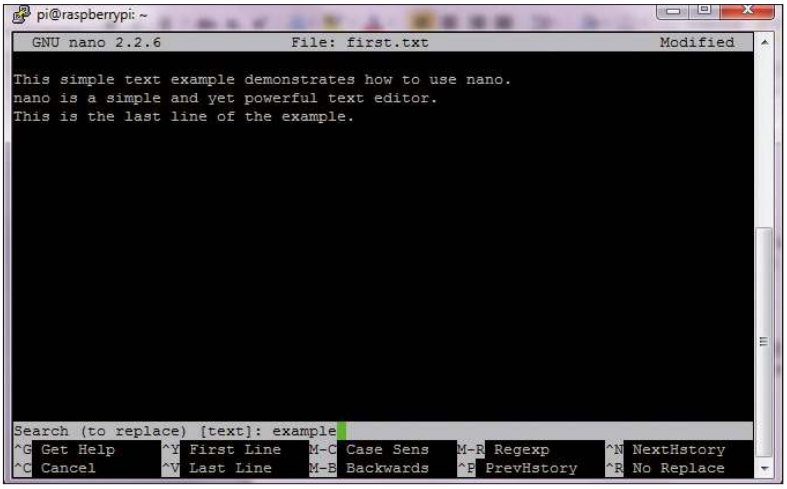


Figure 5.9 Replacing text.

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

Step 10: Display the contents of the file:

```
beagle@beagle: ~ $ cat first.txt
This simple text file demonstrates how to use nano.
```

```
Nano is a simple and yet powerful text editor
This is the last line of the example.

beagle@beagle: ~ $
```

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

5.3 The vi Text Editor

The **vi** text editor has been around for many years and it has been the standard Unix operating system default text editor. The **vi** editor is a fully featured, powerful text editor for doing many different tasks. The only problem with using **vi** is that it is not very user-friendly, and learning may take some time. In this section, we will be looking at the basic features of this editor and show how we can use it in simple editing applications.

Notice that you cannot use the keyboard arrow keys with the **vi** editor. Some of the useful **vi** editor commands are listed below:

ZZ	saves the changes and exits vi
:wq	saves the changes and exits vi
:q!	exits without saving the changes
h	moves the cursor left (backwards)
j	moves the cursor down
k	moves the cursor up
l	moves the cursor right (spacebar)
$	moves to the last column on the current line
o	moves the cursor to the first column on the current line
w	moves the cursor to the beginning of the next word
b	moves the cursor to the beginning of the previous word
H	moves the cursor to the top of the screen
M	moves the cursor to the middle of the screen
L	moves the cursor to the bottom of the screen
G	moves to the last line in the file
nG	moves to line n
r	replaces the character under the cursor with the next character typed
i	inserts before the cursor
a	appends after the cursor
A	appends at the end of the line
x	deletes the character under the cursor
dd	deletes the line under the cursor
dw	deletes the word under the cursor

/	searches for a word (forwards)
?	searches for a word (backwards)
:s	searches and replaces a word in the current line

Start the **vi** text editor by typing **vi** followed by the name of the file to be created or modified. In this example, it is assumed that a new file called **myfile.txt** is to be created:

```
beagle@beagle: ~ $ vi myfile.txt
```

You should see the **vi** text editor screen displayed as in Figure 5.10. The name of the file being edited is displayed at the bottom of the screen.

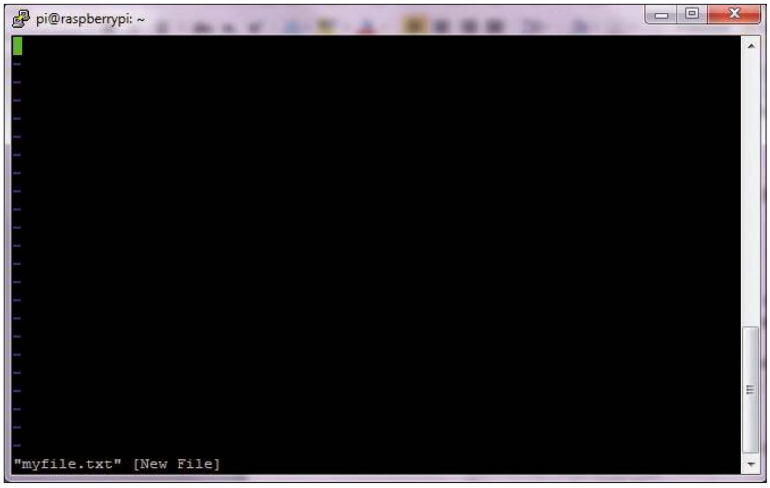


Figure 5.10 **vi** *text editor screen.*

The **vi** editor is different from most other text editors in that it is not possible to start typing inside the editor window. The steps for editing this file are given below:

Step 1: The **vi** editor has different modes and you must be in the insert mode to be able to write to the window. Press **i** to move to the insert mode. Then type in the following text (see Figure 5.11):

```
The vi text editor is a very powerful text editor.
But it is not easy to use this editor.
This exercise should help you understand the basic commands.
```

Figure 5.11 Entering the text.

Step 2: To come out of the insert mode, press the **ESC** key. To save the file, type characters **:w**. You can exit the editor after saving the changes by typing **:q**. Alternatively, you can type **ZZ** (note upper case) to save and exit. If you make changes to the file and attempt to quit without saving, you will get an error message. If you want to exit without saving the changes, simply type **:q!**

Step 3: Make sure you are in the command mode and type the character **/** followed by a word to search for this word in the text. For example, type **/editor** to search for the word **editor** (see Figure 5.12) in the text.

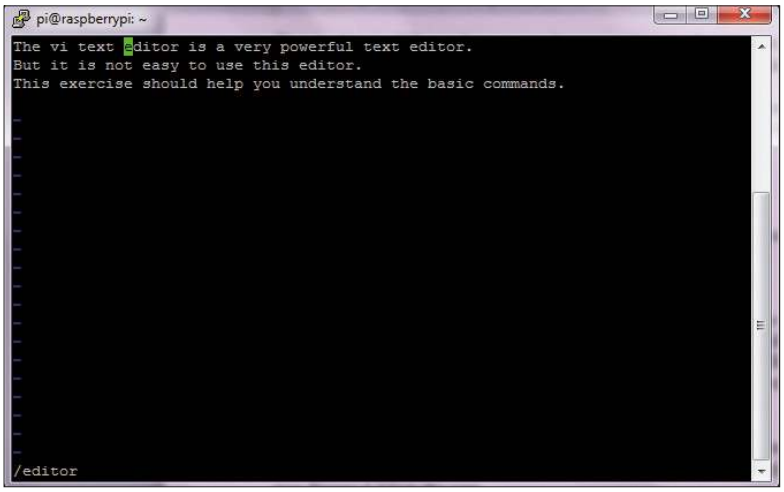


Figure 5.12 Searching for text.

Step 4: Insert the word **is** before the word **editor**. Type **i** followed by **is** and space and terminate insert mode by pressing the ESC key.

Step 5: Move the cursor right by pressing the **l** key. Similarly, move the cursor left by pressing the **h** key. Move the cursor down (to the second line) by pressing the **j** key.

Step 6: Search for the word **this** and delete it. Type **/this** followed by the Enter key. Type **dw** to delete the word.

Step 7: Delete the second line where the cursor is by typing **dd**

Step 8: Search for the word **help** and replace it with the word **guide**. Go to the line where the word help is. Type **/help**, then type **:s/help/guide/**

Step 9: You can search and replace a word in any line other than the current line. For this example, position the cursor on the first line. Change the word **basic** in the second line to **BASIC**. Type:

```
:1,2s/basic/BASIC/
```

Notice that you can specify the range of lines by separating them with a comma. In this example, the search starts from line 1 and terminates at line 2.

5.4 Using Thonny

Thonny is more than a simple text editor. It is an **I**ntegrated **D**evelopment **E**nvironment (IDE) that can be used to write programs and then upload them to your BeagleY-AI (see Figure 4.16)

5.4.1 The Thonny IDE

Start the Thonny IDE from the GUI Desktop under the **Development** menu.

The screen consists of two parts: the upper part is where you write your programs and the lower part is the shell where small interactive Python program codes can be written. This part is mainly used for testing small codes.

The upper part contains the following menu items:

File: click to create a new program (or a text file), open an existing program, save a program, or print a program.

Edit: click to cut, copy, paste, select, and find & replace text or characters in a file.

View: click to view files, heap, exceptions, program tree, stack, variables, program arguments, focus editor, and change the font size.

Run: click to configure the Python interpreter, run a program, debug a program, and send EOF/Soft reboot.

Tools: click to manage packages, open the system shell, open the Thonny folder, and manage plug-ins.

Help: click to display the help contents, version history, report problems, and About Thonny.

The Thonny IDE must be configured before it is used to write and upload programs to your BeagleY-AI. The details of this are given in a later chapter.

5.5 The gedit Text Editor

The gedit text editor is the default editor in some Linux systems, and it is easy to use while offering many options. To use gedit on BeagleY-AI, you need to install it first. Enter the following command to install gedit:

```
sudo apt install gedit
```

5.5.1 Using gedit

As an example, let us create a simple text file named test.dat, consisting of 3 lines. Enter the following command to start gedit:

```
gedit test.dat
```

A new screen will pop up and wait for you to type your text (or the program). Type the following lines (Figure 5.13):

```
this is line 1
this is line 2
this is line 3
```

You can use the three lines next to the Save button to perform the following:

- Save the file created
- Find text inside the file
- Find and replace text inside the file
- Go to a specified line
- Check spelling (in Tools)
- Set the language (in Tools)
- Document statistics (in Tools)
- Display line numbers (in Preferences)
- Display status bar (in Preferences)
- Text wrapping (in Preferences)
- Highlight text (in Preferences)
- Keyboard shortcuts
- Help

Click **Save** to save your text file.

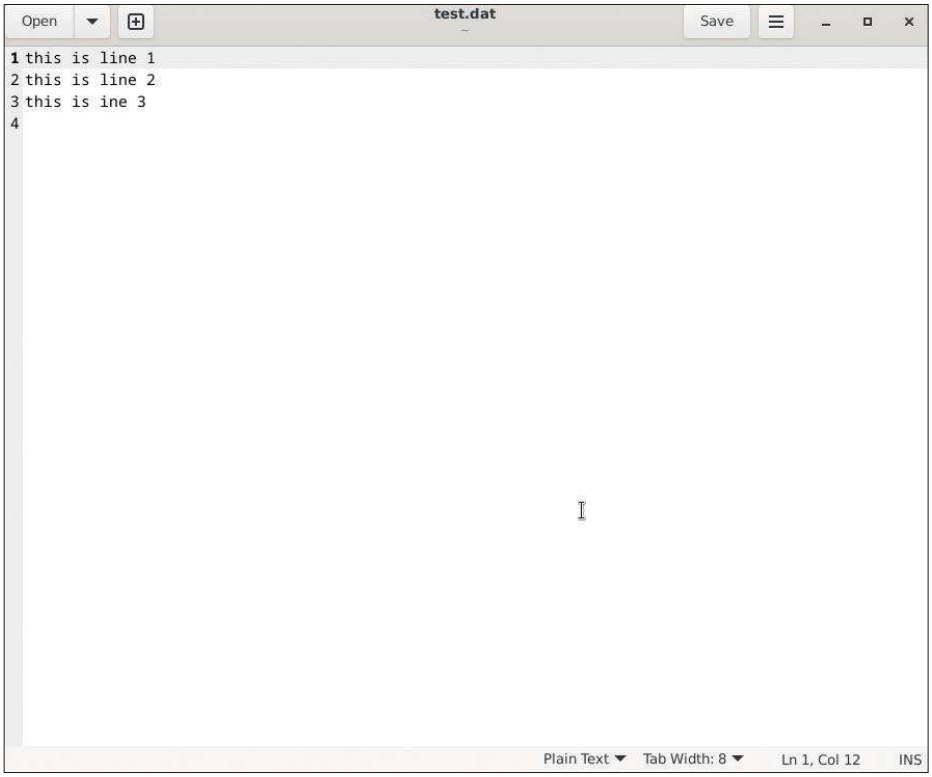


Figure 5.13 Text file using gedit.

Chapter 6 • Creating and Running a Python Program

6.1 Overview

Several Integrated Development Environments (IDEs) can be used to program the Beagle-Y AI. Visual Studio (VS) includes support for 36 programming languages, including C++, C#, BASIC, and several other others. It also provides open-source support for Python through the Python Development and Data Science workloads, as well as the free Python Tools for Visual Studio extension.

You will be programming your BeagleY-AI computer using the Python programming language. It is worthwhile to look at the creation and running of a simple Python program on your BeagleY-AI computer. In this Chapter, the message **Hello From BeagleY-AI** will be displayed on your PC screen.

As described below, there are three methods that you can create and run Python programs on your BeagleY-AI.

6.2 Method 1 – Interactively from Command Prompt in Console Mode

In this method, you will log in to your BeagleY-AI through a PC using SSH, or start the Terminal Emulator in GUI Desktop mode. Here, you will create and run the Python program interactively. This method is excellent for small programs. The steps are as follows:

- Log in to your BeagleY-AI using SSH or start the Terminal Emulator.
- At the command prompt, enter **python**. You should see the Python command mode, which is identified by three characters **>>>**
- Type the program:

```
print ("Hello From BeagleY-AI")
```

- The text will be displayed interactively on the screen, as shown in Figure 6.1. Note that, at the time of writing this book, the Python version was: 3.11.2.

```
beagle@beagle:~$ python
Python 3.11.2 (main, Aug 26 2024, 07:20:54) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from BeagleY-AI")
Hello from BeagleY-AI
>>>
```

Figure 6.1 Running a program interactively.

- Type **Ctrl+z** to exit from the program.

6.3 Method 2 – Create a Python File in Console Mode

In this method, you will log in to your BeagleY-AI using SSH as before, or start the Terminal Emulator in GUI Desktop mode and then create a Python file. A Python file is simply a text file with the extension **.py**. You can use a text editor, e.g., the **nano** text editor to create your file. In this example, a file called **hello.py** is created using the **nano** text editor. Figure

6.2 shows the contents of the file **hello.py**. This figure also shows how to run the file under Python. Notice that the program is run by entering the command:

python hello.py

```
beagle@beagle:~$ cat hello.py
print("Hello From BeagleY-AI")

beagle@beagle:~$ python hello.py
Hello From BeagleY-AI
beagle@beagle:~$
```

Figure 6.2 Creating and running a Python file.

6.4 Method 3 – Create a Python File in GUI Desktop Mode

In this method, you will be using the Thonny IDE to create and run your program. The Thonny IDE must be configured before it can be used to write and upload programs to your BeagleY-AI. Click the bottom-right corner of the screen to select your processor type and choose **Local Python 3**. You are now ready to write your program.

The steps are:

- Type the following code to the upper part of the screen:

```
print("Hello From BeagleY-AI")
```

- Click **File** -> **Save** and save with the name **hello.py**
- Click the **Run** icon (green menu button at the top) to run the program. The output of the program will be displayed at the bottom of the screen, as shown in Figure 6.3.

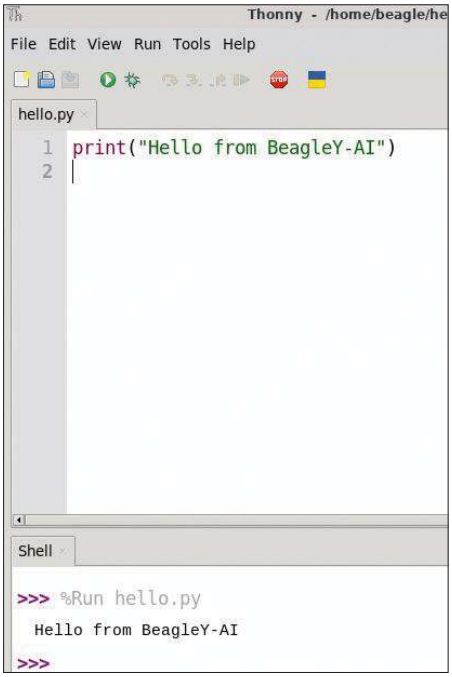


Figure 6.3 Run the program (part of the display is shown).

You can run small programs in interactive mode by entering them at the lower part of the screen called **Shell**. The results will be displayed immediately under the **shell** window.

6.5 Which Method?

The choice of a method depends upon the size and complexity of a program. Small programs can be run interactively without creating a program file. Larger programs can be created as Python files and then they can run either in the console mode or in Desktop GUI mode under the Thonny IDE. Running under the Thonny IDE has the advantage that code justification is corrected automatically as you write the code. In this book, the Thonny IDE is used for small programs, and the nano editor is used for larger programs to create the program files.

Chapter 7 • Python Programming and Simple Programs

7.1 Overview

Python is an interpreted, interactive, and object-oriented programming language. It was developed by Guido van Rossum in the 1980s at the National Institute for Mathematics and Computer Science in the Netherlands. It is derived from many other languages, including C, C++, Modula-3, Smalltalk, and Unix shell. The language is now maintained by a team at the Institute.

Python is interactive which means that you can issue a command and see the result immediately without having to compile the command. It is interpreted, which means no pre-compilation is required before it is run.

Python supports object-oriented techniques of programming. It is a beginner-friendly language, easy to learn, and maintain. Beginners can easily learn programming in a relatively short period of time. Python supports a large library of functions, which makes it very powerful. The language is portable, meaning that it can run on several different popular platforms.

In this and the next chapters, you will learn the details of the Python programming language on the BeagleY-AI computer and see how you can write programs using this language. Many example programs are given to show how electronic engineers can use Python to help them in their calculations.

7.2 Variable Names

Python variable names are case sensitive and can start with a letter **A** to **Z** or **a** to **z** or an underscore character **"_"**, followed by more letters or numbers 0 to 9. Some valid and invalid example variable names are given below:

SUM	-	valid
Sum	-	valid
SUm	-	valid
_total	-	valid
Cnt5	-	valid
8tot	-	invalid
%int	-	invalid
&xyz	-	invalid
My_Number	-	valid
@loop	-	invalid
_Account	-	valid

Note that variables **total**, **Total**, **TOTAL**, **ToTaL**, or **toTAL** are all different.

7.3 Reserved Words

There are some words which are reserved for use by the Python interpreter and thus cannot be used as variable names by programmers. A list of these reserved words is given below.

Notice that all the reserved words contain lowercase letters:

and	for	raise
assert	from	return
break	global	try
class	if	while
continue	import	with
def	in	yield
del	is	
elif	lambda	
else	not	
except	or	
exec	pass	
finally	print	

7.4 Comments

Comment lines in Python start with a hash sign **"#"**. All characters after the **#** sign are ignored by the Python interpreter. An example comment line is shown below:

```
# This is a comment line
```

Comments can also be inserted after a statement:

```
Sum = 0                    # Another comment
```

7.5 Line Continuation

The line continuation character "\" can be used to continue a statement on following lines. An example is shown below:

```
Sum = a +\
        b +\
        c
```

Which is equivalent to:

```
Sum = a + b + c
```

7.6 Blank Lines

A line containing no statements is ignored by the Python interpreter.

7.7 More Than One statement on a Line

It is permissible to have more than one statement on a single line by separating the statements with a semicolon character. An example is given below:

```
cnt = 5; sum = 0; tot = 20;
```

7.8 Indentation

In most programming languages blocks of code are identified by using braces at the beginning and end of the block, or by identifying the end of the block using a suitable statement. e.g., END, WEND, or ENDIF. In Python, there are no braces or statements to indicate the start and end of a block. Instead, blocks of code are identified by line indentation. All statements within a block must be indented by the same amount. The actual number of spaces used to indent a block is not important, as long as all the statements in the block use the same number of spaces.

A valid block of code is given below (don't worry at this stage what the code does):

```
if j == 5:
        a = a + 1
        b = a + 2
else:
        a = 0
        b = 0
```

The following block of code is not valid since the indentation is not correct:

```
if j == 5:
        a = a + 1
    b = a + 2
else:
        a = 0
          b = 0
```

7.9 Python Data Types

Python supports the following data types:

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Sets
- Files

7.10 Numbers

Python supports the following numeric variable types:

- **int** - signed integer
- **long** - long integer
- **float** - floating point real number
- Complex numbers

Numbers can be represented in decimal, Octal, binary, or hexadecimal. Long integers are shown with an upper case letter **L**.

Some example numbers are shown below:

Integer

100	-	decimal
-67	-	decimal
500	-	decimal
0x20	-	hexadecimal
0b10000001	-	binary
0o2377	-	octal
202334567L	-	long decimal
0x3AEEFAE	-	hexadecimal

Floating point

2.355
23.780
-45.6
1.298
24.45E4

Complex

24.4+2,6j
0.78-4.2j
23.7j

We can assign numeric values to variables. These variable objects are created when values are assigned to them:

```
sum = 28
a = 0
```

We can delete a variable object by using the **del** statement:

```
del sum, a
```

We can assign a value to several variables at the same time:

```
w = x = y = z = 0
```

Similarly, we can have statements of the form:

```
w, x, y = 3, 5, 8
```

Which is equivalent to:

```
w = 3
x = 5
y = 8
```

We can perform the following mathematical operations on numbers:

Expression operators

+	addition
-	subtraction
*	multiplication
/	division
>>	shift right
<<	shift left
**	power (exponentiation)
%	remainder

Bitwise operators

|	bitwise OR
&	bitwise AND
^	bitwise exclusive-or
~	bitwise complement

Some mathematical functions

pow(x, y)	same as x**y
abs(x)	absolute value of x
round(x, n)	round x to n digits from the decimal point
floor(x)	largest integer not greater than x
int(x)	convert x to integer
hex(x)	hexadecimal equivalent of integer x
bin(x)	binary equivalent of integer x
exp(x)	exponential of x
factorial(n)	factorial of number n
ceil(x)	smallest integer not less than x
log(x)	natural logarithm of x (base 2)
log10(x)	logarithm of x (base 10)

Some mathematical utility libraries

random	random number library
math	mathematics library

Figure 7.1 to Figure 7.3 show examples of using numbers in Python. The statement **import** is used to import a library to a Python program. The **math** library contains a large number of mathematical functions, such as logarithmic functions, trigonometric functions, square

root, hyperbolic functions, angular conversion, and so on. Further details on these functions can be obtained from the following link:

https://docs.python.org/3/library/math.html

The **random** library is useful for generating random numbers. The function **randint(a, b)** in this library generates a random integer between integers **a** and **b** (inclusive). Details of functions available in the random library can be obtained from the following link:

https://docs.python.org/2/library/random.html

```
>>> 28 + 35
63
>>> 22 * 6
132
>>> 2 ** 5
32
>>> 2 << 3
16
>>> 5 % 2
1
>>> abs(-100)
100
>>> 0x10
16
>>> 0o17
15
>>> 0b00001111
15
>>> (2 + 3j) * 3
(6+9j)
>>> hex(20)
'0x14'
>>> bin(15)
'0b1111'
>>>
```

Figure 7.1 Using numbers in Python.

```
>>> int(23.256)
23
>>> float(4)
4.0
>>> 1/3.0
0.3333333333333333
>>> 10/4.0
2.5
>>> import math
>>> math.sqrt(16)
4.0
>>> math.pi
3.141592653589793
>>> math.floor(-3.5)
-4.0
>>> math.trunc(-4.5)
-4
>>> math.sin(30.0 * math.pi/180)
0.49999999999999994
>>> pow(2, 4)
16
>>> max(2,5,12,8)
12
>>> min(2,4,6,8)
2
```

Figure 7.2 Using numbers in Python.

```
>>> a = 0b00001110
>>> bin(a & 0b11)
'0b10'
>>> bin(a | 0b11)
'0b1111'
>>> math.e
2.718281828459045
>>> math.floor(-2.7)
-3.0
>>> sum((1,2,3,4,5,6,7,8,9,10))
55
>>> import random
>>> random.randint(1, 5)
3
>>> random.randint(1, 5)
4
>>> random.randint(1, 5)
5
>>> (2 + 4j) + (4 + 3j)
(6+7j)
>>> (2.4 * 3), (5.0 / 2.0), math.sqrt(12.0)
(7.199999999999999, 2.5, 3.4641016151377544)
```

Figure 7.3 Using numbers in Python.

7.11 Strings

In Python, strings are declared by enclosing characters between a pair of single or double quotation marks. An example is given below:

```
myname = "James Booth"
```

We can manipulate strings by extracting characters, joining two strings, assigning a string to another string, and so on. Some commonly used string manipulation operations are shown in Figure 7.4 and Figure 7.5.

```
>>> name = "John"
>>> surname = "Adams"
>>> full_name = name + surname
>>> print(full_name)
JohnAdams
>>> initial = name[0]
>>> print(initial)
J
>>> initials = name[0] + surname[0]
>>> print(initials)
JA
>>> print(name[0:3])
Joh
>>> print(name[:2])
Jo
>>> print(name[2:])
hn
>>> print(name*2)
JohnJohn
>>> print(name[0:2] + surname[2:4] + "end")
Joamend
>>> print(name + " " + surname)
John Adams
>>> print(len(name))
4
```

Figure 7.4 String manipulation operations.

```
>>> name = "Smith"
>>> print(name[-1])
h
>>> print(name[-2])
t
>>> print(name.find('i'))
2
>>> name.replace('i', 'k')
'Smkth'
>>> numbers = "111,222,333,444,555"
>>> numbers.split(',')
['111', '222', '333', '444', '555']
>>> name
'Smith'
>>> name[0] = 's'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> name = 's' + name[1:]
>>> name
'smith'
```

Figure 7.5 String manipulation operations.

Notice that a third index, as the step, can be used in string slicing operation. The step is added to the first offset until the second offset and the character at this position is extracted. In the following example, the characters at positions 0, 2, 4, and 6 are extracted:

```
>>> a = "computer"
>>> b = a[0:7:2]
>>> print(b)
cmue
```

7.11.1 String functions

Python supports a large number of string functions. Some commonly used string functions are given below:

- capitalize() — changes the first letter of a string to upper case and all other characters to lower case.
- count(str,beg,end) — finds how many times **str** occurs in a string. Starting and ending positions should be specified.
- find(str,beg,end) — determines if **str** occurs in a string. Starting and ending positions should be specified. The index is returned if the **str** is found, otherwise, **-1** is returned.
- len(string) — returns the length of a string.
- isalpha() — returns **True** if the string contains all alphabetical numeric characters.
- isalnum() — returns **True** if the string contains alphabetical and numeric characters.

- isdigit() returns **True** if the string contains all digits.
- islower() returns **True** if the string contains all lowercase letters.
- isupper() returns **True** if the string contains all uppercase letters.
- lower() converts all uppercase characters to lowercase.
- upper() converts all lowercase characters to uppercase.
- lstrip() removes all leading whitespaces.
- rstrip() removes all trailing whitespaces.
- swapcase() changes the case of all letters.

Figure 7.6 shows examples of using some of the string functions.

```
>>> a = "computer"
>>> b = a.capitalize()
>>> print(b)
Computer
>>> print(a.count('p', 0, len(a)))
1
>>> print(a.find('p', 0, len(a)))
3
>>> print(len(a))
8
>>> b = a.upper()
>>> print(b)
COMPUTER
>>> a.isalpha()
True
>>> a.isdigit()
False
>>> b = a.swapcase()
>>> print(b)
COMPUTER
```

Figure 7.6 Using the string functions.

7.11.2 Escape sequences

Escape sequences are special non-printable characters used to generate functions such as newline, tab, formfeed, carriage return, etc. Escape sequences start with the character **"\"**. A list of the commonly used escape sequences is given below:

- \n newline
- \a bell
- \b backspace
- \f formfeed
- \r carriage return
- \t horizontal tab
- \v vertical tab
- \xhh character with 2 hexadecimal value hh

As an example, the following statement will display the letter **a** followed by two newlines:

```
print("a\n\n")
```

7.12 Print Statement

The print statement is one of the most commonly used statements. It displays text or numbers on the screen. Text is displayed by enclosing it in quotes. Numeric data is displayed by simply entering the variable name. The data to be displayed is enclosed in round brackets. Text and numeric data can be mixed in display outputs and the type of the variable to be displayed can be declared using formatting characters. A list of the commonly used formatting characters is given below:

- %c character
- %s string
- %d signed integer
- %u unsigned integer
- %x lower case hexadecimal number
- %X upper case hexadecimal number
- %f floating point number
- %E exponential notation

Figure 7.7 shows some examples of using the print statement.

```
>>> first, last = 1, 100
>>> print("First = %d Last = %d" %(first, last))
First = 1 Last = 100
>>>
>>> name, age = "John", 21
>>> print("Name = %s Age = %d" %(name, age))
Name = John Age = 21
>>>
>>> a = 2.345
>>> print("a is %E" %(a))
a is 2.345000E+00
>>>
>>> a, b = 5, 10
>>> print("a is %d\n b is %d" %(a, b))
a is 5
 b is 10
>>>
>>> a = 100
>>> print("%X" %(a))
64
```

Figure 7.7 Using the print statement.

7.13 List Variables

List variables are variables separated by commas and enclosed in square brackets. The variables in a list can be of different types. The contents of a list can be accessed using square brackets to index the required item inside the list. Indexing starts from 0. As with the strings, the "*" character can be used for repetition and the "+" character can be used for concatenation. Some examples are given below:

```
mylist = ['John', 'Adam', 230, 12.25, 'Peter', 89]
second = [30, 23]

s = mylist[0]              # s = 'John'
s = mylist[2]              # s = 230
s = mylist[2:4]            # s = 230, 12.25
s = mylist[3:]             # s = 12.25, 'Peter', 89
s = mylist * 2             # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 'John',
                           'Adam', 230, 12.25, 'Peter', 89
s = mylist + second        # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 30, 23
```

The contents of a list can be modified by assigning a new value to the required index position. For example, we can change the 2nd element of the list **mylist** from 230 to 100 as:

```
mylist[2] = 100
```

Python does not allow referencing items that are not present in a list. For example, the following statement gives an error message:

```
mylist[200]
```

Lists can be nested to form two dimensional matrices. An example is given below:

```
M = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
```

The nested list is indexed starting from [0][0]. For example, the elements of row 1 can be accessed as follows:

```
>>> M[1]                 # Elements of row 1
[4, 5, 6]

>>> M[1][1]              # Element at row 1, column 1
5
```

The statement **L = [ ]** creates an empty list called **L**.

7.13.1 List functions

Python language supports a large number of list functions. Some commonly used list functions are given below:

- del([i:j]) deletes elements from i to j-1
- list.append(x) appends an item to the end of a list
- list.extend([x,y,z]) adds several items to the list
- cmp(L1,L2) compares elements of lists L1 and L2

- len(L) returns the length of list L.
- max(L) returns the item with the maximum value
- min(L) returns the item with the minimum value
- list.count(x) returns how many times x occurs in a list
- list.index(x) returns the position of the first occurrence of x
- list.insert(i,x) inserts x at position i in the list
- list.remove(x) removes the indexed item from the list
- list.reverse() reverses a list
- list.sort() sorts a list
- list.pop() deletes and returns the last item

Figure 7.8 shows some examples of using the print statement.

```
>>> lst = ['A', 'B', 'C', 'D', 'E']
>>> del(lst[2])
>>> print(lst)
['A', 'B', 'D', 'E']
>>> del(lst[1:3])
>>> print(lst)
['A', 'E']
>>> lst.append('Z')
>>> print(lst)
['A', 'E', 'Z']
>>> lst.extend(['a', 'b', 'c'])
>>> print(lst)
['A', 'E', 'Z', 'a', 'b', 'c']
>>> print(lst.index('a'))
3
>>> print(lst.reverse())
None
>>> print(lst)
['c', 'b', 'a', 'Z', 'E', 'A']
>>> lst.sort()
>>> print(lst)
['A', 'E', 'Z', 'a', 'b', 'c']
>>> print(len(lst))
6
```

Figure 7.8 Using the list functions.

7.14 Tuple Variables

Tuples are similar to lists but their contents cannot be changed. i.e., they are read-only. Also, tuple variables are enclosed in round brackets (parentheses). Some examples are given below:

```
mytuple = ['John', 'Adam', 230, 12.25, 'Peter', 89]

second = [30, 23]

s = mytuple[0]          # s = 'John'
s = mytuple[2]          # s = 230
s = mytuple[2:4]        # s = 230, 12.25
s = mytuple[3:]         # s = 12.25, 'Peter', 89
s = mytuple * 2         # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 'John',
                        'Adam', 230, 12.25, 'Peter', 89
```

```
s = mytuple + second      # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 30, 23
```

The following statement is not valid since we cannot change the contents of a tuple:

```
mytuple[2] = 200
```

7.15 Dictionary Variables

Dictionaries are similar to hash tables with keys and values. Each key is separated from its value by a colon sign, the items are separated by commas, and the whole thing is enclosed in curly brackets. The keys in a dictionary must have data types of numbers, strings, or tuples. The values can be of any data type. An example is given below:

```
mydict = {'Name': 'John', 'Surname': 'Adams', 'Age': 25}
s = mydict['Name']                  # s = 'John'
s = mydict['Age']                   # s = 25
s = mydict.keys()                   # s = ['Age', 'Surname', 'Name']
s = mydict.values()                 # s = [125, 'Adams', 'John']
```

7.15.1 Dictionary functions

Python language supports a large number of dictionary functions. Some commonly used dictionary functions are given below:

- cmp(d1, d2) compares two dictionaries d1 and d2
- len() returns the number of items in a dictionary
- del(d[key]) deletes an item from the dictionary
- d.clear removes all items from the dictionary
- d.keys() returns a list of dictionary keys
- d.values() returns a list of dictionary values

Figure 7.9 shows some examples of using the print statement.

```
>>> d = {'A':1, 'B':2, 'C':3, 'D':4}
>>> print(len(d))
4
>>> print(d.keys())
['A', 'C', 'B', 'D']
>>> print(d.values())
[1, 3, 2, 4]
>>> del(d['C'])
>>> print(d)
{'A': 1, 'B': 2, 'D': 4}
>>> d.clear()
>>> print(d)
{}
```

Figure 7.9 Using the dictionary functions.

7.16 Keyboard Input

Python provides the following function for reading data from the keyboard:

- input provides a prompted read. The data from the keyboard is returned as a string

Figure 7.10 shows examples of using the keyboard input function. Notice that the function returns a string. Therefore, if numeric data is entered then it should be converted into a numeric data type before being used in mathematical operations.

```
>>> name = input("Enter your name: ")
Enter your name: John Smith
>>> a = input("Enter a number: ")
Enter a number: 5
>>> b = input("Enter another number: ")
Enter another number: 4
>>> c = int(a) + int(b)
>>> print(c)
9
>>> a = int(input("Enter a number: "))
Enter a number: 2
>>> b = int(input("Enter another number: "))
Enter another number: 4
>>> c = a * b
>>> print(c)
8
>>>
```

Figure 7.10 Keyboard input examples

7.17 Comparison Operators

Valid Python comparison operators are:

- == checks if two operands are equal
- != checks if two operands are not equal
- > checks if the left operand is greater than the right one
- < checks if the left operand is less than the right one
- >= checks if the left operand is greater than or equal to the right one
- <= checks if the left operand is less than or equal to the right one

7.18 Logical Operators

Valid Python logical operators are:

- and logical AND of the two operands
- or logical OR of the two operands
- not logical inverse of the operand

7.19 Assignment Operators

- = assignment operator
- += compound add operator
- -= compound subtract operator
- *= compound multiply operator
- /= compound divide operator

7.20 Control of Flow

In normal program flow, statements are executed sequentially one after another one. The flow control statements are used to make decisions and change the order of execution depending on the results of these decisions.

Python programming language supports the following flow control statements:

- if
- if-else
- elif
- for
- while
- break
- continue
- pass

7.20.1 The if, if..else, and elif

The general format of the if statement is:

```
        if expression: statement
or
        if expression:
                Statement 1
                Statement 2
        else:
                Statement 1
                Statement 2
```

Notice the use of indentation inside the **if** blocks and the colon character at the end of the **if** and **else** statements.

An example use of the if statement is:

```
        if a == 5: print('a is 5')
```

if there is only one statement after the if, then it can be typed on the same line. If there is more than one statement, then all the statements must be written on the next lines with the same amount of indentation. An example is given below:

```
        if a == 100:
                x = 0
                y = 0
        else:
                x = 1
                y = 10
```

The **elif** statement is used to check for different conditions in an **if** block. An example is given below:

```
if a > 10:
        b = 0
        c = 0
elif a == 10:
        b = 2
        c = 4
```

Notice that the if statements can be nested as shown in the following example:

```
if a == 100:
        c = 0
        k = 1
        if b == 10:
                c = 20
                m = 1
else:
        c = 23
```

7.20.2 The for statement

The for statement is used to create loops (iteration) in programs. The general format of this statement is:

```
for variable in sequence:
        statements
```

Here, the sequence is evaluated first and the first item in the sequence is assigned to the variable and the statements are executed. Then the second item is assigned to the variable and the statements are executed. This continues until there are no more items in the sequence. An example use of the for statement is shown below:

```
for letter in "COMPUTER":
        print(letter)
```

The following will be displayed on the screen:

```
C
O
M
P
U
T
E
R
```

The **for** statement is commonly used to create loops in programs. The **range** statement denotes the range of the variable as in the following example:

```
for cnt in range(0, 5):
        print(cnt)
```

The following will be displayed on the screen:

```
0
1
2
3
4
```

Notice that the upper value of the range is one less than the specified value i.e., in the above example, the range is from 0 to 4 and not to 5.

We can specify a step size in the last parameter when using the range statement, in the following example, the step size is 5, and the list takes values 0, 5, 10, 15, 20, 25:

```
List(range(0, 30, 5))
```

The **for** statement can be nested if desired.

7.20.3 The while statement

The **while** statement can also be used to create loops (iteration) in programs. The general format of this statement is:

```
while expression:
        statements
```

The statements are executed while the expression evaluates to True. An example is given below:

```
cnt = 0
while cnt < 5:
        print(cnt)
        cnt = cnt + 1
```

The output of the program is as follows:

```
0
1
2
3
4
```

Notice that the statements that belong to the **while** statement must be indented. It is important to make sure that the expression is modified inside the loop, otherwise, an infinite loop will be formed, as shown in the following example:

```
cnt = 0
while cnt < 5:
        print(cnt)
```

7.20.4 The continue statement

The **continue** statement is used in **for** and **while** loops and this statement skips all the remaining statements in a loop and returns to the beginning of the loop. An example is given below. In this example, number 3 is not displayed by the print statement:

```
cnt = 0
while cnt < 5:
        cnt = cnt + 1
        if cnt == 3:
                continue
        print(cnt)
```

The output of this example is as follows:

```
1
2
4
5
```

7.20.5 The break statement

The **break** statement is used in **for** and **while** loops and this statement terminates the loop and execution continues with the next statement. An example is given below:

```
cnt = 0
while cnt < 5:
        cnt = cnt + 1
        if cnt == 3:
                break
        print(cnt)
```

The output of this example will be:

```
1
2
```

7.20.6 The pass statement

The **pass** statement is used when a statement is required syntactically but you do not want any command or code to execute. The **pass** statement is a null operation and nothing happens when it executes. An example is given below:

```
for letter in 'COMPUTER':
        if letter == 'P':
                pass
                print('Passed')
        print(letter)
```

The output of this program is:

```
C
O
M
Passed
P
U
T
E
R
```

We have covered the basic statements of the Python programming language. We will now develop example programs using the knowledge we have gained so far.

7.21 Example 1 – 4 Band Resistor Color Code Identifier

In this example, the user enters the three colors of a 4-band resistor, and the program calculates and displays the value of the resistor in Ohms. The tolerance of the resistor is not displayed.

Background Information: Resistor values are identified by the following color codes:

Black:	0
Brown:	1
Red:	2
Orange:	3
Yellow:	4
Green:	5
Blue:	6
Violet:	7
Grey:	8
White:	9

The first two colors determine the first two digits of the value, while the last color determines the multiplier. For example, **red red red** corresponds to $22 \times 10^2 = 2200$ Ohms.

Program Listing: Figure 7.11 shows the program listing (program: **resistor.py**). At the beginning of the program, a list called **color** is created, which stores the valid resistor colors. Then a heading is displayed, and a **while** loop is created which runs as long as the string variable **yn** is equal to y. Inside the loop, the program reads the three colors from the keyboard using the **input function** and stores them as strings in variables **FirstColor**, **SecondColor,** and **ThirdColor**. These strings are then converted into lowercase so that they are compatible with the values listed in the list box. The index values of these colors in the list are then found using function calls of the form **colors.index**. Remember that the index values start from 0. As an example, if the user entered **red**, then the corresponding index value will be 2. The resistor value is then calculated by multiplying the first color number by 10 and adding it to the second color number. The result is then multiplied by the power of 10 of the third color index. The final result is displayed on the screen. The program then asks whether or not the user wants to continue. If the answer is **y**, then the program returns to the beginning, otherwise the program is terminated.

```
#================================================
#               RESISTOR COLOR CODES
#               --------------------
#
# The user enters the three colors of a resistor
# and the program calculates and displays the value
# of the resistor in Ohms
#
# Program: resistor.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#================================================
colors = ['black','brown','red','orange','yellow','green',\
'blue','violet','grey','white']

print("RESISTOR VALUE CALCULATOR")
print("=========================")
yn = "y"

while yn == 'y':
  FirstColor = input("Enter First Color: ")
  SecondColor = input("Enter Second Color: ")
  ThirdColor = input("Enter Third Color: ")
#
# Convert to lowercase
#
  FirstColor = FirstColor.lower()
  SecondColor = SecondColor.lower()
  ThirdColor = ThirdColor.lower()
#
# Find the values of colors
```

```
#
  FirstValue = colors.index(FirstColor)
  SecondValue = colors.index(SecondColor)
  ThirdValue = colors.index(ThirdColor)
#
# Now calculate the value of the resistor
#
  Resistor = 10 * FirstValue  + SecondValue
  Resistor = Resistor * (10 ** ThirdValue)
  print("Resistance = %d Ohms" % (Resistor))
#
# Ask for more
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Figure 7.11 Program listing.

The program was created using the **nano** text editor and then run from the command line by entering the following command:

beagle@beagle:~ $ **python resistor.py**

Figure 7.12 shows a typical run of the program.

```
beagle@beagle:~$ python resistor.py
RESISTOR VALUE CALCULATOR
=========================
Enter First Colour: red
Enter Second Colour: red
Enter Third Colour: yellow
Resistance = 220000 Ohms

Do you want to continue?: n
beagle@beagle:~$
```

Figure 7.12 Typical run of the program.

You could also write and then run the program using the Thonny IDE.

7.22 Example 2 – Series or Parallel Resistors

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

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

Program Listing: Figure 7.13 shows the program listing (program: **serpal.py**). At the beginning of the program a heading is displayed, and the program enters into a **while** loop. Inside this loop, the user is prompted to enter the number of resistors in the circuit and whether they are connected in series or in parallel. The function **str** converts a number into its equivalent string (e.g., the number 5 is converted into string "5"). If the connection is **in series** (mode equals **'s'**), the program accepts the value of each resistor from the keyboard, and the total resistance is calculated by summing the resistance of each resistor. On the other hand, if the connection is **in parallel** (mode equals **'p'**), the program accepts the value of each resistor from the keyboard, and the reciprocal of each resistor's value is added to the total. After all resistor values have been entered, the program calculates and displays the resultant resistance.

```
#================================================
#         RESISTORS IN SERIES OR PARALLEL
#         ------------------------------
#
# This program calculates the total resistance of
# serial or parallel connected resistors
#
# Program: serpal.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#================================================
print("RESISTORS IN SERIES OR PARALLEL")
print("==============================")
yn = "y"

while yn == 'y':
  N = int(input("\nHow many resistors are there?: "))
  mode = input("Are the resistors series (s) or parallel (p)?: ")
  mode = mode.lower()
#
# Read the resistor values and calculate the total
#
  resistor = 0.0

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

  elif mode == 'p':
    for n in range(0,N):
      s = "Enter resistor " + str(n+1) + " value in Ohms: "
```

```
      r = float(input(s))
      resistor = resistor + 1 / r
    print("Total resistance = %.2f Ohms" %(1 / resistor))
#
# Check if the user wants to exit
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Figure 7.13 Program listing.

Figure 7.14 shows a typical run of the program.

```
beagle@beagle:~$ python serpal.py
RESISTORS IN SERIES OR PARALLEL
================================

How many resistors are there?: 2
Are the resistors series (s) or parallel (p)?: s
Enter resistor 1 value in Ohms: 22
Enter resistor 2 value in Ohms: 33
Total resistance = 55 Ohms

Do you want to continue?: y

How many resistors are there?: 2
Are the resistors series (s) or parallel (p)?: p
Enter resistor 1 value in Ohms: 10
Enter resistor 2 value in Ohms: 10
Total resistance = 5.00 Ohms

Do you want to continue?: n
beagle@beagle:~$
```

Figure 7.14 Typical run of the program.

7.23 Example 3 - Resistive Potential Divider

Description: This case study calculates the resistances in a resistive potential divider circuit.

Background Information: Resistive potential divider circuits consist of two resistors. These circuits are used to lower a voltage to a desired value. Figure 7.15 shows a typical resistive potential divider circuit. Here, Vin and Vo are the input and output voltages, respectively. R1 and R2 are the resistor pair used to lower the voltage from Vin to Vo. A large number of resistor pairs can be used to get the desired output voltage. Choosing large resistors draws little current from the circuit while choosing small resistors draws larger currents. In this design, the user specifies Vin, Vo, and R2. The program calculates the required R1 value to lower the voltage to the desired level. Additionally, the program displays the output voltage with the chosen physical resistors.

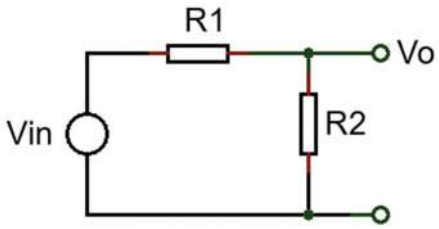


Figure 7.15 Resistive potential divider circuit.

The output voltage is given by:

$$Vo = Vin \times R2 / (R1 + R2)$$

R1 is then given by:

$$R1 = (Vin - Vo) \times R2 / Vo$$

The above formula is used to calculate the required value of R1, given Vin, Vo, and R2.

Program Listing: Figure 7.16 shows the program listing (program: **divider.py**). At the beginning of the program, a heading is displayed. The program then reads Vin, Vo, and R2 from the keyboard. The program calculates R1 and displays R1 and R2. The user is then asked to enter a chosen physical value for R1. With the chosen value of R1, the program displays Vin, Vo, R1, and R2, and asks the user whether or not the result is acceptable. If the answer to this question is **y,** the program terminates. If, on the other hand, the answer is **n** then the user is given the option to try again.

```
#=======================================================
#             RESISTIVE POTENTIAL DIVIDER
#             ---------------------------
#
# This is a resistive potential divider circuit program.
# The program calculates the resistance values that will
# lower the input voltage to the desired value
#
# Program: divider.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#=======================================================
print("RESISTIVE POTENTIAL DIVIDER")
print("===========================")
R1flag = 1
R2flag = 0

while R1flag == 1:
  Vin = float(input("\nInput voltage (Volts): "))
```

```
  Vo = float(input("Desired output voltage (Volts): "))
  R2 = float(input("Enter R2 (in Ohms): "))
#
# Calculate R1
#
  R1 = R2 * (Vin - Vo) / Vo
  print("\nR1 = %3.2f Ohms R2 = %3.2f Ohms" %(R1, R2))
#
# Read chosen physical R1 and display actual Vo
#
  NewR1 = float(input("\nEnter chosen R1 (Ohms): "))

#
# Display and print the output voltage with chosen R1
#
  print("\nWith the chosen R1,the results are:")
  Vo = R2 * Vin / (NewR1 + R2)
  print("R1 = %3.2F R2 = %3.2f Vin = %3.2f Vo = %3.3f" %(NewR1,R2,Vin,Vo))
#
# Check if happy with the values ?
#
  happy = input("\nAre you happy with the values? ")
  happy = happy.lower()
  if happy == 'y':
    break
  else:
    mode = input("Do you want to try again? ")
    mode = mode.lower()
    if mode == 'y':
      R1flag = 1
    else:
      R1flag = 0
      break
```

Figure 7.16 Program listing.

Figure 7.17 shows a typical run of the program.

```
beagle@beagle:~$ python divider.py
RESISTIVE POTENTIAL DIVIDER
===========================

Input voltage (Volts): 12
Desired output voltage (Volts): 5
Enter R2 (in Ohms): 1000

R1 = 1400.00 Ohms R2 = 1000.00 Ohms

Enter chosen R1 (Ohms): 1400

With the chosen R1,the results are:
R1 = 1400.00 R2 = 1000.00 Vin = 12.00 Vo = 5.000

Are you happy with the values? y
beagle@beagle:~$
```

Figure 7.17 Typical run of the program.

7.24 Trigonometric Functions

Python supports a large number of trigonometric functions. The arguments to trigonometric functions must be in radians. The **math** library must be imported into the program before these functions can be used:

- sin(x) trigonometric sine
- cos(x) trigonometric cosine
- tan(x) trigonometric tangent
- asin(x) trigonometric arc sin
- atan(x) trigonometric arc tangent
- atan2(y, x) trigonometric atan(y/x)
- degrees(x) convert degrees into radians
- radians(x) convert radians into degrees

Some examples of using the trigonometric functions are given in Figure 7.18.

```
>>> import math
>>> print(math.radians(45))
0.785398163397
>>> print(math.degrees(1))
57.2957795131
>>> print(math.sin(math.radians(30)))
0.5
>>> print(math.cos(math.radians(60)))
0.5
>>> print(math.degrees(math.asin(0.5)))
30.0
```

Figure 7.18 Trigonometric function examples.

7.25 User Defined Functions

Functions are like small programs within a program. We can use functions to break up a complex program into several manageable sections, where each section can be implemented as a function. Functions enable us to reuse parts of our programs. For example, we can create a function to calculate the cube root of a number and then call this function from different parts of our program. Another advantage of using functions is that they make it easier to maintain and update our programs.

A function that we create can be called from anywhere in a program. Functions have their own variables and their own commands. As we have seen in earlier parts of this chapter, Python has a large number of built-in functions for various operations such as arithmetic, trigonometric, string manipulation, and so on. User-defined functions are created by programmers. In this section, we shall be looking at how functions can be created and used in our programs.

A user-defined function consists of the following:

- functions begin with the keyword **def**, followed by function name, and round brackets, followed by a colon sign.
- Input arguments to the function must be placed inside the brackets at the beginning of the function definition.
- The body of a function must be indented with the same number of spaces on the left-hand side
- An optional text message can be displayed at the first line of a function to describe what the function does.
- A function must be terminated with the return statement

An example function, named **Mult** is given below. This function takes two numbers first and second as its arguments, multiplies them, and returns the result:

```
def Mult(first, second):
        "This is a simple multiplication function"
        result = first * second
        return result
```

A function is called from the main program by specifying the name of the function and enclosing any arguments in a pair of brackets. For example, to call the above function to multiply numbers 5 and 3 and store the result in a variable called a, we include the following statement in our program:

```
a = Mult(5, 3)
```

We can also call a function by specifying the keyword arguments. i.e.:

```
a = Mult(first = 5, second = 3)
```

Figure 7.19 shows the above example in a Python program.

```
>>> def Mult(first, second):
...     "This is a simple multiplication function"
...     result = first * second
...     return result
...
>>> a = Mult(5, 3)
>>> print(a)
15
```

Figure 7.19 Creating and calling a function.

Another example is shown in Figure 7.20. In this example, the function displays a string passed as an argument. Notice that there is no data returned from this function.

```
>>> def Prnt(strng):
...     print(strng)
...     return
...
>>> Prnt("Hello there")
Hello there
```

Figure 7.20 A function displaying a string.

The variables used in a function are local to that function. Thus, for example, if there are two variables with the same name, one inside the function and the other outside, changing the one inside the function does not change the one outside. Variables outside a function are called **global** variables, whereas the ones inside a function are called **local** variables. See Figure 7.21 for an example where the contents of variable **res** are not changed outside the function.

```
>>> def Mult(first, second):
...     res = first * second
...     return res
...
>>> res = 2
>>> a = Mult(3, 8)
>>> print(a)
24
```

Figure 7.21 Variables in a function are local.

The rules for global variables are as follows:

- Global variables are variables assigned at the top of the program outside the function definitions.
- Global names must be declared only if they are assigned within a function.
- Global names may be referenced within a function without being declared.

Therefore, by declaring a variable outside the functions and also inside a function but with the global keyword, we can change its contents inside the function. An example is given below, which identifies the use of global variables:

```
cnt = 10                        # variable cnt is global
def tstfunc():                  # function declaration
        global cnt              # variable cnt defined as global
        cnt = 200               # value of global cnt is changed

tstfunc()                       # function is called
print(cnt)                      # value of cnt is 200
```

As explained above, if the value of a global variable is not changed inside a function, then there is no need to define it as global. In the following code, there is no need to define **x** as global inside the function:

```
x = 10
y = 4

def tst():
        global y
        y = x + 2
```

It is important to note that the variables in a function call are passed **by value**. This means that the value of a parameter cannot be changed inside a function. An example is shown in Figure 7.22. In this example, notice that the value of variable **cnt** is not changed inside the function call.

```
>>> cnt = 2
>>> def Mult(first, second):
...     cnt = 5
...     return(first * second)
...
>>> a = Mult(5, 6)
>>> print(a)
30
>>> print(cnt)
2
```

Figure 7.22 Variables are passed by value.

A function normally returns only one item back to the calling program. In some applications, we may want to return more than one item to the calling program. This is easily done by returning a tuple and then unpacking it in the main program. An example is shown in Figure 7.23. In this example, the function **MyFunc** is declared with two arguments. The arguments are added and stored in a local variable called **sum**. Similarly, the difference of the arguments is stored in variable **diff**. The function returns both **sum** and **diff** as a tuple. The calling main program unpacks the returned data and stores it in variables **x** and **y**.

```
>>> def MyFunc(a, b):
...     sum = a + b
...     diff = a - b
...     return sum, diff
...
>>> x, y = MyFunc(12, 5)
>>> print(x, y)
(17, 7)
```

Figure 7.23 Returning more than one variable from a function,

7.26 Examples

Example 4

Write a program to read an angle from the keyboard in degrees and display the trigonometric sine of this angle. Repeat until the user stops the program.

Solution 4

The required program listing and example output are shown in Figure 7.24 (program: **trig.py**). The angle entered by the user is converted into a floating point and is stored in a variable angle. Then the trigonometric sine of this angle is displayed. The program continues until the user enters **n** in response to the prompt **Any more?**

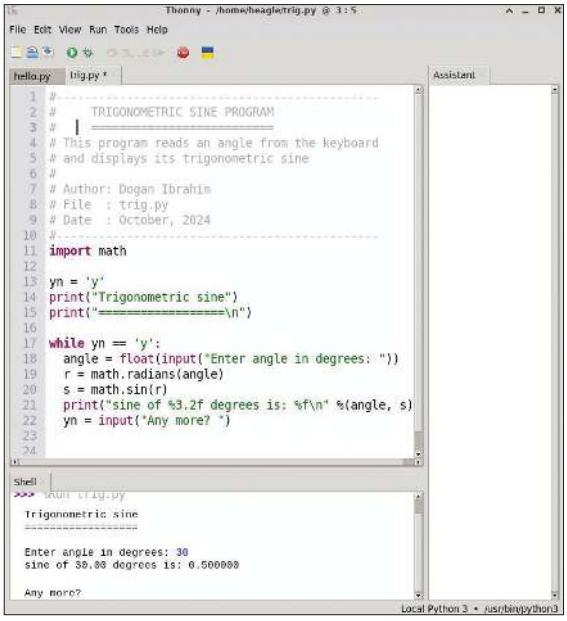


Figure 7.24 Program listing.

This program was created and run using the Thonny IDE.

Example 5

Modify the program in Example 4 so that the user can choose between sine, cosine, and tangent.

Solution 5

The modified program listing and example output are shown in Figure 7.25 and Figure 7.26 (program: **trigall.py**). The user is given a menu with four choices: sine, cosine, tangent, and exit. The angle is read from the keyboard and is converted into radians. The program then calculates the trigonometric value and displays it on the screen. This process is repeated until the user selects the **exit** option.

```
#-------------------------------------------------
#    TRIGONOMETRIC SINE,COSINE,TANGENT PROGRAM
#    =========================================
#
# This program reads an angle from the keyboard
# and displays its trigonometric sine, cosine, or
# tangent depending on user choice. The angle is
# read in degrees,converted into radians and then
# the required trigonometric function is calculated
#
# Author: Dogan Ibrahim
# File  : trigall.py
# Date  : October, 2024
#-------------------------------------------------
import math

choice = '1'
while choice != '0':
  print("Trigonometric Sine, Cosine, or Tangent")
  print("======================================\n")
  print("1. Sine")
  print("2. Cosine")
  print("3. Tangent")
  print("0. Exit")
  choice = input("Enter choice: ")

  if choice != '0':
    angle = float(input("Enter angle in degrees: "))
    r = math.radians(angle)
    if choice == '1':
      s = math.sin(r)
      strng = "sine"
    elif choice == '2':
      s = math.cos(r)
      strng = "cosine"
    elif choice == '3':
      s = math.tan(r)
      strng = "tangent"
    print(strng + " of %3.2f degrees is: %f\n" %(angle, s))
print("End of program")
```

Figure 7.25 Modified program listing.

```
beagle@beagle:~$ python trigall.py
Trigonometric Sine, Cosine, or Tangent
======================================

1. Sine
2. Cosine
3. Tangent
0. Exit
Enter choice: 2
Enter angle in degrees: 60
cosine of 60.00 degrees is: 0.500000

Trigonometric Sine, Cosine, or Tangent
======================================

1. Sine
2. Cosine
3. Tangent
0. Exit
Enter choice: 0
End of program
beagle@beagle:~$
```

Figure 7.26 Example output.

This program was created using the **nano** text editor and then run using the command:

beagle@beagle:~ $ **python trigall.py**

Example 6
Write a program to tabulate the trigonometric sines of angles from 0º to 90º in steps of 5º.

Solution 6
The required program listing is shown in Figure 7.27 (program: **sinetable.py**). After displaying a heading, the for statement is used to create a loop. Variable angle takes values from 0 to 90 (inclusive) in steps of 5. The trigonometric sine is calculated and displayed.

```
#-------------------------------------------------
#      TRIGONOMETRIC SINE TABLE
#      ========================
#
# This program tabulates the trigonometric sine of
# angles from 0 to 90 degrees in steps of 5 degress
#
# Author: Dogan Ibrahim
# File  : sinetable.py
# Date  : October, 2024
#-------------------------------------------------
import math

print("TABLE OF TRIGONOMETRIC SINE")
print("========================\n")
print("  ANGLE      SINE")
```

```
for angle in range(0, 95, 5):
  r = math.radians(angle)
  s = math.sin(r)
  print("  %d        %f" %(angle, s))

print("End of program")
```

Figure 7.27 Program listing.

An example run of the program is shown in Figure 7.28.

```
beagle@beagle:~$ python sinetable.py
TABLE OF TRIGONOMETRIC SINE
=========================

  ANGLE       SINE
  0         0.000000
  5         0.087156
  10         0.173648
  15         0.258819
  20         0.342020
  25         0.422618
  30         0.500000
  35         0.573576
  40         0.642788
  45         0.707107
  50         0.766044
  55         0.819152
  60         0.866025
  65         0.906308
  70         0.939693
  75         0.965926
  80         0.984808
  85         0.996195
  90         1.000000
End of program
```

Figure 7.28 Example run of the program.

Example 7
Write a program to read meters from the keyboard. Convert into yards and inches and display the result.

Solution 7
The required program listing and example output are shown in Figure 7.29 and Figure 7.30 respectively (program: **conv.py**). After displaying a heading, meters are read from the keyboard using the **input** statement. The value is then converted into yards and inches by multiplying with 1.0936 and 39.370 respectively. The results are displayed on the screen.

```
#-------------------------------------------------
#                      CONVERSION PROGRAM
#                      ==================
#
# This program reads meters from the keyboard and
# converts and displays in yards and inches
#
# Author: Dogan Ibrahim
# File  : conv.py
# Date  : October, 2024
#-------------------------------------------------
print("Convert meters into yards and inches")
print("===================================")
meters = float(input("Enter meters: "))
yards = 1.0936 * meters
inches = 39.370 * meters
print("%f meters = %f yards, %f inches" %(meters, yards, inches))
print("End of program")
```

Figure 7.29 Program listing.

```
beagle@beagle:~$ python conv.py
Convert meters into yards and inches
===================================
Enter meters: 20
20.000000 meters = 21.872000 yards, 787.400000 inches
End of program
beagle@beagle:~$
```

Figure 7.30 Example output.

Example 8
Repeat Example 7 but do the conversion in a function called **Conv**. Show how this function can be called from the main program.

Solution 8
The required program listing is shown in Figure 7.31 (program: **convfunc.py**). Function **Conv** is declared at the beginning of the program. Meters to be converted into yards and inches are passed as an argument to the function. The function returns the yards and inches in a tuple. The main program reads the meters from the keyboard and calls the function Conv. The result is displayed on the screen.

```
#-------------------------------------------------
#                      CONVERSION PROGRAM
#                      ==================
#
# This program reads metres from the keyboard and
# converts and displays in yards and inches
#
# Author: Dogan Ibrahim
# File  : convfunc.py
# Date  : October, 2024
#-------------------------------------------------
def Conv(m):
  "Convert metres into yards and inches"
  y = 1.0936 * m
  i = 39.370 * m
  return y, i

print("Convert metres into yards and inches")
print("====================================")
metres = float(input("Enter metres: "))
yards, inches = Conv(metres)
print("%f metres = %f yards, %f inches" %(metres, yards, inches))
print("End of program")
```

Figure 7.31 Program listing.

Example 9
Write a function called **Cyl** to calculate the area and volume of a cylinder, given its radius and height. Use this function in a main program.

Solution 9
The area and volume of a cylinder are given by the formula:

$$\text{Area} = 2\pi rh$$
$$\text{Volume} = \pi r^2 h$$

The required program listing and example output are shown in Figure 7.32 and Figure 7.33 respectively (program: **cylinder.py**). The radius and height of the cylinder are passed as arguments to a function, which calculates the area and volume of the cylinder and returns the results to the main program, where they are displayed on the screen.

```
#-----------------------------------------------------
#                      CONVERSION PROGRAM
#                      ==================
#
# This program reads the radius and height of a cylinder
# and calculates and displays its area and volume
#
# Author: Dogan Ibrahim
# File  : cylinder.py
# Date  : October, 2024
#-----------------------------------------------------
import math

def Cyl(r, h):
  "Area and volume of a cylinder"
  area = 2 * math.pi * r * h
  volume = math.pi * r * r * h
  return area, volume

print("Area and Volume of a Cylinder")
print("=============================")
radius = float(input("Enter the radius: "))
height = float(input("Enter the height: "))
A, V = Cyl(radius, height)
print("Area = %f Volume = %f" %(A, V))
print("End of program")
```

Figure 7.32 Program listing.

```
beagle@beagle:~$ python cylinder.py
Area and Volume of a Cylinder
=============================
Enter the radius: 25
Enter the height: 12
Area = 1884.955592 Volume = 23561.944902
End of program
beagle@beagle:~$
```

Figure 7.33 Example output.

Example 10

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

Solution 10

The required program listing is shown in Figure 7.34 (program: **calc.py**). Two numbers are received from the keyboard and stored in variables **n1** and **n2**. Then, the required mathematical operation is received, and it is performed. The result, stored in variable **result**, is displayed on the screen. The user is given the option of terminating the program.

```
#-----------------------------------------------
#              CALCULATOR PROGRAM
#              ==================
#
# This is a simple calculator program that can
# carry out 4 basic arithmetic opertions
#
# Author: Dogan Ibrahim
# File  : calc.py
# Date  : October, 2024
#-----------------------------------------------
any = 'y'
while any == 'y':
   print("\nCalculator Program")
   print("==================")

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

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

Figure 7.34 Program listing.

An example run of the program is shown in Figure 7.35.

```
beagle@beagle:~$ python calc.py

Calculator Program
==================
Enter first number: 25
Enter second number: 50
Enter operation (+-*/): +
Result = 75.000000

Any more (yn): y

Calculator Program
==================
Enter first number: 12
Enter second number: 2
Enter operation (+-*/): *
Result = 24.000000

Any more (yn): n
beagle@beagle:~$
```

Figure 7.35 Example output.

Example 11

Write a program to simulate double dice. i.e., to display two random numbers between 1 and 6 every time it is run.

Solution 11

The required program listing and example output are shown in Figure 7.36 (program: **dice.py**). Here, the random number generator **randint** is used to generate random numbers between 1 and 6 when the Enter key is pressed. The program is terminated when the letter **X** is entered.

dice.py

```
#-----------------------------------------------
#              DOUBLE DICE
#              ===========
#
# This program displays two random dice numbers
# between 1 and 6 when the Enter key is pressed
#
# Author: Dogan Ibrahim
# File  : dice.py
# Date  : October, 2023
#-----------------------------------------------
import random
strt = 'a'

while strt.upper() != 'X':
  strt = input("Pres ENTER to start, X to exit ")
  first = random.randint(1, 6)
  second = random.randint(1, 6)
  print("%d   %d" %(first, second))
```

Shell

```
>>> %Run dice.py

  Pres ENTER to start, X to exit
  6   1
  Pres ENTER to start, X to exit
  1   5
  Pres ENTER to start, X to exit
  4   1
  Pres ENTER to start, X to exit x
  3   2

>>>
```

Figure 7.36 Program listing and example output.

Example 12
Write a program to use functions to calculate and display the areas of shapes: square, rectangle, triangle, circle, and cylinder. The sizes of the required sides should be received from the keyboard.

Solution 12
The areas of the shapes to be used in the program are as follows:

Square: side = a	area = a^2
Rectangle: sides a, b	area = ab
Circle: radius r	area = πr^2
Triangle: base b, height h	area = bh/2
Cylinder: radius r, height h	area = $2\pi rh$

The required program listing is shown in Figure 7.37 (program: **areas.py**). A different function is used for each shape, and the sizes of the sides are received inside the functions. The main program displays the calculated area for the chosen shape.

```
#-----------------------------------------------
#              AREAS OF SHAPES
#              ===============
#
# This program calculates and displays the areas
# of various geometrical shapes
# of numbers in a list
#
# Author: Dogan Ibrahim
# File  : areas.py
# Date  : October, 2024
#-----------------------------------------------
import math

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

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

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

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

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

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

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

shape = shape.lower()
if shape == 's':
  a = float(input("Enter a side of the square: "))
  area = Square(a)
  s = "Square"
elif shape == 'r':
  a = float(input("Enter one side of the rectangle: "))
  b = float(input("Enter other side of the rectangle: "))
  area = Rectangle(a, b)
  s = "Rectangle"
elif shape == 'c':
  radius = float(input("Enter radius of the circle: "))
  area = Circle(radius)
  s = "Circle"
elif shape == 't':
  base = float(input("Enter base of the triangle: "))
  height = float(input("Enter height of the triangle: "))
  area = Triangle(base, height)
  s = "Triangle"
elif shape == 'y':
  radius = float(input("Enter radius of cylinder: "))
  height = float(input("Enter height of cylinder: "))
  area = Cylinder(radius, height)
  s = "Cylinder"

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

Figure 7.37 Program listing.

An example run of the program is shown in Figure 7.38.

```
beagle@beagle:~$ python areas.py
AREAS OF SHAPES
===============

What is the shape?:
Square (s)
Rectangle(r)
Circle(c)
Triangle(t)
Cylinder(y): r
Enter one side of the rectangle: 12
Enter other side of the rectangle: 3
Area of Rectangle is 36.000000
beagle@beagle:~$
```

Figure 7.38 Example output.

7.27 Recursive Functions

Recursive functions are functions that call themselves either directly or indirectly, and such functions are supported by Python. Although the topic of recursive functions is an advanced topic, an example is given in Figure 7.39 to illustrate the principles of such functions. This recursive function implements the factorial operation. Detailed analysis of recursive functions is beyond the scope of this book.

```
>>> def factorial(n):
...     if n == 1:
...       return 1
...     else:
...       return n * factorial(n-1)
...
>>>
>>> factorial(4)
24
>>> factorial(6)
720
>>>
```

Figure 7.39 Recursive factorial function.

7.28 Exceptions

There may be major errors in our programs, such as dividing by zero, file permission errors, and so on. Normally, when Python encounters such errors, it cannot handle them, and the program crashes.

One way to handle such errors orderly and avoid crashes is to use exception handling in our programs. The basic method is that whenever an error occurs, the program detects this error and takes appropriate measures to handle the error, and continues to execute normally. Exception handling is also useful if we wish to terminate a running program in an orderly manner, such as shutting down any input-output operations when the program is terminated asynchronously by the user (e.g., by pressing the Ctrl+C key).

The statements **try** and **except** are used to handle unexpected errors or terminations in our programs. The general format of exception handling is as follows:

```
try:
```

```
        Normal program statements
        Normal program statements
except condition 1:
        if condition 1 type error occurs then execute this block of code
                ……..
                ……..
except condition 2:
        if condition 2 type error occurs then execute this block of code
                ……..
                ……..
else:
        if there are no errors detected then execute this block of code
                ……
                ……
```

We can use the except statement with no condition in order to handle any type of exception. Some of the commonly used exceptions are:

exception EOFError: the end-of-file condition is reached while reading data

exception ImportError: import statement could not load a module

exception IndexError: sequence subscript is out of range

exception KeyError: a dictionary key is not found in the set of existing keys

exception KeyboardInterrupt: user hits the interrupt key (normally the Cntrl+C or Delete key)

exception MemoryError: operation ran out of memory

exception OverFlowError: arithmetic operation resulted in overflow

exception RuntimeError: an error is detected that does not fall into any other categories

exception ValueError: an operation or function receives an argument that has the right type but an inappropriate value

exception ZeroDivisionError: a division by zero occurred

Some examples of using exceptions in programs are given below.

Example 13
Write a program to wait for an input from the keyboard. Terminate the program orderly when the **Cntrl+C** keys are pressed on the keyboard.

Solution 13
Figure 7.40 shows the program listing (program: **except1.py**). The exception **KeyboardInterrupt** is used in this program. The message **End of Program** is displayed when the **Ctrl+C** key combination is pressed on the keyboard.

```
#=====================================================
#      KeyboardInteerupt EXCEPTION
#
# This program detects the keyboard entry Ctrl+C and
# the program is teminated orderly after the message
# End of Program is displayed
#
# Author : Dogan Ibrahim
# File   : except1.py
# Date   : October, 2024
#=====================================================
try:
   mode = input("Enter Ctrl+C to terminate the program: ")
except KeyboardInterrupt:
   print("\nEnd of Program")
```

Figure 7.40 Program listing.

Example 14
Write a program to detect division by zero and to display the message **Divide by Zero** when this exception is detected.

Solution 14
Figure 7.41 shows the program listing (program: **except2.py**). In this example, the program is forced to divide a number by zero, which is detected as an exception, and the program displays a message when this occurs.

```
#=====================================================
#      ZeroDivisionError EXCEPTION
#
# This program detects when a number is divided by zero
# and generates an exception to display a message
##
# Author : Dogan Ibrahim
# File   : except2.py
# Date   : July, 2024
#=====================================================
```

```
print("Divide by zero exception")

try:
   s = 10 / 0
except  ZeroDivisionError:
   print("Divide by Zero")
```

Figure 7.41 Program listing.

When the program is run it displays the following message:

```
Divide by zero exception
Divide by Zero
```

7.29 try/final Exceptions

The statement **finally** can be used in exception handling. The Try/finally combination specifies an exception where the block beginning with finally is always executed on the way out, regardless of whether an exception occurs in the try block. An example is given below:

Example 15

Write a program to look for **KeyboardInterrupt** exception and display the message **Exception not occurred** if an exception has not occurred.

Solution 15

Figure 7.42 shows the program listing (program: **except3.py**). The block inside **finally** is executed regardless of whether an exception occurs.

```
#=====================================================
#          try/finally In EXCEPTION
#
# This program detects the keyboard entry Ctrl+C and
# displays the message Keyboard Interrupt if interrupt
# occurs. Message Continue is displayed regardless of
# whether an exception occurred
#
# Author : Dogan Ibrahim
# File   : except3.py
# Date   : July, 2024
#=====================================================
try:
   mode = raw_input("Enter Ctrl+C to terminate the program: ")
except KeyboardInterrupt:
   print("\nKeyboard Interrupt")
finally:
   print("\nContinue")
```

Figure 7.42 Program listing.

When the program is run the following is displayed

```
Enter Ctrl+C to terminate the program:
```

After entering Ctrl+C:

```
Keyboard Interrupt
```

7.30 Date and Time

In some applications, it may be necessary to get the current date and time. Python supports a number of functions to get the current date and time. The module **time** must be imported before these functions can be used. Some of the commonly used date and time functions are as follows:

- time.localtime() returns the current date and time in the following format:
- time.struct_time(tm_year=2013,tm_mon=12,tm_mday=18,
- tm_hour=12,tm_min=45,tm_sec=3,tw_wday=2,tm_yday=352,
- tm_isdst=0)
- time.asctime() returns the date and time in a standard readable format
- time.clock() returns the current CPU time in seconds
- time.ctime() returns the current date and time
- time.time() returns the current time in seconds since the epoch
- time.sleep(x) suspends the calling program for x seconds

Some examples are given in Figure 7.43.

```
>>> import time
>>> print(time.localtime())
time.struct_time(tm_year=2023, tm_mon=10, tm_mday=6, tm_hour=14, tm_min=31,
ec=0, tm_wday=4, tm_yday=279, tm_isdst=1)
>>>
>>> print(time.asctime())
Fri Oct  6 14:31:05 2023
>>>
>>> print(time.ctime())
Fri Oct  6 14:31:15 2023
>>>
>>> print(time.time())
1696599083.4537978
>>>
```

Figure 7.43 Example date and time functions.

The **datetime** module can also be used for date and time functions. This module must be imported in order to use these functions. Some examples of **date** functions are shown in Figure 7.44.

```
>>> from datetime import date
>>> print(date.today())
2023-10-06
>>>
>>> print(date.today().year)
2023
>>>
>>> print(date.today().month)
10
>>>
>>> print(date.today().day)
6
>>>
```

Figure 7.44 Examples of using the datetime date functions.

The function **strftime(format)** is very useful as it can be used to format a date and time string. Some examples of using this function are given in Figure 7.45.

```
>>> from datetime import datetime
>>> print(datetime.now().strftime("%Y:%m"))
2023:10
>>>
>>> print(datetime.now().strftime("%H:%M:%S"))
14:41:08
>>>
>>> print(datetime.now().strftime("%d:%m:%Y"))
06:10:2023
>>>
>>> print(datetime.now().strftime("%d:%m:%Y_%H:%M:%S"))
06:10:2023_14:42:07
>>>
```

Figure 7.45 Examples of using strftime.

7.31 Creating Your Own Modules

In some applications, we may want to create our own Python modules and import them into our programs. Python modules are simply .py program files. Writing a module is just like writing any other Python program. Modules can contain functions, classes, and variables.

A simple module called **msg.py** is shows below:

```
def hello():
    print("Hello there!")
```

We can now import this module into our Python programs. An example program called **myprog.py** is shown below:

```
import msg
msg.hello()
```

Running the program: **python myprog.py** will display the following output:

```
Hello there!
```

We can also modify our program **myprog.py** and import and then call the module as follows:

```
from msg import hello
hello()
```

We can use variables in our module as shown below:

msg.py

```
def hello():
   print("Hello there!")

name = "Jones"
```

myprog.py

```
import msg
msg.hello()
print(msg.name)
```

The program will display:

```
Hello there!
Jones
```

Aliases can be created for modules. This is shown in the following code:

myprog.py

```
import msg as tst
tst.hello()
```

Will display the output:

```
Hello there!
```

An example module is given below that calculates the cube of a number.

Example 16
Write a module that calculates the cube of the integer number passed to it. Show how this module can be imported and used in a program.

Solution 16
Figure 7.46 shows the module listing (program: **cubeno.py**). The function **cube** inside **cubeno.py** has the number as its argument. The cube of this number is calculated and returned. Figure 7.47 shows the program (program: **myprog.py**). As an example, when the number is 3, the output from the program is:

Cube of 3 is: 27

```
def cube(N):
  r = N * N * N
  return r
```

Figure 7.46 Program cubeno.py listing.

```
import cubeno
n= 3
res = cubeno.cube(n)
print("Cube of %d is: %d" %(n,res))
```

Figure 7.47 Program myprog.py.

Module Search Path: When a module is to be imported, Python looks at the following folders in the order given:

- The folder from which the module is called (where the calling main program is)
- The list of directories contained in the PYTHONPATH environment variable.
- Installation dependent list of directories configured when Python was installed

Python's search path can be displayed by entering the following command interactively:

>>> import sys
>>> sys.path

The display on the author's computer is shown in Figure 7.48.

```
beagle@beagle:~$ python
Python 3.11.2 (main, Aug 26 2024, 07:20:54) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', '/usr/lib/python311.zip', '/usr/lib/python3.11', '/usr/lib/python3.11/lib-d
ynload', '/usr/local/lib/python3.11/dist-packages', '/usr/lib/python3/dist-packa
ges', '/usr/lib/python3.11/dist-packages']
>>>
```

Figure 7.48 Python path display.

To make sure that your module is found by Python, you can do one of the following:

- Put the module program file in the folder where your main program is.
- Modify the PYTHONPATH environment variable to contain the folder where the module program is.
- Put the module program in one of the folders already contained in the PYTHONPATH.

Chapter 8 • BeagleY-AI LED Projects

8.1 Overview

This Chapter is about the BeagleY-AI hardware interface and using LEDs in simple projects. The BeagleY-AI is connected to external electronic circuits and devices using its GPIO (General Purpose Input Output) port connector. This is a 2.54 mm, 40-pin expansion header, arranged in a 2x20 strip as shown in Figure 8.1. The I/O ports are numbered as GPIOnn.

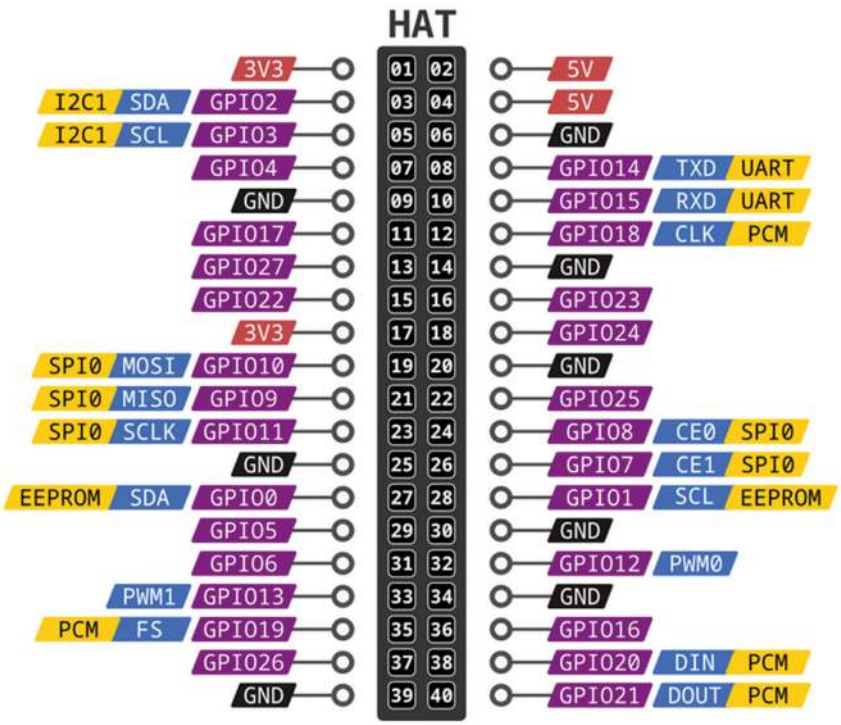


Figure 8.1 BeagleY-AI GPIO pins.

8.2 BeagleY-AI GPIO pin Definitions

When the GPIO connector is at the far side of the board, the pins starting from the left of the connector are numbered as 1, 3, 5, 7, and so on, while the ones at the top are numbered as 2, 4, 6, 8 and so on (Figure 8.2).

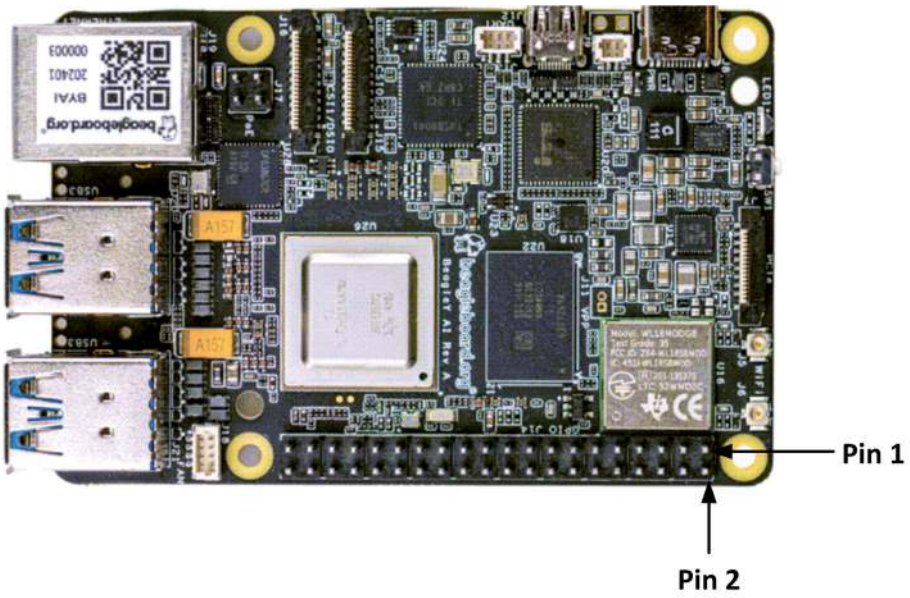


Figure 8.2 GPIO pin numbering.

The GPIO provides 26 general-purpose bi-directional I/O pins. Some of the pins have multiple functions. For example, GPIO10 is a general-purpose I/O pin, shared with the SPI MOSI and SPI0.

Two power outputs are provided: +3.3 V and +5.0 V. The GPIO pins operate at +3.3 V logic levels (unlike many other computer circuits that 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). BeagleY-AI is normally operated using an external power supply (e.g., a mains adapter) with +5 V output. 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 BeagleY-AI. The amount of current that can be supplied by the 5 V pin depends on many factors such as the current required by the device itself, current taken by the USB peripherals, camera current, micro 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 BeagleY-AI board as there is no over-voltage protection circuitry.

8.3 Project 1 – Flashing an LED

Description: This is perhaps the easiest hardware project you can design using your BeagleY-AI. In this project you will connect an LED to one of the ports of the BeagleY-AI and then flash the LED at a rate of once a second. The aim of this project is to show how a

simple Python program can be written and then run from a file. The project also shows how to connect an LED to a BeagleY-AI GPIO pin. In addition, the project shows how to use the GPIO library to configure and set a GPIO pin to logic 0 or 1.

Block diagram: The block diagram of the project is shown in Figure 8.3

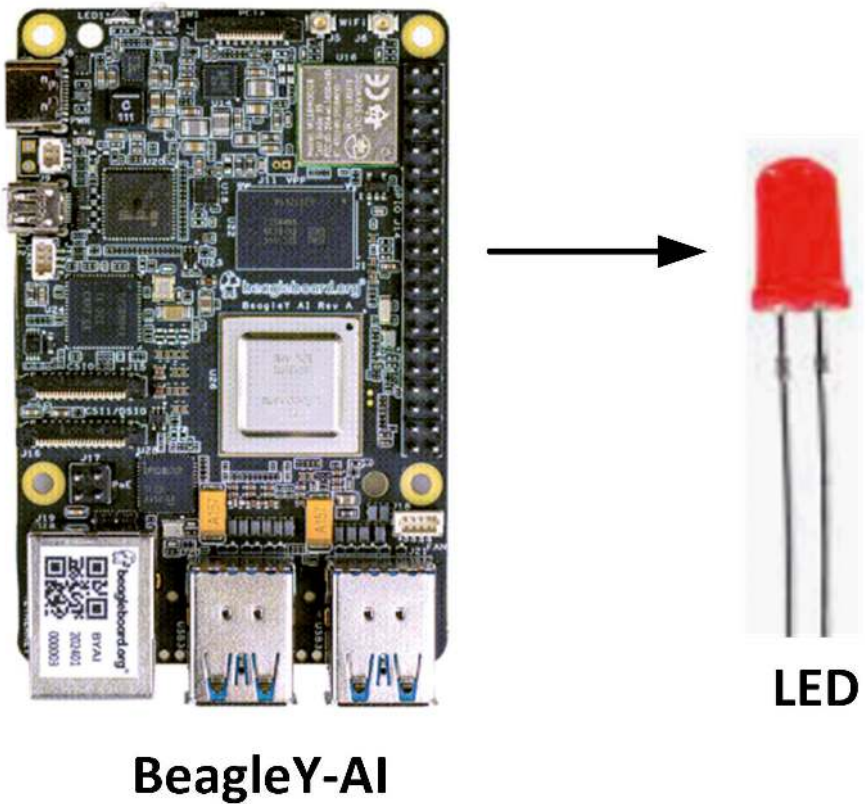


Figure 8.3 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.4. A small low-current LED is connected to port pin GPIO17 (pin 11) of the BeagleY-AI through a current-limiting resistor. The value of the current limiting resistor is calculated as follows:

The output high voltage of a GPIO pin is 3.3 V. The voltage across an LED is approximately 1.8 V. The current through the LED depends upon the type of LED used and the amount of required brightness. Assuming that we are using a small LED, we can assume a forward LED current of about 3 mA. Then, the value of the current limiting resistor is:

$$R = (3.3\ V - 1.8\ V) / 3\ mA = 500\ \Omega.$$ We can choose a 470 Ω resistor.

In Figure 8.4, the LED is operated in current sourcing mode, where a high output from the GPIO pin drives the LED. The LED can also be operated in current sinking mode, where the other end of the LED is connected to the +3.3 V supply and not to ground. In current sinking mode, the LED is turned ON when the GPIO pin is at logic low.

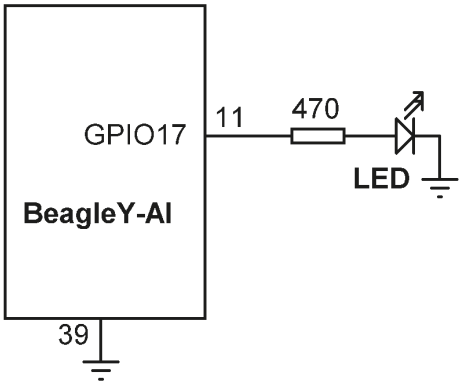


Figure 8.4 Circuit diagram of the project.

Construction: The project is constructed on a breadboard as shown in Figure 8.5. Jumper wires are used to connect the LED to the GPIO port. Notice that the short side of the LED must be connected to ground.

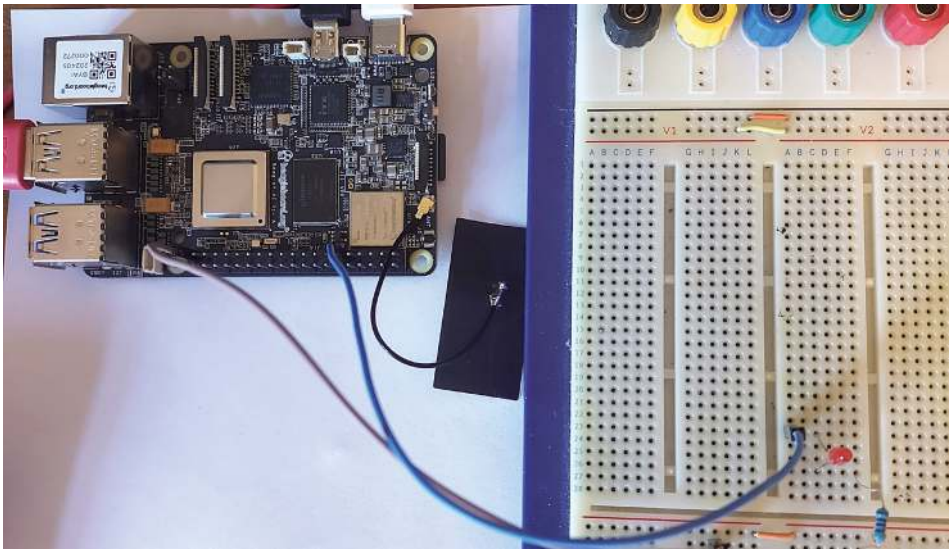


Figure 8.5 Constructing the project on a breadboard.

Program listing: The libgpiod library which is required for I/O programming should already be installed in the default image, in case it's not installed, then install it by using the command below:

```
sudo apt-get install python3-libgpiod
```

The program is called **led.py**, and the listing is shown in Figure 8.6. The program was written using the **nano** text editor. At the beginning of the program, the **gpiod** and **time** libraries are imported. Then, the **LED** is linked to GPIO port 17 and it is configured as an output with default value of 0, i.e., the LED is OFF at the beginning of the program. Then an endless **while** loop is formed where the LED is turned ON and OFF with a one-second delay between each output. Function **set_value()** sets the value of a GPIO pin to 0 or 1.

```
#-------------------------------------------------------
#
#                      FLASHING LED
#                      ============
#
# In this project a small LED is connected to GPIO17 of
# the BeagleY-AI. The program flashes the LED every
# second.
#
# Program: led.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time library

led = gpiod.find_line('GPIO17')
led.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

while True:
  led.set_value(1)                      # turn ON LED
  time.sleep(1)                         # wait 1 second
  led.set_value(0)                      # turn OFF LED
  time.sleep(1)                         # wait 1 second
```

Figure 8.6 Program listing of the project.

The program is run from the console mode as follows:

```
beagle@beagle: ~ $ python led.py
```

If you wish to run the program from the GUI Desktop environment, you should use the Thonny IDE. Type in the program if it is not already in your default directory. Click **Run** to run the program. You should see the LED flashing every second. To terminate the program, close the screen by clicking on the **STOP** button.

Note: You can copy the programs from your BeagleY-AI home directory to your PC using the **winSCP** file copy program (available free of charge on the Internet).

8.4 Project 2 – Alternately Flashing LEDs

Description: This project is similar to the previous one but here two LEDs are used and they flash alternately every second. The aim of this project is to show how more than one LED can be connected to BeagleY-AI.

Block diagram: The block diagram of the project is shown in Figure 8.7

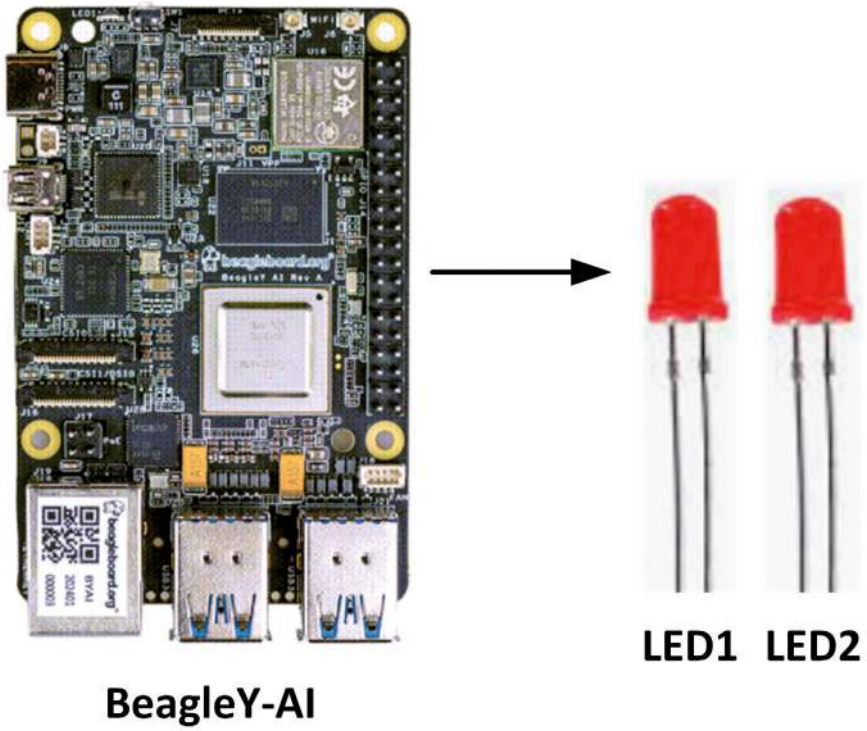


Figure 8.7 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.8. Two small LEDs are connected to port pins GPIO17 (pin 11) and GPIO27 (pin 13) of the BeagleY-AI through current limiting resistors.

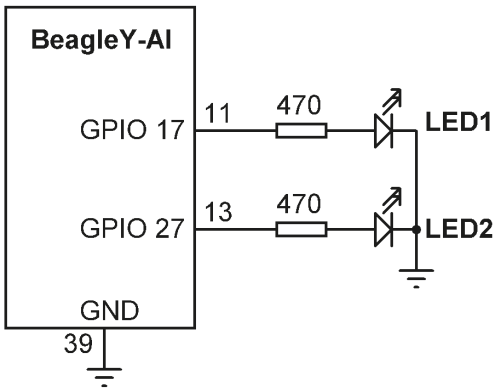


Figure 8.8 Circuit diagram of the project.

Program listing: The program is called **alternate.py** and the listing is shown in Figure 8.9. The program was written using the **nano** text editor. At the beginning of the program, **led1** and **led2** are linked to GPIO ports 17 and 27 respectively, and configured as outputs. The rest of the program is executed indefinitely in a **while** loop where the LEDs are turned on and off alternately, with a one-second delay between each output. Enter Ctrl+C to terminate the program.

```
#------------------------------------------------------
#
#                     FLASHING LEDs
#                     =============
#
# In this project two small LED is connected to GPIO17 and
# GPIO27 of BeagleY-AI. The program flashes the LEDs alternately
# every second.
#
# Program: alternate.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time library

led1 = gpiod.find_line('GPIO17')
led2 = gpiod.find_line('GPIO27')
led1.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
led2.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

while True:
  led1.set_value(1)                     # turn ON led1
  led2.set_value(0)                     # turn OFF led2
  time.sleep(1)                         # wait 1 second
```

```
led1.set_value(0)                    # turn OFF led1
led2.set_value(1)                    # turn ON led2
time.sleep(1)                        # wait 1 second
```

Figure 8.9 Program listing of the project.

8.5 Project 3 – Binary Counting with 8 LEDs

Description: In this project, 8 LEDs are connected to the BeagleY-AI GPIO pins. The LEDs count up in binary every second. The aim of this project is to show how 8 LEDs can be connected to the GPIO pins. In addition, the project shows how to group the LEDs as an 8-bit port and control them as a single port.

Block diagram: The block diagram of the project is shown in Figure 8.10.

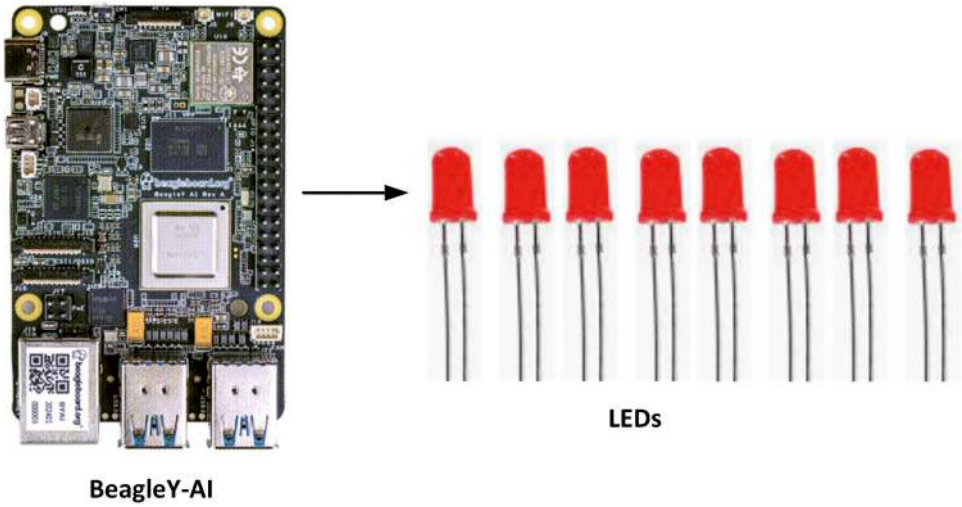


Figure 8.10 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.11. The LEDs are connected to 8 GPIO pins through 470 Ohm current-limiting resistors. The following 8 GPIO pins are grouped as an 8-bit port, where GPIO2 is configured as the LSB and GPIO9 is configured as the MSB:

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

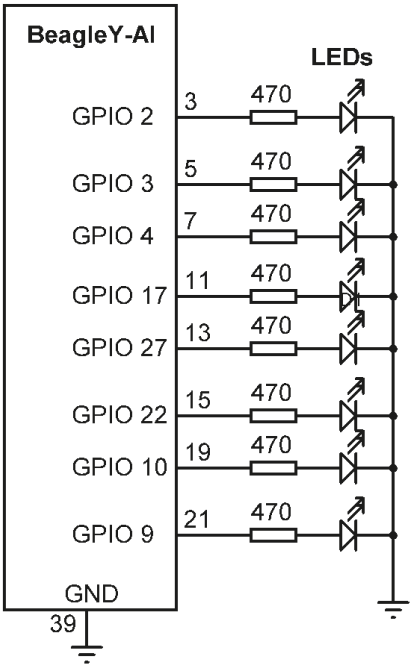


Figure 8.11 Circuit diagram of the project.

Construction: The project is constructed on a breadboard as shown in Figure 8.12. Notice that in this project, a **T-Cobbler** (Figure 8.13) connects to the 40-pin GPIO header of the BeagleY-AI through a ribbon cable. A T-type connector is used at the other side of this ribbon cable, which is plugged into a breadboard. This setup makes it very easy to connect to the BeagleY-AI header, especially when there are many connections to make. The GPIO pin names are written on the T-cobbler for ease of access.

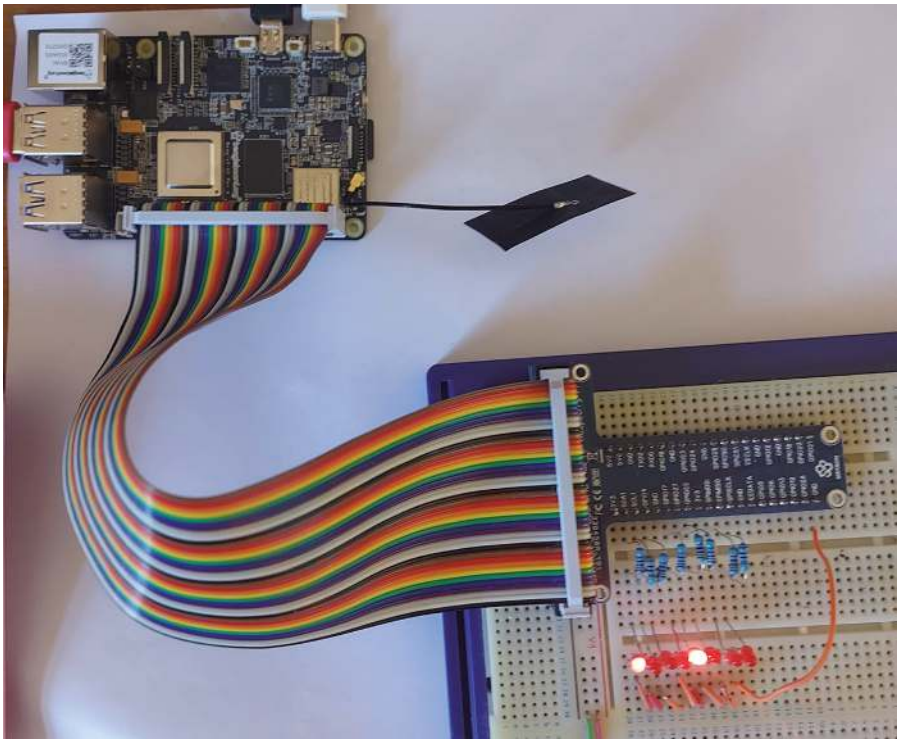


Figure 8.12 Constructing the project on a breadboard.

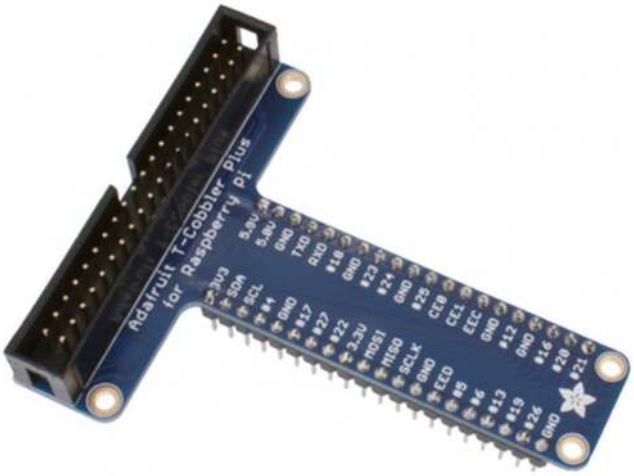


Figure 8.13 The T-Cobbler.

Program listing: The program is called **LEDCNT.py** and the listing is shown in Figure 8.14. The program was written using the **nano** text editor. At the beginning of the program, the LEDs are linked to GPIO ports and are configured as outputs. Inside the main program, a loop is formed to execute forever, and inside this loop, the LEDs count up by one in binary.

The variable **cnt** is used as the counter. Function **Port_Output** is used to control the LEDs. This function can take integer numbers from 0 to 255 and it converts the input number (x) into binary using the built-in function **bin**. Then the leading "0b" characters are removed from the output string **b** (the **bin** function inserts characters "0b" at the beginning of the converted string). Afterward, the converted string **b** is made up of 8 characters by inserting leading 0s. The string is then sent to the PORT bit by bit, starting from the least-significant bit (GPIO2) position. The result is that the 8 LEDs count up in binary.

```
#------------------------------------------------------------
#
#                      BINARY UP COUNTING LEDs
#                      =======================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# the LEDs count up in binary with one second delay between
# each output.
#
# Program: LEDCNT.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time

#
# LED connections
#
PORT = [0] * 8
PORT[0] = gpiod.find_line('GPIO9')
PORT[1] = gpiod.find_line('GPIO10')
PORT[2] = gpiod.find_line('GPIO22')
PORT[3] = gpiod.find_line('GPIO27')
PORT[4] = gpiod.find_line('GPIO17')
PORT[5] = gpiod.find_line('GPIO4')
PORT[6] = gpiod.find_line('GPIO3')
PORT[7] = gpiod.find_line('GPIO2')

PORT[0].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[1].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[2].request(consumer='beag;e',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[3].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
```

```
PORT[4].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[5].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[6].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
PORT[7].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

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

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].set_value(1)          # bit ON
      else:
         PORT[i].set_value(0)          # bit OFF
   return
#
# Main program loop. Count up in binary every second
#
cnt = 0

while True:
  Port_Output(cnt)                     # send cnt to port
  time.sleep(1)                        # wait 1 second
  cnt = cnt + 1                        # increment cnt
  if cnt > 255:
     cnt = 0
```

Figure 8.14 Program listing.

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

Modified Program

The program shown in Figure 8.14 can be modified and made more friendly by storing the LED port numbers in a list. The modified program, **LEDCNT2.py**, is shown in Figure 8.15. In this version, the LED port numbers are stored in the list **LED**. **PORT** is defined as a list having 8 elements. Inside the function **Configure()**, the LEDs are linked to GPIO ports and they are configured as outputs. Then, the function **Port_Output** is used, as before, to send the port data to the LEDs.

```
#-------------------------------------------------------------
#
#                      BINARY UP COUNTING LEDs
#                      =======================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# the LEDs count up in binary with one second delay between
# each output.
#
# In this version of the program the LEDs are grouped as an
# 8 bit port
#
# Program: LEDCNT2.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time

LED = [9, 10, 22, 27, 17, 4, 3, 2]
PORT = [0] * 8
#
# This function initializes the ports
#
def Configure():
  for i in range(8):
    PORT[i] = gpiod.find_line('GPIO'+str(LED[i]))
    PORT[i].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

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

   for i in range (0, 8):
      if b[i] == "1":
```

```
            PORT[i].set_value(1)          # bit ON
         else:
            PORT[i].set_value(0)          # bit OFF
      return
#
# Main program loop. Count up in binary every second
#
cnt = 0
Configure()

while True:
  Port_Output(cnt)                        # send cnt to port
  time.sleep(1)                           # wait 1 second
  cnt = cnt + 1                           # increment cnt
  if cnt > 255:
     cnt = 0
```

Figure 8.15 Modified program.

8.6 Project 4 – Christmas Lights (Random Flashing 8 LEDs)

Description: In this project, 8 LEDs are connected to BeagleY-AI GPIO pins, just as in Project 3. The LEDs flash randomly every 0.5 seconds just like fancy Christmas lights. The aim of this project is to show how to generate random numbers between 1 and 255.

The block diagram and circuit diagram of the projects are the same as those in Figure 8.10 and Figure 8.11 respectively.

Program listing: The program is called **xmas.py** and the listing is shown in Figure 8.16. The program was written using the **nano** text editor. At the beginning of the program, the **random** module and other required modules are imported to the program. Then, a loop is created to execute forever and inside this loop, a random number is generated between 1 and 255, and this number is used as an argument to function **Port_Output.** The binary pattern corresponding to the generated number is sent to the port, which turns the LEDs **on** or **off** in a random manner.

```
#-------------------------------------------------------------
#
#                      CHRISTMAS LIGHTS
#                      ================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port.The LEDs
```

```
# turn ON and OFF randomly after generating a random number 1-255
#
# Program: xmas.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time
import random                           # import random

LED = [9, 10, 22, 27, 17, 4, 3, 2]
PORT = [0] * 8
#
# This function initializes the ports
#
def Configure():
  for i in range(8):
    PORT[i] = gpiod.find_line('GPIO'+str(LED[i]))
    PORT[i].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

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

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].set_value(1)           # bit ON
      else:
         PORT[i].set_value(0)           # bit OFF
   return
#
# Main program loop
#

Configure()

while True:
  numbr = random.randint(1, 255)        # generate random number
  Port_Output(numbr)                    # send cnt to port
  time.sleep(0.5)                       # wait 0.5 second
```

Figure 8.16 Program listing.

Recommended modifications: Modify the program such that 10 LEDs can be connected to the BeagleY-AI board and flashed randomly.

8.7 Project 5 – Chasing LEDs

Description: In this project 8 LEDs are connected to the BeagleY-AI GPIO pins as in the previous project. As shown in Figure 8.17, the LEDs rotate (chase each other) from the LSB to MSB with one second delay between each output.

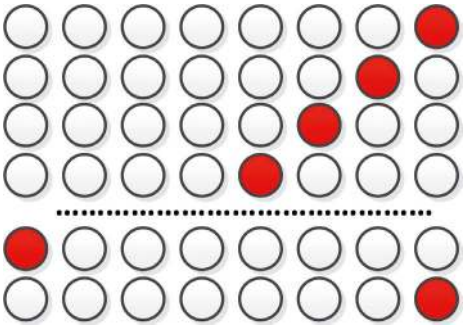


Figure 8.17 Chasing LEDs.

The block diagram and circuit diagram of the projects are same as in Figure 8.10 and Figure 8.11 respectively.

Program listing: The program is called **rotate.py** and the listing is shown in Figure 8.18. The program was written using the **nano** text editor. Inside the main program, a loop is created to execute indefinitely, and inside this loop, the variable **rot** is used as an argument to the **Port_Output** function. This variable is shifted left at each iteration, and thus the LED **on** sequence is from left to right (from LSB to MSB). A one-second delay is inserted between each output.

```
#-------------------------------------------------------------
#
#                      ROTATING LEDs
#                      =============
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port.The LEDs
# turn ON and OFF as if they are chasing each other
#
# Program: rotate.py
# Date   : October, 2024
# Author : Dogan Ibrahim
```

```
#--------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time

LED = [9, 10, 22, 27, 17, 4, 3, 2]
PORT = [0] * 8
#
# This function initializes the ports
#
def Configure():
  for i in range(8):
    PORT[i] = gpiod.find_line('GPIO'+str(LED[i]))
    PORT[i].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

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

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].set_value(1)           # bit ON
      else:
         PORT[i].set_value(0)           # bit OFF
   return
#
# Main program loop
#

Configure()
rot = 1

while True:
  Port_Output(rot)                      # send rot to port
  time.sleep(1)                         # wait 1 second
  rot = rot << 1                        # rotate rot
  if rot > 128:                         # if at the end
     rot = 1
```

Figure 8.18 Program listing.

8.8 Project 6 – Rotating LEDs with Pushbutton Switch

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

Block diagram: The block diagram of the project is shown in Figure 8.19.

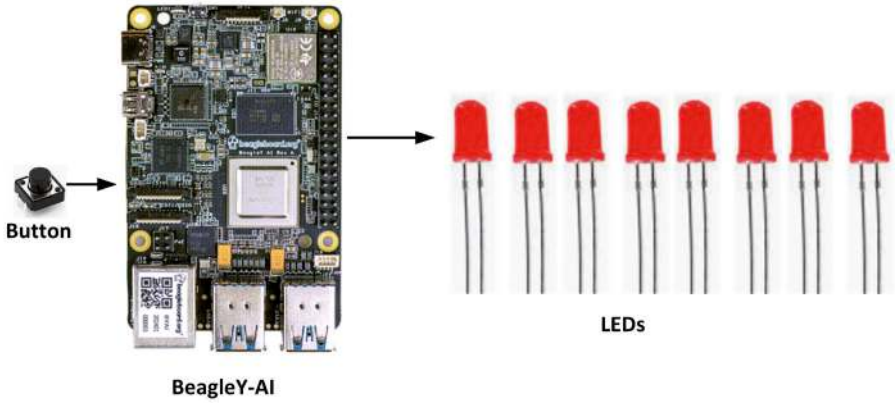


Figure 8.19 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.20. The LEDs are connected to 8 GPIO pins through 470-ohm current limiting resistors, as in the previous projects. The push-button switch is connected to GPIO11 (pin 23) of BeagleY-AI. The pushbutton switch is connected through a 10K and a 1K resistor. When the switch is not pressed, the input is at logic 1. When the switch is pressed, the input changes to logic 0. Notice that the 1K resistor is used here for safety in case the input channel is configured as an output by mistake. If this happens, without a resistor, the output would be short-circuited, which could damage the BeagleY-AI hardware.

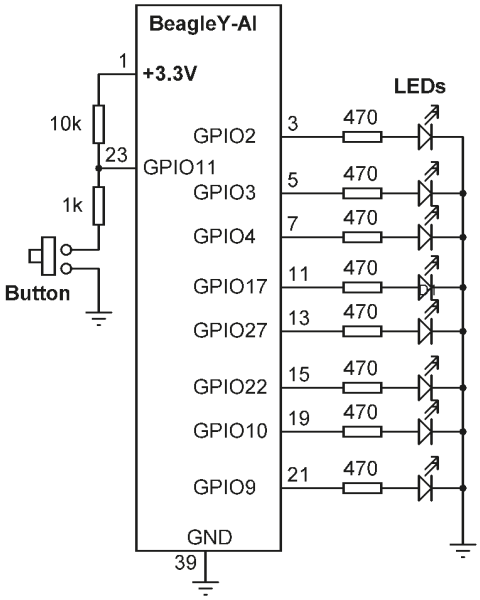


Figure 8.20 Circuit diagram of the project.

Construction: The project is constructed on a breadboard, as shown in Figure 8.21.

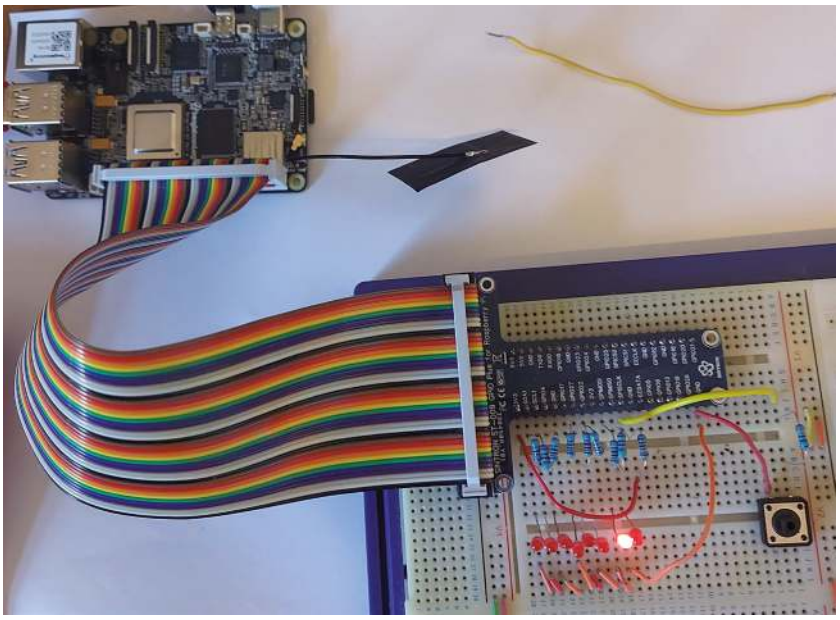


Figure 8.21 Project constructed on a breadboard.

Program listing: The program is called **buttonled.py** and the listing is shown in Figure 8.22. The program was written using the **nano** text editor. The LEDs are assigned as in the previous project. The button is assigned to port GPIO11 and is configured as an input. A **while** loop is created to execute indefinitely, and inside this loop, the variable **rot** is used as an argument to the **Port_Output** function. If the button is not pressed, then **rot** is shifted right, and the LED **on** sequence is from left to right (from MSB to LSB). If on the other hand, the button is pressed, then the LED **on** sequence is from right to left (from LSB to MSB). A one-second delay is inserted between each output.

```
#
#               ROTATING LEDs ITH PUSH BUTTON
#               =============================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# In addition, a puch button switch is connected to GPIO11.
# Normally the button is at logic 1 and goes to 0 when pressed.
# The LEDs rotate in one direction and when the button is pressed
# the direction of rotation is reversed. One second delay is inserted
#
# Program: buttonled.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time

LED = [9, 10, 22, 27, 17, 4, 3, 2]
PORT = [0] * 8
#
# This function initializes the ports
#
def Configure():
  for i in range(8):
    PORT[i] = gpiod.find_line('GPIO'+str(LED[i]))
    PORT[i].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

#
# Configure the button as input
#
button = gpiod.find_line('GPIO11')
button.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
```

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

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].set_value(1)          # bit ON
      else:
         PORT[i].set_value(0)          # bit OFF
   return
#
# Main program loop
#

Configure()
rot = 1

while True:
  Port_Output(rot)                     # send rot to port
  time.sleep(1)                        # wait 1 second
  if button.get_value()== 0:           # button=0 when pressed
    rot = rot << 1                     # rotate left
    if rot > 128:
        rot = 1
  else:
    rot = rot >> 1                     # rotate right
    if rot == 0:
      rot = 128
```

Figure 8.22 Program listing.

Note that the internal pull-up resistors can be enabled on input ports.

8.9 Project 7 – Morse Code Exerciser with LED or Buzzer

Description: In this project, an LED or a buzzer is connected to GPIO17 (pin11) of the BeagleY-AI. The user enters a text from the keyboard. The buzzer is then turned **on** and **off** to sound the letters of the text in Morse code.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.23, where an active buzzer is connected to GPIO11 of the BeagleY-AI.

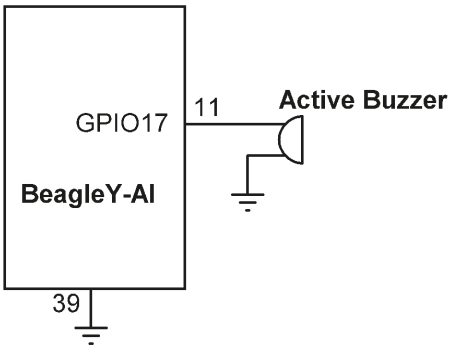


Figure 8.23 Circuit diagram of the project.

Morse Code: In Morse code, each letter is made up of dots and dashes. Figure 8.24 shows the Morse code of all the letters in the English alphabet (this table can be extended by adding the Morse code for numbers and punctuation marks). The following rules apply to the timing of dots and dashes:

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

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

Time (ms) = 1200/wpm

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

Letter	Morse code
A:	.-
B :	-...
C :	-.-.
D :	-..
E :	.
F :	..-.
G :	--.

```
H :     ....
I :     ..
J :     .---
K :     -.-
L :     .-..
M :     --
N :     -.
O :     ---
P :     .--.
Q :     --.-
R :     .-.
S :     ...
T :     -
U :     ..-
V :     ...-
W :     .--
X :     -..-
Y :     -.--
Z :     --..
```

Figure 8.24 Morse code of English letters.

Program listing: The program is called **morse.py** and the listing is shown in Figure 8.25. The Morse code alphabet is stored in list **Morse_Code**. Function **DO_DOT** implement a single dot with a duration of one unit time. The function **DO_DASH** implements a single dash with duration of 3 unit times. The function **DO_SPACE** implements a space character with duration of 7 unit times. The rest of the program is executed in a loop where a text is read from the keyboard, and the buzzer sounds in such a way to represent the Morse code of this text. The program terminates if the user enters the text **QUIT**.

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

```
beagle@beagle:~ $ python morse.py
```

```
#------------------------------------------------------------------
#
#                  MORSE CODE EXERCISER
#                  ====================
#
# This project can be used to learn the Morse code. A buzzer is
# connected to GPIO17 of the BeagleY-AI.
#
# The program reads a text from the keyboard and then sounds the
# buzzer to simulate sending or receiving the Morse code of this
# text.
#
# In this project the Morse code speed is assumed to be 10 wpm,
```

```
# but can easily be changed by changing the parameter wpm.
#
# File  : morse.py
# Date  : October, 2024
# Author: Dogan Ibrahim
#-------------------------------------------------------------------
import gpiod
import time

Buzzer = gpiod.find_line('GPIO17')
Buzzer.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

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

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

#
# This function sends a DOT (unit time)
```

```
#
def DO_DOT():
   Buzzer.set_value(1)
   time.sleep(unit_time)
   Buzzer.set_value(0)
   time.sleep(unit_time)
   return

#
# This function sends a DASH ( 3*unit time)
#
def DO_DASH():
   Buzzer.set_value(1)
   time.sleep(3*unit_time)
   Buzzer.set_value(0)
   time.sleep(unit_time)
   return

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

#
# Main program code
#
text = ""
while text != "QUIT":
   text = input("Enter text to send: ")
   if text != "QUIT":
      for letter in text:
         if letter == ' ':
            DO_SPACE()
         else:
            for code in Morse_Code[letter.upper()]:
               if code == '-':
                  DO_DASH()
               elif code == '.':
                  DO_DOT()
               time.sleep(unit_time)
         time.sleep(3*unit_time)
   time.sleep(2)
```

Figure 8.25 Program listing of the project.

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

8.10 Project 8 – Electronic Dice

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

Block diagram: The block diagram of the project is shown in Figure 8.26.

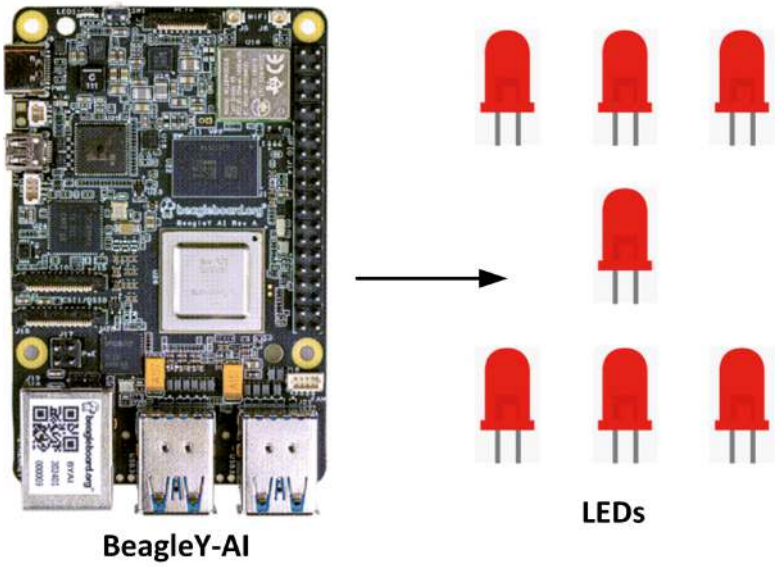


Figure 8.26 Block diagram of the project.

Figure 8.27 shows the LEDs that should be turned **on** to display the 6 dice numbers.

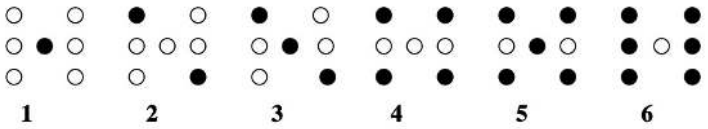


Figure 8.27 LED Dice.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.28. Here, 8 GPIO pins are collected together to form a PORT. The following pins are used for the LEDs (there are 7 LEDs, but 8 port pins are used in the form of a byte where the most-significant bit position is not used):

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

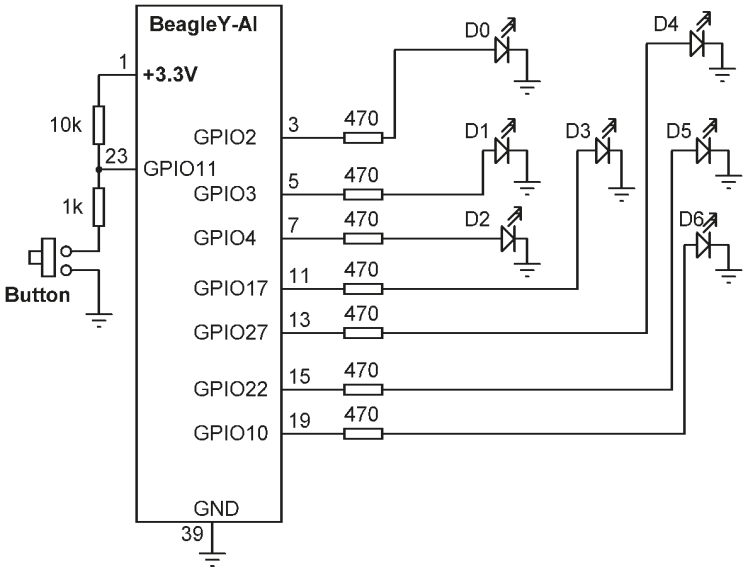


Figure 8.28 Circuit diagram of the project.

The push-button switch is connected to GPIO port pin GPIO11.

Table 8.1 gives the relationship between a dice number and the corresponding LEDs to be turned **on** to imitate the faces of a real dice. For example, to display number 1 (i.e., only the middle LED is **on**), you have to turn LED D3 **on**. Similarly, to display number 4, you have to turn **on** D0, D2, D4 and D6.

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

Table 8.1 Dice number and LEDs to be turned **on**

The relationship between the required number and the data to be sent to the PORT to turn on the correct LEDs is given in Table 8.2. For example, to display dice number 2, you have to send hexadecimal 0x41 to the PORT. Similarly, to display number 5, we have to send hexadecimal 0x5D to the PORT and so on.

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

Table 8.2 Required number and PORT data

Program listing: The program is called **dice.py** and the listing is shown in Figure 8.29. The bit pattern to be sent to the LEDs corresponding to each dice number is stored in hexadecimal format in a list called DICE_NO (see Table 8.2). GPIO 1 is configured as a button pin, and the push-button switch is connected to this pin to simulate the "throwing" of a dice. The main program waits until a button is pressed. Then, a random number is generated between 1 and 6 and stored in variable **n**. The bit pattern corresponding to this number is found and sent to function **Port_Output** so that the required LEDs are turned on to represent the dice number. This process is repeated after 3-seconds of delay.

```
#-------------------------------------------------------------
#
#               ELECTRONIC DICE WITH LEDs
#               =========================
#
# Yhis is an electronic dice project. A button is connected to
# GPIO11 of the BeagleY-AI. A random number is generated between
# 1 and 6 when the button is pressed. The dice number is displayed
# on 7 LEDs configured as the faces of a dice.
#
# Program: dice.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import gpiod                            # import gpiod
import time                             # import time
import random                           # import random

DICE_NO = [0, 0x08, 0x41, 0x49, 0x55, 0x5D, 0x77]
LED = [9, 10, 22, 27, 17, 4, 3, 2]
PORT = [0] * 8

#
# This function initializes the ports
#
def Configure():
```

```
  for i in range(8):
    PORT[i] = gpiod.find_line('GPIO'+str(LED[i]))
    PORT[i].request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

#
# Configure the button as input
#
button = gpiod.find_line('GPIO11')
button.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

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

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].set_value(1)          # bit ON
      else:
         PORT[i].set_value(0)          # bit OFF
   return
#
# Main program loop
#

Configure()

while True:
  if button.get_value() == 0:          # if button is pressed
    n = random.randint(1, 6)           # generate random no
#    print(n)                          # display it
    pattern=DICE_NO[n]                 # get the pattern
    Port_Output(pattern)               # display the pattern
    time.sleep(3)                      # wait 3 sconds
    Port_Output(0)                     # clear display
```

Figure 8.29 Program listing of the project.

8.11 Project 9 – Varying the LED Flashing Rate

Description: In this project, an LED and two pushbuttons are connected to the BeagleY-AI board. Normally, the LED flashes every second. Pressing the **Faster** button increases the flashing rate. Similarly, pressing the **Slower** button decreases the flashing rate. The aim of this project is to show how more than one pushbutton can be connected to the BeagleY-AI board.

Block diagram: The block diagram of the project is shown in Figure 8.30.

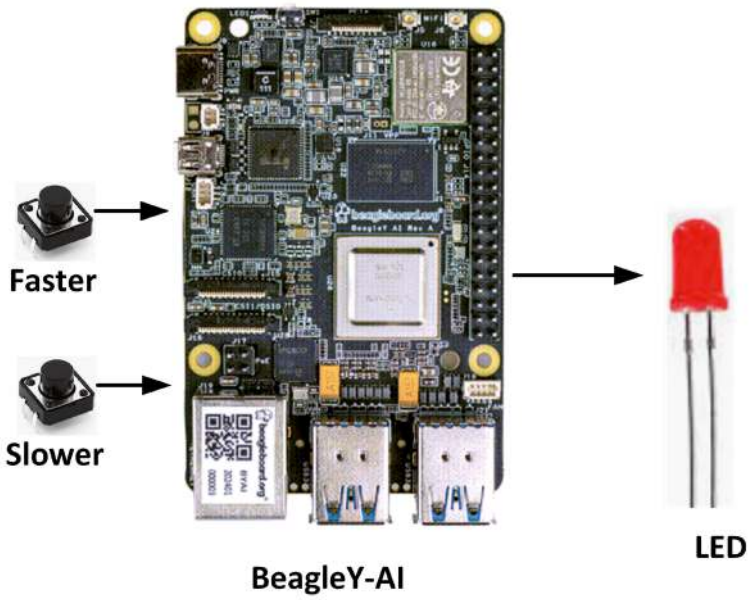


Figure 8.30 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.31. Here, the LED is connected to GPIO17, the **Faster** button to GPIO9, and the **Slower** button to GPIO11. The button states are at logic 1 and go to logic 0 when pressed.

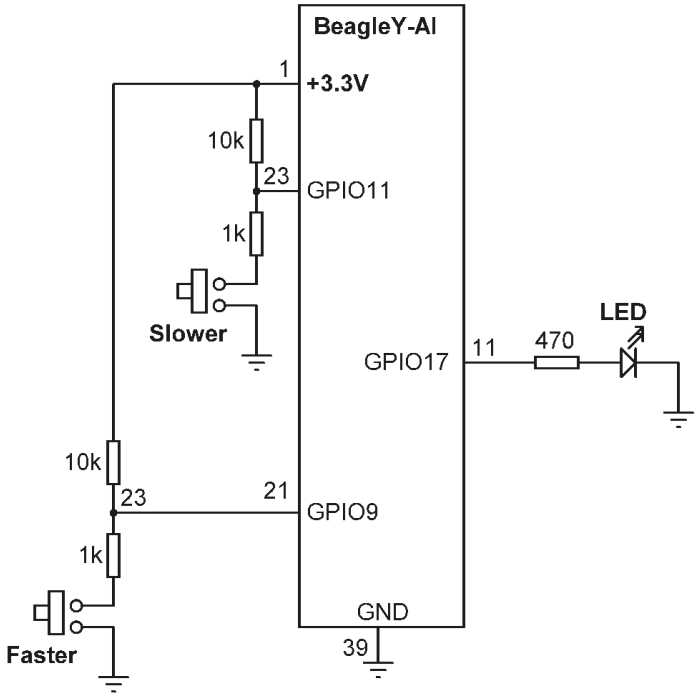


Figure 8.31 Circuit diagram.

Program listing: The program is called **sfled.py** and the listing is shown in Figure 8.32. At the beginning of the program, the LED and button ports are identified. The LED is configured as output, and the two buttons are configured as inputs. The remainder of the program runs in a loop. Here, pressing the **Faster** button increases the flashing rate by decreasing the delay by 0.2 seconds, and pressing the **Slower** button increases the delay by 0.2 seconds.

```
#------------------------------------------------------------
#
#               VARYING LED FLASHING RATE
#               =========================
#
# In this project an LEDs and two buttons named Faster and Slower
# are connected to the BeagleY-AI. Normally the LED flashes every
# second. Pressing Faster increases the flashing rate. Similarly,
# pressing Slower decreases the flashing rate.
#
# Program: sfled.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#------------------------------------------------------------
import gpiod                            # import gpiod
```

```
import time                              # import time

LED = gpiod.find_line('GPIO17')
LED.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

Faster = gpiod.find_line('GPIO9')
Slower = gpiod.find_line('GPIO11')
Faster.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
Slower.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

#
# Main program loop
#
dl = 1                                   # default delay

while True:
  LED.set_value(1)                       # LED ON
  time.sleep(dl)                         # delay
  LED.set_value(0)                       # LED OFF
  time.sleep(dl)                         # delay
  if Faster.get_value() == 0:            # request for faster
    dl = dl - 0.2                        # lower delay
    if dl <= 0:
      dl = 0.1
  if Slower.get_value() == 0:            # request for slower
    dl = dl + 0.2                        # increase delay
```

Figure 8.32 Program listing.

Testing: Keep the **Faster** button pressed and you should see the LED flashing faster. Similarly, keep the **Slower** button pressed, and the LED should flash slower.

Chapter 9 • Using an I²C LCD

9.1 Overview

The I2C (also known as I²C) bus is commonly used in microcontroller-based projects. In this chapter, you will be looking at the use of this bus on BeagleY-AI. Some other interesting projects are also given in this chapter. The aim is to make the reader familiar with the I²C bus library functions and to show how they can be used in a real project. Before looking at the details of the projects, it is worthwhile to look at the basic principles of the I²C bus.

9.2 The I²C Bus

The I²C bus is one of the most commonly used microcontroller communication protocols for communicating with external devices such as sensors and actuators. The I²C bus is a single master, multiple slave bus, and it can operate at standard mode: 100 Kbit/s, full speed: 400 Kbit/s, fast mode: 1 Mbit/s, and high speed: 3.2 Mbit/s. The bus consists of two open-drain wires, pulled up with resistors:

SDA: data line
SCL: clock line

Figure 9.1 shows the structure of an I²C bus with one master and three slaves.

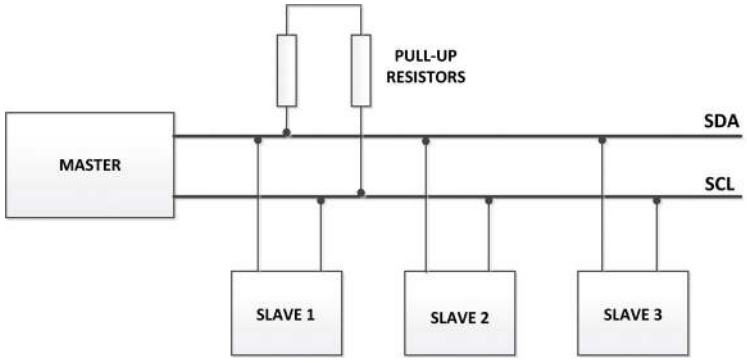


Figure 9.1 I²C bus with one master and three slaves.

Because the I²C bus is based on just two wires, there should be a way to address an individual slave device on the same bus. For this reason, the protocol defines that each slave device provides a unique slave address for the given bus. This address is usually 7-bits wide. When the bus is free, both lines are **high**. All communication on the bus is initiated and completed by the master which initially sends a **start** bit and completes a transaction by sending a **stop** bit. This alerts all the slaves that some data is coming on the bus, and all the slaves listen on the bus. After the **start** bit, 7 bits of the unique slave address are sent. Each slave device on the bus has its own address, and this ensures that only the addressed slave communicates on the bus at any time to avoid any collisions. The last sent bit is a **read/write (R/W)** bit, such that if this bit is **0**, it means that the master wishes to write to the bus (e.g., to a register of a slave); if this bit is **1**, it means that the master wishes to read from the bus (e.g., from the register of a slave). The data is sent on

the bus with the **MSB** (most significant bit) first. An **acknowledgment (ACK)** bit takes place after every byte, and this bit allows the receiver to signal to the transmitter that the byte was received successfully, allowing the transmission of another byte may be sent. The **ACK bit** is sent at the 9th clock pulse.

The communication over the I²C bus is simply as follows:

- The master sends on the bus the address of the slave it wants to communicate with.
- The LSB is the R/W bit which establishes the direction of data transmission, i.e., from **master to slave (R/W = 0)**, or from **slave to master (R/W = 1)**.
- Required bytes are sent, each interleaved with an ACK bit, until a stop condition occurs

Depending on the type of slave device used, some transactions may require separate transactions. For example, the steps to read data from an I²C compatible memory device are:

- The master starts the transaction in write mode (R/W = 0) by sending the slave address on the bus.
- The memory location to be retrieved is then sent as two bytes (assuming 64Kbit memory).
- The master sends a STOP condition to end the transaction.
- The master starts a new transaction in read mode (R/W = 1) by sending the slave address on the bus.
- The master reads the data from the memory. If reading the memory in sequential format, then more than one byte will be read.
- The master sets a stop condition on the bus.

9.3 I²C Pins of BeagleY-AI

BeagleY-AI I²C port is at the following GPIO pins:

GPIO2	SDA1	pin 3
GPIO3	SCL1	pin 5
GPIO25	SDA4	pin 22
GPIO22	SCL4	pin 15

There are also I²C pins at the GPIO0 and GPIO1, but these are shared with other modules on the board, and using them as an I2C is not recommended.

2.2 kΩ pull-up resistors are used from the I²C pins to +3.3 V. Notice that because the I²C pins are pulled-up to +3.3 V and BeagleY-AI pins are not +5 V compatible, it is necessary to use voltage level converter circuits if the I²C LCD operates with +5 V.

9.4 Project 1 – Using an I²C LCD – Seconds Counter

Description: In this project, an I²C type LCD is connected to the BeagleY-AI. The program counts up in seconds and displays on the LCD. The aim of this project is to show how an I²C-type LCD can be used in projects.

The I²C LCD

The I²C LCD has 4 pins: GND, +V, SDA, and SCL. SDA can be connected to pin GPIO2, and SCL to pin GPIO 3. The +V pin of the display should be connected to the **+**5 V (pin 2) of the BeagleY-AI. BeagleY-AI GPIO pins are not +5 V tolerant, but the I²C LCD operates with +5V where its SDA and SCL pins are pulled to +5 V. It is not a good idea to connect the LCD directly to BeagleY-AI as it can damage its I/O circuitry. There are several solutions here. One solution is to remove the I²C pull-up resistors on the LCD module. The other option is to use an LCD that operates with +3.3 V. Another solution is to use a bidirectional +3.3 V to +5 V logic level converter chip. In this project, you will use the TXS0102 bidirectional logic

level converter chip, like the one shown in Figure 9.2.

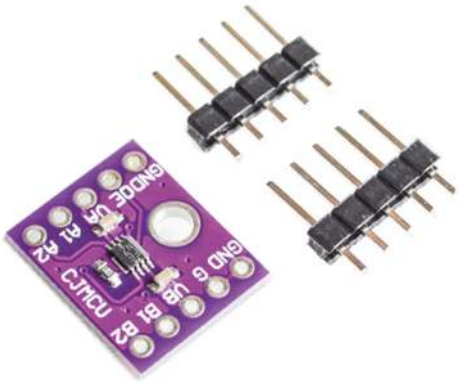


Figure 9.2 Logic level converter.

Block diagram: Figure 9.3 shows the block diagram of the project.

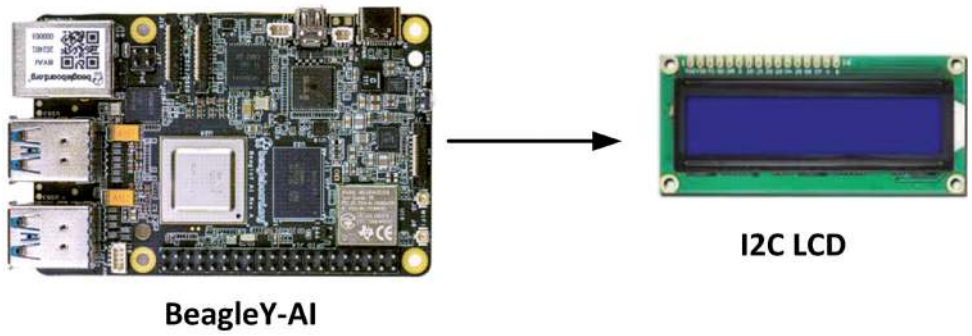


Figure 9.3 Block diagram.

Circuit diagram: The circuit diagram is shown in Figure 9.4.

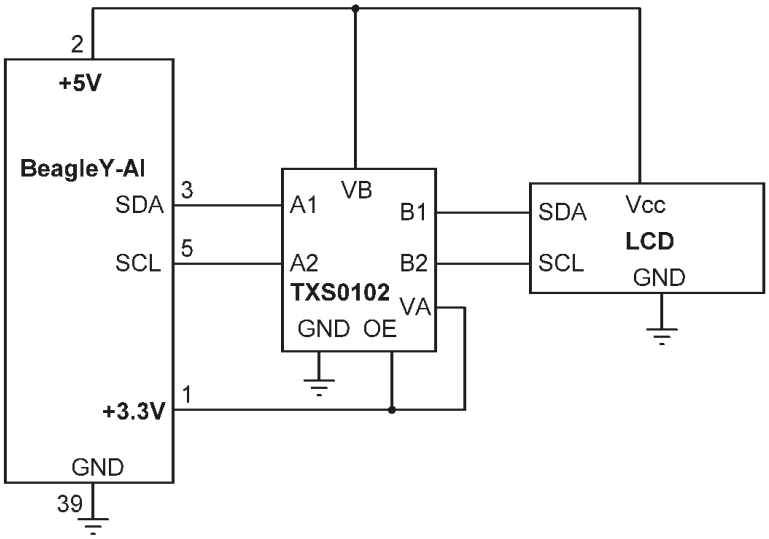


Figure 9.4 Circuit diagram of the project.

Figure 9.5 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.

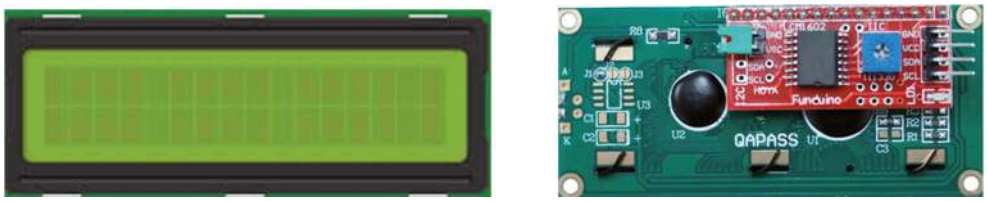


Figure 9.5 I²C-based LCD (front and back views)

Program Listing: Before developing the program, make sure that you have the latest version of the i2c tools and smbus. Enter the following command:

```
beagle@beagle:~ $ sudo apt-get install i2c-tools
beagle@beagle:~ $ sudo apt-get install python3-smbus
```

Connect the LCD to your BeagleY-AI as shown in Figure 9.4. Then, enter the following command to ensure that the LCD is detected by your BeagleY-AI:

```
beagle@beagle:~ $ sudo i2cdetect –r –y 1
```

You should see a table similar to the one shown below. A number in the table means that the LCD has been recognized correctly and the I^2C slave address of the LCD is shown in the table. In this example the LCD address is 27:

```
     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: --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

You should now install an I^2C LCD library so that you can send commands and data to the LCD. There are many Python libraries available for the I^2C type LCDs. The one chosen here is on GitHub from Dave Hylands. This library is installed as follows:

- Go to the following web link:

  https://github.com/dhylands/python_lcd/tree/master/lcd

- Copy the following files to your home directory **/home/beagle** using WinSCP:

  i2c_lcd.py
  lcd_api.py

- Check to make sure that the file is copied successfully. You should see the file listed with the command:

  beagle@beagle: ~ $ **ls**

You are now ready to write the program. Figure 9.6 shows the program listing (**lcd.py**). At the beginning of the program, the LCD driver libraries **lcd_api** and **i2c_lcd** are imported into the program. The heading SECONDS COUNTER is displayed at the top row (row 1) and

the program enters a loop. Inside this loop, variable **cnt** is incremented every second and the total value of **cnt** is displayed on the LCD continuously in the following format:

```
SECONDS COUNTER
nn
```

```
#-------------------------------------------------------
#             I2C LCD SECONDS COUNTER
#             ======================
#
# In this program an I2C LCD is connected to the BeagleY-AI.
# The program counts up in seconds and displays on the LCD.
#
# At the beginning of the program the text SECONDS COUNTER is
# displayed
#
# Program: lcd.py
# Date   : October 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------
import smbus
import time
from lcd_api import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)

mylcd.clear()                                  # clear LCD
mylcd.putstr("SECONDS COUNTER")                # display string

cnt = 0                                        # initialize cnt

while True:                                    # infinite loop
  cnt = cnt + 1                                # increment count
  mylcd.move_to(0,1)
  mylcd.putstr(str(cnt))                       # display cnt
  time.sleep(1)                                # wait one second
```

Figure 9.6 Program listing.

Figure 9.7 shows the display.

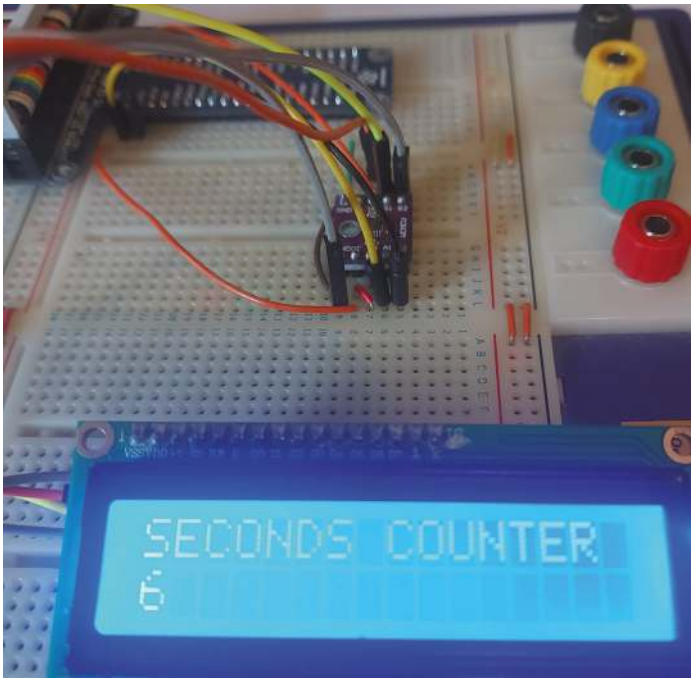


Figure 9.7 The LCD display.

The I²C LCD library supports many functions. Some of the most commonly used ones include (refer to the LCD library documentation for further details):

clear()	clear LCD and set to home position
show_cursor()	show cursor
hide_cursor()	hide cursor
blink_cursor_on()	blink cursor
blink_cursor_off()	stop blinking cursor
display_on()	display on
display_off()	display off
backlight_on()	backlight on
backlight_off()	backlight off
move_to(x, y)	move cursor to (x, y)
putchar()	display a character
putstr()	display a string

9.5 Project 2 – Using an I²C LCD – Display Time

Description: In this project, an I²C type LCD is connected to the BeagleY-AI as in the previous project. The program displays the current time on the LCD.

The block diagram and circuit diagram are as in Figure 9.3 and Figure 9.4 respectively.

Program listing: Figure 9.8 shows the program listing (**LCDtime.py**). At the beginning of the program, **time**, **datetime**, and **I2C LCD** modules are imported into the program. The LCD is cleared, and the program enters a loop. Inside this loop, the current time is extracted using the **strftime()** function, and the current time is then displayed on the top row of the LCD every second in the following format:

hh:mm:ss

```
#             I2C LCD TIME DISPLAY
#             ======================
#
# This program displays the current time on the LCD.
#
# Program: LCDtime.py
# Date   : October 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
from datetime import datetime
from  lcd_api  import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()                           # clear LCD

while True:                             # infinite loop
  now = datetime.now()
  time = now.strftime("%H:%M:%S")
  mylcd.move_to(0,0)
  mylcd.putstr(str(time))
  sleep(1)                              # wait one second
  mylcd.clear()
```

Figure 9.8 Program listing.

Figure 9.9 shows the display.

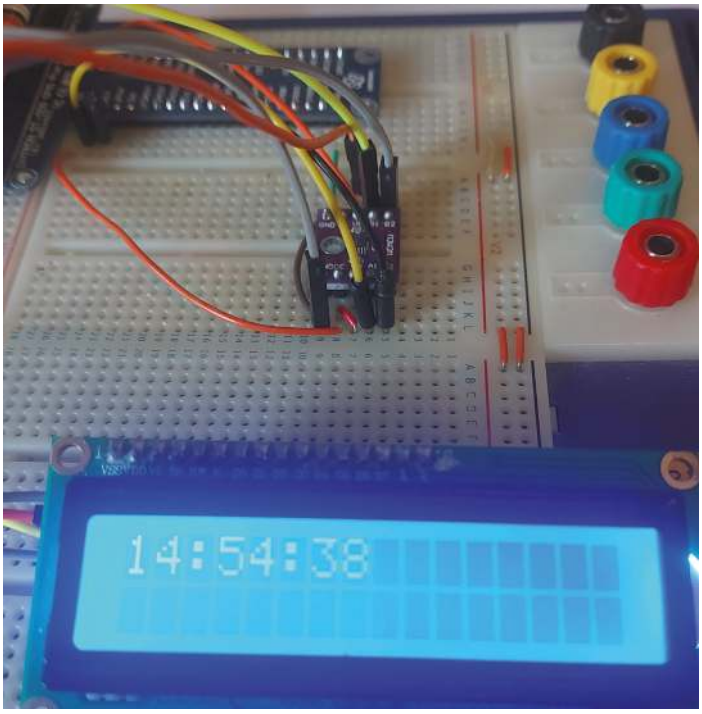


Figure 9.9 LCD display.

9.6 Project 3 – Using an I²C LCD – Display the IP address of BeagleY-AI

Description: In this project, an I²C-type LCD is connected to the BeagleY-AI as in the previous projects. The IP address of the BeagleY-AI is displayed on the top row of the LCD.

The block diagram and circuit diagram are as in Figure 9.3 and Figure 9.4, respectively.

Program listing: Figure 9.10 shows the program listing (**LCDip.py**). The IP address is extracted using the **hostname** command with the **–I** option. The IP address is then displayed on the LCD in the following format:

```
192.168.3.196
```

```
#------------------------------------------------------------
#
#             I2C LCD IP DISPLAY
#             ==================
# This program displays the IP address on the LCD.
#
# Program: LCDip.py
# Date   : October 2024
# Author : Dogan Ibrahim
```

```
#--------------------------------------------------------
from time import sleep
from subprocess import check_output
from  lcd_api import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

ip = check_output(["hostname", "-I"],encoding="utf-8").split()[0]
mylcd.putstr(str(ip))

while True:
   pass
```

Figure 9.10 Program listing.

9.7 Project 4 – Reaction Timer – Output to Screen

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

Block Diagram: Figure 9.11 shows the block diagram of the project.

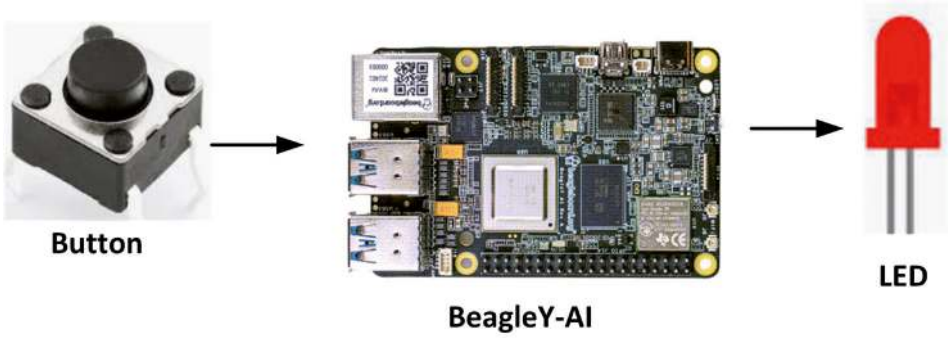


Figure 9.11 Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is very simple, and it consists of an LED and a push-button switch. The LED and the button are connected to GPIO17 and GPIO27, respectively. The button is connected using two resistors as shown in Figure 9.12.

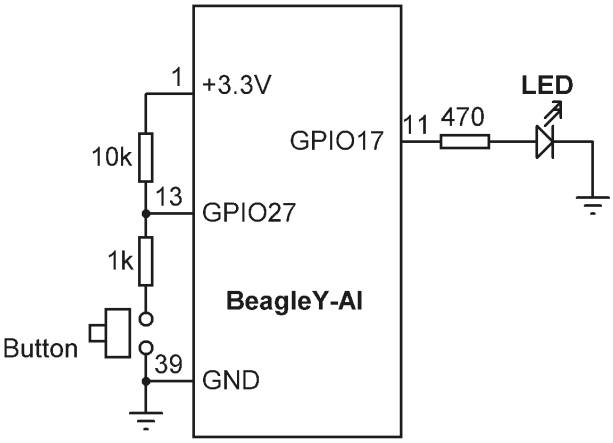


Figure 9.12 Circuit diagram of the project.

Program listing: The program is called **reaction.py** and its listing is shown in Figure 9.13. At the beginning of the program, the random library and other used libraries are imported. The program runs in a loop where the system time is recorded as soon as the LED is turned **on**. The program waits for the user to press the button, and the system time is read again at this moment. The difference between the second time and the first time is displayed as the reaction time of the user. This process repeats after a random delay of 1 to 10 seconds. Note that the floating point function **time.time()** returns the time in seconds since the epoch.

```
#------------------------------------------------------------
#
#                 REACTION TIMER
#                 ==============
#
# This is a reaction timer program. The user presses a button
# as soon as he/she see a light. The time between seeing the
# light and pressing the button is measured and is displayed
# in milliseconds as the reaction time of the user. The light
# comes ON after a random number of seconds between 1 and 10
# seconds.
#
# Program: reaction.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#------------------------------------------------------------
import time
import random
import gpiod
import time
```

```
button = gpiod.find_line('GPIO27')
led = gpiod.find_line('GPIO17')
led.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
button.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

# Start of main program
#
while True:
   T = random.randint(1, 10)                    # generate random no
   time.sleep(T)
   led.set_value(1)                             # LED ON
   start_time = time.time()                     # start time
   while(button.get_value() == 1):              # wait until pressed
      pass
   end_time = time.time()
   diff_time = 1000.0*(end_time - start_time)
   diff_int = int(diff_time)
   print(«Reaction time=%d ms» %diff_int)
   led.set_value(0)                             # LED OFF
   time.sleep(3)                                # wait 3 seconds
```

Figure 9.13 Program listing.

An example output is shown in Figure 9.14.

```
beagle@beagle:~$ python reaction.py
Reaction time=678 ms
Reaction time=2242 ms
Reaction time=4076 ms
Reaction time=387 ms
```

Figure 9.14 Example output.

9.8 Project 5 – Reaction Timer – Output to LCD

Description: This project is very similar to the previous one but here the output is sent to LCD instead of the screen. As before, the user presses a button as soon as he/she sees a LED lighting. The time delay between seeing the light and pressing the button is measured and displayed on the LCD. The LED then turns OFF and the process is repeated after a random delay of 1 to 10 seconds.

Block Diagram: Figure 9.15 shows the block diagram of the project.

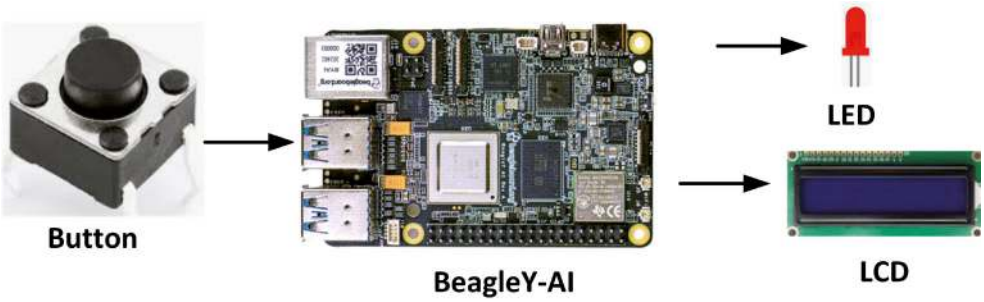


Figure 9.15 Block diagram of the project.

Circuit Diagram: The circuit diagram of the project, shown in Figure 9.16, is very simple and it consists of an LED, a push-button switch, and an LCD display. The LED and the button are connected to GPIO17 and GPIO27 respectively, as in the previous project.

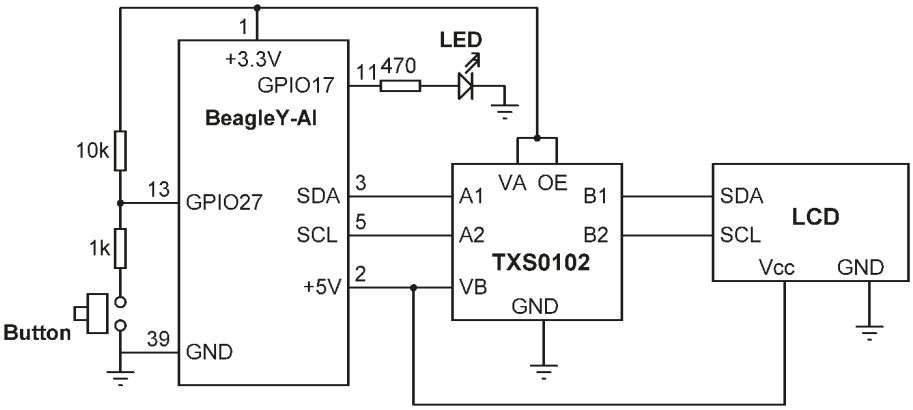


Figure 9.16 Circuit diagram of the project.

Program listing: The program is called **LCDreaction.py** and its listing is shown in Figure 9.17. The program is basically the same as the one in Figure 9.27, but here the output is sent to the LCD.

```
#
#                 REACTION TIMER - OUTPUT TO LCD
#                 ==============================
#
# This is a reaction timer program. The user presses a button
# as soon as he/she see a light. The time between seeing the
# light and pressing the button is measured and is displayed
# on LCD in milliseconds as the reaction time of the user. The
# light comes ON after a random number of seconds between 1 and
# 10 seconds.
#
```

```
# Program: LCDreaction.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import time
import random
import gpiod

from  lcd_api import  LcdApi
from i2c_lcd import  I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
mylcd.clear()

button = gpiod.find_line('GPIO27')
led = gpiod.find_line('GPIO17')
led.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
button.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

# Start of main program
#
while True:
   T = random.randint(1, 10)
   time.sleep(T)
   led.set_value(1)                          # LED ON
   start_time = time.time()                  # start time
   while(button.get_value() == 1):
     pass
   end_time = time.time()
   diff_time = 1000.0*(end_time - start_time)
   diff_int = str(int(diff_time)) + " ms"
   mylcd.move_to(0, 0)
   mylcd.putstr(diff_int)
   led.set_value(0)                          # LED OFF
   time.sleep(3)
   mylcd.clear()
```

Figure 9.17 Program listing.

9.9 Project 6 – Automatic Dusk Lights

Description: In this project, a light dependent resistor (LDR) is used to sense the darkness, and a relay is activated when the ambient light intensity falls below the required level. It is possible to connect e.g. lights to the relay so that they turn **on** automatically when, for example, it is dusk. The aim of this project is to show how to use an LDR in a BeagleY-AI project, and also how to connect and activate a relay.

Block Diagram: Figure 9.18 shows the block diagram of the project.

Figure 9.18 Block diagram of the project.

Circuit Diagram: As shown in Figure 9.19, the circuit diagram of the project is simple and it consists of an LDR, a 10-kilo ohm potentiometer, and a relay. The LDR is connected to GPIO4, and the relay to GPIO17.

The resistance of an LDR increases as the light level falls. The response of a typical LDR is shown in Figure 9.20. The LDR is connected as a resistive potential divider circuit. The voltage across the LDR increases as the light level falls. When dark, logic 0 will be sent to the BeagleY-AI which in turn will activate the relay. In light conditions, logic 1 will be sent to the BeagleY-AI, which will deactivate the relay. The potentiometer can be adjusted so that the relay is activated at the required light level. This process will require some trial and error.

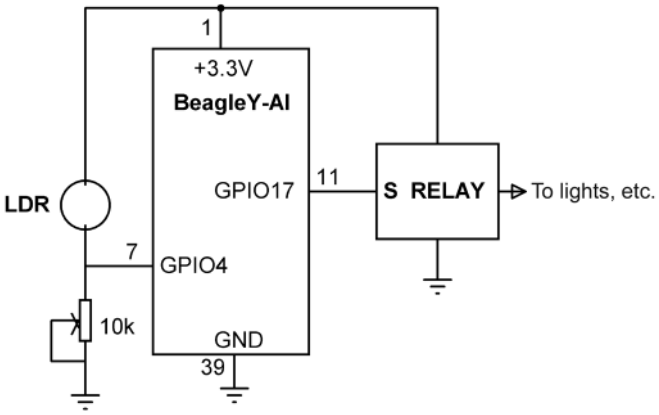


Figure 9.19 Circuit diagram of the project.

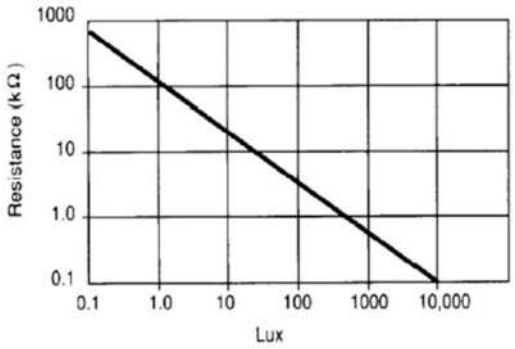


Figure 9.20 Response of a typical LDR.

Program listing: Figure 9.21 shows the program listing (program: **dusklight.py**). The LDR is the input, and the relay is the output. The program detects the voltage at its GPIO4 pin and if it is at logic 0 (i.e. dark) then it deactivates the relay, otherwise, the relay is activated. The potentiometer can be used to adjust the required light trigger level.

```
#-------------------------------------------------------------
#
#                   DUSK LIGHT
#                   ==========
#
# In this project a light dependent resistor (LDR) is used to
# detect the ambient light level. When the light level falls
# below the required value, a relay is activated which turns
# ON the lights.
#
# The potentiometer can be used to adjust the triggering
# light level of the project.
#
```

```
# Program: dusklight.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import gpiod

LDR = gpiod.find_line('GPIO4')
LDR.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
RELAY = gpiod.find_line('GPIO17')
RELAY.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

RELAY.set_value(0)                    # RELAY OFF)

while True:
   if LDR.get_value() == 0:
      RELAY.set_value(1)              # At logic 0 (dark)
   else:
      RELAY.set_value(0)              # At logic 1 (light)
```

Figure 9.21 Program listing.

9.10 Project 7 – Ultrasonic Distance Measurement

Description: This project uses an ultrasonic transmitter/receiver pair to measure the distance in front of the sensor. The distance is displayed on the screen. The aim of the project is to show how ultrasonic sensors can be attached to a BeagleY-AI and how distance can be measured using these sensors.

Block diagram: Figure 9.22 shows the block diagram of the project.

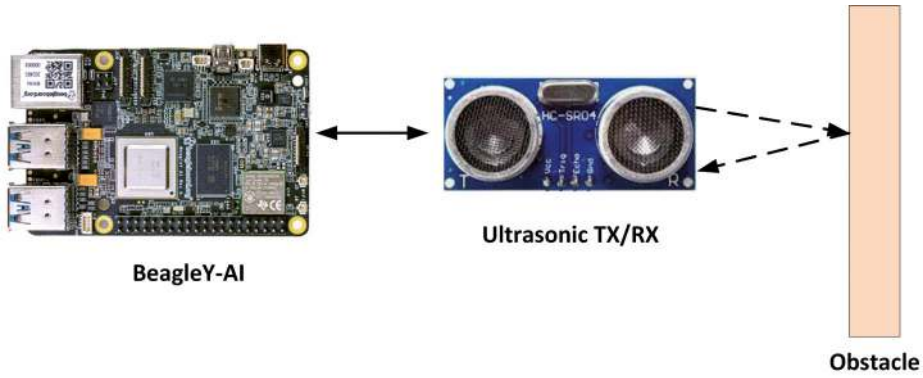


Figure 9.22 Block diagram of the project.

Circuit Diagram: An ultrasonic sensor is used to sense the distance in front of the sensor. The outputs of the ultrasonic sensors are +5 V and therefore are not compatible with the inputs of the BeagleY-AI. A resistive potential divider circuit is used to lower the voltage to +3.3 V. The voltage at the output of the potential divider resistor is:

Vo = 5 V × 2 K / (2 K + 1 K) = 3.3 V

In this project, an HC-SR04-type ultrasonic transmitter/receiver module is used (see Figure 9.23). These modules have the following specifications:

- Operating voltage (current): 5 V (2 mA) operation
- Detection distance: 2 cm – 450 cm
- Input trigger signal: 10 us TTL
- Sensor angle: not more than 15 degrees

The sensor modules have the following pins:

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

Figure 9.23 Ultrasonic transmitter/receiver module.

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

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

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

The speed of sound is 343 m/s, or 0.0343 cm/μs

Therefore,

Distance to object (in cm) = (time to received echo in μs) × 0.0343 / 2

or,

Distance to object (in cm) = (time to received echo in μs) × 0.01715

Figure 9.24 shows the principle of operation of the ultrasonic sensor module. For example, if the time to receive the echo is 294 microseconds, then the distance to the object is calculated as:

Distance to object (cm) = 294 × 0.01715 = 5.04 cm

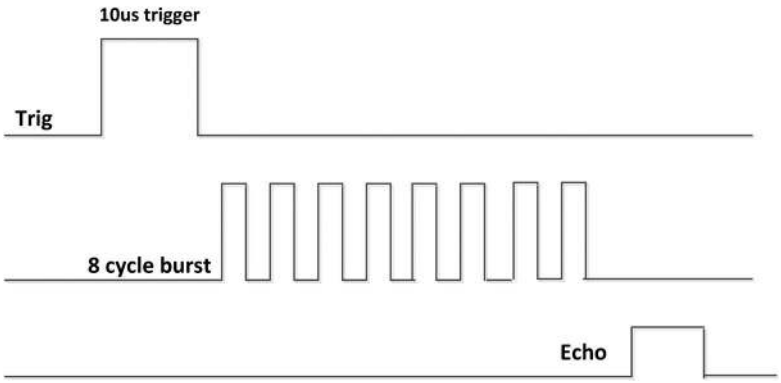


Figure 9.24 Operation of the ultrasonic sensor module.

Figure 9.25 shows the circuit diagram of the project. The trig and echo pins of the sensor are connected to GPIO4 and GPIO17 respectively. The echo output of the ultrasonic sensor is connected to the BeagleY-AI through a resistive potential divider circuit to drop the voltage level to +3.3 V.

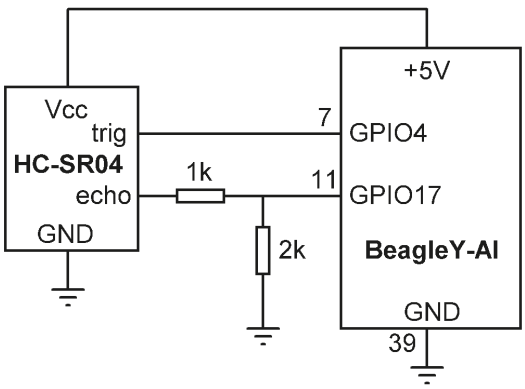


Figure 9.25 Circuit diagram of the project.

Program listing: Figure 9.26 shows the program listing (**ultrasonic.py**). At the beginning of the program, the **echo** and **trigger** pins are defined. Function **GetDistance()** calculates the distance to the obstacle and returns it to the main program after rounding it to 2 decimal places. A 10ms trigger pulse is sent and the program waits until it receives the echo signal. The elapsed time is multiplied by 17150 to calculate the distance in centimeters. The remainder of the program runs in a loop where the distance is measured continuously and displayed on the screen. Figure 9.41 shows an example output from the program.

```
#-------------------------------------------------------
#
#             ULTRASONIC DISTANCE SENSOR
#             ==========================
#
# This program uses a HC-SR04 type ultrasonic transmitter/receiver
# to measure the distance to an obstacle in-front of the sensor.
# The measured distance is displayed on the screen.
#
# Program: ultrasonic.py
# Date   : October 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------
import gpiod
import time

trigger = gpiod.find_line('GPIO4')
echo = gpiod.find_line('GPIO17')
trigger.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
echo.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

#
# This function calculates the distance
#
def GetDistance(trig, echo):
  trig.set_value(0)                    # trig=0
  time.sleep(0.08)                     # waiting for sensor to settle
  trig.set_value(1)                    # send trigger
  time.sleep(0.00001)                  # wait 10ms
  trig.set_value(0)                    # trig=0

  while echo.get_value() == 0:
    start_time = time.time()

  while echo.get_value() == 1:
    end_time = time.time()

  pulse_width = end_time - start_time  # calculate elapsed time
```

```
  distance = pulse_width * 17150       # calculate distance
  distance = round(distance, 2)        # round
  return distance

while True:
  obstacle = GetDistance(trigger, echo)
  print("Distanc (cm)=", obstacle)
  time.sleep(1)
```

Figure 9.26 Program listing.

```
beagle@beagle:~$ python ultrasonic.py
Distanc (cm)= 161.36
Distanc (cm)= 161.38
Distanc (cm)= 161.4
Distanc (cm)= 161.77
Distanc (cm)= 7.73
Distanc (cm)= 7.85
Distanc (cm)= 5.69
Distanc (cm)= 16.59
Distanc (cm)= 17.73
Distanc (cm)= 21.63
Distanc (cm)= 25.22
Distanc (cm)= 21.43
Distanc (cm)= 25.71
Distanc (cm)= 108.42
Distanc (cm)= 206.82
```

Figure 9.27 Example output.

9.11 Project 8 – Car Parking Sensors

Description: This is a parking sensors project to help a person park a car safely and easily. A pair of ultrasonic transmitter/receiver sensors are mounted in the front and back of a vehicle to sense the distance to the objects, and an active buzzer sounds if the sensors are too close to the objects in front of them. In this project, a safe distance is assumed to be 10cm.

Block Diagram: Figure 9.28 shows the block diagram of the project.

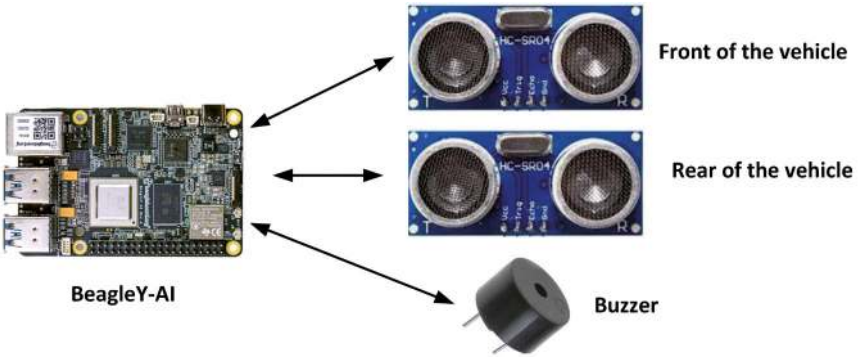


Figure 9.28 Block diagram of the project.

Circuit Diagram: Figure 9.29 shows the circuit diagram. The **trig** and **echo** pins of the Front ultrasonic sensor are connected to GPIO4 and GPIO17, respectively, as in the previous project. Similarly, the **trig** and **echo** pins of the rear ultrasonic sensor are connected to GPIO27 and GPIO22, respectively. The echo outputs of the ultrasonic sensors are connected to the BeagleY-AI through resistive potential divider resistors to drop the voltage levels to

+3.3 V. The active buzzer is connected to GPIO10 of the BeagleY-AI.

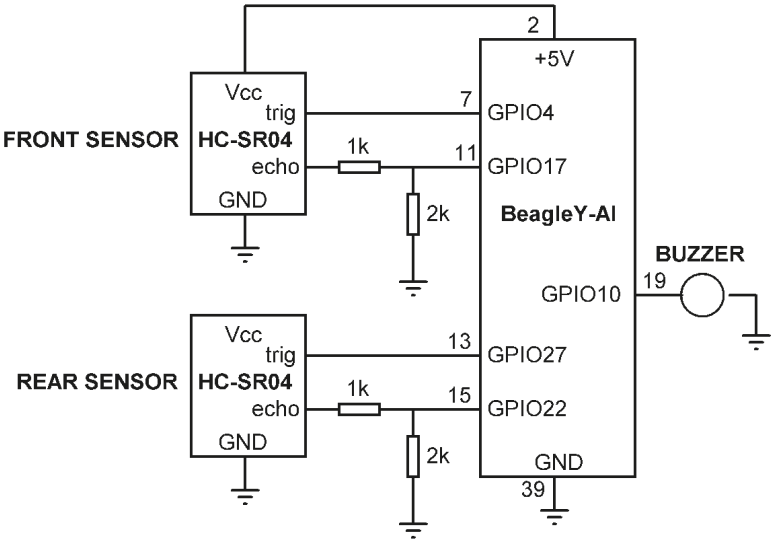


Figure 9.29 Circuit diagram of the project.

Program listing: Figure 9.30 shows the program listing (program **parking.py**). At the beginning of the program, the trigger and echo pins are defined. The triggers are configured as outputs, and the echoes are configured as inputs. If the distance from either sensor to an object is less than or equal to the **Allowed_Distance** (set to 10cm) then the buzzer is sounded to indicate that the vehicle is too close to an object (either at the front or the rear).

```
#-------------------------------------------------------------------
#                          PARKING SENSORS
#                          ===============
#
# This is a parking sensors project. Ultrasonic tranamitter/receiver
# sensors are attached to the front and rear of a vehicle. In addition
# an active buzzer is connected to the BeagleY-AI. The program senses
# the objects in the front and rear of the vehicle and sounds the buzzer
# if the vehicle is too close to the objects. In this project a distance
# less than 10cm is considered to be too close.
#
# File  : parking.py
# Date  : October, 2024
```

```
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
import time
import gpiod

Buzzer = gpiod.find_line('GPIO10')
forwardecho = gpiod.find_line('GPIO17')
rearecho = gpiod.find_line('GPIO22')
forwardtrig = gpiod.find_line('GPIO4')
reartrig = gpiod.find_line('GPIO27')

Buzzer.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
forwardtrig.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
reartrig.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
forwardecho.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
rearecho.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

#
# Calculate the distance to the obstacle
#
def Get_Distance(trig, echo):
   trig.set_value(0)
   time.sleep(0.08)
   trig.set_value(1)
   time.sleep(0.00001)
   trig.set_value(0)

   while echo.get_value() == 0:
      start_time = time.time()

   while echo.get_value() == 1:
      end_time = time.time()

   pulse_width = end_time - start_time
   distance = pulse_width * 17150
   return distance

Allowed_Distance = 10                          # allowed 10cm

Buzzer.set_value(0)                            # buzzer OFF

# obstacle_f and obstacle_r are the distances to the obstacles in the
# front and rear, respectively
#
while True:
  obstacle_f = Get_Distance(forwardtrig, forwardecho)
```

```
obstacle_r = Get_Distance(reartrig, rearecho)
if obstacle_f <= Allowed_Distance or obstacle_r <= Allowed_Distance:
   Buzzer.set_value(1)
else:
   Buzzer.set_value(0)
```

Figure 9.30 Program listing.

Chapter 10 • Plotting Graphs With Python and BeagleY-AI

10.1 Overview

In this chapter, you will learn how to draw graphs using the Python programming language. Additionally, examples and projects are given on drawing graphs for simple electronic circuits.

10.2 The Matplotlib Graph Plotting Library

Matplotlib is a Python plotting library that is used to create two-dimensional graphs. Before using this package, it must be installed on your Raspberry Pi 5 using the following command:

```
beagle@beagle:~ $ sudo apt-get install python3-matplotlib
```

You must import the matplotlib module at the beginning of our programs before you can use Matplotlib. Use the following statement:

```
import matplotlib.pyplot as plt
```

Perhaps the easiest way to learn how to use Matplotlib is to look at an example.

Note that graphs can only be plotted in GUI Desktop mode. Write your programs using Thonny or the Terminal Emulator in console mode.

Example 1

Write a program to draw a line graph passing from the following (x, y) points:

x: 2 4 6 8
y: 4 8 12 16

Solution 1

The required program listing is shown in Figure 10.1 (program: **graph1.py**). This program is very simple. The function call **plt.plot** plots the graph with the specified x and y values. The graph is shown on the GUI Desktop when the statement **plt.show()** is executed. The program was written in console mode on the GUI Desktop.

```
beagle@beagle:~ $ python graph1.py
```

```
#-----------------------------------------------
#              SIMPLE LINE GRAPH
#              =================
#
# This program draws a line graph passing from
# the following points:
#
# x = 2 4 6  8
```

```
# y = 4 8 12 16
#
# Author: Dogan Ibrahim
# File  : graph1.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt

x = [2, 4, 6, 8]
y = [4, 8, 12, 16]

plt.plot(x, y)
plt.show()
```

Figure 10.1 Program listing.

Figure 10.2 shows the graph plotted by the program. Notice that at the bottom of the graph, we have several buttons to control the graph, such as zoom, save, etc.

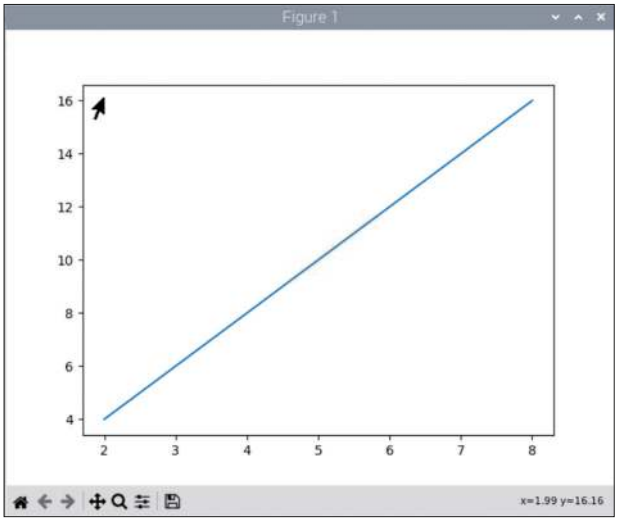


Figure 10.2 Line graph drawn by the program.

You can add titles, axis labels, and grid to your graph using the following functions:

```
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Simple X-Y Graph")
plt.grid(True)
```

The new graph is shown in Figure 10.3.

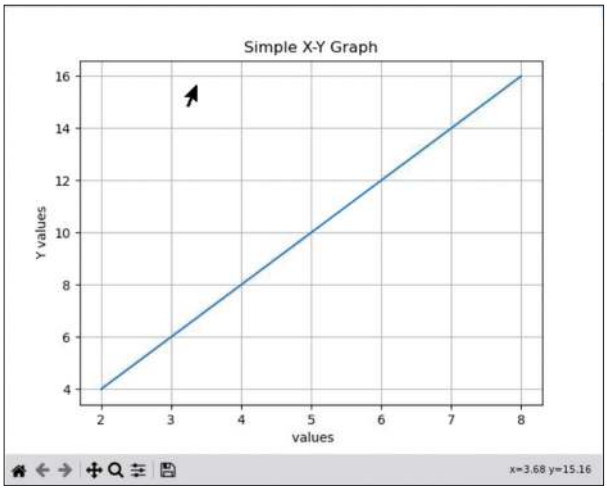


Figure 10.3 Graph with labels, title, and grid.

Matplotlib supports a large number of functions (see web link: https://matplotlib.org/2.0.2/api/pyplot_summary.html for a full description of all the functions). Some commonly used functions are:

- bar: make a bar plot
- box: turn the axis box on or off
- boxplot: make a box plot
- figtext: add text to the figure
- hist: plot a histogram
- legend: place a legend on the axes
- loglog: make a logarithmic plot
- pie: plot a pie chart
- polar: make a polar plot
- plotfile: plot data in a file
- semilogx: logarithmic plot with log on x-axis
- semilogy: logarithmic plot with log on y-axis
- suptitle: add a cantered title to the plot
- tick_params: change the appearance of ticks and tick labels

Example 2
Write a program to draw a sine curve from 0 to 2π.

Solution 2
You have to use **NumPy** arrays to store your data points before plotting. Figure 10.4 shows the program listing (program: **graph2.py**).

```
#-----------------------------------------------
#              SINE GRAPH
#              ==========
#
# This program draws a sine graph from 0 to 2pi
#
# Author: Dogan Ibrahim
# File  : graph2.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 2 * np.pi, 0.1)
y = np.sin(x)

#
# Now plot the graph
#
plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Sin(X)")
plt.title("Sine Wave")
plt.grid(True)
plt.show()
```

Figure 10.4 Program listing.

The graph drawn by the program is shown in Figure 10.5.

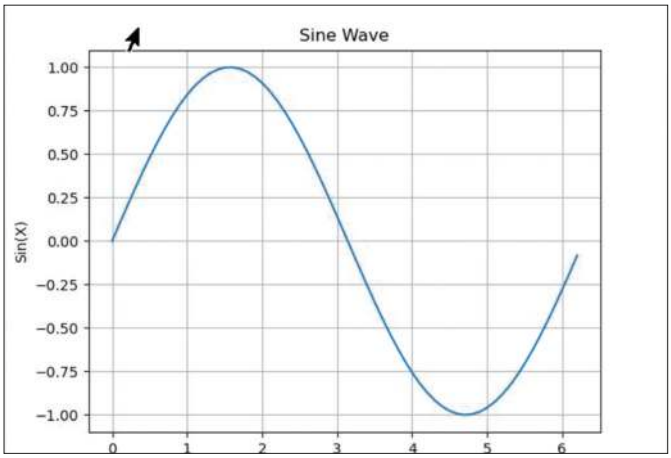


Figure 10.5 Graph drawn by the program.

Example 3
Draw the graph of the following function as x is varied from 0 to 4:

$$y = 2x^2 + 3x + 2$$

Solution 3
Figure 10.6 shows the program listing (program: **graph3.py**). After calculating the x and y values, the graph is drawn as shown in Figure 10.7.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the function:
#
#    y = 2x2 + 3x + 2 from x=0 to x = 4
#
# Author: Dogan Ibrahim
# File  : graph3.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 4, 0.1)
y = [(2 * i * i + 3 * i + 2) for i in x]
```

```
#
# Now plot the graph
#
plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=2x2 + 3x + 2")
plt.grid(True)
plt.show()
```

Figure 10.6 Program listing.

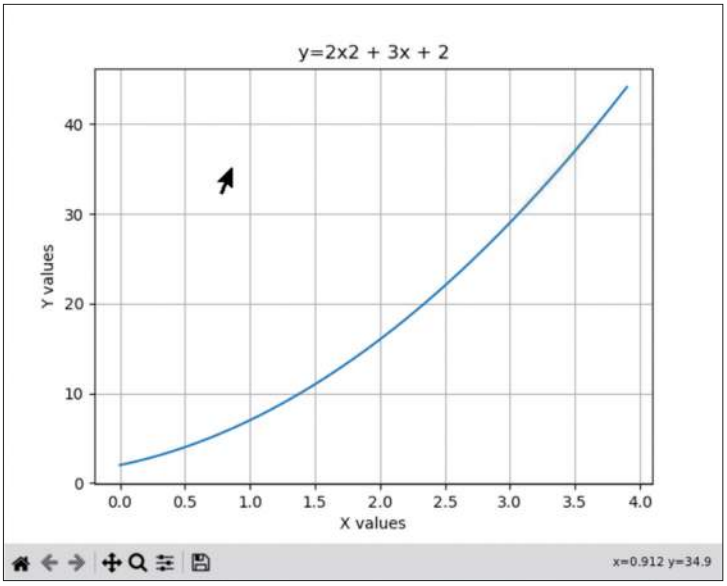


Figure 10.7 Graph drawn by the program.

Example 4

This is an example of drawing two graphs on the same axes. Write a program to draw the graphs of the following two functions as x is varied from 0 to 3:

$$y = x^2 + 2$$
$$y = x^2 + 4$$

Solution 4

Figure 10.8 shows the program listing (program: **graph4.py**). After calculating the x and y values the graphs are drawn as shown in Figure 10.9.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the functions:
#
#   y = x2 + 2
#   y = x2 + 4 from x=0 to x = 3
#
# Author: Dogan Ibrahim
# File  : graph4.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 3, 0.1)
y1 = [(i * i + 2) for i in x]
y2 = [(i * i + 4) for i in x]

#
# Now plot the graph
#
plt.plot(x, y1, linestyle='solid')
plt.plot(x, y2, linestyle='dashed')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=x2+2 and y=x2+4")
plt.grid(True)
plt.show()
```

Figure 10.8 Program listing.

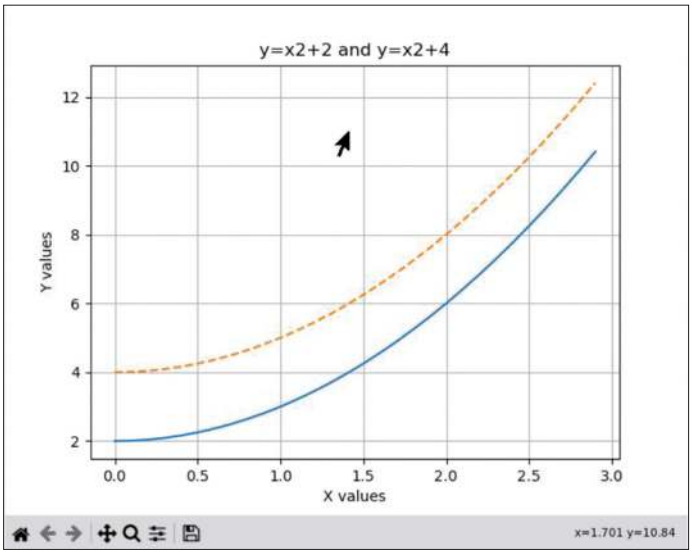


Figure 10.9 Graph drawn by the program.

In order to identify the individual graphs in a multi-graph drawing, you can plot each graph with a different color, or with different types of lines. Some examples are shown below:

```
plt.plot(x, y1, color='blue')
plt.plot(x, y2, color='green')
```

or

```
plt.plot(x, y1, linestyle='solid')
plt.plot(x, y2, linestyle='dashed')
```

Figure 10.10 shows the graph in Figure 10.9 drawn with different line styles.

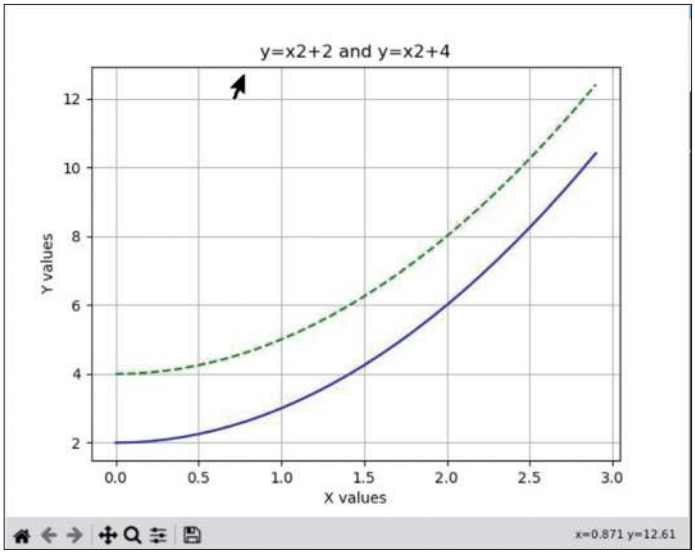


Figure 10.10 Using different line styles.

Example 5

In this example, you will use legends to identify multiple graphs in a multi-graph drawing. The functions to be drawn are the same as the ones given in the previous example.

Solution 5

Figure 10.11 shows the program listing (program: **graph5.py**). The **label** parameter is used to identify the two graphs. Also, the statement **plt.legend()** must be specified to draw the legend.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the functions:
#
#   y = x2 + 2
#   y = x2 + 4 from x=0 to x = 3
#
# In this program the graphs are identified
#
# Author: Dogan Ibrahim
# File  : graph5.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
```

```
#
# Calculate the data points in np
#
x = np.arange(0, 3, 0.1)
y1 = [(i * i + 2) for i in x]
y2 = [(i * i + 4) for i in x]

#
# Now plot the graph
#
plt.plot(x, y1, linestyle='solid', label='x2+2')
plt.plot(x, y2, linestyle='dashed', label='x2+4')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=x2+2 and y=x2+4")
plt.grid(True)
plt.legend()
plt.show()
```

Figure 10.11 Program listing.

Figure 10.12 shows the graph drawn by the program.

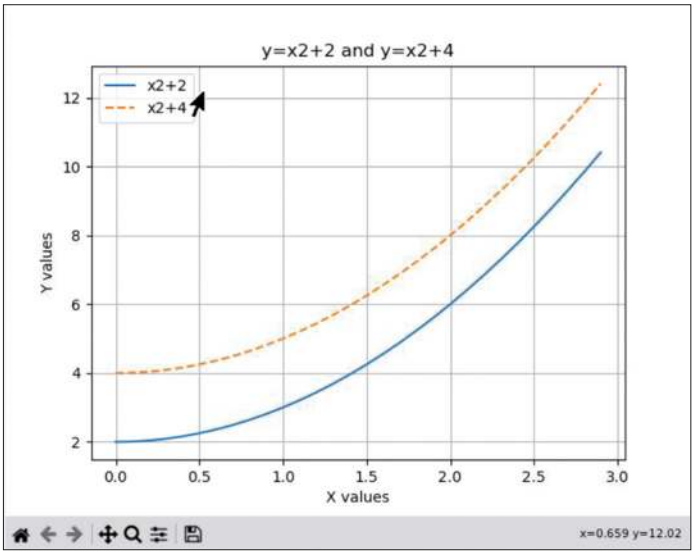


Figure 10.12 Graph drawn by the program.

Example 6
Write a program to draw a pie chart for the following data;

France = 15%, Germany = 20%, Italy = 20%, UK = 45%

Solution 6
Figure 10.13 shows the program listing (program: **graph6.py**). The Pie chart is drawn with an equal aspect ratio so that is a circle.

```
#-----------------------------------------------
#              Pie Chart
#              =========
#
# This program draws a pie chart for the data:
#
#   France=15%, Germany=20%,Italy=20%,UK=45%
#
# Author: Dogan Ibrahim
# File  : graph6.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = "France", "Germany", "Italy", "UK"
sizes = [15, 20, 20, 45]

x, chrt = plt.subplots()
chrt.pie(sizes, labels=labels)
chrt.axis('equal')
plt.show()
```

Figure 10.13 Program listing.

The Pie chart drawn by the program is shown in Figure 10.14.

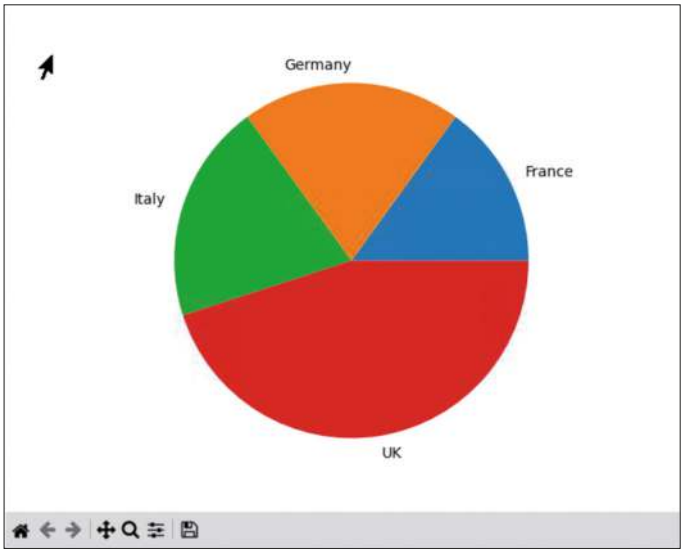


Figure 10.14 Pie chart drawn by the program.

We can explode parts of the Pie chart by specifying the parts to be exploded. For example, to explode the fourth item in our example, we can issue the statement:

```
Explode = (0, 0, 0, 0.1)          # specify the amount to be exploded
```

The amount of explosion is determined by the value we specify. Also, the percentages of each part can be written inside the Pie chart elements by using the statement:

```
autopct='%1.1f%%'                 # specify 1 digit after the decimal point
```

Parts of the pie chart can be shadowed if desired to give it a 3D effect. This can be done using the statement:

```
shadow=True
```

The program shown in Figure 10.15 (program: **graph7.py**) makes use of the above features, and the resulting pie chart is shown in Figure 10.16.

```
#-----------------------------------------------
#                     Pie Chart
#                     =========
#
# This program draws a pie chart for the data:
#
#   France=15%, Germany=20%,Italy=20%,UK=45%
#
# Part UK is exploded in this graph. Also, the
```

```
# percentage of each part is written inside the
# corresponding parts and pats are shadowed
#
# Author: Dogan Ibrahim
# File  : graph7.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = "France", "Germany", "Italy", "UK"
sizes = [15, 20, 20, 45]
explode = (0, 0, 0, 0.1)

x, chrt = plt.subplots()
chrt.pie(sizes, labels=labels, explode=explode,\
autopct='%1.1f%%',shadow=True)
chrt.axis('equal')
plt.show()
```

Figure 10.15 Program listing.

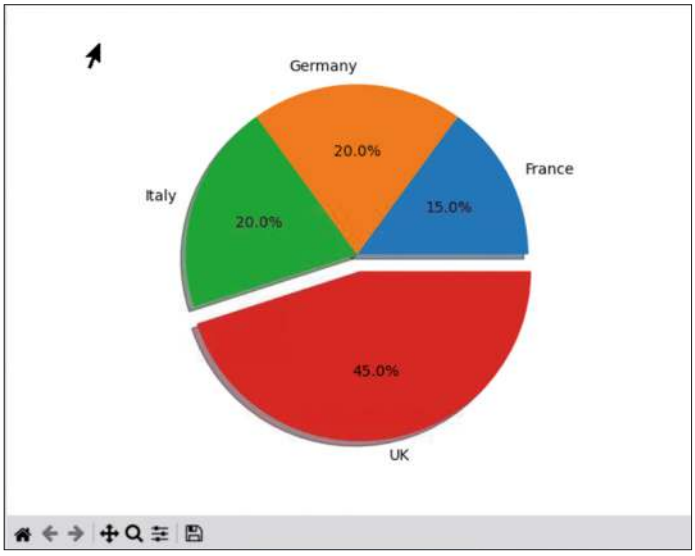


Figure 10.16 Pie chart drawn by the program.

Example 7

Write a program to draw a bar chart for the following data:

France = 10, Italy = 8, Germany = 6, UK = 2

Solution 7

Figure 10.17 shows the program listing (program: **graph8.py**). After specifying the values for each bar, the bar chart is drawn.

```
#-----------------------------------------------
#              Bar Chart
#              =========
#
# This program draws a bar chart for the data:
#   France=10, Italy=8,Germany=6,UK=2
#
# Author: Dogan Ibrahim
# File  : graph8.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = ("France", "Germany", "Italy", "UK")
pos = np.arange(len(labels))
values = [10, 8, 6, 2]

plt.bar(pos, values, align='center',alpha=0.5)
plt.xticks(pos, labels)
plt.ylabel('MB/s')
plt.title('Internet Speed')
plt.show()
```

Figure 10.17 Program listing.

Figure 10.18 shows the graph drawn by the program.

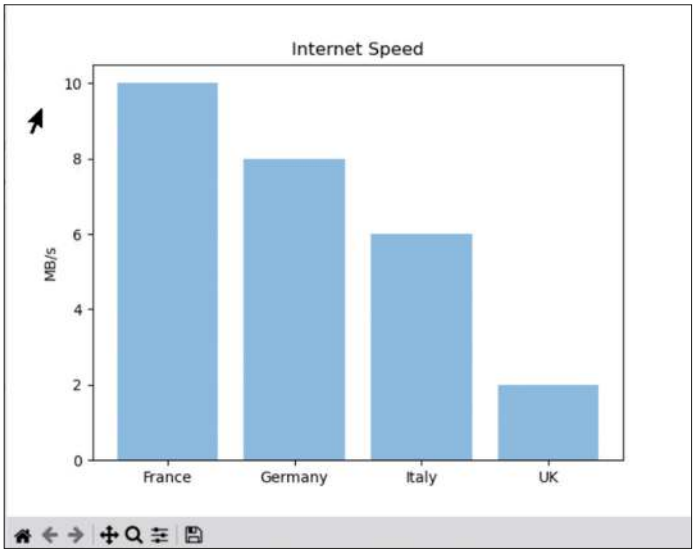


Figure 10.18 Graph drawn by the program.

You can plot a horizontal bar chart by replacing the statement **plt.bar** with **plt.barh**.

10.3 Project 1 – RC Transient Circuit Analysis - Charging

Description: This project is about analyzing a charging RC transient circuit by plotting its time response.

Background Information: RC circuits are used in many radio and communications circuits. A typical RC transient circuit consists of a resistor in series with a capacitor, as shown in Figure 10.19. When the switch is closed, the voltage across the capacitor rises exponentially with a time constant, T = RC.

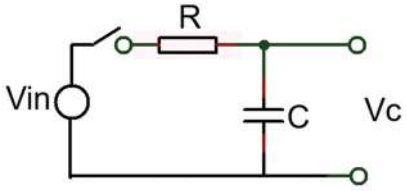


Figure 10.19 Charging RC circuit.

Expressed mathematically, assuming that initially the capacitor is discharged, when the switch is closed the voltage across the capacitor rises a given by the following formula:

$$Vc = Vin\left(1 - e^{-t/RC}\right) \tag{10.1}$$

Initially, the voltage across the capacitor is 0V, and in a steady state, the voltage across the capacitor becomes equal to Vin. The time constant is the time at which the output voltage rises to around 63.2% of its final value.

Program Listing: Figure 10.20 shows the program listing (program: **RCrise.py**). After displaying the heading, the values of the input voltage Vin, and resistor and capacitor values are read from the keyboard. The program then calculates the time constant as **T=RC** and displays the time constant and also draws the time response of the circuit. The graph is drawn as the time value (x-axis) changes from 0 to 6T and 50 points are taken to draw the graph. The time constant is also written on the graph at the point (Time constant, Vin / 2). The horizontal axis is in seconds, while the vertical axis is in volts.

```
#-----------------------------------------------
#              RC TRANSIENT RESPONSE
#              =====================
#
# This program reads the R and C values and then
# calculates and displays the time conctant. Also,
# the time response of the circuit is drawn
#
# Author: Dogan Ibrahim
# File  : RCrise.py
# Date  : October, 2024
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math

print("RC Transient Response")
print("=====================")

#
# Read Vin, R and C
#
Vin = float(input("Enter Vin in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0

#
# Calculate and display time constant
#
T = R * C
F = 6.0 * T
N = F / 50.0
print("Time constant = %f seconds" %(T))
```

```
#
# Now plot the time response
#
x = np.arange(0, F, N)
y = [(Vin * (1.0 - math.exp(-i/T))) for i in x]

plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Capacitor Volts")
plt.title("RC Response")
plt.grid(True)
TC = "T="+str(T)+"s"
plt.text(T, Vin/2, TC)
plt.show()
```

Figure 10.20 Program listing.

Figure 10.21 shows an example graph displayed by the program. In this example, the following input values were used (see Figure 10.22):

Vin = 10 volts
R = 100 ohm
C = 10 microfarad

The time constant was calculated to be 0.1 seconds.

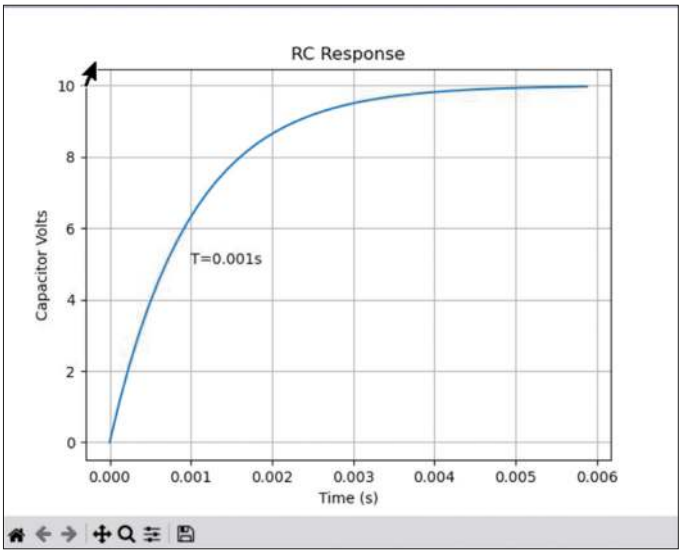


Figure 10.21 Graph plotted by the program.

```
pi@raspberrypi:~ $ python RCrise.py
RC Transient Response
=====================
Enter Vin in Volts: 10
Enter R in Ohms: 100
Enter C in microfarads: 10
Time constant = 0.001000 seconds
```

Figure 10.22 Input values to the example program.

10.4 Project 2 – RC Transient Circuit Analysis - Discharging

Description: This case study is about analyzing a discharging RC transient circuit by plotting its time response.

Background Information: In this case study, an RC circuit is used as in Figure 10.23. We assume that the capacitor is fully charged after switch s1 is closed. We then close switch s2 so that the capacitor discharges through resistor R. The time response of the voltage across the capacitor is given by the following formula:

$$Vc = Vo\, e^{-t/RC} \tag{10.2}$$

Where Vo is the initial voltage across the capacitor (normally the same as Vin) before s2 is closed. Again, T=RC is known as the time constant of the circuit.

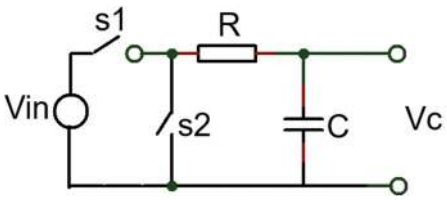


Figure 10.23 Discharging RC circuit.

Program Listing: Figure 10.24 shows the program listing (program: **RCfall.py**). After displaying the heading, the values of the initial voltage across the capacitor (Vo), and the resistor and capacitor are read from the keyboard. The program then calculates the time constant as **T=RC, d**isplays the time constant, and also draws the time response of the circuit. The graph is drawn as the time value (x-axis), changes from 0 to 6T, and 50 points are taken to draw the graph. The time constant is also written on the graph at the point (Time constant, Vo / 2). The horizontal axis is in seconds, while the vertical axis is in volts.

```
#-----------------------------------------------
#              RC TRANSIENT RESPONSE
#              =====================
#
# This program reads the R and C values and then
# calculates and displays the time constant. Also,
# the time response of the circuit is drawn as the
```

```
# capacitor is discharged
#
# Author: Dogan Ibrahim
# File  : RCfall.py
# Date  : October, 2024
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math

print("RC Transient Response")
print("=====================")

#
# Read Vo, R and C
#
Vo = float(input("Enter Initial Capacitor Voltage in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0

#
# Calculate and display time constant
#
T = R * C
F = 6.0 * T
N = F / 50.0
print("Time constant = %f seconds" %(T))

#
# Now plot the time response
#
x = np.arange(0, F, N)
y = [(Vo * (math.exp(-i/T))) for i in x]

plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Capacitor Volts")
plt.title("RC Response")
plt.grid(True)
TC = "T="+str(T)+"s"
plt.text(T, Vo/2, TC)
plt.show()
```

Figure 10.24 Program listing.

Figure 10.25 shows an example graph displayed by the program. In this program, the following input values were used (see Figure 10.26):

Initial capacitor voltage = 10 volts
R = 1000 ohms
C = 100 microfarads

The time constant was calculated to be 0.1 seconds.

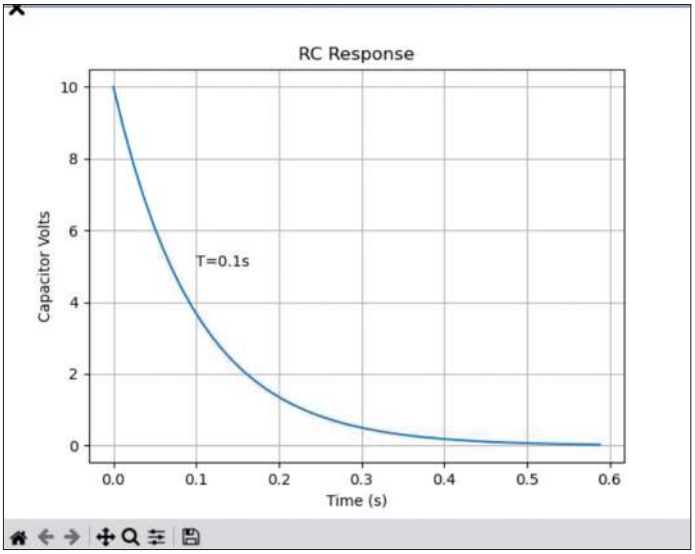


Figure 10.25 Graph plotted by the program.

```
pi@raspberrypi:~ $ python RCfall.py
RC Transient Response
=====================
Enter Initial Capacitor Voltage in Volts: 10
Enter R in Ohms: 1000
Enter C in microfarads: 100
Time constant = 0.100000 seconds
```

Figure 10.26 Input values to the example program.

10.5 Transient RL Circuits

The time response of a transient resistor-inductor circuit is similar to the RC circuit. When the circuit is connected to a DC supply of value Vin, the current in the circuit rises exponentially and is given by the following formula:

$$i = \frac{Vin}{R}\left(1 - e^{-Rt/L}\right) \qquad (10.3)$$

Where, Vin is in volts, R in ohms, L in henries, and t in seconds. The time constant of this circuit is given by T = L/R.

After the current reaches its steady state value, disconnecting the DC supply and shorting the leads causes the current in the circuit to fall exponentially, given by the following formula:

$$i=\frac{Vo}{R}e^{-Rt/L} \tag{10.4}$$

Where, Vo is the initial voltage across the inductor.

The transient response of RL circuits is similar to those of the RC circuits and therefore is not covered further in this book.

10.6 Project 3 – RCL Transient Circuit Analysis

Description: This case study is about analyzing the transient response of a second-order, series-connected RLC circuit by plotting its time response.

Background Information: An RLC circuit (Figure 10.27) is a second-order system that can have three modes of operation depending on the values of the components when a DC voltage is applied across its terminals.

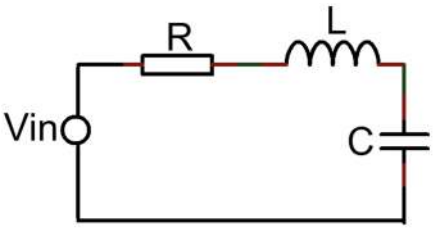


Figure 10.27 RLC circuit.

Underdamped mode: This mode is identified when the following condition holds true:

$$R<2\sqrt{\frac{L}{C}} \tag{10.5}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i(t)=\frac{Vin}{WL\sqrt{1-\xi^2}}e^{-\xi Wt}\sin\left(Wt\sqrt{1-\xi^2}\right) \tag{10.6}$$

Where:

$$W=\frac{1}{\sqrt{LC}} \quad \text{and,} \quad \xi=\frac{R}{2}\sqrt{\frac{C}{L}}<1 \tag{10.7}$$

Critically damped mode: In this mode of operation, the following is satisfied:

$$R=2\sqrt{\frac{L}{C}} \tag{10.8}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i|t|=\frac{Vin}{L}t\,e^{-Wt} \tag{10.9}$$

Where:

$$W=\frac{1}{\sqrt{LC}} \quad \text{and,} \quad \xi=\frac{R}{2}\sqrt{\frac{C}{L}}=1 \tag{10.10}$$

Overdamped mode: In this mode of operation, the following is satisfied:

$$R>2\sqrt{\frac{L}{C}} \tag{10.11}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i|t|=\frac{Vin}{WL\sqrt{\xi^2-1}}e^{-\xi Wt}\sinh\left(Wt\sqrt{\xi^2-1}\right) \tag{10.12}$$

Where:

$$W=\frac{1}{\sqrt{LC}} \quad \text{and,} \quad \xi=\frac{R}{2}\sqrt{\frac{C}{L}}>1 \tag{10.13}$$

Program Listing: Figure 10.28 shows the program listing (program: **RLC.py**). At the beginning of the program, a heading is displayed and then the values of the input voltage, resistor, capacitor, and inductor are read and stored in variables Vin, R, C, and L, respectively. The program then finds out in which mode the circuit will be operating based on the value of ξ. Then, three functions are used, one for each mode, to calculate and plot the transient response of the circuit. The mode of the circuit is displayed on the graph at the coordinate (3T, 0), where T = 2π/W. In all the graphs, 80 points are used to draw the points from 0 to 6T.

```
#-----------------------------------------------
#              RLC TRANSIENT RESPONSE
#              ======================
#
# This program reads the R,L,C values and then
# calculates and displays the transient response
#
# Author: Dogan Ibrahim
# File  : RLC.py
# Date  : October, 2024
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math
global x, y, z

def critically_damped():
  global x,y
  x = np.arange(0, F, N)
  y = [Vin*((1/L) * i * math.exp(-i*w)) for i in x]

def underdamped():
  global x,y,z
  x = np.arange(0, F, N)
  zeta = math.sqrt(1 - z*z)
  y = [Vin*(1/(w*L*zeta)*(math.exp(-z*w*i))*math.sin(w*i*zeta)) for i in x]

def overdamped():
  global x,y,z
  x = np.arange(0, F, N)
  y = [Vin*(1/(w*L*(math.sqrt(z*z-1))))*(math.exp(-z*w*i))*\
math.sinh(w*i*math.sqrt(z*z-1)) for i in x]


print("RLC Transient Response")
print("======================")

#
# Read Vin, R,C and L
#
Vin = float(input("Enter Vin in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0
L = float(input("Enter L in millihenries: "))
L = L / 1000.0
```

```
w = math.sqrt(1/(L * C))
z = (R/2) * math.sqrt(C / L)
T = (2.0 * math.pi) / w
F = 6 * T
N = F / 80.0

#
# Find the mode of operation
#
mode = R - 2.0 * math.sqrt(L / C)
if abs(mode) < 0.01:
  case = 2
  md = "Critically Damped"
  critically_damped()
elif mode < 0:
  case = 1
  md = "Underdamped"
  underdamped()
elif mode > 0:
  case = 3
  md = "Overdamped"
  overdamped()

#
# Now plot the time response
#
plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Current")
plt.title("RLC Response")
plt.grid(True)
plt.text(3*T,0,  md)
plt.show()
```

Figure 10.28 Program listing.

Figure 10.29 shows a typical run of the program with the following values:

Vin = 10 volts
R = 10 ohms
C = 100 microfarads
L = 200 microhenries

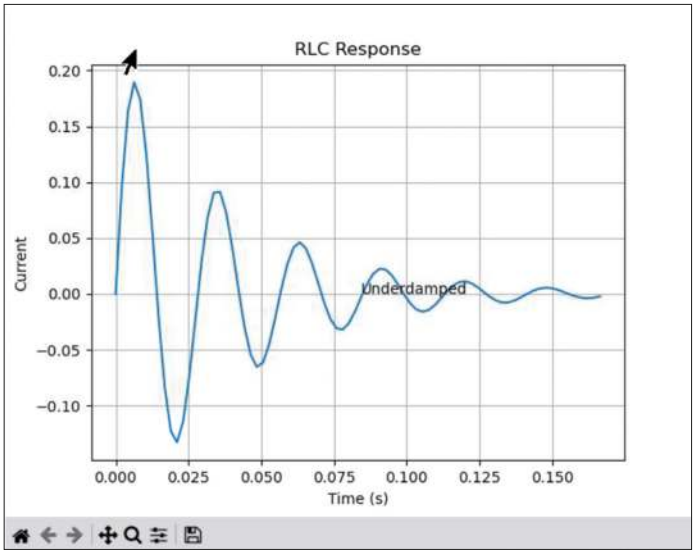


Figure 10.29 Response of the circuit.

10.7 Project 4 – Temperature, Pressure, and Humidity Measurement – Display on the Screen

Description: In this project, the BME280 sensor module is used to read the ambient temperature, pressure, and humidity, and to display the readings on the screen.

Block diagram: Figure 10.30 shows the block diagram of the project.

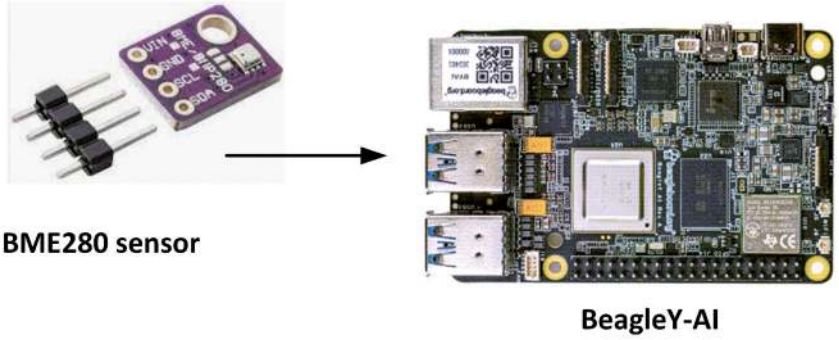


Figure 10.30 Block diagram of the project.

The BME280 Sensor Module

The BME280 module (Figure 10.31) is a low-cost sensor developed for measuring the ambient temperature, atmospheric pressure, and humidity. This module operates with the I^2C (or SPI) bus interface and has the pins SDA, SCL, Vin, and GND. The basic specifications of this module are:

- Operating voltage: 1.2 to 3.6 V
- Interface I^2C or SPI
- Current consumption: 1.8 µA
- Humidity sensor response time: 1 s
- Humidity sensor accuracy: ±3%
- Pressure sensor range: 300 to 1100 hPa
- Temperature range: -40 to +85ºC

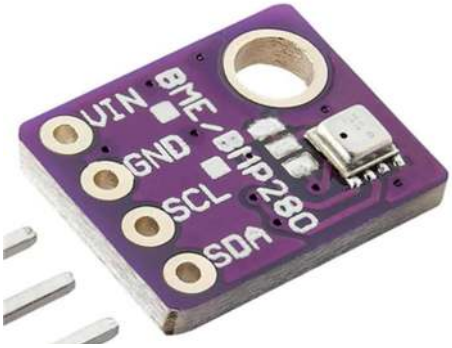


Figure 10.31 The BME280 sensor module.

Circuit diagram: The project circuit diagram is shown in Figure 10.32. The module is connected to BeagleY-AI SDA (pin 3) and SCL (pin 5) pins. +3.3 V power is applied from pin 1.

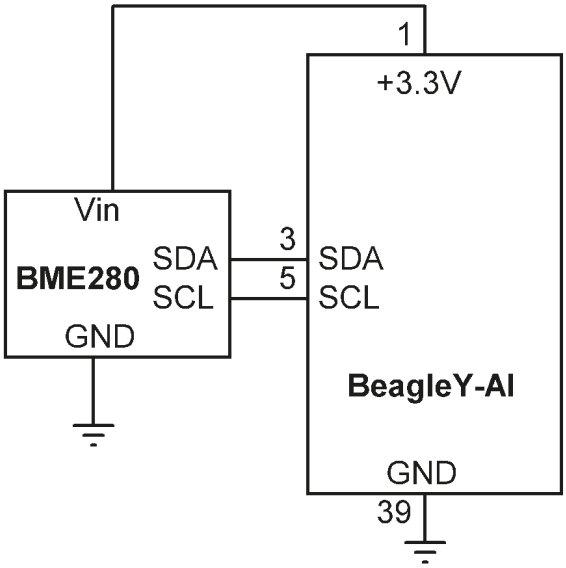


Figure 10.32 Circuit diagram of the project.

The default address of the BME280 is 0x76. This can be confirmed by entering the following command after the circuit is built (Figure 10.33):

```
i2cdetect -r–y 1
```

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
beagle@beagle:~$
```

Figure 10.33 Checking the I²C bus for the sensor module.

Program listing: Figure 10.34 shows the program listing (**bme280.py**). Before running the program, it is necessary to load the BME280 library. The steps are (ignore the warning messages):

- git clone https://github.com/MarcoAndreaBuchmann/bme280pi.git
- cd bme280pi
- sudo python3 setup.py install

The sensor library can be imported to your Python programs as follows:

```
from bme280pi import Sensor
sensor = Sensor()
```

At the beginning of the program, the BME280 sensor library is imported as above, and the sensor address is specified, Inside the main program loop the temperature, atmospheric pressure, and humidity are read and displayed on the screen every 5 seconds.

```
#-------------------------------------------------------------
#         TEMPERATURE,ATMOSPHERIC PRESSURE AND HUMIDITY
#         ============================================
#
# This program reads the ambient temperature, atmospheric
# pressure, and humidity using a BME280 sensor module. The
# readings are dislayed on the screen every 5 seconds
#
# Program: bme280.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
from bme280pi import Sensor
```

```
sensor = Sensor(address = 0x76)

while True:                                    # infinite loop
  data = sensor.get_data()                     # get sensor data
  temperature = data['temperature']            # temperature
  pressure = data['pressure']                  # pressure
  humidity = data['humidity']                  # humidity
  print("Temperature = %5.2f C" %temperature)
  print("Pressure = %d hPa" %pressure)
  print("Humidity = %d" %humidity)
  print("")
  sleep(5)
```

Figure 10.34 Program listing.

Figure 10.35 shows an example output from the program.

```
beagle@beagle:~$ python bme280.py
Temperature = 21.84 C
Pressure = 1016 hPa
Humidity = 0

Temperature = 21.83 C
Pressure = 1015 hPa
Humidity = 0

Temperature = 22.47 C
Pressure = 1015 hPa
Humidity = 0

Temperature = 23.33 C
Pressure = 1015 hPa
Humidity = 0
```

Figure 10.35 Output from the program.

10.8 Project 5 – Temperature, Pressure, and Humidity Measurement – Plotting the Data

Description: This project is very similar to the previous one, but here the data is plotted on the GUI Desktop.

The block diagram and circuit diagram of the project are the same as in Figure 10.30 and Figure 10.32.

Program listing: Figure 10.36 shows the program listing (**bme280plot.py**). The sensor data is collected for 60 seconds where the temperature, pressure, and humidity are stored in **t[]**, **p[]**, and **h[]**. The time in seconds is stored in **tim[]**. When the program runs, the message **Collecting data...** is displayed. The collected data is plotted as shown in Figure 10.37. Note that you can adjust the position of the graphs on the screen using the horizontal arrow tool at the bottom of the screen.

```
#-------------------------------------------------------------
#     PLOT TEMPERATURE, ATMOSPHERIC PRESSURE AND HUMIDITY
#     ===================================================
#
# This program reads the ambient temperature, atmospheric
# pressure, and humidity using a BME280 sensor module. The
# readings are plotted on the Desktop
#
# Program: bme280plot.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
from bme280pi import Sensor
import matplotlib.pyplot as plt

sensor = Sensor(address = 0x76)

p = [0]*60
t=[0]*60
h=[0]*60
data = [0]*60
tim=[0]*60
print("Collecting data...")

for i in range(60):
   data=sensor.get_data()
   tim[i]=i
   p[i] =int(data['pressure'])
   t[i] = int(data['temperature'])
   h[i] = int(data['humidity'])
   sleep(0.1)

plt.figure()
plt.subplot(2, 2, 1)
plt.plot(tim,t)
plt.title("Temperature (C)")
plt.grid()

plt.subplot(2, 2, 2)
plt.plot(tim,p)
plt.title("Pressure (hPa)")
plt.grid()

plt.subplot(2, 2, 3)
plt.plot(tim,h)
```

```
plt.title("Relative Humidity (%)")
plt.grid()
plt.show()
```

Figure 10.36 Program listing.

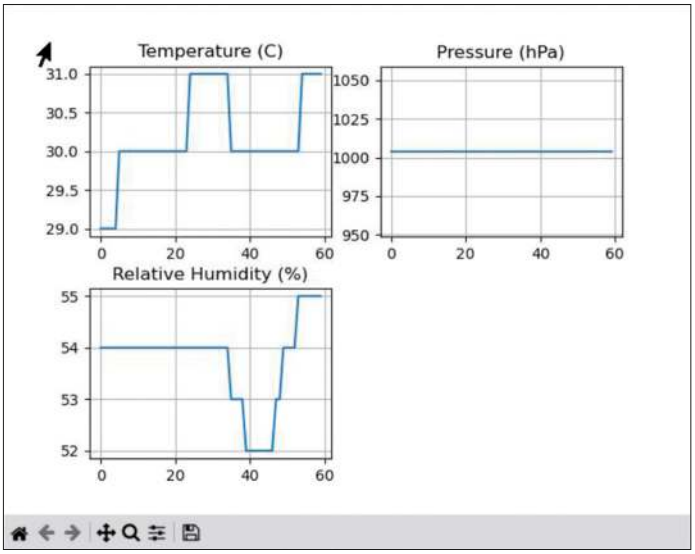


Figure 10.37 Example output from the program.

Chapter 11 • Using a 4 x 4 Keypad

11.1 Overview

Keypads are useful devices for entering data into microcontroller-based systems. They are especially useful in portable applications where the user has to enter data or make a choice. In this chapter, you will learn to use a 4x4 keypad in your BeagleY-AI projects.

11.2 Project 1 – Using a 4x4 Keypad

Description: This is a 4x4 keypad program. The program reads the key pressed by the user and displays its code on the screen. The aim of the project is to show how a 4x4 keypad can be used with a BeagleY-AI project.

The 4x4 Keypad: Several types of keypads can be used in microcontroller-based projects. In this project, a 4x4 keypad (see Figure 11.1) is used. This keypad has keys for numbers 0 to 9, as well as the letters A, B, C, D, *, and #. The keypad is interfaced with the processor using 8 wires, labeled R1 to R4 (representing the rows) and C1 to C4 (representing the columns)(see Figure 11.2).

Figure 11.1 4x4 keypad.

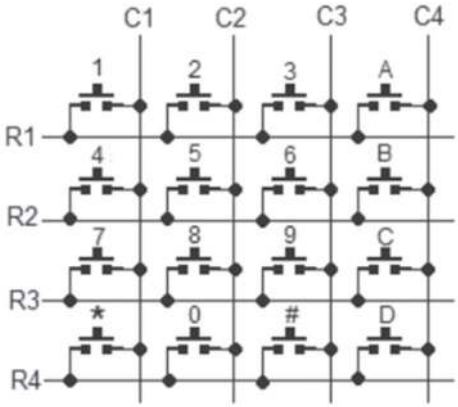


Figure 11.2 Circuit diagram of the 4x4 keypad.

The operation of the keypad is very simple: the columns are configured as inputs and they are all set High, while the rows are configured as outputs. The pressed key is identified by using column scanning. In this process, one row is forced Low while the other rows remain High. Then, the state of each column is scanned, and if a column is found to be Low, the intersection of that column and row is the key pressed. This process is repeated for all the rows.

Block diagram: Figure 11.3 shows the block diagram.

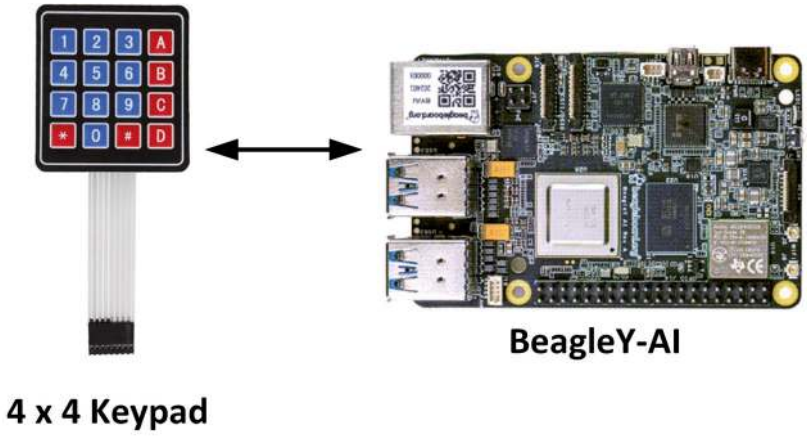


Figure 11.3 Block diagram.

Circuit diagram: The circuit diagram of the project is shown in Figure 11.4. The 4x4 keypad is connected to the following GPIO pins of the BeagleY-AI. The column pins are held

High by using external 10 Kilo-ohm resistors to +3.3 V:

Keypad pin	BeagleY-AI pin
R1	GPIO 4
R2	GPIO17
R3	GPIO27
R4	GPIO22
C1	GPIO10
C2	GPIO9
C3	GPIO11
C4	GPIO5

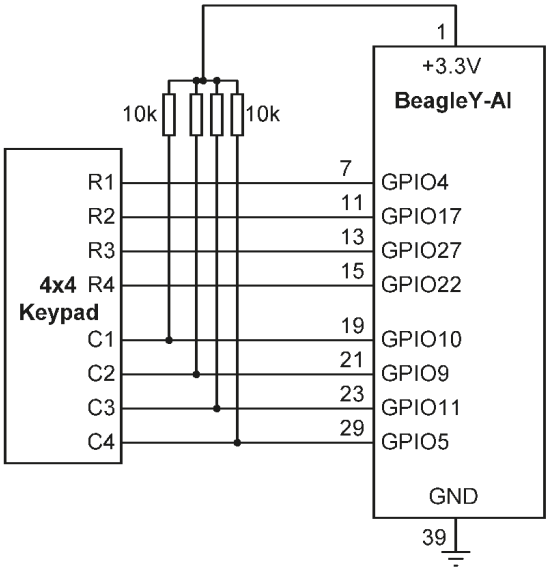


Figure 11.4 Circuit diagram.

Figure 11.5 shows the pin configuration of the 4x4 keypad used in the project.

Figure 11.5 Pin configuration of the 4x4 keypad.

Program listing: Figure 11.6 shows the program listing (program: **keypad.py**). At the beginning of the program, column and row pins are configured. Rows are configured as outputs and columns as inputs. All the rows are set High initially. The function **GetChar()** waits until a key is pressed and then returns the key to the calling code. This function calls the function **ReadRow()**. **ReadRow()** takes two arguments: the row number and the keypad characters on that row. It scans the columns, and if a column is in a Low state, the function returns the keypad character corresponding to that column. The program then calls **GetChar()** and displays the pressed key on the screen.

```
#-----------------------------------------------------------------
#
#               4 x 4 KEYPAD
#               ============
#
# In this program a 4 x 4 keypad is connected to BeagleY-AI.
# the program displays the key pressed on the screen
#
# Program: keypad.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import gpiod
from time import sleep

#
# ROW pins
#
```

```
ROW1 = gpiod.find_line('GPIO4')
ROW2 = gpiod.find_line('GPIO17')
ROW3 = gpiod.find_line('GPIO27')
ROW4 = gpiod.find_line('GPIO22')

#
# COLUMN pins
#
COL1 = gpiod.find_line('GPIO10')
COL2 = gpiod.find_line('GPIO9')
COL3 = gpiod.find_line('GPIO11')
COL4 = gpiod.find_line('GPIO5')

#
# ROWS as outputs
#
ROW1.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW2.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW3.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW4.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

ROW1.set_value(1)
ROW2.set_value(1)
ROW3.set_value(1)
ROW4.set_value(1)

#
# COLUMNS as inputs and (pulled HIGH in hardware)
#
COL1.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL2.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL3.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL4.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

#
# This function sets a row  to 0 and then finds out which
# key is pressed on a column
#
def ReadRow(line, char):
   x = 'E'
   line.set_value(0)
   if COL1.get_value() == 0:
      x = char[0]
   if COL2.get_value() == 0:
      x = char[1]
   if COL3.get_value() == 0:
```

```
      x = char[2]
   if COL4.get_value() == 0:
      x = char[3]
   line.set_value(1)
   return x

#
# This function waits until a character is pressed on keypad
#
def GetChar():
  r = 'E'
  while r == 'E':
     a = ReadRow(ROW1, ["1","2","3","A"])
     b = ReadRow(ROW2, ["4","5","6","B"])
     c = ReadRow(ROW3, ["7","8","9","C"])
     d = ReadRow(ROW4, ["*","0","#","D"])
     if  a != 'E':
        r = a
     elif b !='E':
        r = b
     elif c != 'E':
        r = c
     elif d != 'E':
        r = d
     sleep(0.1)
  return r

c = GetChar()                  # Wait for key press
print (c)                      # Display the pressed key
```

Figure 11.6 Program listing.

Importing the keypad functions in a program

It is easier to import the keypad function under a file instead of writing them every time you want to use them. This can be done by collecting all the functions in a single file and then importing that file at the beginning of your Python programs. Figure 11.7 shows a program called **keypadfuncs.py**, which can be imported into your programs. This file must be placed in your default directory (**/home/beagle**). Note that the keypad rows and columns must be connected to the same BeagleY-AI GPIO pins as given in this project.

```
#-----------------------------------------------------------------------
#
#               4 x 4 KEYPAD
#               ============
#
# In this program a 4 x 4 keypad is connected to BeagleY-AI.
# the program displays the key pressed on the screen
#
# Program: keypad.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-----------------------------------------------------------------------
import gpiod
from time import sleep

#
# ROW pins
#
ROW1 = gpiod.find_line('GPIO4')
ROW2 = gpiod.find_line('GPIO17')
ROW3 = gpiod.find_line('GPIO27')
ROW4 = gpiod.find_line('GPIO22')

#
# COLUMN pins
#
COL1 = gpiod.find_line('GPIO10')
COL2 = gpiod.find_line('GPIO9')
COL3 = gpiod.find_line('GPIO11')
COL4 = gpiod.find_line('GPIO5')

#
# ROWS as outputs
#
ROW1.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW2.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW3.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)
ROW4.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

ROW1.set_value(1)
ROW2.set_value(1)
ROW3.set_value(1)
ROW4.set_value(1)

#
# COLUMNS as inputs and (pulled HIGH in hardware)
```

```
#
COL1.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL2.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL3.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)
COL4.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_IN)

#
# This function sets a row  to 0 and then finds out which
# key is pressed on a column
#
def ReadRow(line, char):
   x = 'E'
   line.set_value(0)
   if COL1.get_value() == 0:
      x = char[0]
   if COL2.get_value() == 0:
      x = char[1]
   if COL3.get_value() == 0:
      x = char[2]
   if COL4.get_value() == 0:
      x = char[3]
   line.set_value(1)
   return x

#
# This function waits until a character is pressed on keypad
#
def GetChar():
  r = 'E'
  while r == 'E':
     a = ReadRow(ROW1, ["1","2","3","A"])
     b = ReadRow(ROW2, ["4","5","6","B"])
     c = ReadRow(ROW3, ["7","8","9","C"])
     d = ReadRow(ROW4, ["*","0","#","D"])
     if  a != 'E':
        r = a
     elif b !='E':
        r = b
     elif c != 'E':
        r = c
     elif d != 'E':
        r = d
     sleep(0.2)
  return r
```

Figure 11.7 Program: keypadfuncs.py.

Figure 11.8 shows a program (**keypadtest.py**) that imports the keypad functions.

```
#-------------------------------------------------------------------
#
#               4 x 4 KEYPAD TEST
#               =================
#
# This program imports the keypad functions
#
# Program: keypadtest.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------------
from keypadfuncs import GetChar

c = GetChar()                   # Wait for key press
print (c)                       # Display the pressed key
```

Figure 11.8 Program: keypadtest.py.

11.3 Project 2 – Security Lock with Keypad and LCD

Description: This is an electronic lock project where a relay is used to open a door. A 4-digit secret code is set up in the program. The user must enter the secret code for the door to open.

Block diagram: Figure 11.9 shows the block diagram of the project.

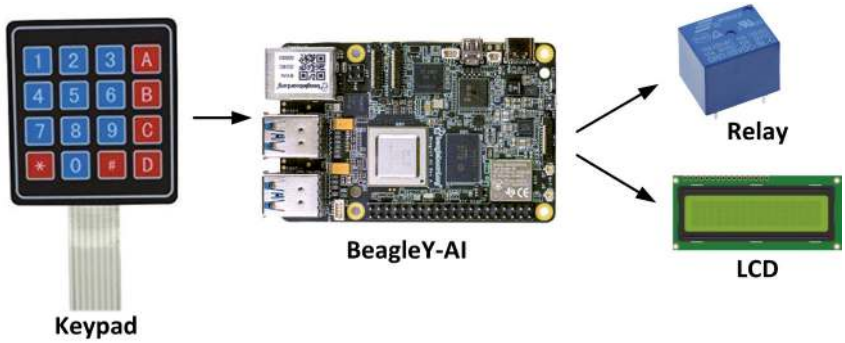


Figure 11.9 Block diagram.

Circuit diagram: The circuit diagram is shown in Figure 11.10. The LCD is connected as in the previous LCD-based projects. The keypad is connected as in the previous project. A relay is connected to GPIO21 (pin 40) of the BeagleY-AI.

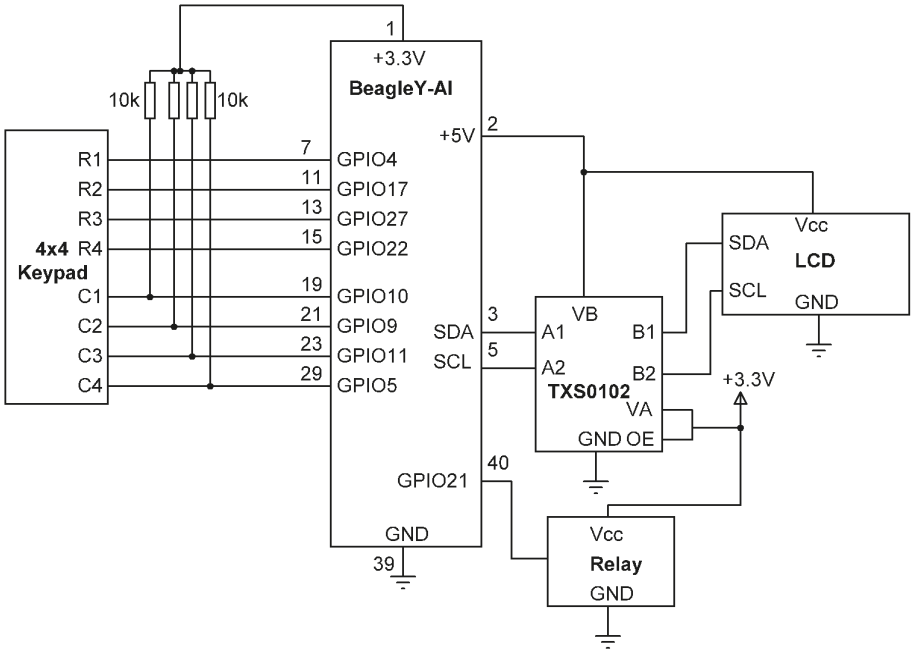


Figure 11.10 Circuit diagram.

Program listing: Figure 11.11 shows the program listing **(lock.py**). At the beginning of the program, the LCD is initialized. The secret code is set to "1357". The program then displays **Code:** and expects the user to enter the correct code. If the correct code is entered, the message **Door Opened** is displayed and the relay is turned On for 20 seconds. After this time, the relay is deactivated. If the wrong code is entered, the message **Error** is displayed for 5 seconds, and the user is asked to enter the correct code again.

```
#----------------------------------------------------------------------
#
#               KEYPAD OPERATED LOCK
#               ====================
#
# In this program a door (or a safe) is opened via a relay.
# The user is required to enter the correct secret code for
# the door  to open. Once opened the door stays open for 20 secs
#
# Program: lock.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#----------------------------------------------------------------------
import gpiod
from time import sleep
from lcd_api import LcdApi
```

```
from i2c_lcd import I2cLcd
from keypadfuncs import GetChar

Relay = gpiod.find_line('GPIO21')
Relay.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

Relay.set_value(0)

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
mylcd.clear()

Codea = "1"
Codeb = "3"
Codec = "5"
Coded = "7"

while True:
  mylcd.move_to(0, 0)
  mylcd.putstr("Code: ")
  a = GetChar()
  b = GetChar()
  c = GetChar()
  d = GetChar()

  if (a == Codea and b == Codeb and c == Codec and d == Coded):
     mylcd.clear()
     mylcd.putstr("Door Opened")
     Relay.set_value(1)
     sleep(20)
     Relay.set_value(0)
     mylcd.clear()
```

Figure 11.11 Program listing.

Suggested modification: Modify the program in Figure 11.11 so that the lock is disabled for 10 minutes if the wrong code is entered 3 times.

Chapter 12 • I²C, SPI Bus, and PWM Projects

12.1 Overview

I²C and SPI buses are two very important peripheral buses used in microcontroller systems. Most sensors in microcontroller applications are based on using one of these buses. In this chapter, you will learn how to use these buses with the BeagleY-AI in some simple projects.

12.2 Project 1 - I²C Port Expander

Description: A simple project is given in this section to show how the I²C functions can be used in a program. In this project, the I²C bus-compatible Port Expander chip (MCP23017) is used to give an additional 16 I/O ports to BeagleY-AI. This is useful in some applications where a large number of I/O ports may be required. In this project, an LED is connected to MCP23017 port pin GPA0 (pin 21), and the LED is flashed On and Off every second to verify the program's operation. A 470-ohm current limiting resistor is used in series with the LED.

The aim: This project aims to show how the I²C bus can be used in BeagleY-AI projects.

Block diagram: The block diagram of the project is shown in Figure 12.1.

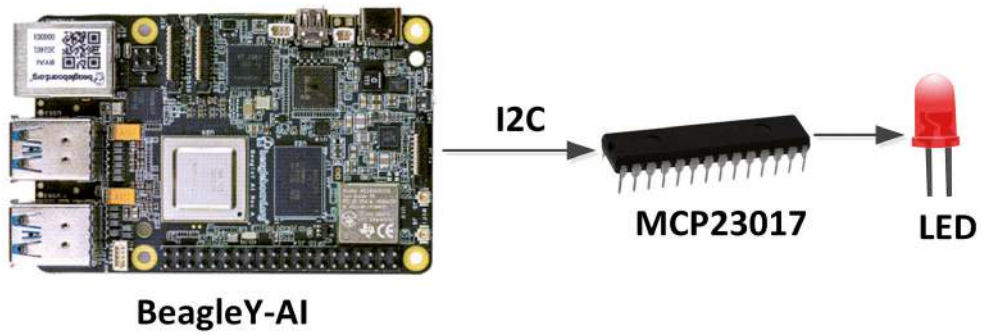


Figure 12.1 Block diagram of the project.

The MCP23017

The MCP23017 is a 28-pin chip with the following features. The pin configuration is shown in Figure 12.2:

- 16 bi-directional I/O ports
- Up to 1.7 MHz operation on I2C bus
- Interrupt capability
- External reset input
- Low standby current
- +1.8 V to +5.5 V operation
- 3 address pins, so that up to 8 devices can be used on the I2C bus
- 28-pin DIL package

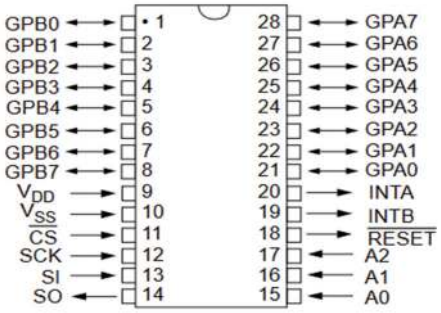


Figure 12.2 Pin configuration of the MCP23017.

The pin descriptions are given in Table 12.1.

Pin	Description
GPA0-GPA7	Port A pins
GPB0-GPB7	Port B pins
VDD	Power supply
VSS	Ground
SDA	I2C data pin
SCL	I2C clock pin
RESET	Reset pin
A0-A2	I2C address pins

Table 12.1 MCP23017 pin descriptions

The MCP23017 is addressed using pins A0 to A2. Table 12.2 shows the address selection. In this project, the address pins are connected to ground, thus the address of the chip is 0x20. The chip address is 7 bits wide, with the low bit set or cleared depending on whether we wish to read data from the chip or write data to the chip respectively. Since in this project, we will be writing to the MCP23017, the low bit should be 0, making the chip byte address (also called the **device opcode**) 0x40.

A2	A1	A0	Address
0	0	0	0x40
0	0	1	0x21
0	1	0	0x22
0	1	1	0x23
1	0	0	0x24
1	0	1	0x25
1	1	0	0x26
1	1	1	0x27

Table 12.2 Address selection of the MCP23017

The MCP23017 chip has 8 internal registers that can be configured for its operation. The device can either be operated in 16-bit mode or in two 8-bit modes by configuring bit IOCON.BANK. On power-up, this bit is cleared, which selects the two 8-bit mode by default.

The I/O direction of the port pins is controlled with registers IODIRA (at address 0x00) and IODIRB (at address 0x01). Clearing a bit to 0 in these registers sets the corresponding port pin(s) as output(s). Similarly, setting a bit to 1 in these registers sets the corresponding port pin(s) input(s). GPIOA and GPIOB register addresses are 0x12 and 0x13 respectively. This is shown in Figure 12.3.

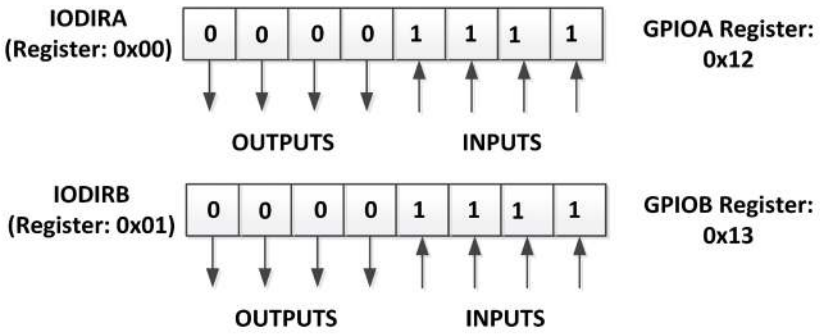


Figure 12.3 Configuring the I/O ports.

Circuit diagram: Figure 12.4 shows the circuit diagram of the project. Notice that I^2C pins of the port expander are connected to pins GPIO2 (SDA) and GPIO3 (SCL) of the BeagleY-AI. The LED is connected to port pin GPA0 of the MCP23017 (pin 21). The address select bits of the MCP23017 are all connected to ground.

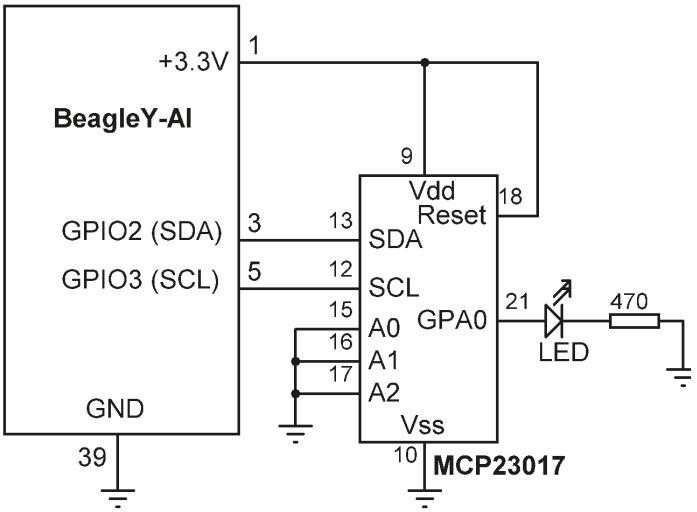


Figure 12.4 Circuit diagram of the project.

More information on the MCP23017 chip can be obtained from the datasheet:

http://docs-europe.electrocomponents.com/webdocs/137e/0900766b8137eed4.pdf

Program listing: Figure 12.5 shows the program listing (Program: **MCP23017**). At the beginning of the program, GPIOA is configured as an output. Then, an endless loop is created, and the LED is turned On and Off with a one-second delay between each output. The function **bus.write_byte_data()** writes a byte to the specified I^2C device address and the specified register address.

```
#================================================================
#                PORT EXPANDER
#                =============
#
# In this project a MCP23017 type port expander chip is used to
# provide 16 additional I/I ports to the BeagleY-AI. As an example,
# an LED is connected to the chip and the LED is flashed every second
# with one second delay between each output
#
# Author: Dogan Ibrahim
# File  : MCP23017,py
# Date  : November 2024
#================================================================
import smbus
import time

bus = smbus.SMBus(1)                          # Using i2c bus1
addr = 0x20                                   # MCP23017 address
MCP_GPIOA_REG = 0x12                          # MCP23017 GPIOA address
MCP_IODIRA_REG = 0

bus.write_byte_data(addr,MCP_IODIRA_REG,0)    # Configure as output

while True:
   bus.write_byte_data(addr,MCP_GPIOA_REG,1)  # LED ON
   time.sleep(1)
   bus.write_byte_data(addr, MCP_GPIOA_REG,0) # LED OFF
   time.sleep(1)
```

Figure 12.5 Program listing.

12.3 Project 2 - SPI ADC - Voltmeter

Description: This is a voltmeter project. Because the BeagleY-AI does not have any onboard analog-to-digital converters (ADC), an external ADC chip (MCP3002) is used in this project. The voltage to be measured is applied to the ADC, and its value is displayed on the screen.

Block diagram: Figure 12.6 shows the block diagram.

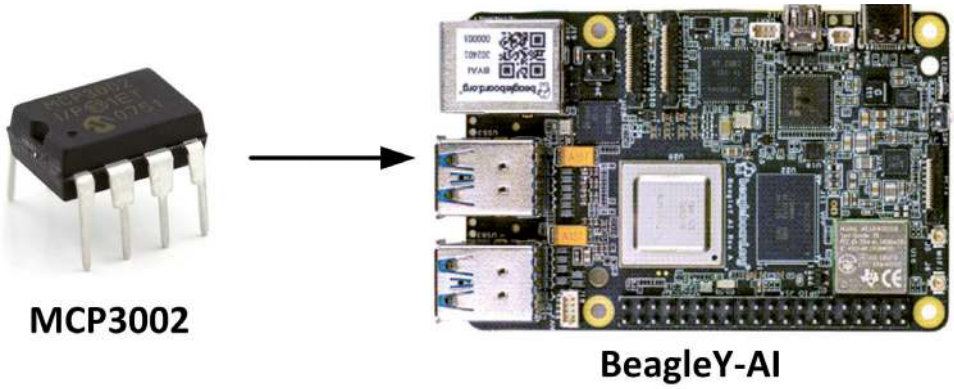


Figure 12.6 Block diagram.

12.3.1 The SPI bus

The MCP3002 ADC chip operates with the SPI bus. It is useful to summarize the operation of the SPI bus in this section.

The SPI bus is a commonly used protocol to connect sensors and many other devices to microcontrollers. The SPI bus is a master-slave type bus protocol. In this protocol, one device (the microcontroller) is designated as the master, and one or more other devices (usually sensors) are designated as slaves. In a minimum bus configuration, there is one master and only one slave. The master establishes communication with the slaves and controls all the activity on the bus.

Figure 12.7 shows an example of an SPI with one master and 3 slaves. The SPI bus uses 3 signals: clock (SCK), data in (SDI), and data out (SDO). The SDO of the master is connected to the SDIs of the slaves, and the SDOs of the slaves are connected to the SDI of the master. The master generates the SCK signals to enable data transfer on the bus. In every clock pulse, one bit of data is transferred from master to slave, or from slave to master. The communication is only between a master and a slave, and the slaves cannot communicate with each other. It is important to note that only one slave can be active at any time because there is no mechanism to identify multiple slaves simultaneously. Thus, slave devices have enable lines (e.g., Chip Select (CS) or Chip Enable (CE)), which are normally controlled by the master. A typical communication between a master and several slaves is as follows:

- Master enables slave 1
- Master sends SCK signals to read or write data to slave 1
- Master disables slave 1 and enables slave 2
- Master sends SCK signals to read or write data to slave 2
- The above process continues as required

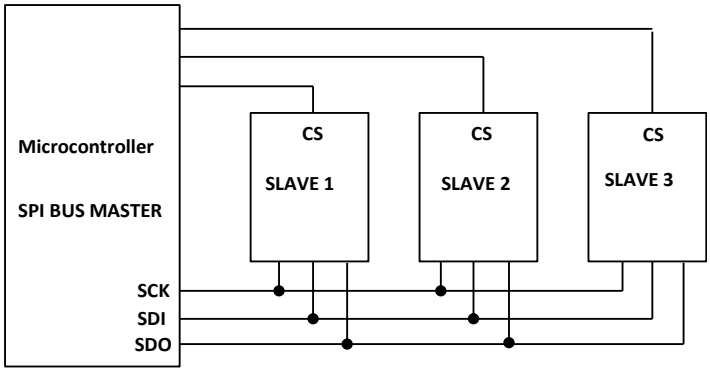


Figure 12.7 SPI bus with one master and 3 slaves.

The SPI signal names are also called MISO (Master In, Slave Out), and MOSI (Master Out, Slave In). Clock signal SCK is also called SCLK, and the CS is also called SSEL or CE. In the SPI projects in this chapter, the development kit is the master, and one or more slaves can be connected to the bus. Transactions over the SPI bus are started by enabling the SCK line. The master then asserts the SSEL line Low so that data transmission can begin. The data transmission involves two registers, one in the master and one in the slave device. Data is shifted out from the master into the slave with the MSB bit first. If more data is to be transferred, then the process is repeated. Data exchange is complete when the master stops sending clock pulses and deselects the slave device.

Both the master and the slave must agree on the clock polarity and phase on the line, which are known as the SPI bus modes. These two settings are named Clock Polarity (CPOL) and Clock Phase (CPHA) respectively. CPOL and CPHA can have the following values:

CPOL	**Clock active state**
1	Clock active High
11	Clock active Low

CPHA	**Clock phase**
1	Clock out of phase with data
2	Clock in phase with data

The four SPI modes are:

Mode	**CPOL**	**CPHA**
0	0	0
1	0	1
2	1	0
3	1	1

When CPOL = 0, the active state of the clock is 1, and its idle state is 0. For CPHA = 0, data is captured on the rising edge of the clock, and data is shifted out on the falling edge. For CPHA = 1, data is captured on the falling edge of the clock and is shifted out on the rising edge of the clock.

When CPOL = 1, the active state of the clock is 0, and its idle state is 1. For CPHA = 0, data is captured on the falling edge of the clock and is output on the rising edge. For CPHA = 1, data is captured on the rising edge of the clock and is shifted out on the falling edge.

BeagleY-AI SPI bus pins are:

MOSI	GPIO10
MISO	GPIO9
SCLK	GPIO11
CE0	GPIO8

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

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

The MCP3002 is a successive approximation 10-bit ADC with an on-chip sample and hold amplifier. The device is programmable to operate as either a differential input pair or as dual single-ended inputs. The device is offered in an 8-pin package. Figure 12.8 shows the pin configuration of the MCP3002.

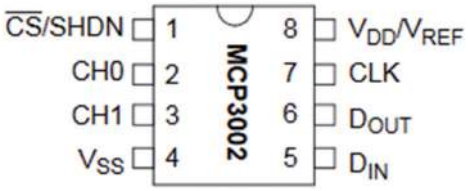


Figure 12.8 Pin configuration of the MCP3002.

The pin definitions are as follows:

Vdd/Vref:	Power supply and reference voltage input
CH0:	Channel 0 analog input
CH1:	Channel 1 analog input
CLK:	SPI clock input
DIN:	SPI serial data in

DOUT: SPI serial data out
CS/SHDN: Chip select/shutdown input

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

Digital output code = 1024 x Vin / 3.3

or, Digital output code = 310.30 x Vin

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

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

Figure 12.9 shows the circuit diagram of the project where the voltage to be measured is applied directly to the CH0 input of the ADC.

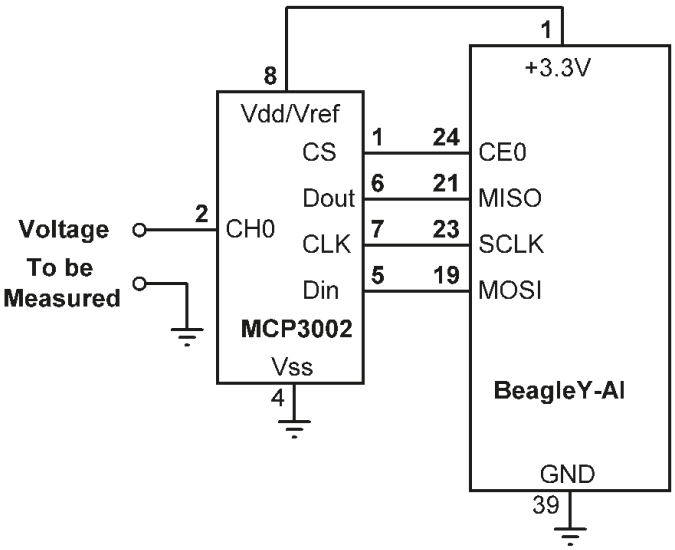


Figure 12.9 Circuit diagram of the project.

Program listing: The spidev must be installed on your BeagleY-AI and SPI overlay defined, and permission given to spidev:

- beagle@beagle:~ $ **sudo apt-get install python3-spidev**
- edit file **/boot/firmware/extlinux/extlinux.conf** and add the following line:

  fdtoverlays /overlays/k3-am67a-beagley-ai-spidev0.dtb0

- beagle@beagle:~ $ **sudo nano /boot/firmware/extlinux/extlinux.conf**
- Enter **Ctrl+X** followed by **Y** to save and exit the editor
- Reboot your BeagleY-AI

  beagle@beagle:~ $ **sudo reboot**

- Give permission to spidev

  beagle@beagle:~ $ **sudo chmod ugo+rwx /dev/spidev***

Note: Overlays are located in the directory: **/boot/firmware/overlays** and should be added to file: **/boot/firmware/extlinux/extlinux.conf**.

Figure 12.10 shows the program listing (**voltmeter.py**). Function **get_adc_data** is used to read the analog data, where the channel number (channel_no) is specified in the function argument as 0 or 1. Notice that 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 8 bits at a time.

The following data can be sent to the ADC (SGL/DIFF = 1 and ODD/SIGN = channel_no) as bytes with leading zeroes for a 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

At the beginning of the program in Figure 12.10, an instance of the SPI is created. The function **get_adc_data** reads the temperature from sensor chip MCP3002 and returns a digital value between 0 and 1023. This value is then converted into millivolts and is displayed on the screen. Figure 12.11 shows an example output from the project where the input CH0 was connected to GND or +3.3 V.

```
#---------------------------------------------------------------
#                      VOLTMETER
#                              =========
#
# This is a voltmeter project. The voltage to be measured is applied
# to CH0 input of the MCP3002 ADC. The measured voltage is displayed
# on the screen using a print statement
#
# Program: voltmeter.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#---------------------------------------------------------------
import spidev
from time import sleep

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

```
spi.max_speed_hz = 4000

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

#
# Start of main program. Read the input voltage and display in mv
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0          # convert to mV
   print("Voltage = %5.2f mV" %mV)     # display voltage in mV
   sleep(1)                            # wait one second
```

Figure 12.10 Program listing.

```
Voltage = 3287.10 mV
Voltage = 3296.77 mV
Voltage = 3290.32 mV
Voltage = 3290.32 mV
Voltage = 3274.19 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage = 3300.00 mV
Voltage = 3280.65 mV
Voltage = 3277.42 mV
Voltage = 3290.32 mV
```

Figure 12.11 Example output from the program.

12.4 Project 3 – Voltmeter – Output to LCD

Description: This project is basically the same as the previous one, but here the measured voltage is displayed on LCD.

Block diagram: Figure 12.12 shows the block diagram.

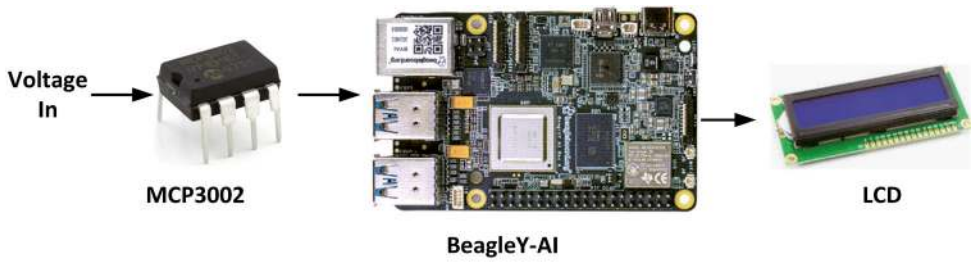


Figure 12.12 Block diagram.

Circuit Diagram: The circuit diagram of the project is shown in Figure 12.13. The LCD and the MCP3002 are connected as in the previous projects.

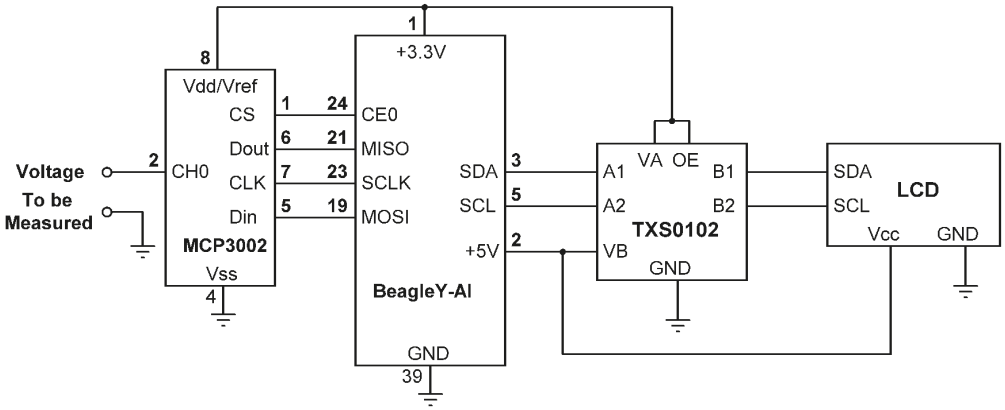


Figure 12.13 Circuit diagram of the project.

Program listing: Figure 12.14 shows the program listing (**LCDvolt.py**). This program is basically the same as the one in Figure 12.10, but here the output is sent to LCD instead of being displayed on the screen. The data is displayed in the following format:

nnnn mV

```
#----------------------------------------------------------------
#                      VOLTMETER WITH LCD DISPLAY
#                      ==========================
#
# This is a voltmeter project. The voltage to be measured is applied
# to CH0 input of the MCP3002 ADC. The measured voltage is displayed
# on the LCD
#
# Program: LCDvolt.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#----------------------------------------------------------------
```

```
import spidev
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
from time import sleep

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

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16


mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

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

#
# Start of main program. Read the voltage and display it
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0          # convert to mV
   disp = str(mV)[:4] + " mV"
   mylcd.move_to(0,0)
   mylcd.putstr(disp)
   sleep(2)
   mylcd.clear()
```

Figure 12.14 Program listing.

12.5 Project 4 – Analog Temperature Sensor Thermometer – Output to the Screen

Description: In this project, an analog temperature sensor chip is used to measure and then display the ambient temperature every second on the screen. The temperature is read using an external ADC as in the previous project. The aim of this project is to show how the ambient temperature can be read and displayed on the monitor using an analog temperature sensor chip.

Block Diagram: Figure 12.15 shows the block diagram of the project.

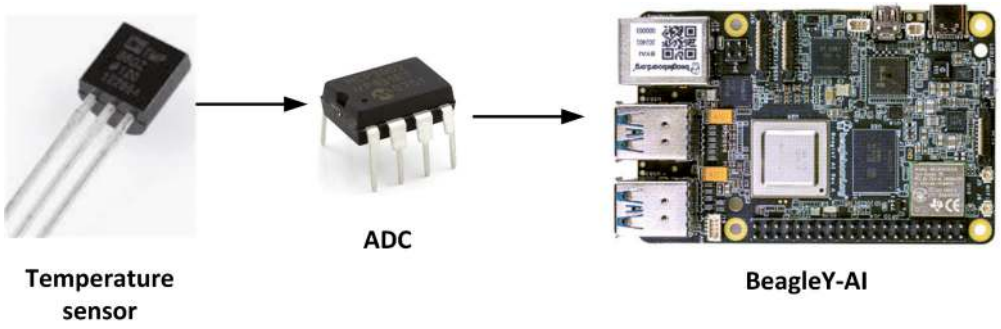


Figure 12.15 Block diagram of the project.

Circuit Diagram: The dual MCP3002 ADC chip is used in this project to provide analog input capability to the BeagleY-AI. Figure 12.16 shows the circuit diagram of the project. A TMP36DZ type analog temperature sensor chip is connected to CH0 of the ADC. TMP36DZ is a 3-terminal small sensor chip with pins: Vs, GND, and Vo. Vs is connected to +3.3 V, GND is connected to the system ground, and Vo is the analog output voltage. The temperature in degrees Centigrade is given by:

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

Where, Vo is the sensor output voltage in millivolts.

CS, Dout, CLK, and Din pins of the ADC are connected to the SPI pins CE0, MISO, SCLK, and MOSI pins of the BeagleY-AI respectively.

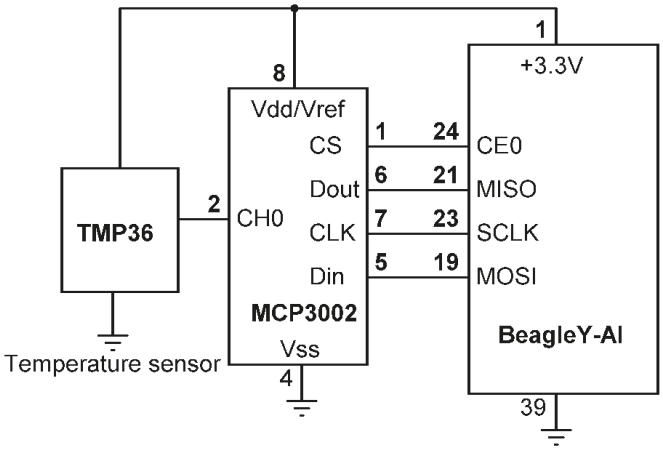


Figure 12.16 Circuit diagram of the project.

Program listing: Figure 12.17 shows the BeagleY-AI Python program listing (program: **tmp36.py**). The function **get_adc_data** is used to read the analog data, where the channel number (channel_no) is specified in the function argument as 0 or 1. The function **get_adc_data** reads the temperature from sensor chip MCP3002 and returns a digital value between 0 and 1023. This value is then converted into millivolts, 500 is subtracted from it, and the result is divided by 10 to find the temperature in degrees Centigrade. The temperature is displayed on the monitor every second.

```
#---------------------------------------------------------------
#               ANALOG TEMPERATURE MEASUREMENT
#               ==============================
#
# This is a thermometer project. Ambient temperature is read using
# an ADC and is then displayed on the screen every second
#
# Program: tmp36.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#---------------------------------------------------------------
import spidev
from time import sleep

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

#
```

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

#
# Start of main program. Read the analog voltage and display it
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0                  # convert to mV
   Temperature = (mV - 500) / 10.0
   print("Temperature = %5.2f C" %Temperature)
   sleep(1)                                    # wait one second
```

Figure 12.17 Python program listing.

A typical display on the monitor is shown in Figure 12.18.

```
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
```

Figure 12.18 Typical display.

12.6 Project 5 – Analog Temperature Sensor Thermometer – Output on LCD

Description: This project is similar to the previous one, but here the temperature is displayed on LCD.

Block diagram: Figure 12.19 shows the block diagram of the project.

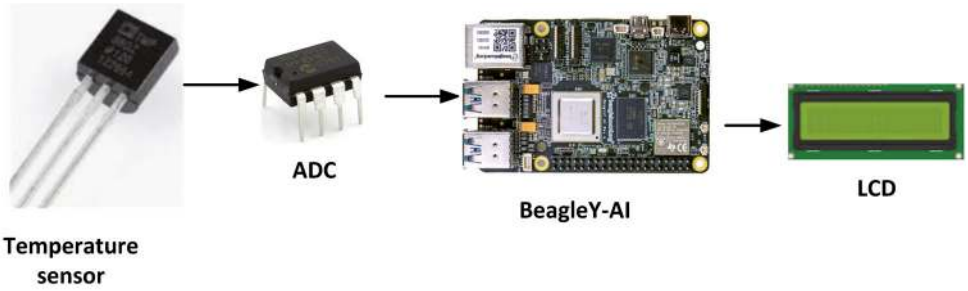


Figure 12.19 Block diagram.

Circuit diagram: The circuit diagram of the project is shown in Figure 12.20. The ADC and the sensor chip are connected as in the previous project.

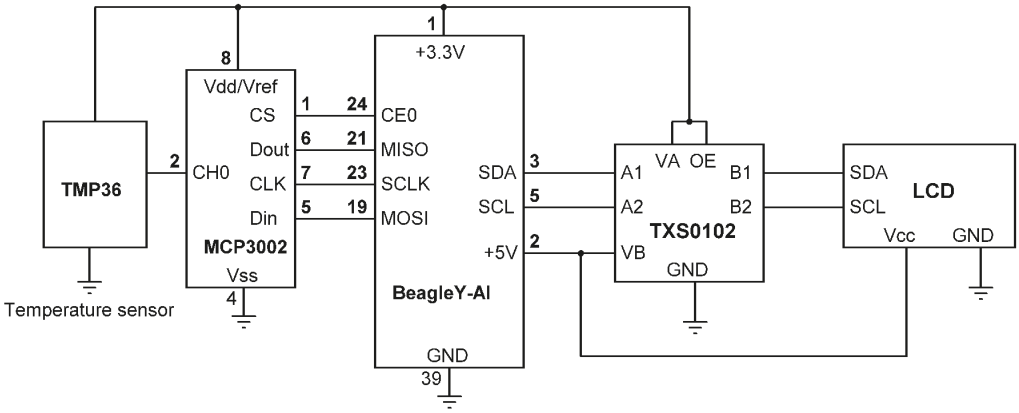


Figure 12.20 Circuit diagram

Program listing: Figure 12.21 shows the program listing (**LCDtmp36.py**). The program is very similar to the previous one, but here the temperature is displayed on LCD.

```
#-----------------------------------------------------------------
#           ANALOG TEMPERATURE MEASUREMENT - OUTPUT ON LCD
#           ==============================================
#
# This is a thermometer project. Ambient temperature is read using
# an ADC and is then displayed on LCD
#
# Program: LCDtmp36.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import spidev
from time import sleep
from lcd_api import LcdApi
```

```
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR, I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

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

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

#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0          # convert to mV
   Temperature = (mV - 500) / 10.0
   T = str(Temperature)[:5] + " C"
   mylcd.move_to(0,0)
   mylcd.putstr(T)
   sleep(5)                            # wait one second
   mylcd.clear()
```

Figure 12.21 Program listing.

If you get a permission error, run the following command:

```
beagle@beagle:~ $ sudo chmod ugo+rwx /dev/spidev*
```

12.7 Using a Digital to Analog Converter (DAC)

A digital-to-analog converter (DAC) is an electronic circuit that converts a digital signal into an analog signal. They are commonly used in music players to convert digital data into analog audio signals. They are also used in mobile phones, televisions, and digital audio processing systems. Waveform generators are important in many electronic communication applications. In this chapter, you will develop projects to generate waveforms such as square, sine, triangular, staircase, etc, by using an external DAC chip and programming the BeagleY-AI. You will be using the popular MCP4921 DAC chip from Microchip.

12.7.1 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 12.22 shows the pin layout of this chip. The basic features are:

- 12-bit operation
- 20 MHz clock support
- 4.5 µs settling time
- External voltage reference input
- Unity or 2x Gain control
- 1x or 2x gain
- 2.7 V to 5.5 V supply voltage
- -40ºC to +125ºC temperature range

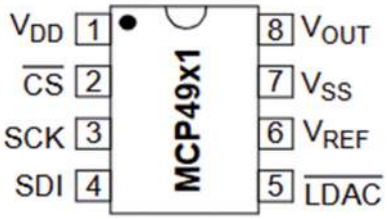


Figure 12.22 MCP4921 DAC.

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:	analog output
Vss:	supply ground

In projects in this chapter, you will be operating the MCP4921 with a gain of 1. 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 mV / 4096 = 0.8 mV

The SPI Bus
As discussed in an earlier chapter, the 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)**: **M**aster **O**ut **S**lave **I**n. This signal is output from the master and is input to a slave.

MISO: **M**aster **I**n **S**lave **O**ut. This signal is output from a slave and input to a master.

SCLK (or **SCK)**: The clock, controlled by the master.

CE (or **CS)**: **C**hip **E**nable (slave select).

The following pins are the SPI bus pins on BeagleY-AI:

GPIO pin	SPI	Physical pin no
GPIO 10	MOSI (SPI0)	19
GPIO 9	MISO (SPI0)	21
GPIO 11	SCLK (SPI0)	23
GPIO 8	CE0 (SPI0)	24

12.7.2 Project 6 - Generating square wave signal with any peak voltage up to +3.3 V

Description: In this project, you will be using the DAC to generate a square wave signal with a frequency of 1kHz, where the required output voltage is a 2 V peak.

Block Diagram: Figure 12.23 shows the block diagram of the project.

Figure 12.23 Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 12.24. The output of the DAC is connected to a PSCGU250 type digital oscilloscope.

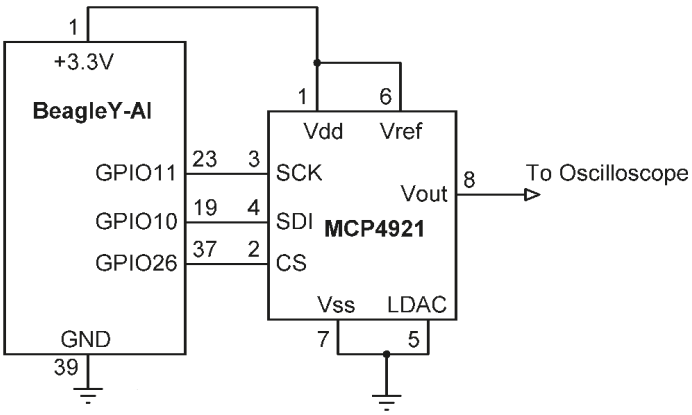


Figure 12.24 Circuit diagram of the project

Program Listing: Data is written to the DAC in 2 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: 2x gain, 1: 1x gain
BUF	0: unbuffered input, 1: buffered input
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 to ensure that the device is active and the gain is set to 1x. Then, we will send the lower byte (D0:D7) of the data. This means that 0x30 should be added to the upper byte before sending it to the DAC.

Figure 12.25 shows the program listing (program: **squaredac.py**). GPIO26 is used as the **CS** pin. Variable **frequency** is set to 1000 which is the required frequency. The function DAC sends the 12-bit input data to the DAC. This function consists of two parts. In the first part, the High byte is sent after adding 0x30 as described above. The function **xfer2** is used to send the data to the DAC. In the second part of the function, the Low byte is extracted and sent to the DAC. Notice that we could have sent 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 2000mv (i.e., 2 V, remember that the DAC is 12 bits /, with 4095 steps, and the reference voltage is set to 3300 mV). The **OFFvalue** is set to 0 V. 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 affects the period and consequently the frequency of the output waveform. Because of this, 2 mV is subtracted from **halfperiod.**

```
#-----------------------------------------------------------------
#                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  : October, 2024
#-----------------------------------------------------------------
from time import sleep
import gpiod
import spidev                                   # Import SPI

CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS.set_value(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):
   CS.set_value(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

   CS.set_value(1)                            # Disable CS

try:
  ONvalue = int(2000*4095/3300)               # 2V output
  OFFvalue = 0

  while True:
     DAC(ONvalue)                            # Send to DAC
     sleep(halfperiod - 0.0002)              # Wait
     DAC(OFFvalue)                           # Send to DAC
     sleep(halfperiod - 0.0002)              # Wait

except KeyboardInterrupt:
     pass
```

Figure 12.25 Program listing.

Figure 12.26 shows the output waveform generated by the program. Notice that the peak output voltage is 2 V as expected.

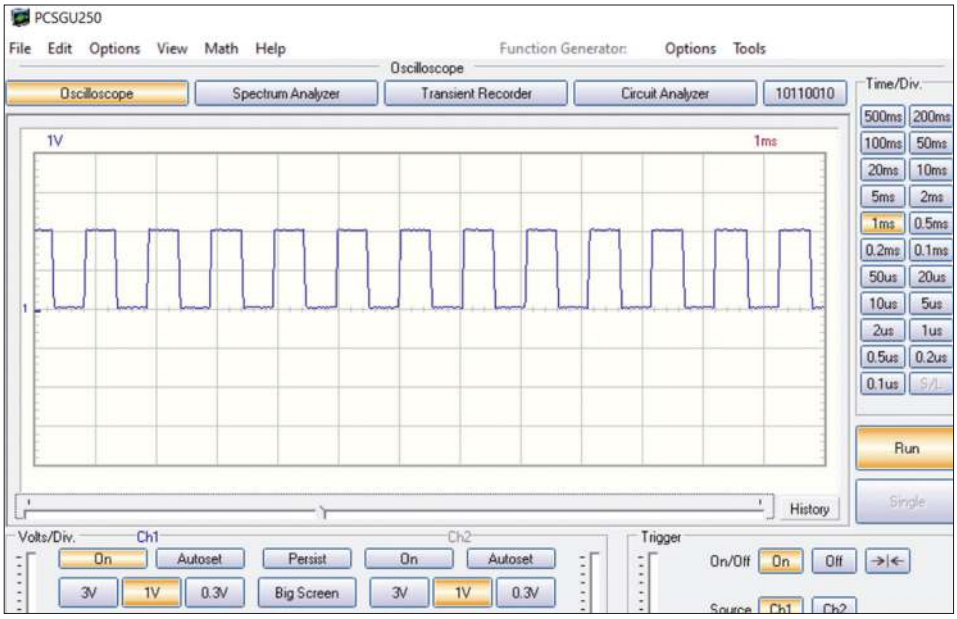


Figure 12.26 Output waveform.

12.7.3 Project 7 - Generating sawtooth wave signal

Description: In this project, you 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

The block diagram and circuit diagram of the project are as in Figure 12.23 and Figure 12.24

Program Listing: Figure 12.27 shows the program listing (program: **sawtooth.py**). The program is very similar to the one given in Figure 12.25.

```
#               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  : October, 2024
#-------------------------------------------------------------------
from time import sleep                          # Import time
```

```
import spidev                                   # Import SPI
import gpiod

CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS.set_value(1)                                 # Disable CS

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.set_value(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

   CS.set_value(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
        sleep(0.0007)                           # Wait
        i = i + 0.2

except KeyboardInterrupt:
    pass
```

Figure 12.27 Program listing.

An example output waveform taken from the oscilloscope is shown in Figure 12.28. Notice that the time delay had to be adjusted experimentally to give the correct timing.

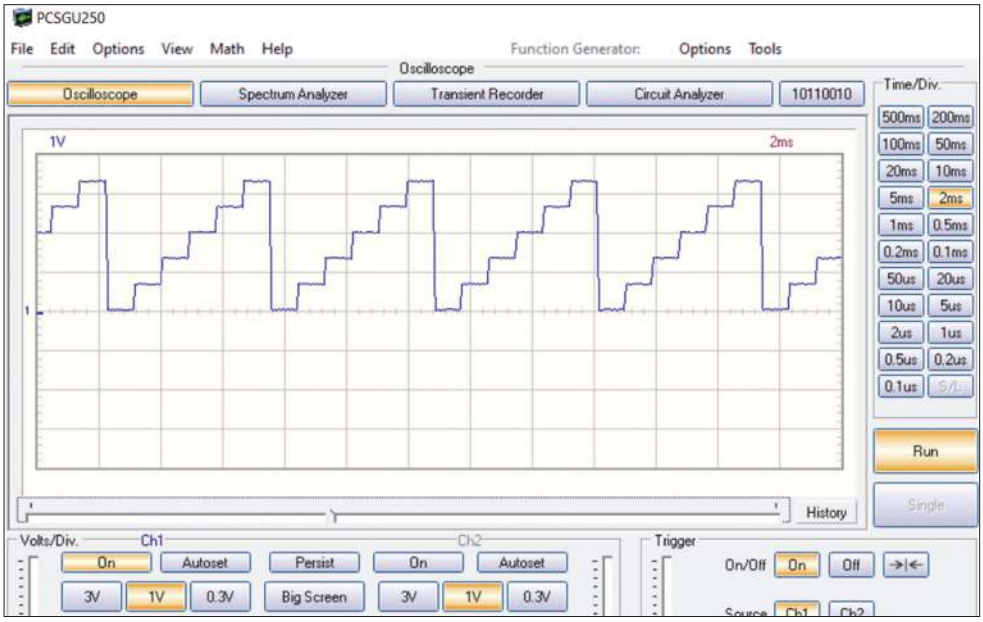


Figure 12.28 Example output waveform.

12.7.4 Project 8 - Generating triangle wave signal

Description: In this project, we will be using the DAC to generate a triangle wave signal.

The block diagram and circuit diagram of the project are as in Figure 12.23 and Figure 12.24

Program Listing: Figure 12.29 shows the program listing (program: **triangle.py**). The program is very similar to the one given in Figure 12.27.

```
#-----------------------------------------------------------------
#                GENERATE TRIANGLE WAVEFORM
#                ==========================
#
# This program generates triangle waveform
#
# Author: Dogan Ibrahim
# File  : triangle.py
# Date  : October, 2024
#-----------------------------------------------------------------
import gpiod
from time import sleep                          # Import time
import spidev                                   # Import SPI
```

```
CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS.set_value(1)                                 # Disable CS
sample = 0
Inc = 0.05

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.set_value(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

   CS.set_value(1)                              # Disable CS

try:
   while True:
        DACValue = int(sample*4095)             # Value  to send
        DAC(DACValue)                           # Send to DAC
        sleep(0.0001)                           # Wait
        sample = sample + Inc                   # Next sample
        if sample > 1.0 or sample < 0:
           Inc = -Inc
           sample = sample + Inc

except KeyboardInterrupt:
    pass
```

Figure 12.29 Program listing.

An example output waveform taken from the oscilloscope is shown in Figure 12.30.

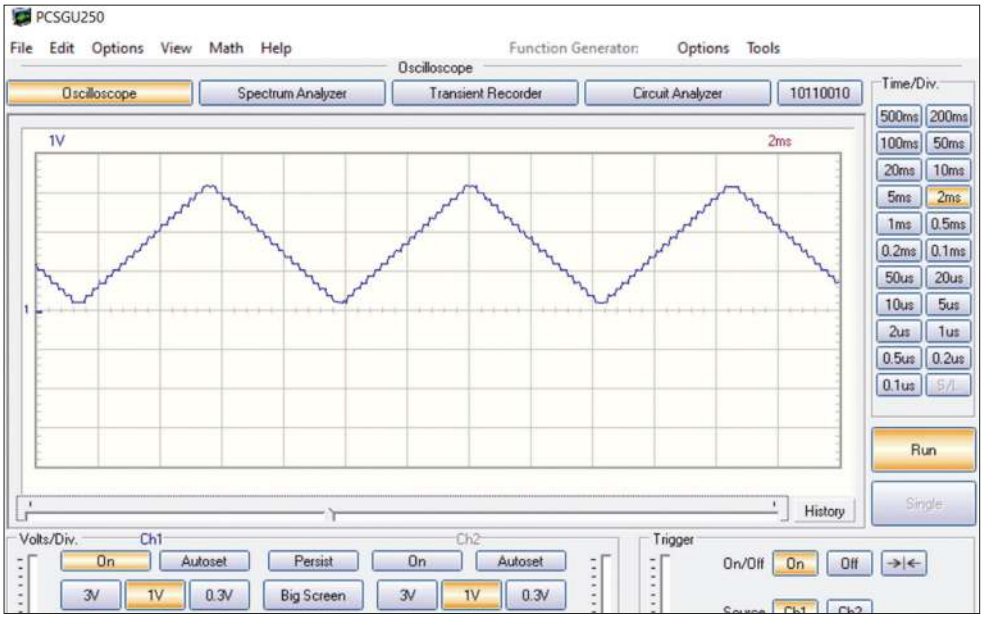


Figure 12.30 Example output waveform.

If you get a permission error, run the following command:

```
beagle@beagle:~ $ sudo chmod ugo+rwx /dev/spidev*
```

12.7.5 Project 9 - Generating arbitrary wave signal

Description: In this project, you 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 12.31. Notice that the waveform has a period of 20 ms.

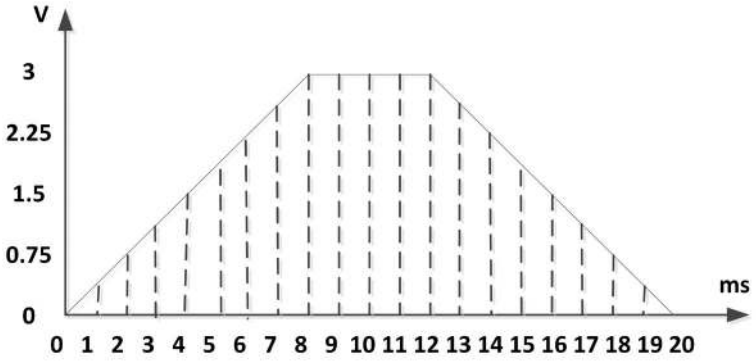


Figure 12.31 Waveform to be generated.

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		

The block diagram and circuit diagram of the project are as in Figure 12.23 and Figure 12.24

Program Listing: Figure 12.32 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 1 ms. It was found by experiment that 0.8ms 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  : October, 2024
#---------------------------------------------------------------------
import gpiod
from time import sleep                          # Import time
import spidev                                   # Import SPI

CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz=3900000

CS.set_value(1)                                 # Disable CS
sample = 0

#
# Waveform sample points
#
wave = [0,0.375,0.75,1.125,1.5,1.875,2.25,2.625,3,3,3,3,3,\
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):
   CS.set_value(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])

   CS.set_value(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
```

```
        sleep(0.0008)                          # Wait
        if sample == 20:                       # If 20 sampes
           sample = 0

except KeyboardInterrupt:
    pass
```

Figure 12.32 Program listing.

An example output waveform taken from the oscilloscope is shown in Figure 12.33.

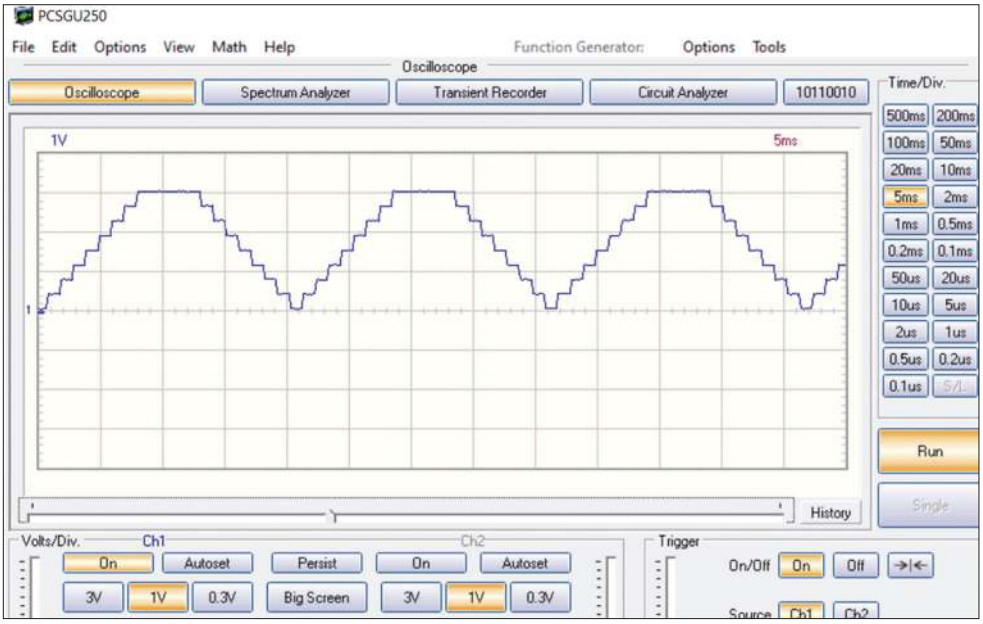


Figure 12.33 Example output waveform.

12.7.6 Project 10 - Generating sine wave signal

Description: In this project, we will be using the DAC to generate a low-frequency sine wave using the built-in trigonometric **sin** function. The generated sine wave will have an amplitude of 1.5 V, a frequency of 100Hz (period = 10ms), and an offset of 1.5 V.

The block diagram and circuit diagram of the project are as in Figure 12.23 and Figure 12.24

Program Listing: The frequency of the sine wave to be generated is 100Hz. This wave has a period of 10 ms or 10,000 µs. If we assume that the sine wave will consist of 100 samples, then each sample should be output at 10,000/100 = 100 µs intervals. The sample values will be calculated using the trigonometric **sin** function of Python.

The **sin** function will have the format:

$$\sin\left(\frac{2\pi \times count}{T}\right)$$

where, T is the period of the waveform and is equal to 100 samples. Thus, the sine wave is divided into 100 samples and each sample is output at 100 µs. The above formula can be re-written as:

$$\sin\left(0.0628 \times count\right)$$

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*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:

$$1861.3 \times \sin\left(0.0628 \times count\right)$$

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 sine wave, which is 1861.3 when the **sin** function above is equal to 1.5. In this project, we are adding a 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:

$$1861.3 + 1861.2 \times \sin\left(0.0628 \times count\right)$$

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 12.34 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 sine wave 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  : October, 2024
#-------------------------------------------------------------------
import gpiod
from time import sleep                          # Import time
import spidev                                   # Import SPI
import math                                     # Import math

CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS.set_value(1)                                 # Disable CS

sample = 0
T = 100
R = 0.0628
amplitude = 1861.3
offset = 1861.3
sins = [None]*101

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.set_value(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

   CS.set_value(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
        sleep(0.0001)                          # Wait
        sample = sample + 1                    # Next sample
        if sample == 100:                      # 100 samples?
            sample  = 0

except KeyboardInterrupt:
    pass
```

Figure 12.34 Program listing.

An example output waveform taken from the oscilloscope is shown in Figure 12.35. Notice that the frequency of the waveform is not very accurate because the delay function of Python is not accurate.

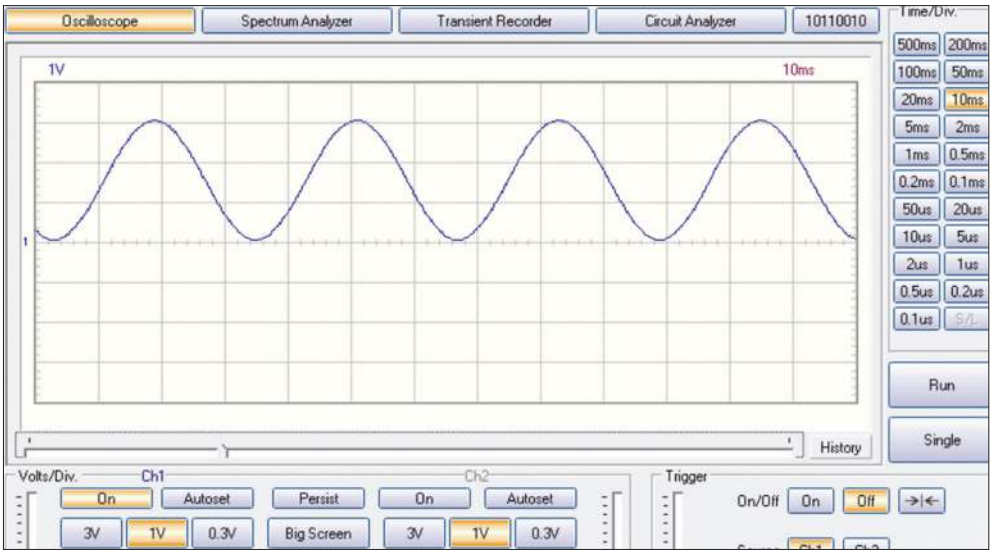


Figure 12.35 Example output waveform.

If you get a permission error, run the following command:

beagle@beagle:~ $ **sudo chmod ugo+rwx /dev/spidev***

12.7.7 Project 11 – SPI Port Expander

Description: This project is very similar to the port expender project given earlier in this chapter. In that project, the I^2C compatible chip MCP23017 was used. In this project, the SPI bus-compatible port expander chip MCP23S17 is used to give an additional 16 I/O ports to the BeagleY-AI. The operation of the MCP23S17 is identical to the operation of MCP23017, except that the MCP23S17 uses the SPI bus. In this project, an LED is connected to MCP23S17 port pin GPA0, and the LED is flashed On and Off every second, as in the I^2C based project. A 470 Ohm current-limiting resistor is used in series with the LED.

Block diagram: The block diagram of the project is the same as in Figure 12.1, but the MCP23017 chip is replaced with MCP23S17.

The MCP23S17

The MCP23S17 is a 28-pin chip with the following features. The pin configuration is shown in Figure 12.36, which is the same as the pin configuration of MCP23017, but SPI pins are used instead of I2C pins:

- 16 bi-directional I/O ports
- Up to 1.7 MHz operation on I2C bus
- Interrupt capability
- External reset input
- Low standby current

- +1.8 V to +5.5 V operation
- 3 address pins, so that up to 8 devices can be used on the SPI bus
- 28-pin DIL package

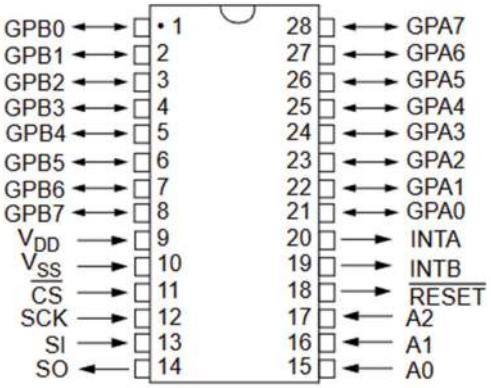


Figure 12.36 Pin configuration of the MCP23S17.

The pin descriptions are given in Table 12.3.

Pin	Description
GPA0-GPA7	Port A pins
GPB0-GPB7	Port B pins
VDD	Power supply
VSS	Ground
SI	SPI MOSI data pin
SCK	SPI clock pin
SO	SPI MISO data pin
CS	SPI SSEL chip enable pin
A0-A2	I2C address pins
RESET	Reset pin
INTA	Interrupt pin
INTB	Interrupt pin

Table 12.3 MCP23S17 pin descriptions

The MCP23S17 is a slave SPI device. The slave address contains four upper fixed bits (0100) and three user-defined hardware address bits (pins A2, A1, and A0) with the read/write bit filling out the control byte. These address bits are enabled/disabled by the control register IOCON.HAEN. By default, the user address bits are disabled at power-up (i.e., IOCON.HAEN = 0) and A2 = A1 = A0 = 0, and the chip is addressed with 0x40. As such, we can use two MCP23S17 chips on SPI0 by connecting one CS bit to CE0, and the other one to CE1, and addressing both chips with 0x40. By setting bit HAEN to 1, we can change the

addresses of the devices in multiple MCP23S17-based applications (e.g., more than 2) by connecting the A2, A1, and A0 accordingly. 16 such chips can be connected (8 to CE0 and 8 to CE1), corresponding to 16x16 = 256 I/O ports. Figure 12.37 and Figure 12.38 show the addressing format. The address pins should be externally biased even if disabled.

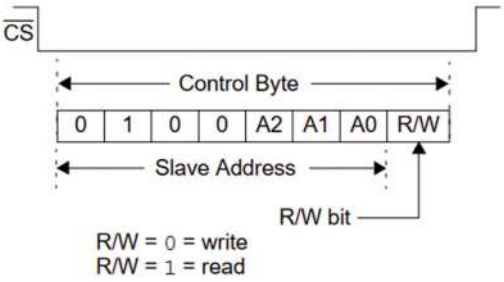


Figure 12.37 MCP23S17 control byte format.

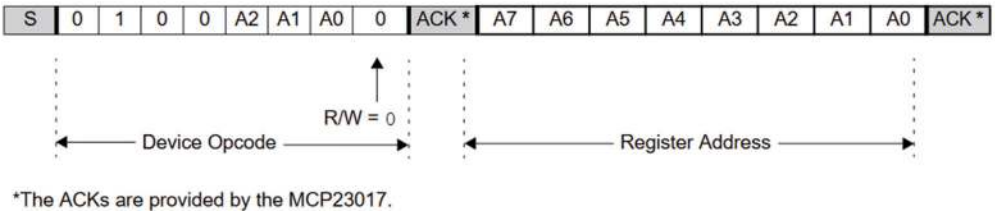


Figure 12.38 MCP23S17 addressing registers.

Like the MCP23017, the MCP23S17 chip has 8 internal registers that can be configured for its operation. The device can either be operated in 16-bit mode or in two 8-bit modes by configuring bit IOCON.BANK. On power-up, this bit is cleared, which chooses the two 8-bit mode by default.

The I/O direction of the port pins is controlled with registers IODIRA (at address 0x00) and IODIRB (at address 0x01). Clearing a bit to 0 in these registers makes the corresponding port pin(s) as output(s). Similarly, setting a bit to 1 in these registers makes the corresponding port pin(s) input(s). GPIOA and GPIOB register addresses are 0x12 and 0x13, respectively. This is shown in Figure 12.39.

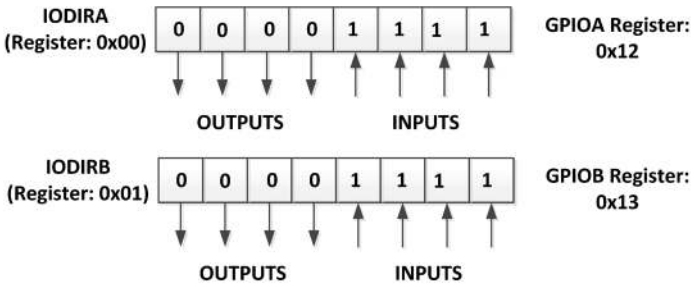


Figure 12.39 Configuring the I/O ports.

Further information on the MCP23S17 chip can be obtained from the Microchip Inc. data sheet at the following website:

http://ww1.microchip.com/downloads/en/DeviceDoc/20001952C.pdf

Circuit diagram: Figure 12.40 shows the circuit diagram of the project. CS is controlled separately and in this project, GPIO26 is used as the CS pin.

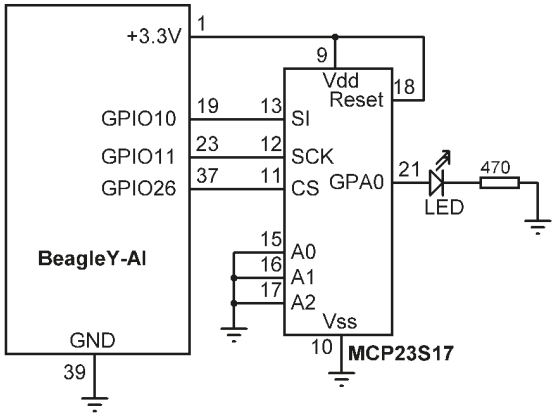


Figure 12.40 Circuit diagram of the project.

Program listing: Figure 12.41 shows the program listing (Program: **MCP23S17**). The programming of the MCP23S17 chip is as follows (notice that not all SPI devices require device addresses):

- Send device address (it is 0x40 in this project)
- Send register address
- Send register data

First of all, we have to program the I/O direction register **IODIRA** to 0 so that PORTA pins are outputs. This register has address 0x0. Then, we should program bit 0 of PORTA (pin GPIOA) where the LED is connected to. The address of register **GPIOA** is 0x12.

At the beginning of the program, the SPI interface signals between the BeagleY-AI and MCP23S17 are defined. The required addresses of the MCP23S17 and the **CS** connection are then defined, and CS is initially set to 1 so that the MCP23S17 chip command mode is disabled (CS must be controlled separately).

The function **Configure** configures PORTA as output. Function **Send** sends data to the specified port register (**RegAddr**) so that the required pin is at logic 1 or 0. Data is either 0 or 1. When 1, the LED is turned On, and when 0 the LED is turned Off. The main program runs in a loop and calls the function **Send** every second to flash the LED.

```
#-------------------------------------------------------
#              SPI BUS PORT EXPANDER
#              =====================
#
# In this project the SPI bus compatible MCP23S17 chip is used
# to add 16 more ports to BeagleY-AI SBC. An LED is connected
# to pin GPA0 of the expander and the LED is flashed every
# second
#
# Author: Dogan Ibrahim
# File  : MCP23S17.py
# Date  : October 2024
#-------------------------------------------------------------
import spidev
import gpiod
from time import sleep

CS = gpiod.find_line('GPIO26')
CS.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=1)

Device_Address = 0x40                  # MCP23S17 SPI address
spi = spidev.SpiDev()
spi.open(0, 0)                         # Bus=0, device=0
spi.max_speed_hz = 3900000

CS.set_value(1)
MCP_GPIOA = 0x12
MCP_IODIRA = 0                         # MCP IODIRA address

#
# This function configures PORTA as output
#
def Configure():
    buff = [0, 0, 0]
    buff[0] = Device_Address
    buff[1] = MCP_IODIRA
    buff[2] = 0
    CS.set_value(0)
    spi.writebytes([buff[0],buff[1],buff[2]])
    CS.set_value(1)

#
# This function sends data to register RegAddr
#
def Send(RegAddr, data):
    buff = [0, 0, 0]
```

```
    buff[0] = Device_Address
    buff[1] = RegAddr
    buff[2] = data
    CS.set_value(0)
    spi.writebytes([buff[0],buff[1],buff[2]])
    CS.set_value(1)

Configure()
#
# Main program flashes the LED every second
#
while True:
    Send(MCP_GPIOA, 1)                          # LED ON
    sleep(1)                                    # 1 second delay
    Send(MCP_GPIOA, 0)                          # LED OFF
    sleep(1)                                    # 1 second delay
```

Figure 12.41 Program listing.

12.8 Pulse Width Modulation (PWM

Pulse Width Modulation (PWM) is a commonly used technique for controlling the power delivered to analog loads using digital waveforms. Although analog voltages (and currents) can be used to control the delivered power, they have several drawbacks. Controlling large analog loads requires large voltages and currents that cannot easily be obtained using standard analog circuits and DACs. Precision analog circuits can be heavy, large, and expensive and they are also sensitive to noise. By using the PWM technique the average value of voltage (and current) fed to a load is controlled by switching the supply voltage On and Off at a fast rate. The longer the power on time, the higher the voltage supplied to the load.

Figure 12.42 shows a typical PWM waveform, which is essentially a repetitive positive pulse. The waveform has a period (**T)**, On time (**$T_{ON)}$**, and an OFF time (**$T - T_{ON}$**). The minimum and maximum values of the voltage supplied to the load are 0 and V_P respectively. The PWM switching frequency is usually set to be very high (usually in the order of several kHz) so that it does not affect the load being powered. The main advantage of PWM is that the load is operated efficiently since the power loss in the switching device is very low. When the switch is On there is practically no voltage drop across the switch, and when the switch is Off there is no current supplied to the load.

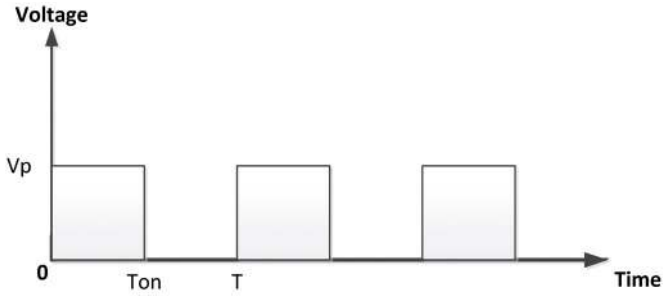


Figure 12.42 PWM waveform.

The duty cycle (or D) of a PWM waveform is defined as the ratio of the On time to its period. Expressed mathematically:

Duty Cycle (D) = T_{ON} / T

The duty cycle is usually expressed as a percentage, as follows:

D = (T_{ON} / T_{OFF}) x 100 %

By varying the duty cycle between 0% and 100%, we can effectively control the average voltage supplied to the load, from 0 and V_p.

The average value of the voltage applied to the load can be calculated by considering a general PWM waveform, as shown in Figure 1. The average value A of waveform f(t) with period T, peak value $y_{max,}$ and minimum value y_{min} is calculated as:

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

or,

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

In a PWM waveform y_{min} = 0, and the above equation becomes:

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

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

As seen from the above equation, the average value of the voltage supplied to the load is directly proportional to the duty cycle of the PWM waveform. By varying the duty cycle, we control the average voltage supplied to the load. Figure 12.43 shows the average voltage for different values of the duty cycle.

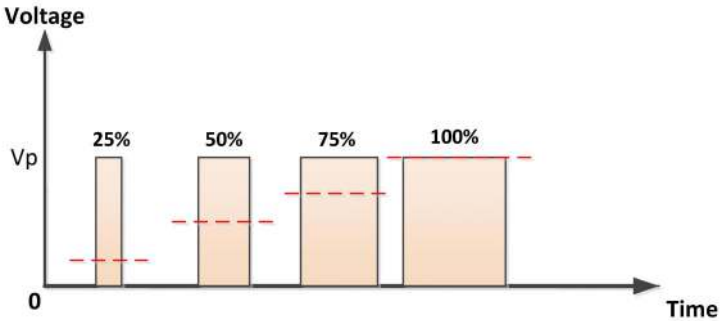


Figure 12.43 Average voltage (shown as dashed line) supplied to a load.

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

12.8.1 PWM channels of BeagleY-AI

The BeagleY-AI has hardware PWM channels at the following GPIO ports:

PWM0-A:	GPIO5, GPIO15
PWM0-B:	GPIO12, GPIO14
PWM1-A:	GPIO6, GPIO21
PWM1-B:	GPIO13, GPIO20

Example projects for using the PWM are given in this section.

12.8.2 Project 12 – Generate 1000Hz PWM waveform with 50% duty cycle

Description: In this project, we will generate a PWM waveform with a frequency of 1000 Hz and a duty cycle of 50% using GPIO20 (Pin 38, or hat-38). The aim of this project is to show how we can use the PWM functions.

To enable the PWM pin, we must include the PWM overlay in the file: **/boot/firmware/extlinux/extlinux.conf.** Run the following command to see a list of the overlays (Figure 12.44):

```
beagle@beagle:~ $ ls /boot/firmware/overlays/ | grep beagley-ai-pwm
```

```
beagle@beagle:~$ ls /boot/firmware/overlays/ | grep beagley-ai-pwm
k3-am67a-beagley-ai-pwm-ecap0-gpio12.dtbo
k3-am67a-beagley-ai-pwm-ecap1-gpio16.dtbo
k3-am67a-beagley-ai-pwm-ecap1-gpio21.dtbo
k3-am67a-beagley-ai-pwm-ecap2-gpio17.dtbo
k3-am67a-beagley-ai-pwm-ecap2-gpio18.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio12.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio14.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio15-gpio12.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio15-gpio14.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio15.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio5-gpio12.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio5-gpio14.dtbo
k3-am67a-beagley-ai-pwm-epwm0-gpio5.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio13.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio20.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio21-gpio13.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio21-gpio20.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio21.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio6-gpio13.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio6-gpio20.dtbo
k3-am67a-beagley-ai-pwm-epwm1-gpio6.dtbo
beagle@beagle:~$
```

Figure 12.44 List of PWM overlays.

Select the overlay for GPIO20. i.e., **k3-am67a-beagley-ai-pwm-epwm1-gpio20.dtbo**

- beagle@beagle:~ $ **sudo nano /boot/firmware/extlinux/extlinux.conf**

  Go to the end of the file and type:

  fdtoverlays /overlays/k3-am67a-beagley-ai-pwm-epwm1-gpio20.dtbo

- Press **Ctrl+X**, then **Y** to save and exit. Figure 12.45 shows the end of the file.

```
fdtdir /
fdt /ti/k3-am67a-beagley-ai.dtb
#fdtoverlays /overlays/k3-am67a-beagley-ai-spidev0.dtbo
#initrd /initrd.img

fdtoverlays /overlays/k3-am67a-beagley-ai-pwm-epwm1-gpio20.dtbo
initrd /initrd.img
```

Figure 12.45 Include the overlay file.

- Reboot your BeagleY-AI

  beagle@beagle:~ $ **sudo reboot**

- Run the following command to make sure that the pwm overlay has been loaded (see Figure 12.46)

  beagle@beagle:~ $ **sudo beagle-version | grep UBOOT**

```
beagle@beagle:~$ sudo beagle-version | grep UBOOT
UBOOT: Booted Device-Tree:[k3-am67a-beagley-ai.dts]
UBOOT: Loaded Overlay:[k3-am67a-beagley-ai-pwm-epwm1-gpio20.kernel]
beagle@beagle:~$
```

Figure 12.46 Check the overlay.

- Export pin GPIO38 (hat-38) as pwm

beagle@beagle@~ $ **sudo beagle-pwm-export --pin hat-38**

Program listing: Figure 12.47 shows the program listing (Program: **PWM1.py**). The period and the duty cycle must be specified in nanoseconds, where the duty cycle is the On time in nanoseconds. For a 1000 Hz signal, the period is 1 ms, or 1,000,000 ns. For a 50% duty cycle, the On time must be 500,000 ns. Figure 12.48 shows the generated waveform on the oscilloscope. Here, the horizontal axis was 500 µs/division, and the vertical axis was 2 V/division. Clearly, the period of the generated waveform is 1 ms (frequency = 1000 Hz), the duty cycle is 50%, and the amplitude is about 3.3 V.

```
#-------------------------------------------------------
#
#                     PWM EXAMPLE
#                     ===========
#
# This is a PWM example where a 1000Hz square wave signal is
# generated with a 50% duty cycle on pin GPIO20 of the BeagleY-AI
#
# Program: PWM1.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------

def write_file(path, value):
   f = open(path, 'w')
   f.write(str(value))
   f.close()

write_file("/dev/hat/pwm/GPIO20/period",1000000)
write_file("/dev/hat/pwm/GPIO20/duty_cycle", 500000)
write_file("/dev/hat/pwm/GPIO20/enable", 1)

while True:
  pass
```

Figure 12.47 Program: **PWM1.py**.

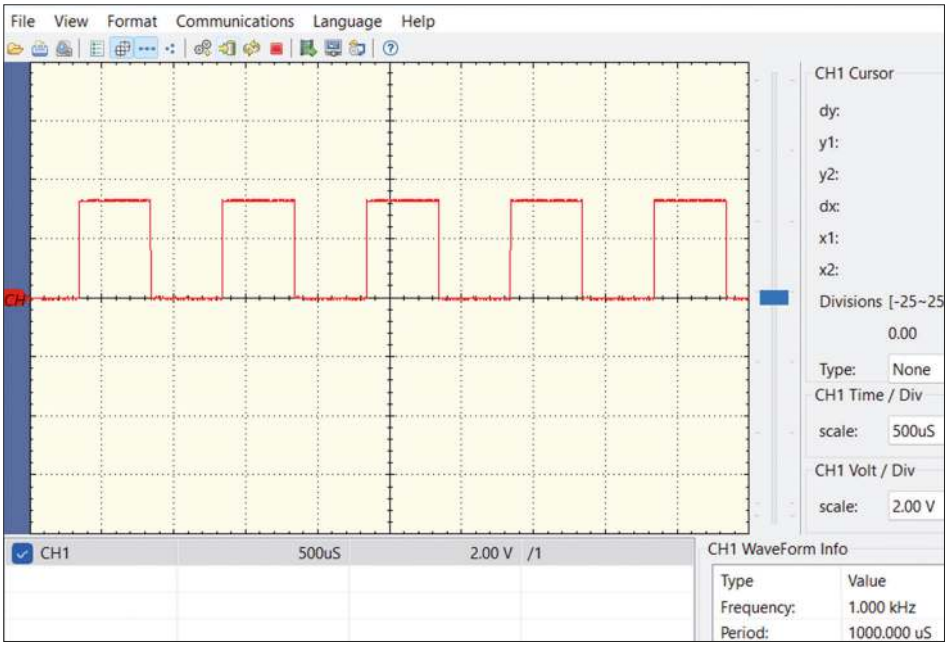


Figure 12.48 Generated PWM waveform.

12.8.3 Project 13 – Changing the brightness of an LED

Description: In this project, an LED is connected to port GPIO20 through a 470 Ohm current-limiting resistor. The program changes the brightness of the LED by adjusting the duty cycle of the PWM voltage sent to the LED. The aim of this project is to show how the PWM can be used in a project.

Program listing: Figure 12.49 shows the program listing (Program: **LEDfade.py**). The frequency is set to 1000 Hz to ensure the LED light remains steady (i.e., not flashing). As the duty cycle is increased from 0% to 100% in steps, the LED brightness gradually increases.

```
#------------------------------------------------------
#
#                      PWM FADE LED
#                      ============
#
# This is a PWM example where an LED is connected to GPIO20
# and this port is configured as PWM port. The frequency of PWM
# is set to 1000 Hz but the duty cycle is changed from 0% to 100%
#
# Program: LEDfade.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#------------------------------------------------------
```

```
import time

def write_file(path, value):
   f = open(path, 'w')
   f.write(str(value))
   f.close()

write_file("/dev/hat/pwm/GPIO20/period",1000000)
write_file("/dev/hat/pwm/GPIO20/duty_cycle", 500000)
write_file("/dev/hat/pwm/GPIO20/enable", 1)

i = 0
while True:
   write_file("/dev/hat/pwm/GPIO20/duty_cycle", i)
   time.sleep(0.4)
   i = i + 80000
   if i > 1000000:
     i = 0
```

Figure 12.49 Program listing.

Export pin GPIO38 (hat-38) as pwm

beagle@beagle@~ $ **sudo beagle-pwm-export --pin hat-38**

12.8.4 Project 14 – Mosquito repeller

Description: The concept of mosquito repeller is very simple. A sound with a frequency higher than 20 kHz is termed ultrasonic. Humans can only hear sounds in the frequency range of 20 Hz to 20 kHz. It is well known that various animals and insects can hear ultrasonic sounds. Male mosquitos emit sounds in the range of 20 kHz to 40 kHz. After breeding, female mosquitos tend to avoid male mosquitos, and therefore, they tend to avoid ultrasonic sounds. Most mosquito repellers generate 40 kHz ultrasonic sounds through an ultrasonic transducer. In this project, a 40 kHz ultrasonic sound is generated using the BeagleY-AI.

Circuit diagram: Figure 12.50 shows the circuit diagram. Basically, an ultrasonic transducer is used through a transistor switch connected to pin GPIO20 of the BeagleY-AI.

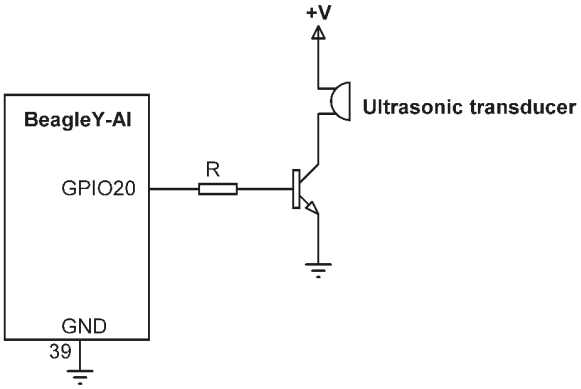


Figure 12.50 Circuit diagram of the project.

Program listing: Figure 12.51 shows the program listing (Program: **Ultra.py**). The program is very similar to the previous one. A 40 kHz sound wave has a period of 2500 ns. Choosing a 50% duty cycle, the waveform On time, or the duty cycle setting should be 12500 ns as shown in Figure 12.51. Figure 12.52 shows the output waveform on an oscilloscope.

```
#-------------------------------------------------------
#
#                      MOSQUITO REPELLER
#                      =================
#
# This is a mosquito repeller program where a 40kHz sound
# is generated and sent to an ultrasonic transducer
#
# Program: Ultra.py
# Date   : October, 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------

def write_file(path, value):
   f = open(path, 'w')
   f.write(str(value))
   f.close()

write_file("/dev/hat/pwm/GPIO20/period",25000)
write_file("/dev/hat/pwm/GPIO20/duty_cycle", 12500)
write_file("/dev/hat/pwm/GPIO20/enable", 1)

while True:
  pass
```

Figure 12.51 Program listing.

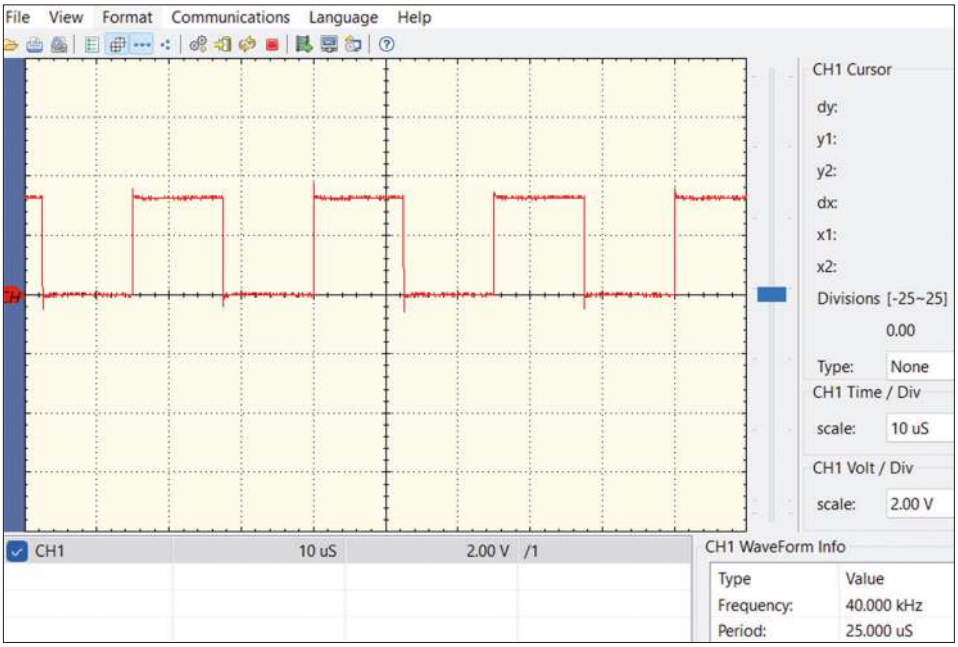


Figure 12.52 Output waveform.

Export pin GPIO38 (hat-38) as pwm

```
beagle@beagle@~ $ sudo beagle-pwm-export --pin hat-38
```

Chapter 13 • Communication Over the Wi-Fi

13.1 Overview

Three major features of BeagleY-AI are its Wi-Fi, Bluetooth communication, and AI capabilities. BeagleY-AI is equipped with a dual-band 2.4GHz IEEE802.11ax wireless LAN module and Bluetooth Low Energy (BLE) 5.4. Without these built-in features, you would typically need to use external network-based hardware communication modules to communicate over the Internet. Network communication is handled using either UDP or TCP protocols. In this chapter, you will learn how to write Python programs using both the UDP and TCP protocols using the on-board Wi-Fi module.

13.2 UDP and TCP

Communication over a Wi-Fi link is in the form of a client and server, and sockets are used to send and receive data packets. The server side usually waits for a connection from the clients, and once a connection is made two-way communication can start. Two protocols are mainly used for sending and receiving data packets over a Wi-Fi link: UDP and TCP. TCP is a connection-based protocol, which guarantees the delivery of packets. Packets are given sequence numbers, and the receipt of all the packets is acknowledged to avoid them arriving in the wrong order. As a result of this confirmation, TCP is usually slow, but it is reliable, as it guarantees the delivery of packets. UDP, on the other hand, is not connection-based. Packets do not have sequence numbers, and as a result of this, there is no guarantee that the packets will arrive at their destinations, or they may arrive in the wrong sequence. UDP has less overhead than TCP and as a result, it is faster. Table 13.1 lists some of the differences between the TCP and UDP protocols.

TCP	UDP
Packets have sequence numbers and delivery Clarity Engagement Delivery Correct article usage client of every packet is acknowledged	There is no delivery acknowledgment
Slow	Fast
No packet loss	Packets may be lost
Large overhead	Small overhead
Requires more resources	Requires less resources
Connection based	Not connection based
Not suitable for multicast	Has multicast capability
More difficult to program	Easier to program
Examples: HTTP, HTTPS, FTP	Examples: DNS, DHCP, Computer games

Table 13.1 TCP and UDP packet communications.

13.2.1 UDP communication

Figure 13.1 shows the UDP communication over a Wi-Fi link:

Server

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

Client

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

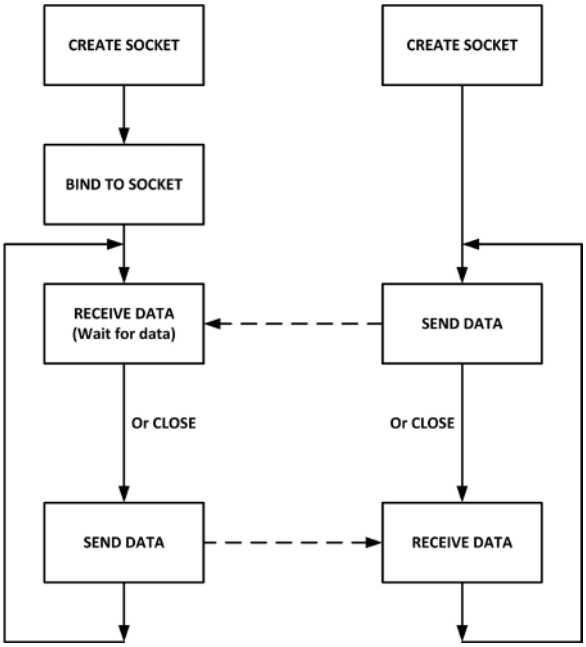


Figure 13.1 UDP communication.

13.2.2 TCP communication

Figure 13.2 shows the TCP communication over a Wi-Fi link:

Server

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

Client

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

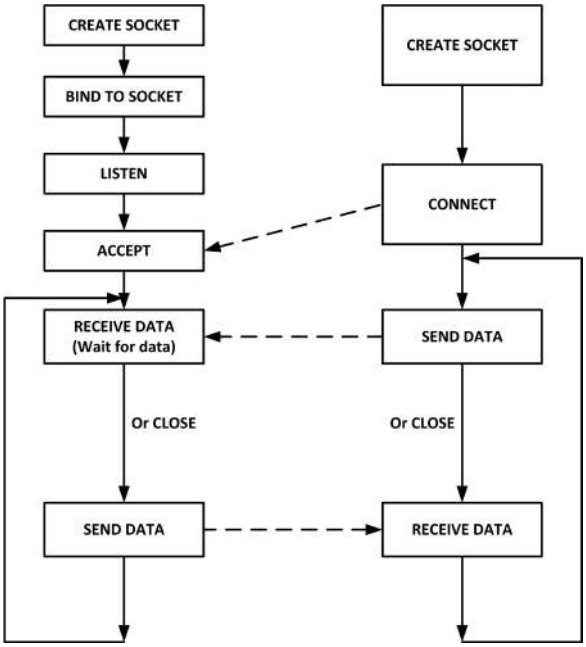


Figure 13.2 TCP communication.

13.3 Project 1 – Sending a Text Message to a Smartphone Using TCP

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

Background Information: Port numbers range from 0 to 65535. Numbers from 0 to 1023 are reserved and are called well-known ports. For example, port 23 is the Telnet port, port 25 is the SMTP port, etc. In this section, you will be using port number 1500 in your program. BeagleY-AI is the Server node in this example, and smartphone is the Client node.

Block diagram: Figure 13.3 shows the project block diagram where the BeagleY-AI and smartphone communicate over a Wi-Fi router.

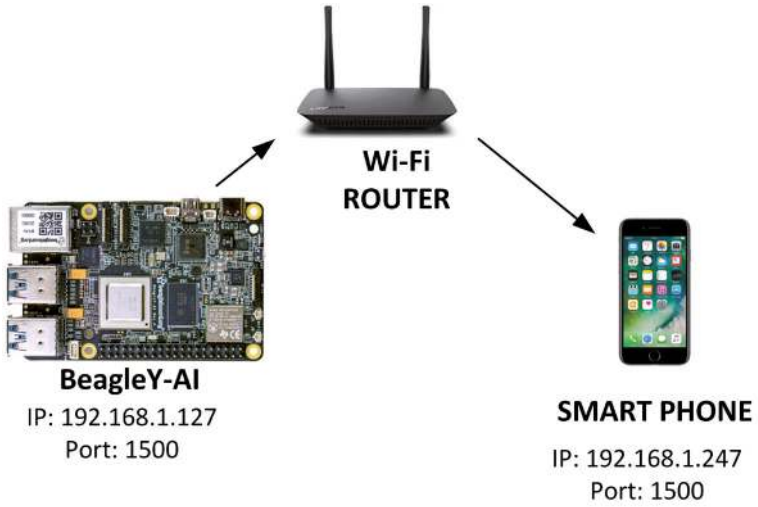


Figure 13.3 Block diagram of the project.

Program listing: In this project, BeagleY-AI is the server. Figure 13.4 shows the program listing (**tcpserver.py**). At the beginning of the program, a TCP/IP socket is created (**sock.SOCK_STREAM**) and is then bound to port 1500. The program listens for a connection. Notice that it is possible for the server to listen to multiple clients, but it can only communicate with one at any time. When the client makes a connection, this is accepted by the server. The server then reads a message from the keyboard and sends it to the client over the Wi-Fi link. Notice that the **setsockopt()** statement ensures that the program can be used again without having to wait for the socket timeout of 30 seconds.

```
#================================================================
#      SEND TEXT MESSAGES USING TCP/IP
#      ===============================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcpserver.py
# Date  : October, 2024
```

```
#============================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.1.127", 1500))
sock.listen(1)
client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

yn = 'y'

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

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

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

Figure 13.4 Program listing.

Testing
You should stop the firewall running on your BeagleY-AI if it is running. Enter the following command to check the status of the firewall:

beagle@beagle:~ $ **sudo ufw status**

If the firewall is running, you should disable it using the following command:

beagle@beagle:~ $ **sudo ufw disable**

There are many TCP apps available free of charge on the Internet for smartphones. In this project, the **TCP Client by JOY S.R.L.** app is used on an Android smartphone. This app is available free of charge in the **Play Store** (see Figure 13.5).

Figure 13.5 Apps used in the project.

The program is run as follows:

- Run the server program first:

  beagle@beagle:~ $ **python tcpserver.py**

- Run the Android app and configure it as shown in Figure 13.6 (click the 3-line settings icon at the top right-hand of the screen), where 192.168.1.127 is the IP address of the BeagleY-AI.

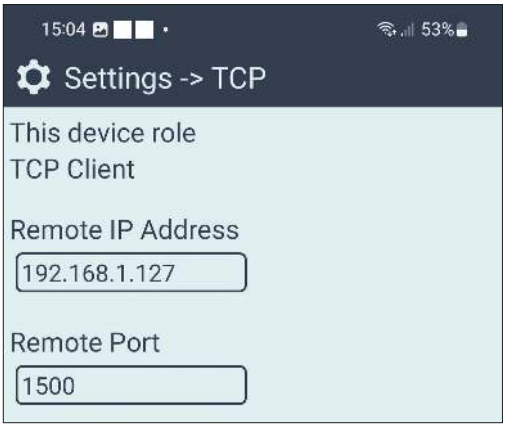


Figure 13.6 Configure the TCP Client app.

- Click **CONNECT** in the settings menu to connect to BeagleY-AI over TCP/IP.

- You should see a connection message on your BeagleY-AI screen and also the IP address of the remote Android smartphone. Now enter a message and press the Enter key. In this example, the message **HELLO FROM BEAGLEY-AI** is sent to the client (Figure 13.7). Figure 13.8 shows the message displayed on the smartphone.

```
beagle@beagle:~$ python tcpserver.py
Connected to client:  ('192.168.1.247', 39638)
Enter your message: HELLO FROM BEAGLEY-AI
Send more messages?: N

Closing connection to client
beagle@beagle:~$
```

Figure 13.7 Enter the message on the keyboard.

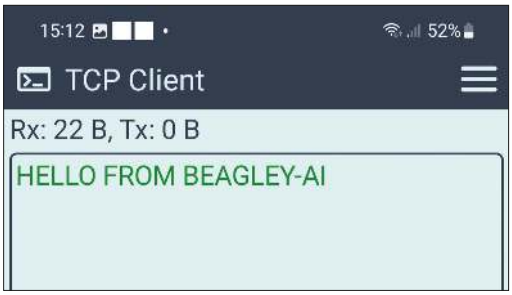


Figure 13.8 Message displayed on the smartphone.

13.4 Project 2 – Two-way Communication with the Smartphone Using TCP

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

The block diagram of the project is the same as in Figure 13.3

Program listing: Figure 13.9 shows the program listing (**tcp2way.py**). Here, port 1500 is used, as in the previous project. The program has been changed to send and receive messages from the smartphone. Socket function **recv(byte count)** sends messages over the TCP/IP link to the connected node.

```
#================================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      ======================================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcp2way.py
# Date  : October, 2024
#================================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.1.127", 1500))
sock.listen(1)

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

yn = 'y'

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

      msg = client.recv(1024)
      print("Received message: ")
      print(msg.decode('utf-8'))

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

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

Figure 13.9 Program listing.

Testing

You will be using the Android app as in Figure 13.5. Start the BeagleY-AI server program and then exchange messages between the smartphone and BeagleY-AI. Example communication is shown in Figure 13.10. In this example, BeagleY-AI sends the message **Message from BEAGLEY-AI**. In return, the Android smartphone sends the message **message from ANDROID**.

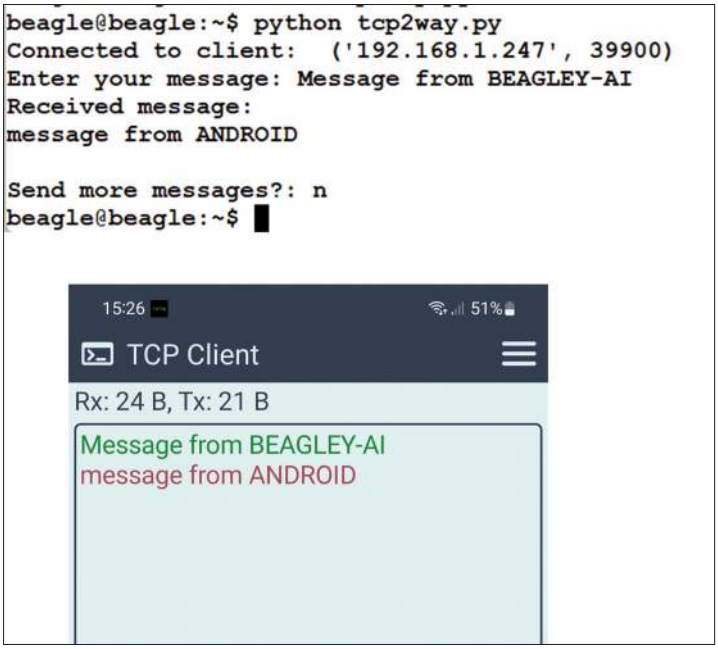


Figure 13.10 Example communication between BeagleY-AI and Android app.

13.5 Project 3 – Communicating with a PC Using TCP

Description: In this project, a TCP/IP-based communication is established between the BeagleY-AI and a PC running Python. Messages are exchanged between the BeagleY-AI and the PC. The aim of this project is to show how TCP/IP communication can be established with a PC.

Background Information: In this project, BeagleY-AI is the server and PC is the client. The programs on both sides are developed using the Python programming language. Python 3.11 is used on the PC. If you do not have Python on your PC, you can install it from the following website:

https://www.python.org/downloads/

Block diagram: Figure 13.11 shows the block diagram.

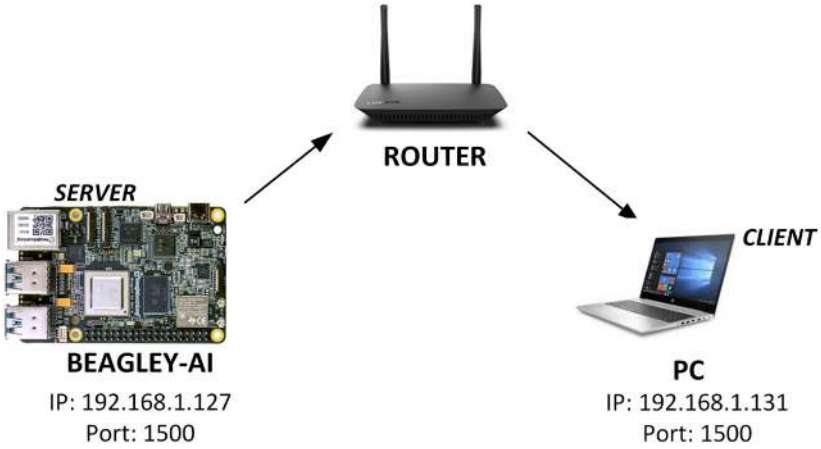


Figure 13.11 Block diagram.

BeagleY-AI program Listing: The BeagleY-AI program listing is shown in Figure 13.12 (**tcppc.py**). The program is very similar to the one given in Figure 13.9, i.e., program: **tcp2way.py**. You should terminate the program by entering **Ctrl+C**.

```
#================================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      =======================================
#
# This is the TCP/IP server program. It communicates with a PC
# running TCP/IP on the same port
#
# Author: Dogan Ibrahim
# File  : tcppc.py
# Date  : October, 2024
#================================================================
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.1.127", 1500))
sock.listen(1)

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

try:
   while True:
      msg = input("Enter your message: ")       # read a message
      msg = msg +"\n"
```

```
      client.send(msg.encode('utf-8'))         # send the message

      msg = client.recv(1024)
      print("Received message: ", msg.decode('utf-8'))

except KeyboardInterrupt:
      print("\nClosing connection to client")
      sock.close()
      time.sleep(1)
```

Figure 13.12 BeagleY-AI program listing.

PC Program Listing: The PC program listing is shown in Figure 13.13 (**client.py**). The program creates a socket and connects to the server. Then, messages are exchanged between the client and the server.

```
#============================================================
#               TCP/IP CLIENT
#               =============
#
# This is the client program on the PC.The program exchanges
# messages with the server on the BeagleY-AI
#
# Author: Dogan Ibrahim
# File  : client.py
# Date  : October, 2023
#============================================================
import socket
import time
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.connect(("192.168.1.127", 1500))

try:
    while True:
        msg = sock.recv(1024)
        print("Received message: ", msg.decode('utf-8'))
        data = input("Enter message to send: ")
        sock.send(data.encode('utf-8'))

except KeyboardInterrupt:
    print("Closing connection to server")
    sock.close()
    time.sleep(1)
```

Figure 13.13 PC program listing.

The steps to run the program are as follows:

- Run the server program on BeagleY-AI
- Run the client program on the PC
- Write messages as desired

Note: You may find that after exiting the program you may not be able to run it again. This is because the socket stays open for about 30 seconds, and the error message saying that the **Address is already in use** may be displayed. You can check the state of the port with the following command:

```
pi@raspberrypi:~ $ netstat –n | grep 5000
```

If the display includes the text **ESTABLISHED,** then it means that the socket has not been closed properly, and you will have to restart your BeagleY-AI to run the program again. If, on the other hand, you see the message with **TIME_WAIT,** then you should wait about 30 seconds before re-starting the program.

13.6 Project 4 – Controlling an LED Connected to BeagleY-AI from a Smartphone Using TCP

Description: In this project, an LED is connected to BeagleY-AI. The LED is turned On and Off by sending commands On and Off respectively from an Android smartphone. The aim of this project is to show how an LED connected to BeagleY-AI can be controlled from an Android smartphone remotely by sending commands using the TCP/IP protocol over a Wi-Fi link. In this project, BeagleY-AI is the server, and the smartphone is the client.

Block diagram: Figure 13.14 shows the block diagram of the project.

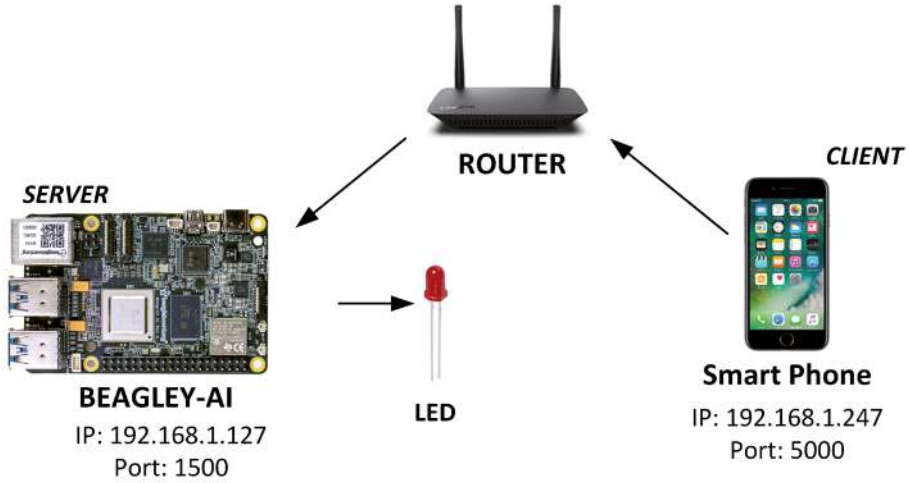


Figure 13.14 Block diagram of the project.

The LED is connected to port pin GPIO21 (pin 40) through a 470-ohm current limiting resistor.

Program Listing: Figure 13.15 shows the program listing (program: **serverled.py**). As in the previous program, a socket is created and port 1500 is used. The LED is assigned to port GPIO pin 21, and it is turned Off at the beginning of the program. The server waits for a connection from the client and then accepts the connection and displays the message **Connected**. It then waits to receive a command from the client. If the command is **On,** then the LED is turned On. If, on the other hand, the command is **Off** then the LED is turned Off. Sending the command **X** terminates the server connection and exits the program.

```
#================================================================
#              CONTROL LED FROM SMART PHONE
#              ============================
#
# In this program TCP/IP is used where BeagleY-AI is the server
# and the smartphone is the client. An LED connected to BeagleY-AI
# GPIO21 and is controlled from the smartphone
#
# Author: Dogan Ibrahim
# File  : serverled.py
# Date  : November, 2024
#================================================================
import socket
import gpiod
from time import sleep

led = gpiod.find_line('GPIO21')
led.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

led.set_value(0)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.1.127", 1500))     # BeagleY-AI IP
sock.listen(1)
client, addr = sock.accept()
print("Connected")

data = [' '] * 10


while data != b'X\n':                  # Terminate?
  data = client.recv(1024)
  if data == b'ON\n':                  # ON
     led.set_value(1)
  elif data == b'OFF\n':               # OFF
```

```
     led.set_value(0)

sock.close()
sleep(1)
```

Figure 13.15 Program listing.

The program can be tested using the Android app **TCP client** (Figure 13.5) used in Project 1. The server program is started, then the client is started. Figure 13.16 shows sending the **On** command to turn On the LED.

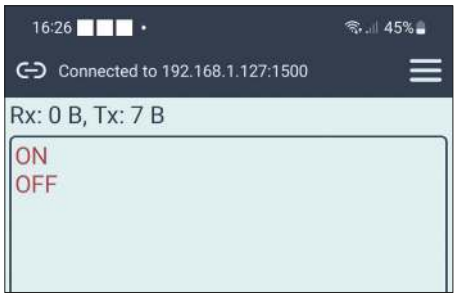


Figure 13.16 Command sent to turn On/Off the LED.

Suggestions: The LED in this project can be replaced, for example, with a relay, and electrical equipment can be controlled remotely over Wi-Fi.

13.7 Project 5 – Sending a Text Message to a Smartphone Using UDP

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

The block diagram is the same as in Figure 13.3.

Program Listing: In this project, BeagleY-AI is the server and the smartphone is the client. Figure 13.17 shows the program listing (**udpserver.py**). At the beginning of the program, a UDP socket is created (**sock.SOCK_DGRAM**) and is then bound to port 1500. The server program then reads text messages sent from the smartphone and displays them on the screen. Messages sent by the BeagleY-AI are displayed on the smartphone.

```
#============================================================
#      SEND TEXT MESSAGES USING UDP
#      ============================
#
# This is the UDP server program running on BeagleY-AI.
# The program exchanges text messages with an Android
# smartphone
```

```
#
# Author: Dogan Ibrahim
# File  : udpserver.py
# Date  : October, 2024
#=======================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.1.127", 1500))

try:
  while True:
    print()
    print("Waiting for messages")
    data, addr = sock.recvfrom(1024)
    print(addr)
    print("Received msg:", data.decode('utf-8'))
    msg = input("Message to send: ")
    sock.sendto(msg.encode('utf-8'), addr)
    print("Message sent")

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

Figure 13.17 Program listing.

There are many UDP apps available free of charge for both Android and iOS smartphones. In this project, **TCP UDP Server & Client** from *Stervs* for Android smartphones is used (Figure 13.18).

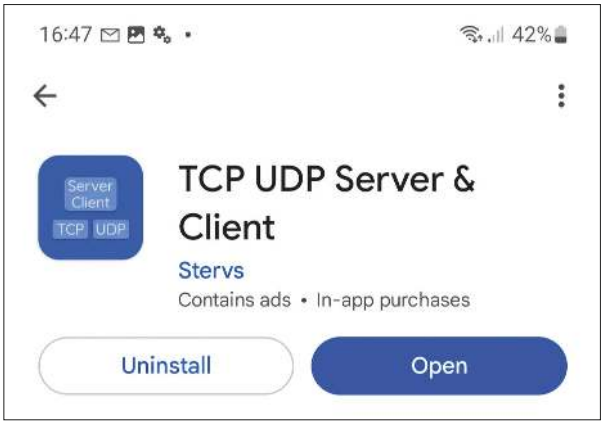


Figure 13.18 UDP app.

The steps to test the program are as follows:

- Start the server program on BeagleY-AI:

  beagle@beagle:~ $ **python udpserver.py**

- Start the smartphone app and click **Create client** at the bottom of the screen. Set the **Server-IP** and **Port** number, and click the arrow to run it (Figure 13.19).

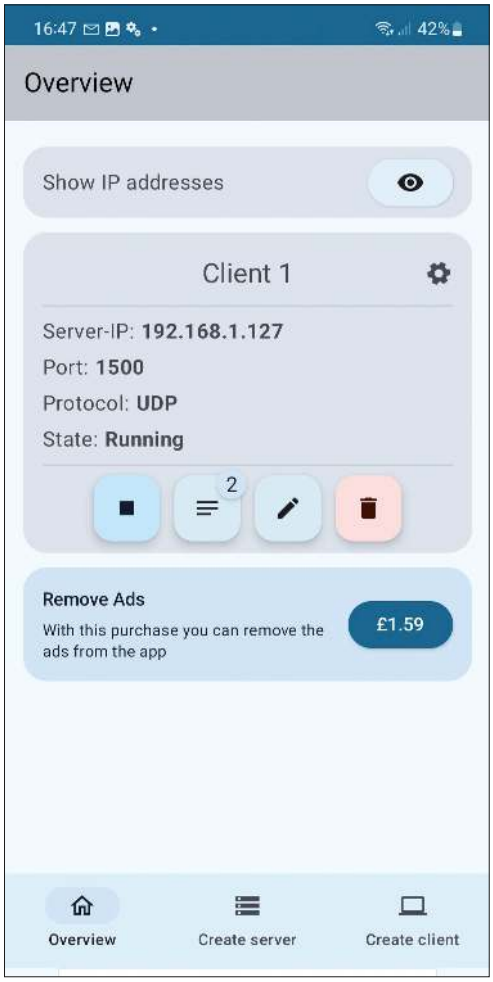


Figure 13.19 Configure the UDP app.

- Click the three lines to open a screen, write a message on the mobile phone app, and then click **SEND**. The message **Hello from Android** was sent as an example (Figure 13.20).

- Write a message on BeagleY-AI, and this message will be displayed on the smartphone. **Hello from BeagleY-AI** was sent for an example (Figure 13.20).

- Enter **Ctrl+C** on BeagleY-AI to close the socket

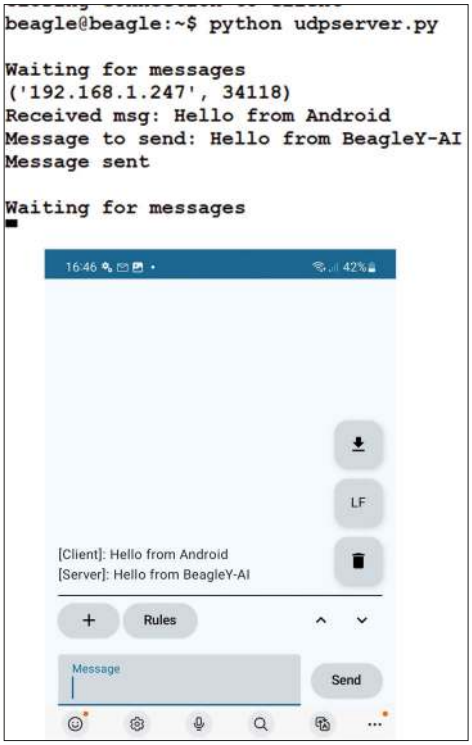


Figure 13.20 Sending and receiving messages.

13.8 Project 6 – Controlling an LED Connected to BeagleY-AI from a Smartphone Using UDP

Description: In this project, an LED is connected to BeagleY-AI port pin GPIO21 (pin 40) through a 470-ohm current limiting resistor. The LED is turned On and Off by sending commands **On** and **Off** respectively from an Android smartphone. The aim of this project is to show how an LED on BeagleY-AI can be controlled from a smartphone by sending commands using the UDP protocol over a Wi-Fi link. Here, BeagleY-AI is the server and the smartphone is the client.

The LED can easily be replaced with a relay, for example, to control electrical appliances from a smartphone.

Program Listing: Figure 13.21 shows the program listing (**udpled.py**). As in the previous program, a socket is created and the server waits to receive commands from a client to control the LED. If the command is **On**, then the LED is turned On. If, on the other hand, the command is **Off**, the LED is turned Off. Command **X** terminates the server program.

```
#=====================================================================
#      CONTROL LED FROM SMARTPHONE
#      ===========================
#
# In this program UDP is used where BeagleY-AI is the server
# and the smartphone is the client. An LED connected to the server
# and is controlled from the smartphone
#
# Author: Dogan Ibrahim
# File  : udpled.py
# Date  : October, 2024
#=====================================================================
import socket
import gpiod
from time import sleep

led = gpiod.find_line('GPIO21')
led.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT, default_val=0)

led.set_value(0)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.1.127", 1500))            # Bind to Zero 2 W IP,port

data = [' '] * 10
while data != b'X':
  data, addr = sock.recvfrom(1024)
  if data == b'ON':                           # ON command
     led.set_value(1)                         # LED ON
  elif data == b'OFF':                        # OFF command
     led.set_value(0)                         # LED OFF

sock.close()
sleep(1)
```

Figure 13.21 Program listing.

The program can be tested using the **UDP Sender/Receiver** app used in Figure 13.18.

The steps to test the program are as follows:

- Construct the circuit on BeagleY-AI with the LED.
- Start the server program on BeagleY-AI:

  beagle@beagle:~ $ **python udpled.py**

- Start and configure the smartphone app.

- Write the command **On** and press **Send** on the smartphone. The LED should turn On. Similarly, write **Off** and the LED should turn Off. Sending **X** should terminate the BeagleY-AI program.

13.9 Communicating with the Raspberry Pi Pico W over Wi-Fi

Raspberry Pi Pico W (it will be called Pico from now on) is a low-cost $6 microcontroller module based on the RP2040 microcontroller chip with a dual-core Cortex-M0+ processor with an on-board Wi-Fi module. Figure 13.22 shows the front view of the Pico hardware module, which is essentially a small board. In the middle of the board is the tiny 7 x 7mm RP2040 microcontroller chip housed in a QFN-56 package. At the two edges of the board, there are 40 gold-colored male GPIO (General-Input-Output) pins with holes. You should solder pins into these holes so that external connections can be made easily to the board. The holes are numbered starting with number 1 at the top left corner of the board and the numbers increase downwards up to number 40, which is at the top right-hand corner of the board. The board is breadboard compatible (i.e., 0.1-inch pin spacing), and after soldering the pins, the board can be plugged into a breadboard for easy connection to the GPIO pins using jumper wires. Next to these holes, you will see bumpy circular cut-outs which can be plugged on top of other modules without having any physical pins fitted.

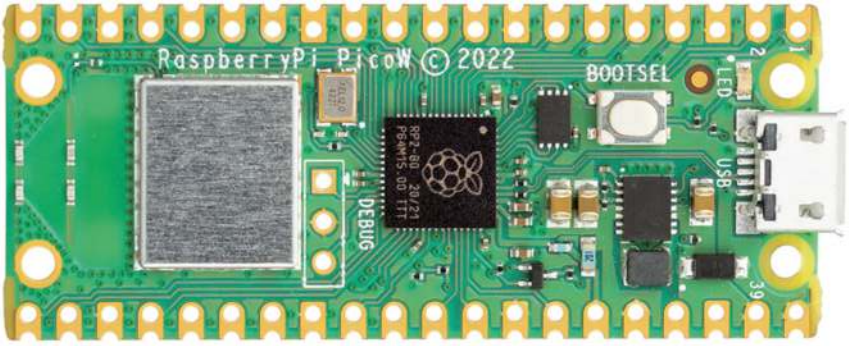


Figure 13.22 Front view of the Pico hardware module.

At one edge of the board, there is the micro-USB B port for providing power to the board and for programming it. Next to the USB port, there is an on-board user LED that can be used during program development. Next to this LED, there is a button named BOOTSEL that is used during the programming of the microcontroller, as we will see in the next chapters. Next to the processor chip, there are three holes where external connections can be made. These are used to debug your programs using Serial Wire Debug (SWD). At the other edge of the board is the single-band 2.4 GHz Wi-Fi module (802.11n). Next to the Wi-Fi module is the on-board antenna.

You will notice the following types of letters and numbers at the back of the board:

GND	-	power supply ground (digital ground)
AGND	-	power supply ground (analog ground)
3V3	-	+3.3 V power supply (output)
GP0 – GP22	-	digital GPIO
GP26_A0 – GP28_A2	-	analog inputs
ADC_VREF	-	ADC reference voltage
TP1 – TP6	-	test points
SWDIO, GND, SWCLK	-	debug interface
RUN	-	default RUN pin. Connect LOW to reset the RP2040
3V3_EN	-	this pin by default enables the +3.3 V power supply. +3.3 V can be disabled by connecting this pin LOW
VSYS	-	system input voltage (1.8 V to 5.5 V) used by the on-board SMPS to generate +3.3 V supply for the board
VBUS	-	micro-USB input voltage (+5 V)

Some of the GPIO pins are used for internal board functions. These are:

GP29 (input)	-	used in ADC mode (ADC3) to measure VSYS/3
GP24 (input)	-	VBUS sense - HIGH if VBUS is present, else LOW
GP23 (output)	-	controls the on-board SMPS Power Save pin

The specifications of the Pico hardware module are as follows:

- 32-bit RP2040 Cortex-M0+ dual-core processor operating at 133 MHz
- 2 MB Q-SPI Flash memory
- 264KB SRAM memory
- 26 GPIO (+3.3 V compatible)
- 3 x 12-bit ADC pins
- Accelerated floating point libraries on-chip
- On-board single-band Infineon CYW43439 wireless chip, 2.4 GHz wireless interface (802.11b/g/n), and Bluetooth 5.2 (not supported at the time of writing)
- Serial Wire Debug (SWD) port
- Micro-USB port (USB 1.1) for power (+5 V) and data (programming)
- 2 x UART, 2 x I2C, 2 x SPI bus interface
- 16 x PWM channels
- 1 x Timer (with 4 alarms), 1 x Real-Time Counter
- On-board temperature sensor
- On-board LED at GPIO0, controlled by the 43439 module
- Castellated module allowing soldering directly to carrier boards
- 8×Programmable IO (PIO) state machines for custom peripheral support
- MicroPython, C, C++ programming
- Drag & drop programming using mass storage over USB

Pico GPIO hardware is +3.3 V compatible, so it is important not to exceed this voltage when interfacing external input devices to the GPIO pins. +5 V to +3.3 V logic converter circuits or resistive potential divider circuits must be used if it is required to interface devices with +5 V outputs to the Pico GPIO pins.

Pico can be programmed using MicroPython or C/C++ languages. It is assumed that the readers have Pico development boards with MicroPython installed. It will also be useful if the readers are familiar with using the Thonny with the Pico. An excellent book entitled: **Raspberry Pi Pico W**, written by the author, is available on the Elektor website, and interested readers should purchase this book for developing Pico-based projects.

Figure 13.23 shows the pin configuration of the Pico.

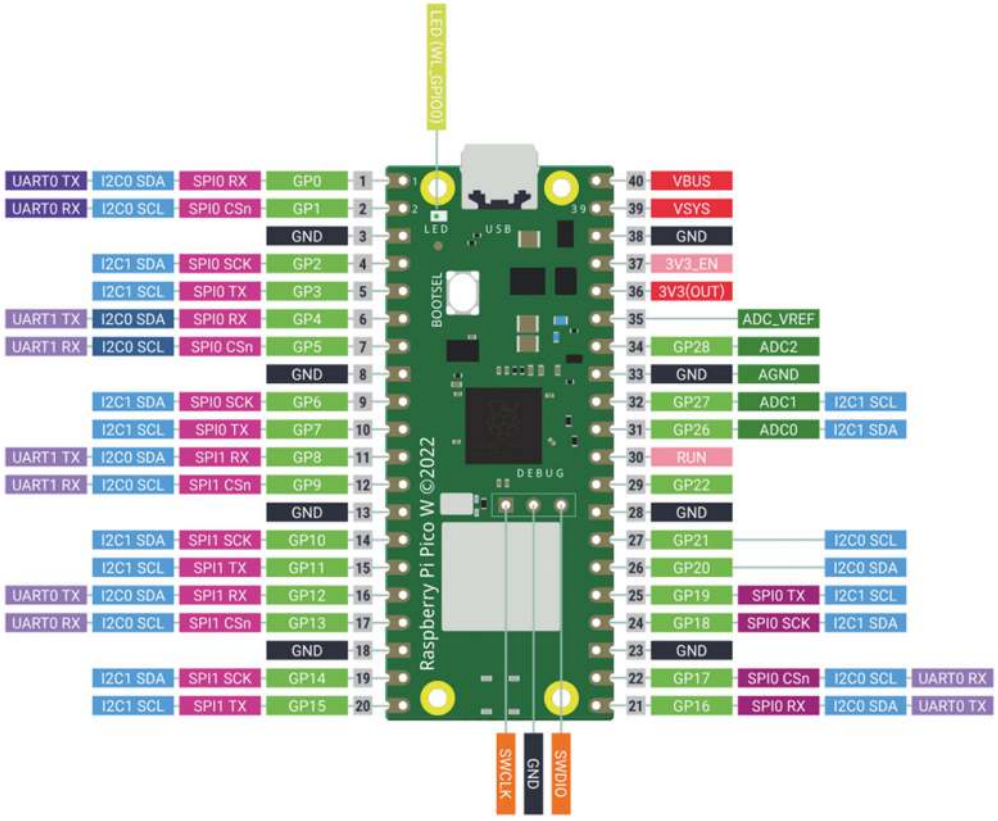


Figure 13.23 Pico pin configuration.

13.9.1 Project 7 – BeagleY-AI and Raspberry Pi Pico W communication – controlling a relay over Wi-Fi

Description: In this project, you have a BeagleY-AI and Raspberry Pi Pico W. A pushbutton is connected to Pico, and a +3.3 V relay is connected to the BeagleY-AI. Pressing the button on the Pico sends a command to BeagleY-AI over the Wi-Fi to activate the relay. The relay

remains active for 5 seconds. In this project, BeagleY-AI and Pico communicate using the UDP protocol, where BeagleY-AI is the server and Pico is the client.

Block diagram: Figure 13.24 shows the block diagram of the project.

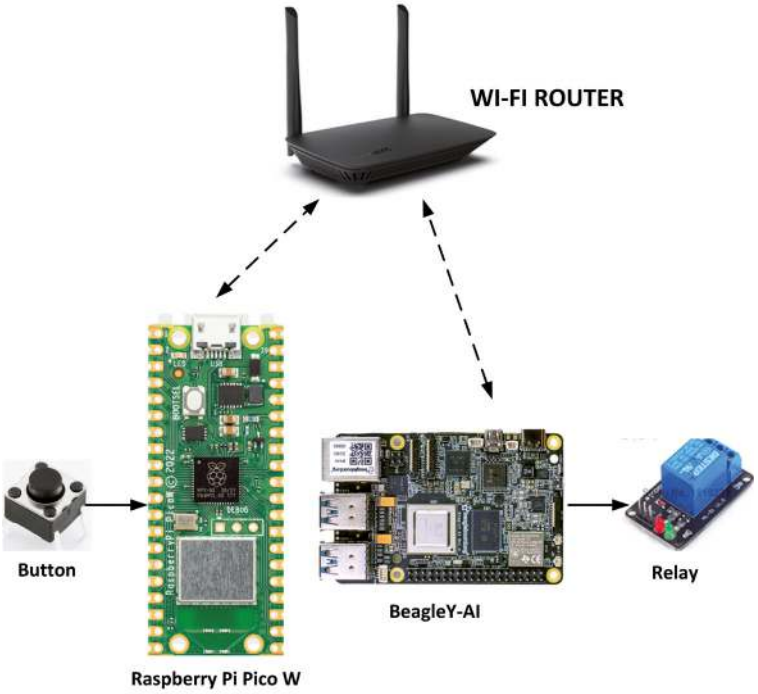


Figure 13.24 Block diagram of the project.

Circuit diagram: The circuit diagram of the project is shown in Figure 13.25, with the button and relay connected to the Pico and BeagleY-AI, respectively.

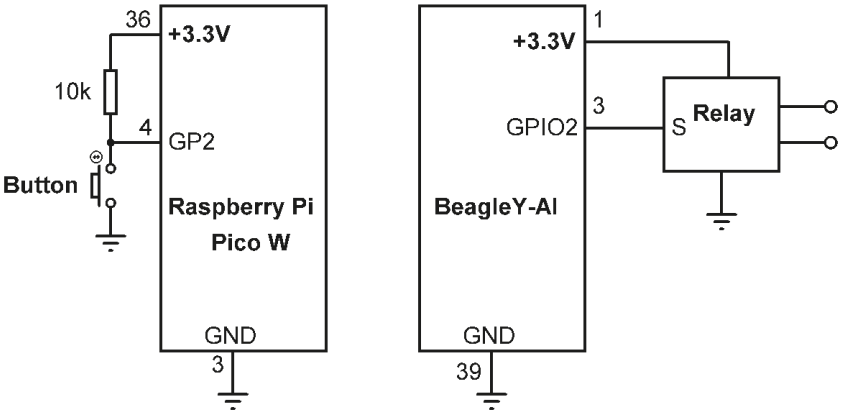


Figure 13.25 Circuit diagram of the project.

Pico program listing: Figure 13.26 shows the Pico program listing (**picoudp.py**). At the beginning of the program, the LED is assigned to port GP2 and is turned Off. The function **Connect()** is called to connect to the local Wi-Fi. Then, a socket is created with port number 2000 and IP address 192.168.1.21. When the button is pressed, the program sends 1 to the BeagleY-AI so that the LED can be turned On. This process is repeated after a 1-second delay.

```
#------------------------------------------------------------
#      RASPBERRY PI PICO W - BEAGLEY-AI COMMS
#      ======================================
#
# In this project a pushbutton is connected to GP2 of PICO W.
# Pressing the button sends a command to BeagleY-AI to
# activate a relay. UDP protocol is used in this project.
#
# Author: Dogan Ibrahim
# File  : picoudp.py
# Date  : October, 2024
#------------------------------------------------------------
from machine import Pin
import network
import socket
import utime
global wlan

BUTTON = Pin(2, Pin.IN)                              # Button at GP2

#
# This function attempts to connect to Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    while wlan.isconnected() == False:
       print("Waiting to be connected")
       wlan.active(True)
       wlan.connect("TP-Link_6138_EXT", "24844604")
       utime.sleep(5)

connect()
print("Connected")
UDP_PORT = 1500                                      # Port used
UDP_IP = "192.168.1.21"                              # Zero 2W IP
cmd = b"1"                                           # Cmd to turn ON
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

```
while True:
    while BUTTON.value() == 1:                # Not pressed
       pass
    while BUTTON.value() == 0:                # Not released
       pass
    sock.sendto(cmd, (UDP_IP, UDP_PORT))      # Send cmd
    print("Command sent")                     # Message
    utime.sleep(1)                            Wait 1 sec
```

Figure 13.26 Raspberry Pi Pico W program listing (picoudp.py).

BeagleY-AI program listing: Figure 13.27 shows the BeagleY-AI program listing (**Beagleudp.py**). At the beginning of the program, the libraries used are imported, and the relay is configured at port GPIO2 and is deactivated. Then, a socket is created, and the program binds to it with the BeagleY-AI IP address. The program then waits to receive a command from the Pico. The received command is stored in variable **data,** and if it is 1, then the relay is activated for 5 seconds. At the end of this time, the relay is deactivated and the program repeats waiting for a command.

```
#============================================================
#      RASPBERRY PI PICO W - BEAGLEY-AI COMMS
#      ======================================
#
# This is the UDP server program running on BeagleY-AI.
# The program receives a command from PICO W and activates a
# relay connected to GPIO2 for 5 seconds.
#
# Author: Dogan Ibrahim
# File  : Beagleudp.py
# Date  : October, 2024
#============================================================
import gpiod
import socket
from time import sleep

RELAY = gpiod.find_line('GPIO2')
RELAY.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.1.127", 1500))

try:
  while True:
    data, addr = sock.recvfrom(1024)   # GEt command
    if data == b'1':                   # Command is 1?
       RELAY.set_value(1)              # Activate Relay
```

```
        sleep(5)                        # 5 seconds delay
        RELAY.set_value(0)              # Deactivate Relay

except KeyboardInterrupt:               # Keyboard interrupt
  print("\nClosing connection to client")
  sock.close()
```

Figure 13.27 BeagleY-AI program listing (Beagleudp.py).

Testing the project
The steps to test the project are:

- Run the server on BeagleY-AI:

  beagle@beagle: ~$ **python Beagleudp.py**

- Run the Pico program in Thonny by clicking the green Run button. You should see the message **Connected** when Pico connects to the local router.

- Push the button on Pico. The message **Command sent** will be displayed on the Pico terminal. A packet will be sent to BeagleY-AI, which will turn ON the LED for 5 seconds

- Enter **Ctrl+C** to terminate the program.

13.10 Project 8 - Storing Ambient Temperature and Atmospheric Pressure Data on the Cloud

Description: In this project, the ambient temperature and atmospheric pressure are read and stored in the Cloud. A BME280-type sensor module (see Chapter 10.7) is used in this project.

Block diagram: The block diagram of the project is shown in Figure 13.28.

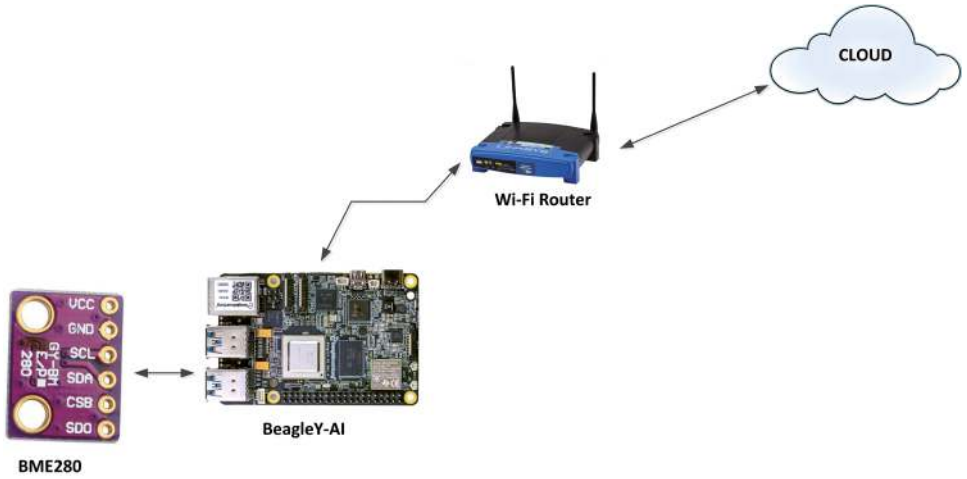


Figure 13.28 Block diagram of the project.

Circuit diagram: Figure 13.29 shows the circuit diagram. SCL and SDA pins of BME280 are connected to the SDA (pin 3) and SCL (pin 5) of the BeagleY-AI. The sensor is powered from +3.3 V.

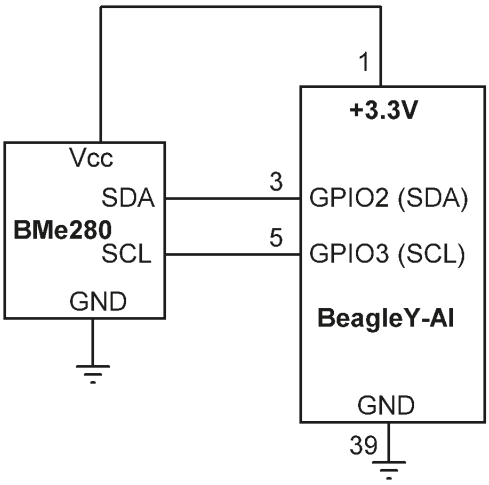


Figure 13.29 Circuit diagram of the project,

The Cloud

Several cloud services can be used to store data (for example, **SparkFun**, **ThingSpeak, Cloudino, Bluemix,** etc). In this project, **ThingSpeak** is used. This is a free cloud service where sensor data can be stored and retrieved using simple HTTP requests. Before using ThingSpeak, we have to create an account from their website and then log in to this account.

Go to the ThingSpeak website:

https://thingspeak.com/

*Click **Get Started For Free** and create an account if you don't already have one*. Then, you should create a New Channel by clicking on **New Channel**. Fill in the form as shown in Figure 13.30. Give the name **BeagleY-AI** to the application, provide a description, and create two **fields** called **Atmospheric Pressure** and **Temperature**. You can optionally fill in the other items as well if you wish.

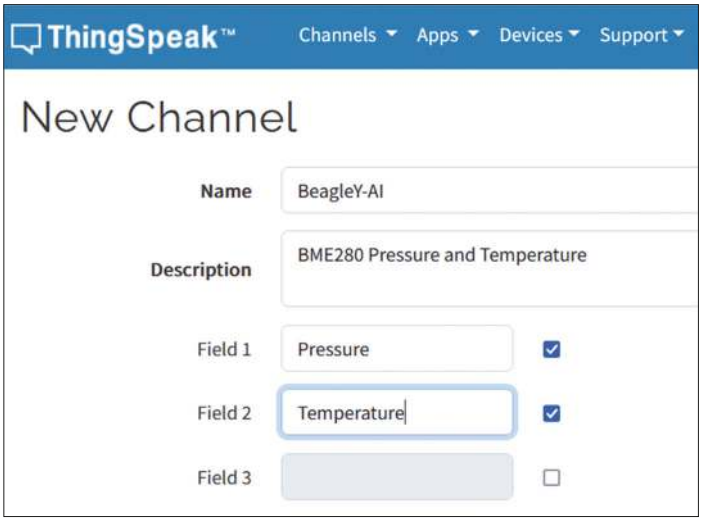


Figure 13.30 Create a New Channel (only part of the form shown).

Click **Save Channel** at the bottom of the form. Your channel is now ready to be used with your data. You will now see tabs with the following names. You can click on these tabs and see the contents to make corrections if necessary:

- **Private View**: This tab displays private information about your channel, where only you can see it.
- **Public View**: If your channel is public, use this tab to display selected fields and channel visualizations.
- **Channel Settings**: This tab shows all the channel options you set at creation. You can edit, clear, or delete the channel from this tab.
- **API Keys**: This tab displays your channel API keys. Use the keys to read from and write to your channel.
- **Data Import/Export**: This tab enables you to import and export channel data.

You should click the **API Keys** tab and save your unique **Write API** and **Read API** keys, and unique **Channel ID**, in a safe place, as you will need to use them in our program. The API Keys and the Channel ID in this project were as in Figure 13.31.

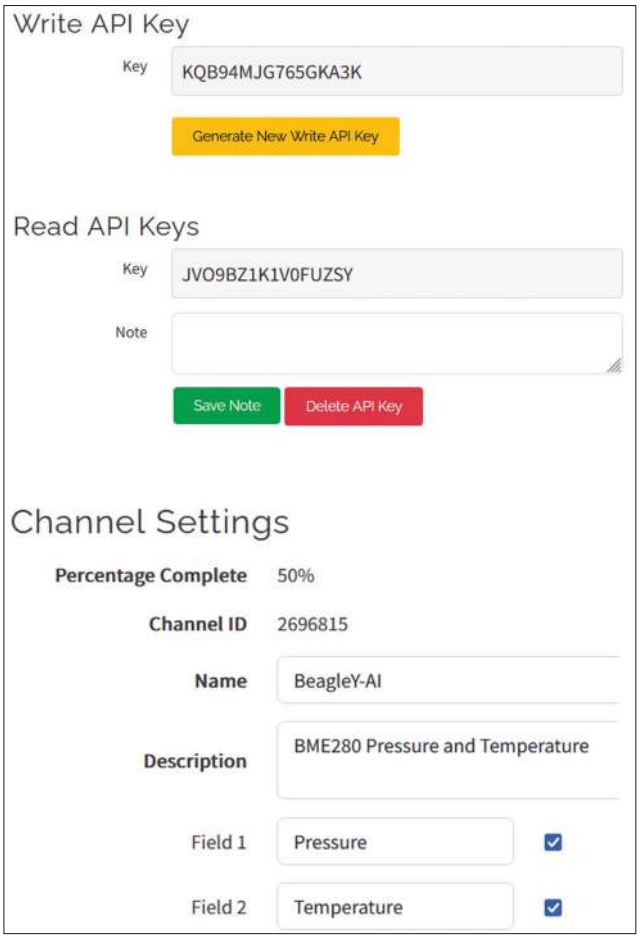


Figure 13.31 Author's Channel ID and API Keys.

Also, select the **Public View** and navigate to **Sharing**. You may select the option **Share channel view with everyone** so that everyone can access your data remotely.

Program listing: In this program, you will be using the BME280 library as described in Chapter 10.7. The steps to install the library are not repeated here.

After constructing the circuit, you should check to make sure that the BME280 is detected by the BeagleY-AI. Enter the following command:

```
beagle@beagle:~ $ sudo i2cdetect -r–y 1
```

You should see the hardware address of the BME280 chip displayed as 76 (see Figure 13.32).

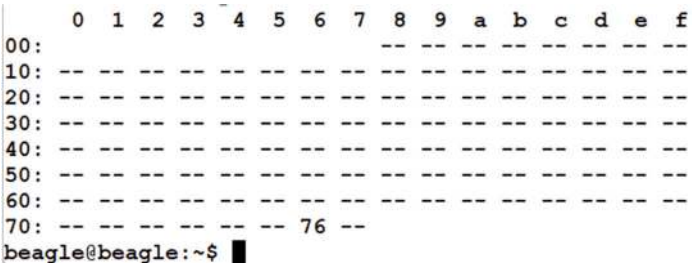


Figure 13.32 BME280 hardware address detected.

Figure 13.33 shows the program listing (**Cloud.py**). At the beginning of the program, the libraries used are imported. ThingSpeak **Write Key** and **Host Address** are defined. The main program loop starts with the **while** statement. Inside this loop, the IP address of the **ThingSpeak** website is extracted, and a connection is made to this site at port 80. Then, the atmospheric pressure and temperature readings are obtained from the BMP280 module and are included in the **path** statement. The **sock.send** statement sends an HTTP GET request to the **ThingSpeak** site and uploads the pressure and temperature values. This process is repeated every 30 seconds.

Figure 13.34 shows the pressure and temperature data plotted by **ThingSpeak**. The Chart Options can be clicked to change various parameters of the charts. For example, Figure 13.35 shows the pressure as a column display. In Figure 13.36, the pressure is shown as a step graph. A title and X-axis label are added in Figure 13.37 to the pressure graph. Figure 13.38 shows the current temperature displayed in a clock format (click **Add Widgets** for this type of display).

Because the Channel was saved as public, you can view the graph from a web browser (see Figure 13.39) by entering the Channel ID. In this project, the link to view the data graphs from a web browser is (this link is only available while the program is running):

https://api.thingspeak.com/channels/**2696815**

We can also export some or all of the fields in CSV format by clicking **Export recent data**, so that it can be analyzed by external statistical packages such as Excel.

```
#-------------------------------------------------------------------------------
#        ATMOSPHERIC PRESSURE AND TEMPERATURE ON THE CLOUD
#       ==================================================
#
# The ambient temperature and pressure sensor BMPE280 is connected to BeagleY-AI.
# The project reads the temperature and atmospheric pressure and sends
# to the Cloud where it can be accessed from anywhere. In addition, change
# of the temperature and the pressure can be plotted in the cloud.
```

```
#
#
# The program uses the ThingSpeak cloud service
#
# Author: Dogan Ibrahim
# File  : Cloud.py
# Date  : October, 2024
#-------------------------------------------------------------------------------
import socket
from time import sleep
from bme280pi import Sensor

sensor = Sensor(address = 0x76)

APIKEY = "KQB94MJG765GKA3K"                    # ThingSpeak API key
host = "api.thingspeak.com"                    # ThingSpeak host

#
# Send data to ThingSpeak. This function sends the temperature and
# humidity data to the cloud every 30 seconds.
#
while True:
   sock = socket.socket()
   addr = socket.getaddrinfo("api.thingspeak.com",80)[0][-1]
   sock.connect(addr)
   data = sensor.get_data()
   p = data['pressure']                        # Pressure in haP
   t = data['temperature']                     # Temperature in C
   path = "api_key="+APIKEY+"&field1="+str(p)+"&field2="+str(t)
   sock.send(bytes("GET /update?%s HTTP/1.0\r\nHost: %s\r\n\r\n"
%(path,host),"utf8"))
   sleep(5)
   sock.close()
   sleep(25)
```

Figure 13.33 Program listing.

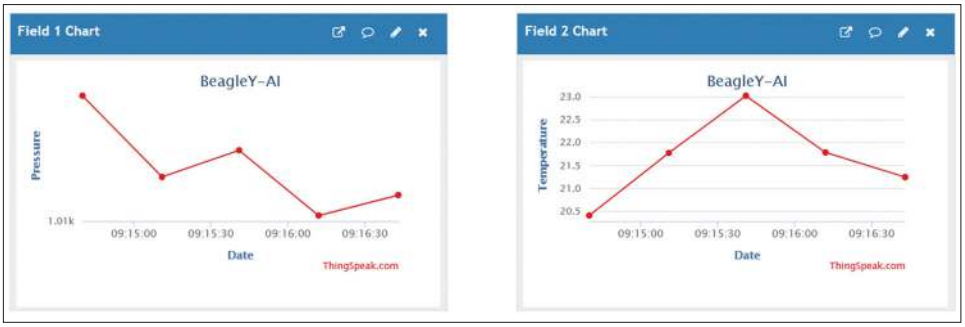


Figure 13.34 Plotting the pressure and temperature.

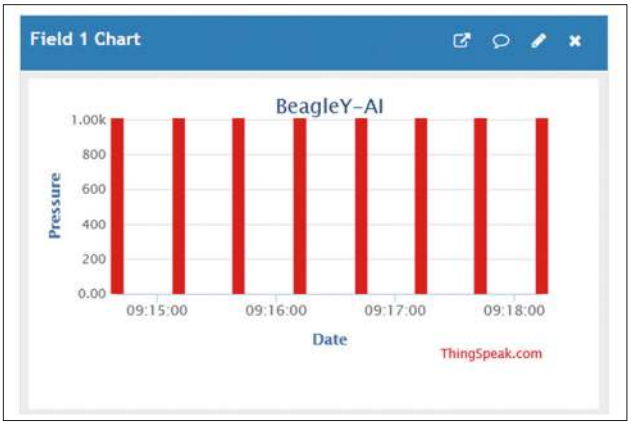


Figure 13.35 Displaying temperature as columns.

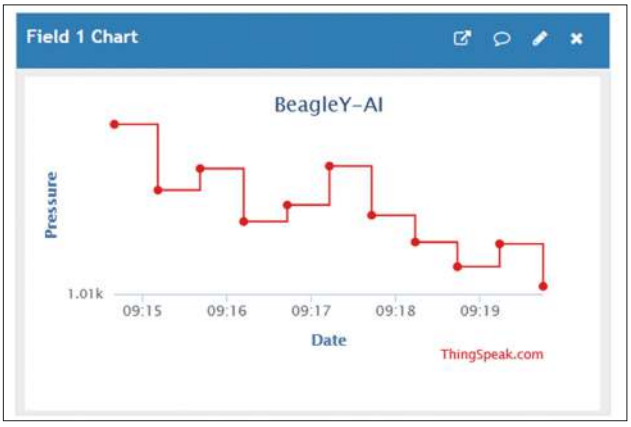


Figure 13.36 Displaying pressure as steps.

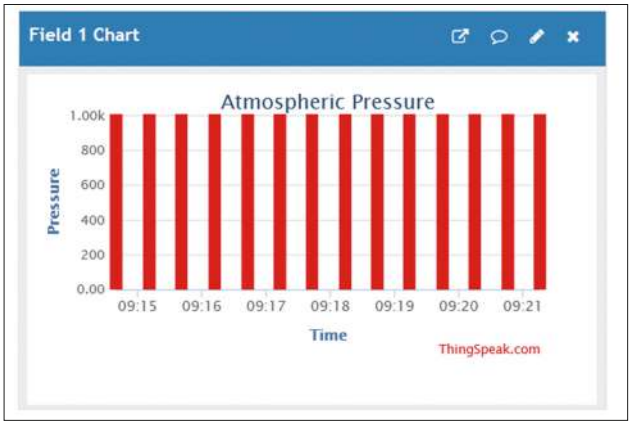


Figure 13.37 Adding title and x-axis label.

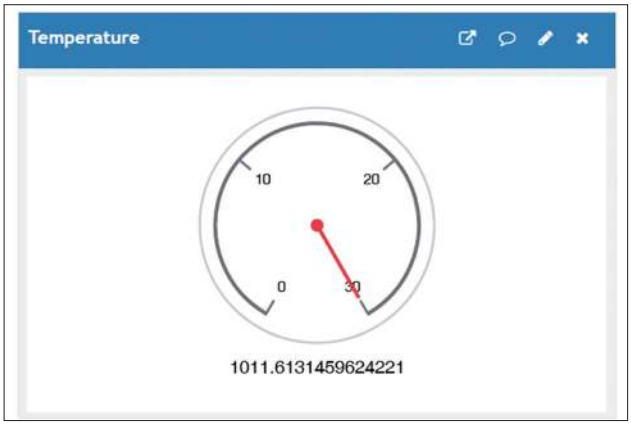


Figure 13.38 Displaying the current temperature in a clock format.

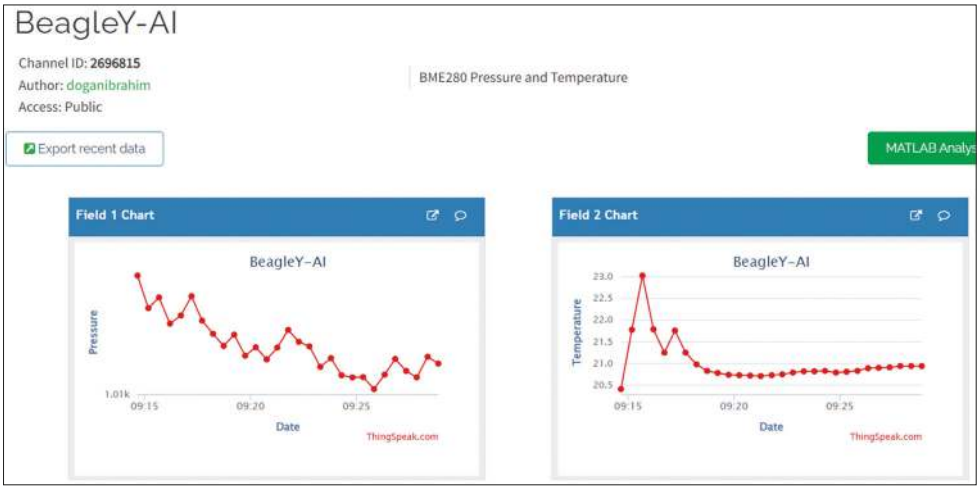


Figure 13.39 Displaying the graphs from a website.

13.11 Using Flask to Create a Web Server to Control BeagleY-AI GPIO Ports from the Internet

Flask is a simple micro-framework written in Python for Python. It is free of charge and can be used to create a web server on BeagleY-AI and other members of the family to control its GPIO ports over the internet. The nice advantage of Flask is that it does not require special tools or libraries, and has no database or any other third-party libraries.

Flask should already be available in Python on your BeagleY-AI, but if not, it can be installed with the following command:

```
beagle@beagle:~ $ sudo apt-get install python3-flask
```

It will be a good idea to create a new folder on your BeagleY-AI and store all of your flask-related documents there. Let's create a folder called **MyFlask** under our default directory **/home/beagle**:

```
beagle@beagle:~ $ mkdir MyFlask
```

Make MyFlask your default directory:

```
beagle@beagle:~ $ cd MyFlask
```

We are now ready to create our first web server application using Flask. To test Flask on your BeagleY-AI single board computer, use the **nano** text editor and create a file called **flasktest.py** with the following lines in it:

```
from flask import Flask                    # import module flask
app = Flask(__name__)                      # create a flask object called app

@app.route('/')
def index():                               # run index when called
  return 'Hello from Flask'                # msg to display when run

if __name__ == '__main__':
  app.run(debug=True, port=8080, host='0.0.0.0')    # listen on port 8080
```

Now, run the above program:

```
beagle@beagle:~ $ sudo python flasktest.py
```

You should see messages similar to the ones shown in Figure 13.40.

```
beagle@beagle:~/MyFlask$ sudo python flasktest.py
 * Serving Flask app 'flasktest'
 * Debug mode: on
WARNING: This is a development server. Do not use it in
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.1.127:8080
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 587-204-632
```

Figure 13.40 Flask messages.

Now, open a web browser (e.g., **Google Chrome**) from a computer connected to the same Wi-Fi router and enter the IP address of your BeagleY-AI followed by :8080 as the port number. In this example use 192.168.1.127:8080. You should see the **Hello from Flask** message appear on a webpage, as shown in Figure 13.41.

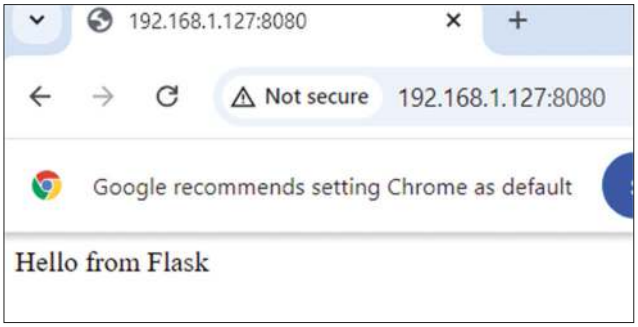


Figure 13.41 Message on the webpage.

We can now create an HTML page and pass variables from a Python program. Create a folder called **templates** under **MyFlask,** move to the **templates** directory, and create a file called **index.html** using the **nano** text editor with the following lines (notice that the variables inside the double curly brackets will have data passed to them from the Python program):

```
<head>
    <title>{{ title }}</title>
</head>
<body>
   <h1>Hello from Flask</h1>
  <h2>The time on the server is: {{ time }}</h2>
</body>
```

We will now modify our **flasktest.py** program under the **MyFlask** directory as follows:

```
from flask import Flask, render_template
import time

app = Flask(__name__)

@app.route('/')

def index():
   now = time.ctime()
   DataToPass = {
         'title' : "TESTING FLASK",
        'time': now
     }
   return render_template('index.html', **DataToPass)

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

The current date and time are obtained using the function call **time.ctime(),** and the result is stored in the variable now. Then, a dictionary called **DataToPass** is created, and the values of title and time are stored in this dictionary. These values will be passed to the items in double curly brackets in the webpage defined by index.html. When the function returns**,** the variables inside the dictionary are passed to the web browser through the dictionary.

Now, run the program **flasktest.py** with the command **sudo python flasktest.py**. Go to a web browser and enter the IP address of your BeagleY-AI followed by the :8080 port number (e.g., for the author's computer: 192.168.1.127:8080). You should see a display similar to the one shown in Figure 13.42.

Figure 13.42 Web page displaying the date and time.

Now that we have learned how to pass variables from a Python program to a web page, we can monitor the status of a GPIO pin or control a GPIO pin from a web page.

13.12 Project 9 – Web Server - Controlling an LED Connected to BeagleY-AI Using the Flask

Description: In this project, an LED is connected to port GPIO21 of the BeagleY-AI through a 470-ohm current limiting resistor. The LED is turned On or Off via remote web pages using Flask. The aim of this project is to show how Flask can be used to control an LED connected to BeagleY-AI.

HTML Template Program Listing: The HTML template **index.html** in folder **/home/beagle/MyFlask/templates** is simple and it consists of a **title** and two buttons: **On** and **Off**. The **title** is in double curly brackets and therefore it expects data to be passed to it from Python. Two buttons are defined, called **LED On** and **LED Off,** with green and red colors respectively, the **LED On** button having reference **/LED/on** and **LED Off** having reference **/LED/off**. Figure 13.43 shows the program listing.

```
<head>
  <title>{{ title }}</title>
</head>

<body>
  <h3>
    <a href="/LED/on"><button type="button"><font color="green">LED ON</button>
</a>
    <a href="/LED/off"><button type="button"><font color="red">LED OFF</button>
</a>
  </h3>
</body>
```

Figure 13.43 HTML template program listing.

BeagleY-AI Program Listing: Figure 13.44 shows the Python program listing on BeagleY-AI in the folder **/home/beagle/MyFlask** (program: **flasktest.py**). The program has the basic Flask-type template as shown earlier with some additional code. Port pin GPIO21 is configured as an output, and the LED is turned OFF at the beginning of the program. The **title** to be passed to **index.html** is named **LED CONTROL** and the function **index** is used to pass this string. Notice that another **app.route** is created with parameters **device** and **action**. In this example, the **device** is **LED**, and its actions are **on** and **off**. Function **action** checks the **device**, and if it is **LED**, then the **actuator** is set to **LED**. For every actuator, we must have an action. If the **action** is **on**, the LED is turned **On**. Otherwise, if the **action** is **off**, the LED is turned **Off**.

```
#=======================================================
#              CONTROLLING LED FROM WEB PAGE
#              ============================
#
# This program turns the LED ON or OFF from a web browser
# activated from any computer on the same Wi-Fi router as
```

```
# the BeagleY-AI. The LED is controlled by clicking buttons
# when the web page is started.
#
# Author: Dogan Ibrahim
# File  : flasktest.py
# Date  : November, 2024
#=========================================================
from flask import Flask,render_template
import gpiod
import time

app=Flask(__name__)

#
# Define GPIO21 as output and turn OFF LED at beginning
#
LED = gpiod.find_line('GPIO21')
LED.request(consumer='beagle',type=gpiod.LINE_REQ_DIR_OUT,default_val=0)

@app.route('/')
def index():
  DataToPass = {
    'title': "LED CONTROL"
  }
  return render_template('index.html', **DataToPass)

@app.route("/<device>/<action>")
def action(device, action):
  if device == "LED":
    actuator = LED

  if action == "on":
    actuator.set_value(1)
  if action == "off":
    actuator.set_value(0)
  return render_template('index.html')

if __name__ == '__main__':
   app.run(debug=False, port=8080,host='0.0.0.0')
```

Figure 13.44 Program: flasktest.py.

To run the program, you should follow these steps:

- Connect an LED to GPIO21 through a current-limiting resistor
- Run program **flasktest.py**

  beagle@beagle:~/MyFlask $ **python flasktest.py**

- Activate a web browser from a computer connected to the same Wi-Fi router and enter the IP address of your BeagleY-AI. As shown in Figure 13.45, you should see two buttons to control the LED. Clicking the buttons turns the LED On or Off accordingly.

Figure 13.45 Controlling the LED from a web page.

- Figure 13.46 shows a typical run of the program.

```
beagle@beagle:~/MyFlask$ python flasktest.py
 * Serving Flask app 'flasktest'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deplo
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.1.127:8080
Press CTRL+C to quit
192.168.1.131 - - [24/Oct/2024 14:58:23] "GET / HTTP/1.1" 200 -
192.168.1.131 - - [24/Oct/2024 14:58:25] "GET /LED/on HTTP/1.1" 200 -
192.168.1.131 - - [24/Oct/2024 14:58:27] "GET /LED/off HTTP/1.1" 200 -
192.168.1.131 - - [24/Oct/2024 14:58:29] "GET /LED/on HTTP/1.1" 200 -
192.168.1.131 - - [24/Oct/2024 15:48:18] "GET /LED/off HTTP/1.1" 200 -
```

Figure 13.46 Run of the program.

- Terminate your program by entering **Ctrl+C**.

Chapter 14 • Using Serial Communication

14.1 Overview

Serial communication is a simple means of sending data by wire to long distances quickly and reliably. Data is transmitted one bit at a time, rather than in parallel, as it reduces the cost of cabling, and also sending in serial format is less affected by electrical noise. Serial communication can be done either in software or by using a UART chip. Using a UART chip has the advantage that the communication can be very high-speed. Error detection is also easily handled in UART-based systems. The most commonly used serial communication method is based on the RS232 standard. In this standard, data is sent over a single line from a transmitting device to a receiving device in bit serial format at a pre-specified speed, also known as the Baud rate, or the number of bits sent/received each second. Typical Baud rates are 4800, 9600, 19200, 38400, etc.

RS232 serial communication is a form of asynchronous data transmission where data is sent character by character using dedicated hardware and without the support of a clock signal. Each character is preceded by a Start bit, seven or eight data bits, an optional parity bit, and one or more stop bits. The most commonly used format is eight data bits, no parity bit, and one stop bit. Therefore, a data frame consists of 10 bits. With a Baud rate of 9600, you can transmit and receive 960 characters every second. The least significant data bit is transmitted first, and the most significant bit is transmitted last. In most asynchronous serial communication systems, a parity bit is used to detect single-bit error in the transmission. The concept of parity is very simple. A parity bit can be Odd, Even, or None. In an Odd parity transmission, the number of 1s transmitted is Odd. If the number is even, a 1 is added as the last bit to make the total Odd. Even parity is the reverse, where the number of 1s transmitted is Even. If the number is not even, a 1 is added as the last bit to make it even.

In standard RS232 communication, logic high is defined to be at -12 V, and logic 0 is at +12 V. Figure 14.1 shows how character **A** (ASCII binary pattern 0010 0001) is transmitted/received over a serial line. The line is normally idle at -12 V. The start bit is first sent by the line going from high to low. Then eight data bits are sent starting from the least significant bit. Finally, the stop bit is sent by raising the line from low to high.

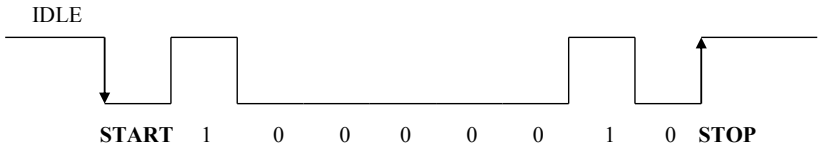


Figure 14.1 Sending character ***A*** *in serial format.*

In serial connection, a minimum of three wires are used for communication: transmit (TX), receive (RX), and ground (GND). Some high-speed serial communication systems use additional control signals for synchronization, such as CTS (Clear To Send), DTR (Data Terminal Ready), and so on. Some systems use software synchronization techniques where a special character (XOFF) is used to tell the sender to stop sending, and another character

(XON) is used to tell the sender to restart transmission. RS232 devices are connected to each other using two types of connectors: a 9-way connector, and a 25-way connector. Table 14.1 shows the TX, RX, and GND pins for each type of connector.

9-pin connector	
Pin	**Function**
2	Transmit (TX)
3	Receive (RX)
5	Ground (GND)

25-pin connector	
Pin	**Function**
2	Transmit (TX)
3	Receive (RX)
7	Ground (GND)

Table 14.1 Minimum pins required for RS232 serial communication.

As described above, RS232 voltage levels are at ±12 V. On the other hand, most microcontroller input-output ports operate at 0 to +5 V or 0 to +3.3 V voltage levels. It is therefore necessary to translate the voltage levels before a microcontroller can be connected to a RS232 compatible device. Thus, the output signal from the microcontroller has to be converted into ±12 V, and the input from an RS232 device must be converted into 0 to +5 V or 0 to +3.3 V before it can be connected to a microcontroller. This voltage translation is normally done using special RS232 voltage converter chips. One such popular chip is the MAX232. This is a popular dual converter chip, which requires a few external capacitors for its operation.

Nowadays, serial communication is done using standard TTL or CMOS logic levels instead of ±12 V, where logic 1 is normally greater than +3 V, and logic 0 is about 0 V. A serial line is idle when the voltage is higher than +3.3 V. The start bit is identified on the high-to-low transition of the line, i.e, the transition from higher voltage to 0 V. Two serial devices supporting the same logic levels can easily be connected to each other. Basically, the TX and the RX pins should be crossed and the ground pins should be connected directly to each other.

14.2 USB – TTL Serial Conversion Modules

USB–TTL serial modules are used to connect a PC to another device (for example, BeagleY-AI, Arduino Uno, Raspberry Pi, etc.) through the USB port. Figure 14.2 shows the USB-TTL converted module used by the author. At one end of this module, there is the USB connector, and at the other end, there is a header for making connections to the device which is to communicate with the PC over the serial port. A jumper on the module selects the serial port voltage levels as +5 V or +3.3 V. In our applications, you should set the jumper so that +3.3 V is selected. The header on the other side of the module has the following pin names:

+5 V (output)
+3.3 V (output)
TXD (serial output from the module)
RXD (serial input to the module)
GND (ground pin)

Figure 14.2 USB-TTL serial converted module.

In a typical application, you should make the connections between the PC and the external device as shown in Figure 14.3.

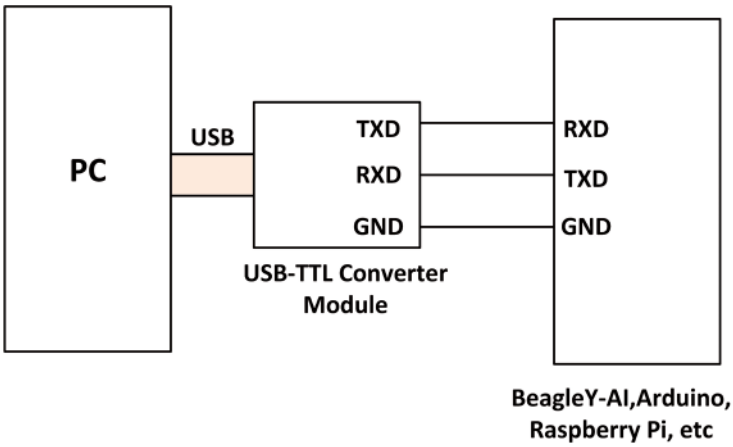


Figure 14.3 USB-TTL converter module connections.

When the USB-TTL converter module is connected to a PC, a serial port with the name **COMx** is created automatically, where x is the port number. The port number can be found on the PC **Device Manager** screen under Ports (COM & LPT). In the author's application in Figure 14.4, the serial port number was COM4. The port number is important when we want to communicate with the PC over the serial port.

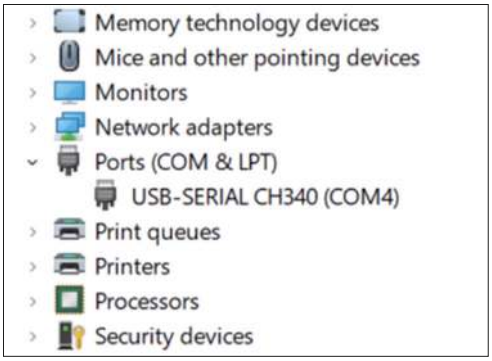


Figure 14.4 Serial port number.

14.3 BeagleY-AI and PC Communication Over Serial Port – Testing the Hardware and Software Configurations

In this example, we will be using serial port pins 8 and 10 of the BeagleY-AI board. Pin 8 is the TXD pin, while pin 10 is the RXD pin of serial port **/dev/ttyAMA0** of the BeagleY-AI.

Before using the serial port on the BeagleY-AI, we must load the ttyAMA0 overlay in the file: **/boot/firmware/extlinux/extlinux.conf.** Run the following command to see a list of the overlays:

```
beagle@beagle:~ $ ls /boot/firmware/overlays/
```

Select the overlay for ttyAMA0. i.e., **k3-am67a-beagley-ai-uart-ttyama0.dtbo**. The steps are:

- beagle@beagle:~ $ **sudo nano /boot/firmware/extlinux/extlinux.conf**

  Go to the end of the file and type:
  fdtoverlays /overlays/k3-am67a-beagley-ai-uart-ttyAMA0.dtbo

- Enter **Ctrl+X** followed by **Y** to save and exit. Figure 14.5 shows the end of the file.

```
label microSD (default)
    kernel /Image
    append console=ttyS2,115200n8 root=/dev/mmcblk1p3 ro rootfst
/dev/mmcblk1p2 rootwait net.ifnames=0 quiet
    fdtdir /
    fdt /ti/k3-am67a-beagley-ai.dtb

    fdtoverlays /overlays/k3-am67a-beagley-ai-uart-ttyama0.dtbo

    #initrd /initrd.img
```

Figure 14.5 Adding the serial port overlay file.

- Reboot your BeagleY-AI

  beagle@beagle:~ $ **sudo reboot**

- Enter the following command to make sure that the serial port overlay has been loaded (see Figure 14.6)

  beagle@beagle:~ $ **sudo beagle-version | grep UBOOT**

```
beagle@beagle:~$
beagle@beagle:~$ sudo beagle-version | grep UBOOT
UBOOT: Booted Device-Tree:[k3-am67a-beagley-ai.dts]
UBOOT: Loaded Overlay:[k3-am67a-beagley-ai-uart-ttyama0.kernel]
beagle@beagle:~$
```

Figure 14.6 Check the serial port overlay.

- Connect the circuit as shown in Figure 14.7 and plug in the USB-TTL converter module into the USB port of your PC. Make sure that a COM port is created on your PC as described earlier.

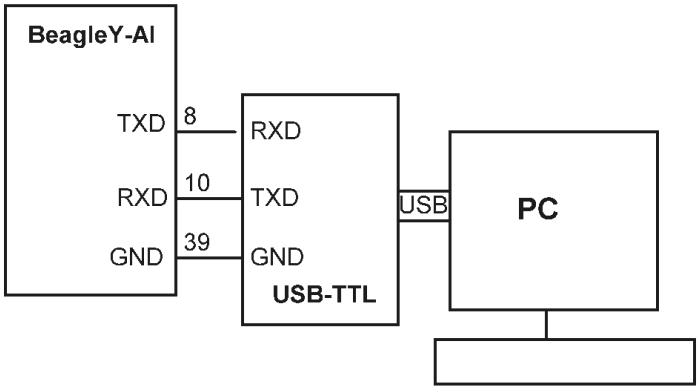


Figure 14.7 Circuit diagram.

- Start the **Terminal Emulator** on the GUI Desktop of your BeagleY-AI (or start remote SSH access to your BeagleY-AI from the PC)

- Start a terminal emulator program on your PC (e.g., Putty, Tera Term, etc.). In this example, the Putty program is used.

- Configure the Putty screen as shown in Figure 14.8, by selecting Serial, Speed 9600, and Serial line to COM4 (this may differ on your PC). It is recommended to configure the Putty screen settings so that, for example, you have a white background with black letters on it.

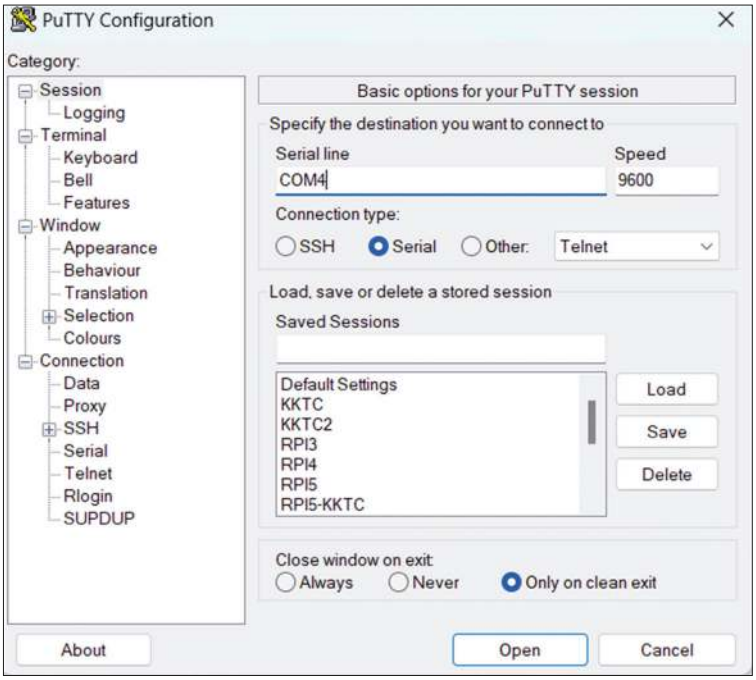


Figure 14.8 Putty configuration.

- Enter the following message on your BeagleY-AI Terminal Emulator screen:

beagle@beagle:~ $ **echo "Hello from BeagleY-AI" > /dev/ttyAMA0**

You should see the message displayed on your Putty screen (Figure 14.9). This proves that the serial hardware and software configurations between the BeagleY-AI and the PC are correct.

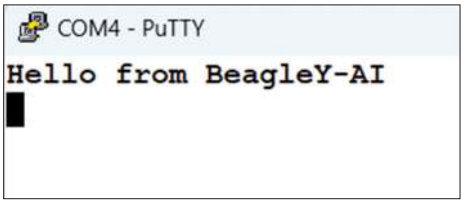


Figure 14.9 Putty screen displaying the message.

14.4 Project 1 – BeagleY-AI – PC Two-Way Communication Over Serial Port – Using Python

Description: In this project, a Python program has been created to establish two-way communication between the BeagleY-AI and the PC. The characters typed on the PC Putty screen are displayed on the BeagleY-AI Terminal Emulator screen where the program is run.

Block Diagram: Figure 14.10 shows the project block diagram.

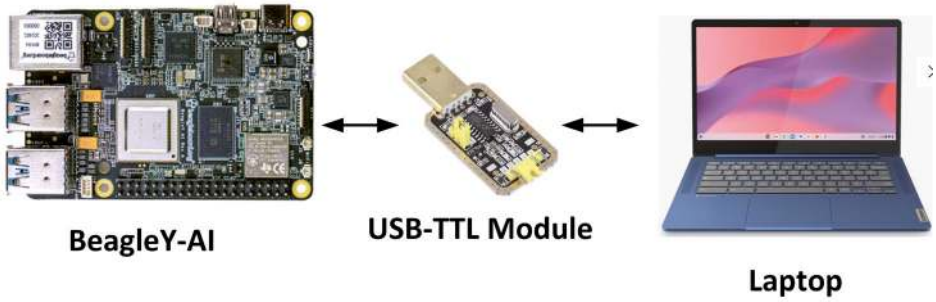


Figure 14.10 Block diagram.

The circuit diagram is the same as in Figure 14.7.

Program listing: For this project, you should use the version of Putty where CR+LF can be added to the end of the data to be sent. This version is available at the following site. Open the zip file and run Putty. There is no need to install it. Select **Terminal -> Keyboard** and set **The Enter key to CR+LF** as shown in Figure 14.11.:

https://www.grzegorz.net/pliki/putty-crlf.zip

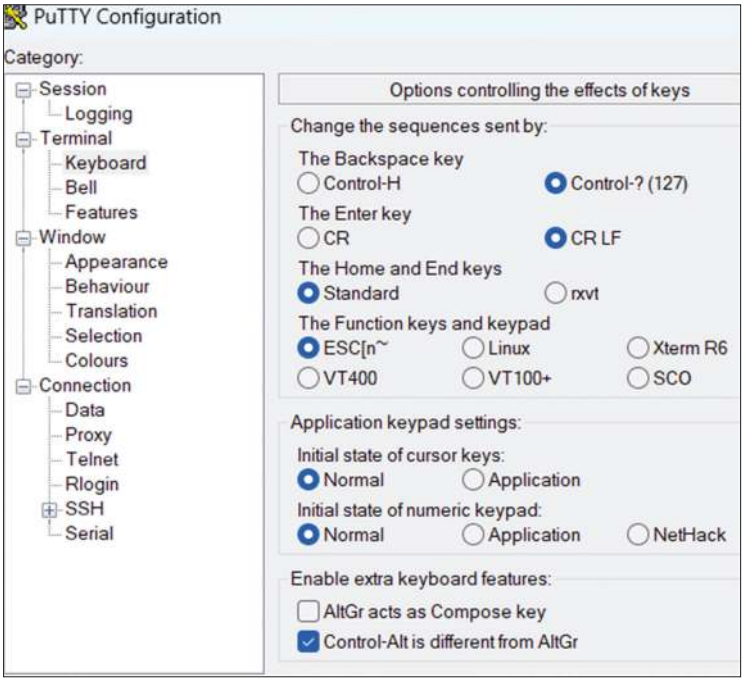


Figure 14.11 Set the Enter key to CR LF.

Figure 14.12 shows the program listing (Program: **serial1.py**). At the beginning of the program, libraries serial and time are imported. The serial port /dev/ttyAMA0 is opened with a 9600 baud rate. The function **write()** sends data to the serial port, while the function **readline()** reads a line terminated with carriage-return (CR) and line-feed (LF).

In this program, the serial port speed is set to 9600, which is the default speed of serial port /dev/ttyAMA0. The speed, number of data bits, etc., can be changed with the following statements (here, the speed is set to 115200):

```
ser = serial.Serial(
      port='/dev/ttyAMA0',
      baudrate = 115200,
      parity=serial.PARITY_NONE,
      stopbits=serial.STOPBITS_ONE,
      bytesize=serial.EIGHTBITS,
      timeout=1
)
```

```
#================================================================
#                 SERIAL COMMUNICATION WITH PC
#                 ============================
#
# This program establishes two-way serial communication with a PC
#
# Author: Dogan Ibrahim
# File  : serial1.py
# Date  @ November 2024
#================================================================
import serial
from time import sleep

md = "Y"

ser = serial.Serial ("/dev/ttyAMA0", 9600)            # open serial port

while md == "Y":
    ser.write("Send data to BeagleY-AI: ".encode())
    received_data = ser.readline()                    # read data
    print (received_data.decode())                    # print received data
    md = input("More data ? ")                        # more?
    md = md.upper()
ser.write("End of communication...".encode())         # end
```

Figure 14.12 Program listing.

Figure 14.13 shows an example of communication between the BeagleY-AI and the PC.

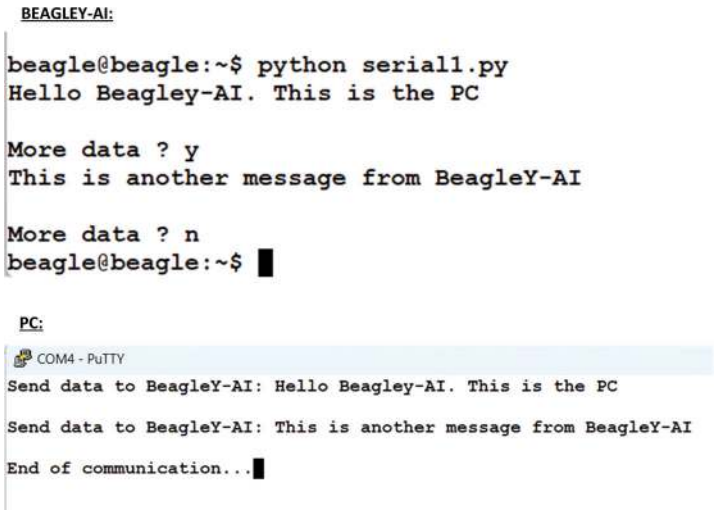


Figure 14.13 Example communication.

14.5 Reading Geographical Coordinates – Using a GPS

There are cases, especially when working mobile, where we may want to know the geographical coordinates (e.g., latitude and longitude) of our location. GPS receivers receive geographical data from the GPS satellites and provide accurate information about the position of the user on Earth. These satellites circle the Earth at an altitude of about 20,000 km and complete two full orbits every day. For a receiver to determine its position, the receiver must communicate with at least 3 satellites. Therefore, if the receiver does not have a 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 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 4 sentences with this heading. These sentences return the number of satellites in view, satellite number, elevation, azimuth, and SNR.

14.5.1 Project 2 – Displaying geographical coordinates on the monitor

Description: In this project, the GPS Click board (www.mikroe.com) is used. This is a small GPS receiver (see Figure 14.14) which is based on the LEA-6S type GPS. This board operates with +3.3 V and provides two types of outputs: I^2C or serial output. In this project, the default serial output is used which operates at 9600 baud rate. An external dynamic antenna can be attached to the board in order to improve its reception for indoor use or for use in places where there may not be a clear view of the sky.

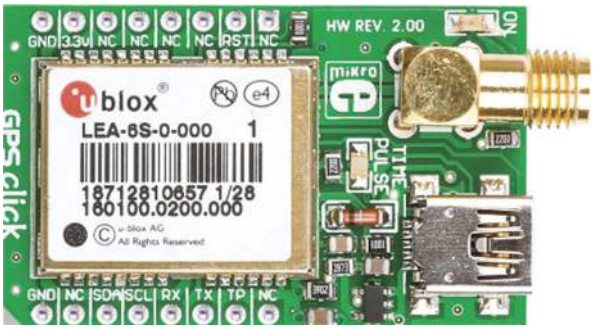


Figure 14.14 GPS Click board.

Figure 14.15 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,00003.13148,E,1,09,1.18,46.5,M,45.4,M,,*66
$GPSA,A,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 14.15 NMEA sentences output from the GPS Click board

GPS Click board is a 2x8 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 sentences, and this is the sentence used in this project to extract the station's 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 14.16 shows the block diagram of the project.

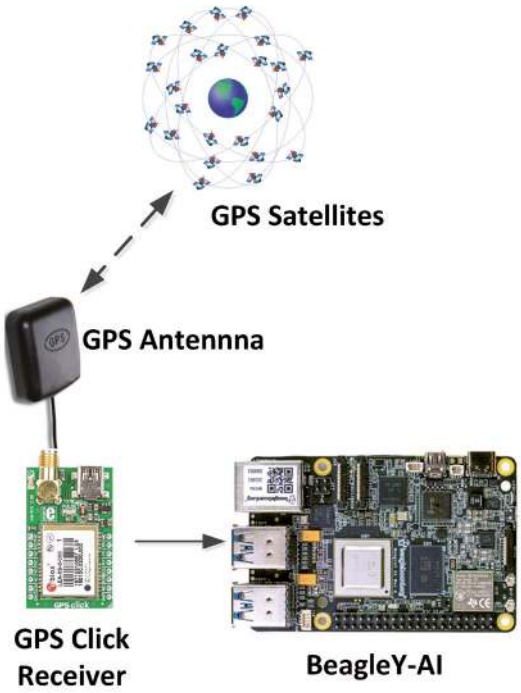


Figure 14.16 Block diagram of the project.

Circuit Diagram: The circuit diagram of the project is shown in Figure 14.17. The UART TX pin of the GPS click board (pin 14) is connected to the serial RXD input (pin 10) of the BeagleY-AI. The GPS click board is powered by the +3.3 V supply of the BeagleY-AI.

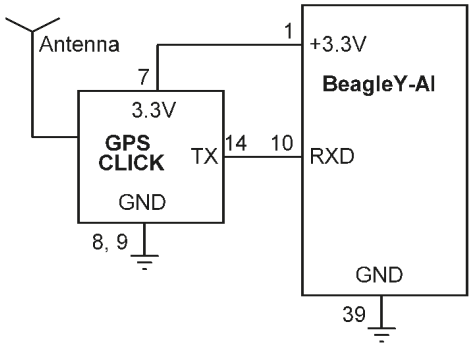


Figure 14.17 Circuit diagram of the project

You can display all the GPS NMEA sentences sent by the GPS click board by installing the PuTTY terminal emulation software on your BeagleY-AI. Enter the following command:

```
beagle@beagle:~ $ sudo apt-get install putty -y
```

The installed software will be available in **GUI Desktop** under **Applications** -> **Internet** -> **PuTTY SSH Client**. Start the application and fill in the details as shown in Figure 14.18.

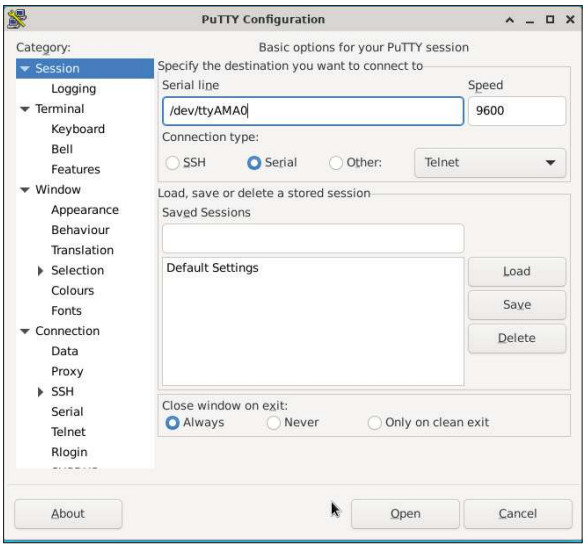


Figure 14.18 Putty details.

Click **Open** to start the terminal emulation software. Figure 14.19 shows an example display of NMEA sentences received from the GPS Click board.

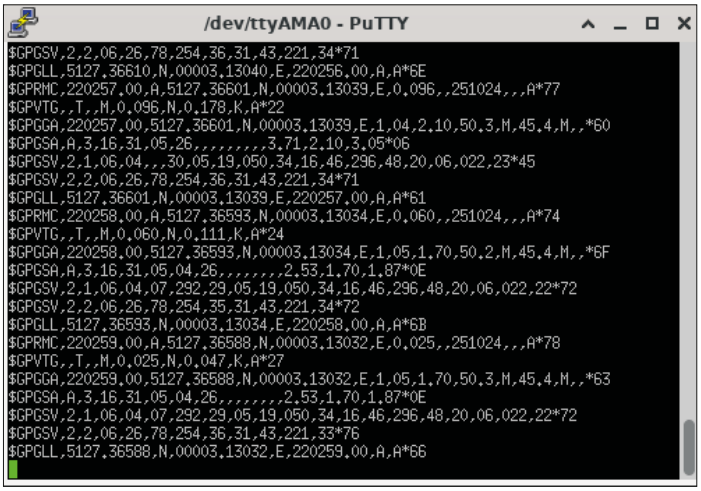


Figure 14.19 Example output of NMEA sentences.

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

Program Listing: Figure 14.20 shows the program listing (program: **gps.py**). At the beginning of the program, the following libraries are imported:

time
serial

Variable **port** is assigned to **/dev/ttyAM0** 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 the sentence is broken down into parts separated by commas using the built-in function **split(",")** and stored in **sdata**. If the 6th field is character **V,** it is assumed that the sentence is not valid (e.g., there is no satellite reception) and the text **NO DATA** is displayed. 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.mmmmmD** which corresponds to **dd** degrees **mm.mmmmm** minutes, and direction **D** which is **N** or **S**. Similarly, the longitude is received in the format: **dddmm.mmmmmD** where **D** is **E** or **W**. The main program separates the degrees and minutes and displays them on the screen. The latitude is displayed in the format: **dd mm.mmmmm D**, and the longitude is displayed as: **ddd mm.mmmmm D**.

```
#----------------------------------------------------------------
#              GEOGRAPHICAL COORDINATES
#              ------------------------
#
# In this project a GPS receiver module (GPS CLICK) is connected
# to the serial input of the BeagleY-AI.The program displays the
# latitude and longitude of the receiver location
#
# Author: Dogan Ibrahim
# File  : gps.py
# Date  : October 2024
#----------------------------------------------------------------
import time                             # Import time library
import serial                           # Import srial
port = "/dev/ttyAMA0"                   # Serial port
lat=latdir=lon=londir = "0"

#
# 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(",")                   # SPlit data
       if sdata[6] == "V":                      # Valid data?
           print("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 screen
#
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)

      print("Latitude : ", latitude)
      print("Longitude: ", longitude)
      print("")

      time.sleep(1)                              # Wait 1 second

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

Figure 14.20 Program: gps.py.

An example display on the screen is shown in Figure 14.21

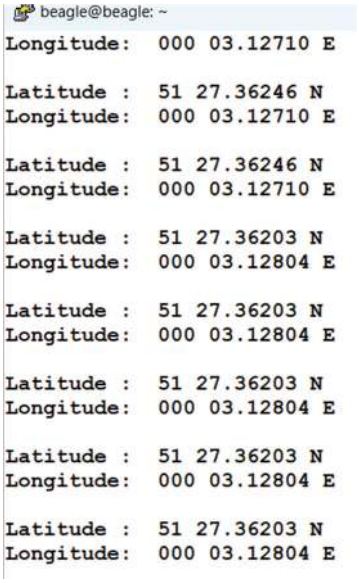


Figure 14.21 Example display of the geographical coordinates.

14.5.2 Project 3 – Displaying geographical coordinates on LCD

Description: This project is similar to the previous one, except that here the geographical coordinates are displayed on an I^2C LCD.

Block diagram: Figure 14.22 shows the project block diagram.

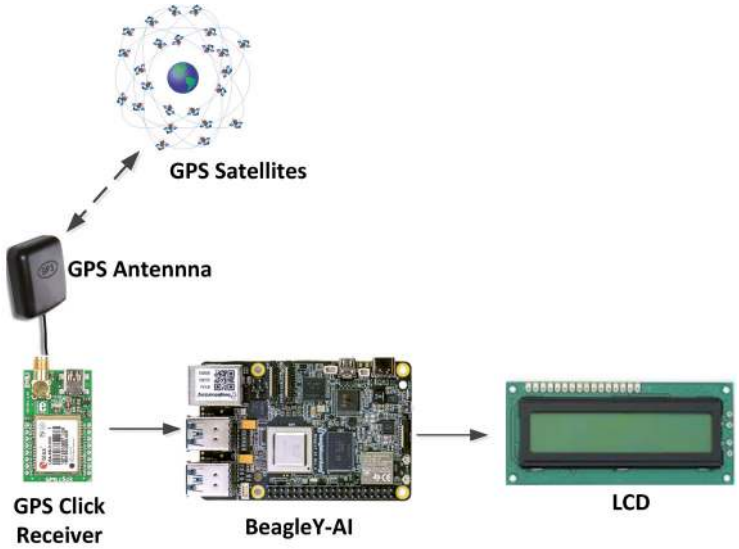


Figure 14.22 Block diagram.

Circuit diagram: The circuit diagram is similar to Figure 14.17, but here, additionally, the LCD is added to the circuit together with the voltage converter module.

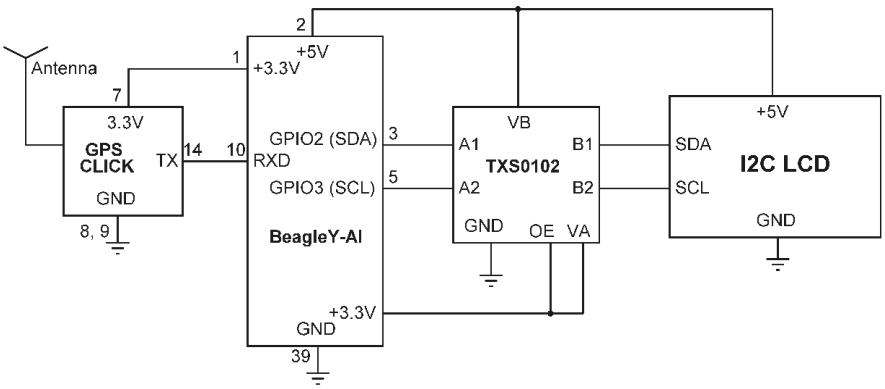


Figure 14.23 Circuit diagram.

Program listing: Figure 14.24 shows the program listing (Program: **gpslcd.py**). The program is very similar to the one given in Figure 14.20, except that here the LCD is initialized, and the latitude and longitude data are displayed on the LCD.

```
#-----------------------------------------------------------------
#              GEOGRAPHICAL COORDINATES ON LCD
#              -------------------------------
#
# In this project a GPS receiver module (GPS CLICK) is connected
# to the serial input RXD of BeagleY-AI SBC. Additionally, an
# I2C LCD is connected. The program displays the latitude and
# longitude of the receiver location on the LCD
#
# Author: Dogan Ibrahim
# File  : gpslcd.py
# Date  : November 2024
#-----------------------------------------------------------------
import smbus
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
import time                             # Import time library
import serial                           # Import srial

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

port = "/dev/ttyAMA0"                   # Serial port
lat=latdir=lon=londir = "0"

#
# 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(",")                   # SPlit data
       if sdata[6] == "V":                      # Valid data?
           mylcd.clear()                        # Clear LCD
           mylcd.move_to(0, 0)                  # At 0,0
           mylcd.putstr("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)

     mylcd.clear()
     mylcd.move_to(0, 0)
     mylcd.putstr(latitude)                     # Display latitude
     mylcd.move_to(0, 1)
     mylcd.putstr(longitude)                    # Display longitude
     time.sleep(1)                              # WAit 1 secons

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

Figure 14.24 Program listing.

An example display is shown in Figure 14.25.

Figure 14.25 Example display.

14.5.3 Project 4 – BeagleY-AI – Raspberry Pi 4 communication over a serial link

Description: In this project, a BeagleY-AI and a Raspberry Pi 4 are used. Raspberry Pi 4 sends a random number to BeagleY-AI. In return, BeagleY-AI increments this number by one and sends it back to the Raspberry Pi 4 where it is displayed on the monitor. The aim of this project is to show how the two computers can communicate over a serial link.

Block diagram: Figure 14.26 shows the project block diagram.

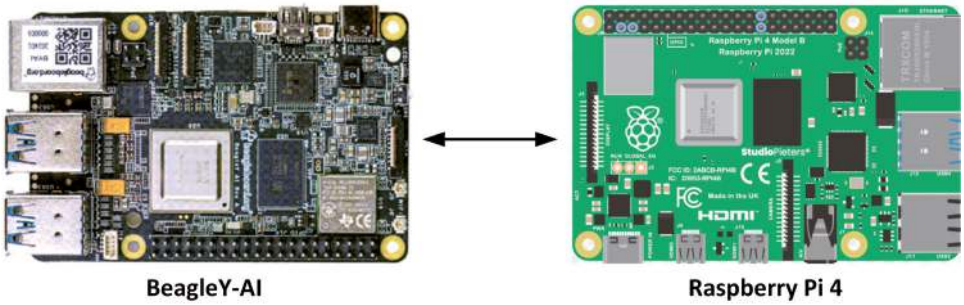


Figure 14.26 Block diagram.

Circuit diagram: The connections between the Raspberry Pi 4 and BeagleY-AI are very simple. As shown in Figure 14.27, the TXD and RXD pins of both computers are interchanged.

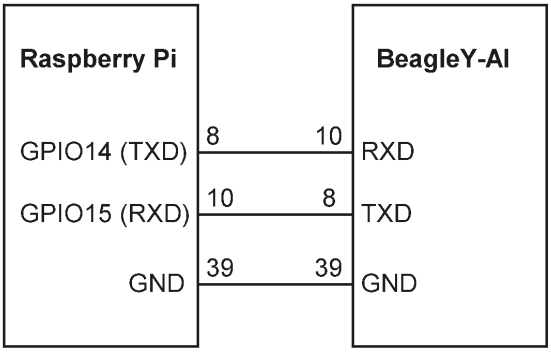


Figure 14.27 Circuit diagram.

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 Pi equipped with the Wireless/Bluetooth modules (e.g., Raspberry Pi 3, Zero W, 4, etc.), the PL011 UART is by default connected to the Bluetooth module, 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. To use the serial port for other purposes, this default configuration must 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 behavior, the serial console setting needs to be removed from the command line. This is easily done by using the **raspi-config** utility by selecting option **3** (Interfacing options), then **I6** (Serial), and selecting **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 finishing).

For Raspberry Pi 3 and 4 the serial port **(/dev/ttyS0**) is routed to GPIO14 (TXD) and GPIO15 (RXD) on the GPIO header. Models earlier than Raspberry Pi 3 use this port for Bluetooth. In this project, we are using the Raspberry Pi 4 whose serial port is: **/dev/ttyS0**. If you are using an earlier than model 3, use the serial port named: **/dev/ttyAMA0**.

Raspberry Pi 4 program: Figure 14.28 shows the Raspberry Pi program (Program: **RPsender.py**). At the beginning of the program, the **serial** library is imported, and the serial line **/dev/ttySerial0** is initialized to work at 9600 Baud. The program generates a random integer number between 1 and 1000 and sends it to BeagleY-AI over the serial link. It then waits to receive the incremented number from the BeagleY-AI and displays the number on the monitor. The user is asked if the program should continue. If the answer is **y** (or **Y**) then the process continues. Otherwise, the program terminates after sending "0" to BeagleY-AI which causes the BeagleY-AI program to terminate as well.

```
#================================================================
#           BEAGLEY-AI - RASPBERRY PI 4 COMMUNICATION
#           ========================================
#
# In this program Raspberry Pi 4 sends a random integer number to
# BeagleY-AI. This number is incremented by one and sent back to
# Raspberry Pi 4 where it is displayed
#
# Author: Dogan Ibrahim
# File  : RPsender.py
# Date  : November 2024
#================================================================
import time
import serial
import random

port = «/dev/ttyS0»
ser = serial.Serial(port, baudrate=9600)
```

```
chk = «Y»

while chk == «Y»:
  rnd = random.randint(1, 1000)                          # random no
  print(«Number sent to BeagleY-AI is: «, rnd)           # display msg
  ser.write(str(rnd).encode())                           # send the msg
  ser.write(«\r\n».encode())                             # CR+LF

  resp = ser.readline()                                  # read back
  respstr = resp.decode()                                # decode
  num = int(respstr)                                     # conv to int
  print(«Number received from BeagleY-AI is: «, num)     # display

  chk = input(«\nContinue?: «)                           # More?
  chk = chk.upper()

ser.write(«0».encode())
ser.write(«\r\n».encode())
time.sleep(2)
ser.close()
```

Figure 14.28 Raspberry Pi 4 program.

BeagleY-AI program: Figure 14.29 shows the BeagleY-AI program (Program: **Beaglercv,py**). At the beginning of the program, the **serial** library is imported, and the serial line **/dev/ttyAMA0** is initialized to work at 9600 Baud. The program then waits to receive a number from Raspberry Pi 4. The received number is incremented by one and sent back to Raspberry Pi 4. The program terminates if a "0" is received from Raspberry Pi 4.

```
#-------------------------------------------------------------
#
#           BEAGLEY-AI - RASPBERRY PI 4 COMMUNICATION
#           ========================================
#
# In this project BeagleY-AI receives a random number from
# Raspberry Pi4, increments the number by 1 and sends it back
# to Raspberry Pi 4
#
# Program: Beaglercv.py
# Date   : November 2024
# Author : Dogan Ibrahim
#-------------------------------------------------------------
import serial

port = "/dev/ttyAMA0"
```

```
ser = serial.Serial(port,baudrate=9600)

num = ""

while True:
  num = ser.readline()                # get the number
  num = num.decode()                  # decode
  numv = int(num)                     # conv to integer
  if numv == 0:                       # to exit
    break
  numv = numv + 1                     # increment
  ser.write(str(numv).encode())       # send to BeagleY-AI
  ser.write("\r\n".encode())

ser.close()
```

Figure 14.29 BeagleY-AI program.

Testing

- Start the BeagleY-AI program, which should block, waiting to receive a number from the Raspberry Pi 4.
- Start the Raspberry Pi 4 program. The generated integer random number and the incremented number sent by BeagleY-AI are displayed on the monitor.
- Enter **y** (or **Y**) to continue running the program, otherwise enter any other character to terminate both programs.
- Figure 14.30 shows an example run of the Raspberry Pi 4 program (it is assumed that the BeagleY-AI program was already started)

```
pi@pi:~ $ python RPsender.py
Number sent to BeagleY-AI is:  655
Number received from BeagleY-AI is:  656

Continue?: y
Number sent to BeagleY-AI is:  924
Number received from BeagleY-AI is:  925

Continue?: y
Number sent to BeagleY-AI is:  295
Number received from BeagleY-AI is:  296

Continue?: y
Number sent to BeagleY-AI is:  408
Number received from BeagleY-AI is:  409

Continue?: n
pi@pi:~ $
```

Figure 14.30 Example run of Raspberry Pi 4 program.

Chapter 15 • Real Time Clock (RTC)

15.1 Overview

Real Time Clocks (RTCs) provide precise and reliable timekeeping, which are beneficial for applications ranging from simple timekeeping to complex scheduling and secure operations.

Without an RTC, a computer must rely on perhaps getting the date and time information from the internet using, for example, the Network Time Protocol (NTP). However, there are many cases where an SBC such as BeagleY-AI may not have a constant or reliable network connection. In situations like these, an RTC allows the board to keep time even if the network connection is severed or the board loses power for an extended period. Fortunately, BeagleY-AI comes with a built-in DS1340 type on-board RTC for timekeeping purposes.

The RTC is useful for the following applications:

- Maintaining accurate time and date
- Timestamping applications and events
- Scheduling tasks accurately at specified times
- Network synchronization with other devices

15.2 The Hardware

A small 1.00 mm pitch, 2-pin JST SH connector is provided on the BeagleY-AI board to connect a coin cell battery (Figure 15.1) to enable the RTC to keep time even when power is lost to the board.

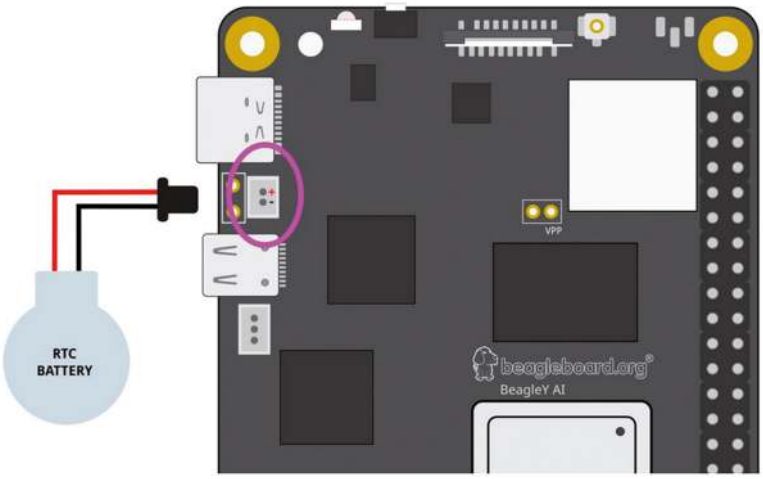


Figure 15.1 BeagleY-AI on-board RTC battery connector (https://docs.beagle.cc/boards/beagley/ai/demos/beagley-ai-using-rtc.html)

15.3 Setting the RTC Time

The RTC time should be set to the current time before it is read. The RTC time can be set accurately using the following command. Here, the date is set to 15 October 2024, and the time to 10:46:00:

```
beagle@beagle:~ $ sudo hwclock --set --date "2024-10-15 22:22:22"
```

There are two different times with different formats:

- System time, which can be displayed using the **date** command
- RTC time, which can be displayed using the **sudo hwclock** command

An example is shown below:

```
beagle@beagle:~ $ date
Tue Oct 15 09:46:26 UTC 2024
beagle@beagle:~ $ sudo hwclock
2024-10-15 10:47:56.279-31+00:00
```

We can set the two times to be the same format using the following date command:

```
beagle@beagle:~ $ date +%Y-%m-%d' '%H:%M:%S.%N%:z
2024-10-15 10:09:49.749529714+00:00
beagle@beagle:~ $ sudo hwclock
2024-10-15 10:09:57.807556+00:00
```

Notice that the two-time readings may differ slightly. Environmental conditions can cause electronics to become slightly out of sync and can cause drift. RTCs are accurate devices that implement various methods to keep the drifts as small as possible for example by compensating for temperature changes.

To sync the system clock, enter the command:

```
beagle@beagle:~ $ sudo hwclock –systohc
```

Now, let's display both the system and the RTC times one after the other one quickly. You should see that there could be about a one-second difference between the two. This is because it may take about a second to query and display the RTC clock.

Chapter 16 • Artificial Intelligence (AI) with the BeagleY-AI

16.1 Overview

Although the BeagleY-AI can be used as a general-purpose single-board computer, it has been developed for artificial intelligence applications. The processor is powered by Texas Instruments AM67A quad-core Cortex-A53 running at 1.4GHz along with an ARM Cortex-R5F processor running at 800MHz for handling general tasks. The processor is equipped with 2 x C7x DSP modules and a Matrix Multiply Accelerator (MMA), enhancing AI performance and making the board suitable for AI-based applications. Each C7x DSP delivers 2 TOPS, thus offering a total of 4 TOPS processing power. Additionally, a graphics accelerator is provided, offering 50GFlops for video and multitasking operations, which are required in AI applications.

Since the BeagleY-AI is a new product, there are currently few AI-based projects utilizing this board. In this chapter, we will develop an AI project that uses TensorFlow Lite for object detection. Links to other AI-based projects using the BeagleY-AI can be found in different sections of this chapter.

16.2 BeagleY-AI Detailed Hardware Specifications

It is interesting to have a look at the detailed hardware specifications of the BeagleY-AI before investigating its application in an AI project. For this, you should install the inxi applications:

```
beagle@beagle:~ $ sudo apt install inxi
```

Run the program by simply entering inxi -F. Figure 16.1 shows the output where the CPU, memory, disk, audio, network, etc. details are listed.

```
beagle@beagle:~$ inxi -F
System:
  Host: beagle Kernel: 6.1.83-ti-arm64-r63 arch: aarch64 bits: 64
    Console: pty pts/0 Distro: Debian GNU/Linux 12 (bookworm)
Machine:
  Type: ARM System: BeagleBoard.org BeagleY-AI details: N/A
CPU:
  Info: quad core model: N/A variant: cortex-a53 bits: 64 type: MCP cache:
    L2: 512 KiB
  Speed: N/A min/max: N/A cores: No per core speed data found.
Graphics:
  Device-1: hdmi-connector driver: display_connector v: N/A
  Device-2: am62p-pvr driver: N/A
  Device-3: am62p-pvr driver: N/A
  Display: x11 server: X.org v: 1.21.1.7 driver: X: loaded: N/A
    unloaded: fbdev,modesetting gpu: display_connector note: X driver n/a
    tty: 80x24 resolution: 1680x1050
  API: OpenGL Message: GL data unavailable in console. Try -G --display
Audio:
  Device-1: hdmi-connector driver: display_connector
  Device-2: simple-audio-card driver: asoc_simple_card
  API: ALSA v: k6.1.83-ti-arm64-r63 status: kernel-api
  Server-1: PipeWire v: 1.0.5 status: active
Network:
  Message: No ARM data found for this feature.
  IF-ID-1: docker0 state: down mac: 02:42:b8:ba:c9:0d
  IF-ID-2: dummy0 state: down mac: b2:1d:1f:ea:8f:94
  IF-ID-3: eth0 state: down mac: c0:d6:0a:f9:cd:86
  IF-ID-4: SoftAp0 state: down mac: 12:ca:bf:d9:e9:b3
  IF-ID-5: usb0 state: down mac: 1c:ba:8c:a2:ed:6b
  IF-ID-6: usb1 state: down mac: 1c:ba:8c:a2:ed:6d
  IF-ID-7: wlan0 state: up mac: 10:ca:bf:d9:e9:b2
Drives:
  Local Storage: total: 29.54 GiB used: 7.44 GiB (25.2%)
  ID-1: /dev/mmcblk1 vendor: Lexar model: LX32G size: 29.54 GiB
Partition:
  ID-1: / size: 24.79 GiB used: 7.38 GiB (29.8%) fs: ext4 dev: /dev/mmcblk1p3
Swap:
  ID-1: swap-1 type: partition size: 4 GiB used: 0 KiB (0.0%)
    dev: /dev/mmcblk1p2
Sensors:
  System Temperatures: cpu: N/A mobo: N/A
  Fan Speeds (RPM): cpu: 0
Info:
  Processes: 190 Uptime: 7m Memory: 3.7 GiB used: 852.3 MiB (22.5%)
  Init: systemd target: graphical (5) Shell: Bash inxi: 3.3.26
beagle@beagle:~$
```

Figure 16.1 Detailed hardware specifications.

16.3 Project 1 - BeagleY-AI TensorFlow Lite Object Detection

This project describes how to set up and run an object detection model using TensorFlow Lite on the BeagleY-AI platform. Full hardware and software details, as well as step-by-step installation instructions for the required software, are provided on the following websites:

https://docs.beagleboard.org/boards/beagley/ai/demos/beagley-ai-object-detection-tutorial.html#

and

https://www.cnx-software.com/2024/10/13/beagley-ai-review-sbc-debian-12-tensorflow-lite-ai-demos/

The following are required for the project:

- Beagle-Y AI board
- USB Webcam. The author used a Full HD 1080P, 12.0 MEGA Pixel F/#2.0, F:4.8mm AUSDOM webcam, but it should work with other webcams as well.
- GUI Desktop connection to your Beagle-Y AI
- Internet connection to your Beagle-Y AI for installing the required software

The steps in developing this project are given below (characters entered by the user are in bold for clarity):

- Open a Terminal session on your Desktop GUI

- Install a lightweight version of Conda using Miniforge/Mambaforge 24.3.0.0:

  wget https://github.com/conda-forge/miniforge/releases/download/24.3.0-0/Mambaforge-24.3.0-0-Linux-aarch64.sh
  bash Mambaforge-24.3.0-0-Linux-aarch64.sh

- Accept the Conda license

- Check that Conda has been installed successfully by entering the command:

  conda –version

  You should see the version number displayed as: conda 24.3.0

- Create a virtual environment with Python 3.9:

  conda create –name myenv python=3.9

- Activate the virtual environment:

  conda activate myenv

- Install the required Python packages:

  pip install https://github.com/google-coral/pycoral/releases/download/v2.0.0/tflite_runtime-2.5.0.post1-cp39-cp39-linux_aarch64.whl
  pip install numpy==1.26.4
  pip install opencv-python
  pip install tflite-runtime

- Create a directory for the object recognition models:

  mkdir object-recognition
  cd object-recognition

- Download a pre-trained model and unzip to directory **TFLite_model**:

  wget https://storage.googleapis.com/download.tensorflow.org/models/tflite/coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip

  unzip coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip -d TFLite_model

- Connect the USB webcam to one of the USB ports of your BeagleY-AI and enter the following command to find your video driver:

  ls –l /dev | grep video

  In the author's application, the result of this command is shown in Figure 16.2 and the video driver was number 0.

```
(base) beagle@beagle:~/object-recognition$ ls -l /dev | grep video
crw-rw----  1 root video    29,   0 Dec 21 16:56 fb0
crw-rw----+ 1 root video   235,   0 Dec 21 16:56 media0
crw-rw----+ 1 root video    81,   0 Dec 21 16:56 video0
crw-rw----+ 1 root video    81,   1 Dec 21 16:56 video1
crw-rw----+ 1 root video    81,   2 Dec 20 21:58 video2
crw-rw----+ 1 root video    81,   3 Dec 20 21:58 video3
crw-rw----+ 1 root video    81,   4 Dec 20 21:58 video4
```

Figure 16.2 Webcam video drivers

- Create a file with the name **object-detection.py** using the nano text editor and copy the program shown in Figure 16.3 to this file. ***Note: This program has been copied from the following site:***

  https://docs.beagleboard.org/boards/beagley/ai/demos/beagley-ai-object-detection-tutorial.html#

  nano object-detection.py

  Exit the nano text editor by entering **Ctrl+X** followed by **Y** to save and exit nano.

```
import os
import argparse
import cv2
import numpy as np
import time
from threading import Thread
```

```
import importlib.util
from typing import List
import sys
from tflite_runtime.interpreter import Interpreter, load_delegate

video_driver_id = 3

class VideoStream:
    «»»Handles video streaming from the webcam.»»»
    def __init__(self, resolution=(640, 480), framerate=30):
        self.stream = cv2.VideoCapture(video_driver_id)
        self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        self.stream.set(3, resolution[0])
        self.stream.set(4, resolution[1])
        self.grabbed, self.frame = self.stream.read()
        self.stopped = False

    def start(self):
        «»»Starts the thread that reads frames from the video stream.»»»
        Thread(target=self.update, args=()).start()
        return self

    def update(self):
        «»»Continuously updates the frame from the video stream.»»»
        while True:
            if self.stopped:
                self.stream.release()
                return
            self.grabbed, self.frame = self.stream.read()

    def read(self):
        «»»Returns the most recent frame.»»»
        return self.frame

    def stop(self):
        «»»Stops the video stream and closes resources.»»»
        self.stopped = True

def load_labels(labelmap_path: str) -> List[str]:
    «»»Loads labels from a label map file.»»»
    try:
        with open(labelmap_path, 'r') as f:
            labels = [line.strip() for line in f.readlines()]
        if labels[0] == '???':
            labels.pop(0)
        return labels
```

```
    except IOError as e:
        print(f»Error reading label map file: {e}»)
        sys.exit()

def main():
    # Argument parsing
    parser = argparse.ArgumentParser()
    parser.add_argument('--modeldir', required=True, help='Folder the .tflite
file is located in')
    parser.add_argument('--graph', default='detect.tflite', help='Name of the
.tflite file')
    parser.add_argument('--labels', default='labelmap.txt', help='Name of the
labelmap file')
    parser.add_argument('--threshold', default='0.5', help='Minimum confidence
threshold')
    parser.add_argument('--resolution', default='1280x720', help='Desired webcam
resolution')
    args = parser.parse_args()

    # Configuration
    model_path = os.path.join(os.getcwd(), args.modeldir, args.graph)
    labelmap_path = os.path.join(os.getcwd(), args.modeldir, args.labels)
    min_conf_threshold = float(args.threshold)
    resW, resH = map(int, args.resolution.split('x'))

    # Load labels and interpreter
    labels = load_labels(labelmap_path)
    interpreter = Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    # Get model details
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height, width = input_details[0]['shape'][1:3]
    floating_model = (input_details[0]['dtype'] == np.float32)

    outname = output_details[0]['name']
    boxes_idx, classes_idx, scores_idx = (1, 3, 0) if 'StatefulPartitionedCall'
in outname else (0, 1, 2)

    # Initialize video stream
    videostream = VideoStream(resolution=(resW, resH), framerate=30).start()
    time.sleep(1)

    frame_rate_calc = 1
    freq = cv2.getTickFrequency()
```

```
    while True:
        t1 = cv2.getTickCount()
        frame = videostream.read()
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame_resized = cv2.resize(frame_rgb, (width, height))
        input_data = np.expand_dims(frame_resized, axis=0)

        if floating_model:
            input_data = (np.float32(input_data) - 127.5) / 127.5

        interpreter.set_tensor(input_details[0]['index'], input_data)
        interpreter.invoke()

        boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0]
        classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0]
        scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0]

        for i in range(len(scores)):
            if min_conf_threshold < scores[i] <= 1.0:
                ymin, xmin, ymax, xmax = [int(coord) for coord in (boxes[i] *
[resH, resW, resH, resW])]
                cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (10, 255, 0), 2)
                object_name = labels[int(classes[i])]
                label = f'{object_name}: {int(scores[i] * 100)}%'
                labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_
SIMPLEX, 0.7, 2)
                label_ymin = max(ymin, labelSize[1] + 10)
                cv2.rectangle(frame, (xmin, label_ymin - labelSize[1] - 10),
(xmin + labelSize[0], label_ymin + baseLine - 10), (255, 255, 255), cv2.FILLED)
                cv2.putText(frame, label, (xmin, label_ymin - 7), cv2.FONT_
HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2)

        cv2.putText(frame, f'FPS: {frame_rate_calc:.2f}', (30, 50), cv2.FONT_
HERSHEY_SIMPLEX, 1, (255, 255, 0), 2, cv2.LINE_AA)
        cv2.imshow('Object detector', frame)

        t2 = cv2.getTickCount()
        time1 = (t2 - t1) / freq
        frame_rate_calc = 1 / time1

        if cv2.waitKey(1) == ord('q'):
            break

    cv2.destroyAllWindows()
    videostream.stop()
```

```
if __name__ == «__main__»:
    main()
```

Figure 16.3 Program: object-detection.py

- Make sure to change your video driver ID in the file **object-detection.py** depending on your video driver. Here, the video driver ID is set to 0.

- Run the program by entering the following command in GUI Desktop mode:

  (base) beagle@beagle:~/object-recognition$ **python3 object_detection.py --modeldir=TFLite_model**

- Move the webcam to focus on an object and you should see the object identification displayed at the top left corner of the screen. Figure 16.4 shows an example of identifying a keyboard. Another example of identifying a laptop is shown in Figure 16.5.

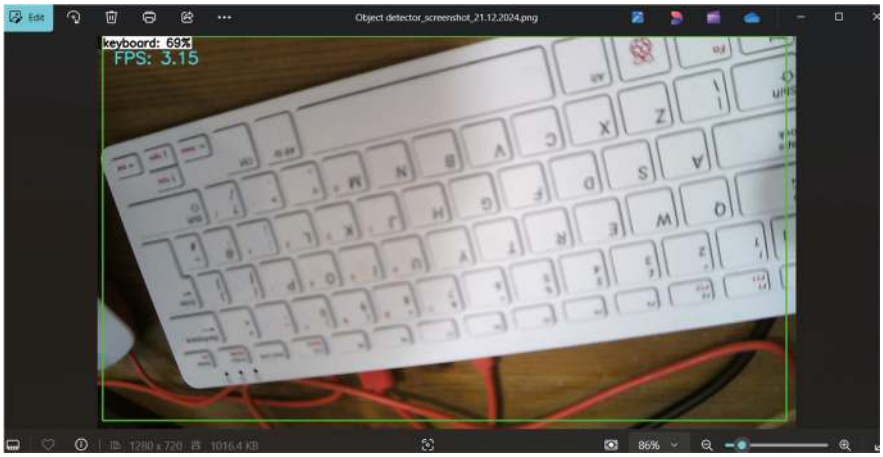


Figure 16.4 Identifying a keyboard

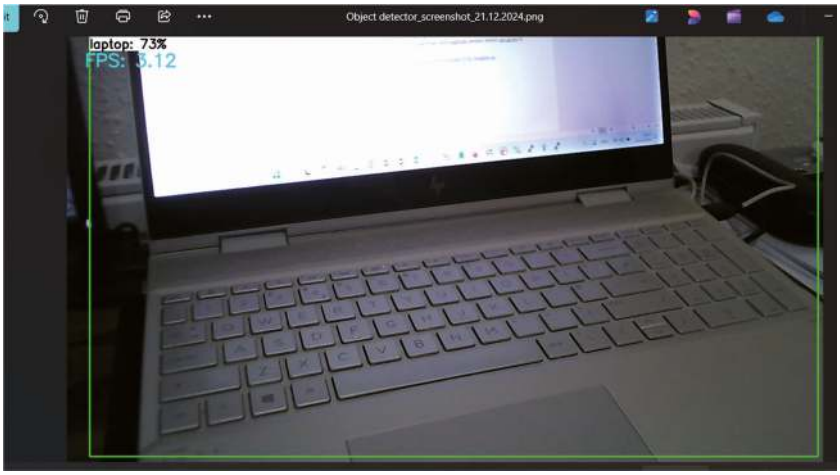


Figure 16.5 Identifying a laptop

In this project, the BeagleY-AI displays a frame rate of only 3 FPS. Some users recommended using the Texas Instruments Deep Learning (TIDL) library for faster processing on the BeagleY-AI.

16.4 BeagleY-AI ChatGPT

This project uses voice input and voice output functionalities with a BeagleY-AI. The project employs ReSpeaker Lite as the audio input and output device, enabling interaction with the ChatGPT and speech-to-text conversion services. Full project details are available on the following website:

https://wiki.seeedstudio.com/respeaker_lite_beagley-ai_chatgpt/

The required hardware for this project are:

- BeagleY-AI
- ReSpeaker Lite USB-2 –Mic Array

The project is based on using Python programming where the program implements a voice assistant that listens for a wake-up word. This word is converted into text, a response is generated using GPT-4. The response is then converted to speech and played back to the user. If the program fails to recognize the command three times, it returns to listening mode and waits until the wake-up word is detected again. Full project details, including the Python program, are given on the above website.

16.5 BeagleY-AI Smart Assistant

This is a smart voice assistant project, where the project responds to a prompt poised by a person. OpenAI is used in the project. Full hardware and software project details are given on the following website:

https://medium.com/@s-kodiganti/designing-a-smart-assistant-with-beagley-ai-139d0451cd15

The following are required for the project:

- BeagleY-AI
- Google AIY voice kit
- USB microphone
- USB monitor+keyboard+mouse
- OpenAI developer account

16.6 BeagleY-AI Robotics

This is a YouTube video that shows the application of BeagleY-AI in robotics. A robotic arm is shown with a suction cup and an air pipe running through the outside. In this project, you drop an object in the center of a box, and it detects the object, picks it up, and places it in the appropriate box.

Full details of this project are available at the following YouTube site:

https://www.youtube.com/shorts/wcF1PEYnZWk

16.7 BeagleY-AI Machine Learning

This YouTube video shows the application of BeagleY-AI in machine learning. Details of the project are available at the following YouTube site:

https://www.youtube.com/watch?v=rPUL7HnFPDI

Chapter 17 • Useful Websites

Further information, projects, tutorials, and documentation are available on the Beagleboard forum website. Readers can search various hardware and software-related items at the following website:

openbeagle.org

A list of useful websites is given in this chapter to help readers gain more information on using the BeagleY-AI.

- https://forum.beagleboard.org/
- https://www.beagleboard.org/collaborate
- https://www.beagleboard.org/boards/beagley-ai
- https://www.ti.com/tool/BEAGLEY-AI
- https://www.elektor.com/products/beagley-ai-sbc-with-gpu-dsp-and-ai-accelerators?srsltid=AfmBOooUO2Nc38wi62qUJwZXDP7l21Nznru-9LY2hDUOhFJ8XVbluzd5R
- https://docs.beagle.cc/books/beaglebone-cookbook/index.html
- https://community.element14.com/products/devtools/single-board-computers/next-genbeaglebone/b/blog/posts/beaglebone-control-stepper-motors-with-pru---part-5-it-works
- https://www.hackster.io/cw-earley/simple-bluetooth-device-detection-0d2469#schematics
- https://docs.beagle.cc/boards/beagley/ai/demos/beagley-ai-using-i2c-oled-display.html
- https://docs.beagleboard.org/boards/beagley/ai/demos/beagley-ai-using-i2c-oled-display.html
- https://www.youtube.com/watch?v=SaIpz00lE84
- https://medium.com/@s-kodiganti/designing-a-smart-assistant-with-beagley-ai-139d0451cd15
- https://www.cnx-software.com/2024/10/13/beagley-ai-review-sbc-debian-12-tensorflow-lite-ai-demos/

Index

The BeagleY-AI Handbook

A Practical Guide to AI, Python, and Hardware Projects

Welcome to your BeagleY-AI journey! This compact, powerful, and affordable single-board computer is perfect for developers and hobbyists. With its **dedicated 4 TOPS AI co-processor** and a **1.4GHz Quad-core Cortex-A53 CPU**, the BeagleY-AI is equipped to handle both **AI applications** and real-time I/O tasks. Powered by the T**exas Instruments AM67A processor**, it offers DSPs, a 3D graphics unit, and video accelerators.

Inside this handbook, you'll find over 50 hands-on projects that cover a wide range of topics—from basic circuits with LEDs and sensors to an AI-driven project. Each project is written in Python 3 and includes detailed explanations and full program listings to guide you. Whether you're a beginner or more advanced, you can follow these projects as they are or modify them to fit your own creative ideas.

What you´ll be working on

Here's a glimpse of some exciting projects included in this handbook:

- **Morse Code Exerciser with LED or Buzzer**
  Type a message and watch it come to life as an LED or buzzer translates your text into Morse code.
- **Ultrasonic Distance Measurement**
  Use an ultrasonic sensor to measure distances and display the result in real time.
- **Environmental Data Display & Visualization**
  Collect temperature, pressure, and humidity readings from the BME280 sensor, and display or plot them on a graphical interface.
- **SPI – Voltmeter with ADC**
  Learn how to measure voltage using an external ADC and display the results on your BeagleY-AI.
- **GPS Coordinates Display**
  Track your location with a GPS module and view geographic coordinates on your screen.
- **BeagleY-AI and Raspberry Pi 4 Communication**
  Discover how to make your BeagleY-AI and Raspberry Pi communicate over a serial link and exchange data.
- **AI-Driven Object Detection with TensorFlow Lite**
  Set up and run an object detection model using TensorFlow Lite on the BeagleY-AI platform, with complete hardware and software details provided.

Prof. Dr. Dogan Ibrahim holds a BSc degree in Electronic Engineering, an MSc degree in Automatic Control Engineering, and a PhD in Digital Signal Processing. He has worked in numerous industrial organizations before returning to academic life. Prof. Ibrahim is the author of over 60 technical books and over 200 technical articles on microcontrollers, microprocessors, and related fields. He is a Chartered Electrical Engineer and a Fellow of the Institution of Engineering and Technology.

Ahmet Ibrahim obtained his BSc degree from the University of Greenwich in London, where he also completed an MSc course. Ahmet has worked at various industrial organizations at different levels and is currently working in a large organization in the IT field. He is the author of several technical books and articles.

Elektor International Media
www.elektor.com

ISBN 978-3-89576-656-5
9 783895 766565

elektor
design > share > earn