Starter Maker PCB

Usage Documentation

version 2.1

Table of contents

Introduction

The **Starter Maker PCB** enables beginner digital makers to explore a range of basic projects and methods. The definition of 'digital making' is somewhat wide but is generally taken to be the combination of Design & Technology (DT) aspects of 'making' with the 'digital' Computer Science (CS) skills of programming and code development. It is therefore all about the creation and development of physical things that are controlled and managed by software. The activity scope is therefore broad, embracing electronics, photography, robotics/mechatronics, music, artwork/installations, and much more!

Digital making with small low-cost single board computers (SBCs) like the Raspberry Pi, or simpler microcontrollers like the Pico, is not only a fun thing to do, but it is increasingly important to encourage these skills in the context of the extraordinarily rapid Fourth Industrial Revolution that is now upon us. Equipping both adults and young people for a changing society, and the 'digital intense' world of work that is emerging, is therefore becoming ever more vital. This is especially important in the context of recent reports which estimate that 90% of <u>all</u> new jobs will require digital skills to some degree.

The **Starter Maker PCB** is a custom printed circuit board (PCB) onto which a small set of electronic components and a connector can be soldered as shown below, allowing the core skill of soldering to be practiced. This allows a beginner digital maker to create a 'module' that connects to the GPIO pins of a Raspberry Pi SBC or one of several different microcontrollers and enables a variety of projects and methods to be explored, all involving software control by the SBC or microcontroller.
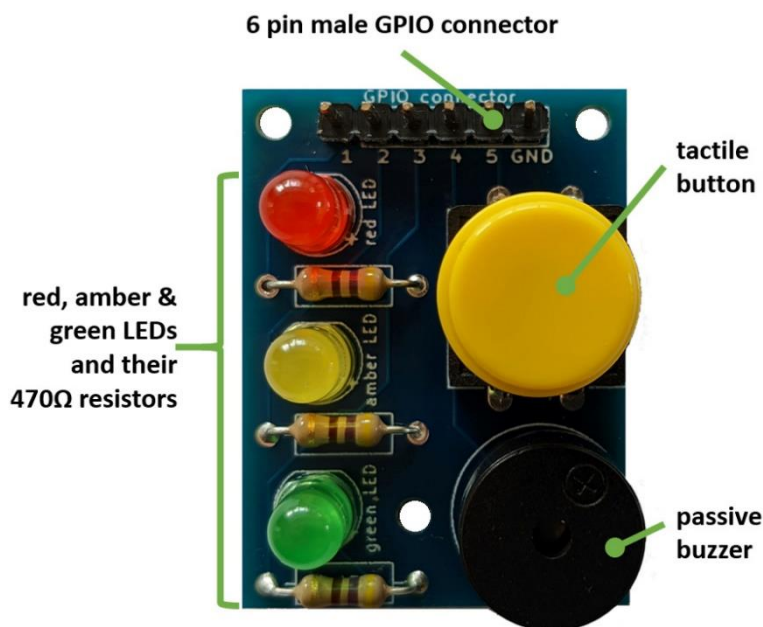
A key aim of this evolving project development is to provide access to a small but expanding library of example code to allow a digital maker to begin to explore and control different components and devices. Once the basics have been explored a digital maker can progress their making experimentation with the use of further, more extensive Maker PCB projects that are available, as well as developing their own custom projects.

Building a permanent assembly from a set of components that uses a custom Printed Circuit Board (PCB), not only allows an extra 'soldering' skill to be practiced, but also allows the more robust populated PCB to be used over an extended period to develop different software projects and options.

Example software is available to be downloaded that provides 'starter' capabilities that a digital maker can use immediately and then build upon to extend their software development skills. As the assembled PCB has been designed to connect to a variety of computing platforms, example programs are available that use a variety of coding languages (Scratch, Python, MicroPython and C/C++). When using a Raspberry Pi SBC both Scratch and Python code have been developed for most of the example projects described in this document so that this 'dual' coding language approach can facilitate a maker progressing from the easy-to-use Scratch 'block programming' to more functional text-based Python coding.

PCB development and component usage

Appendix A: PCB development details provides further information and detail for the **Starter Maker PCB** v1.0 design and a .zip file of the 'Gerber' design files can be downloaded from **here** so that anyone can have small quantities of the PCB manufactured using one of the many low cost online suppliers that are now available.

The PCB design process has used the KiCAD open source software, and the images below show some details of the v1.0 design generated by the software.
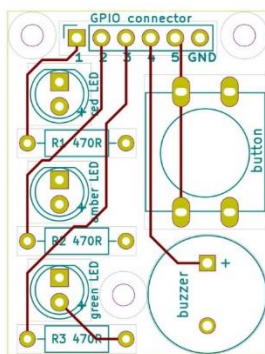
front of the PCB showing the component
footprints and the copper signal interconnections
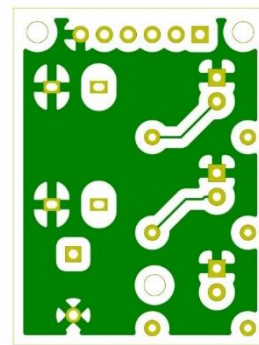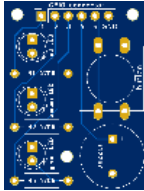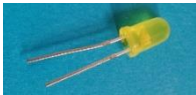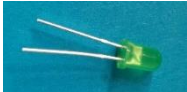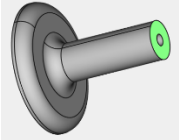
back of the PCB with the 'flooded' ground plane
as well as some interconnections on this side

Assembled PCB component details

Each **Starter Maker PCB** assembly requires all the components in the quantities set out in the table and image below so that a PCB can be completed with soldered in components.

A detailed build description is provided in the *PCB assembly details* section further down this document and additional technical information on each of the components is provided in

Appendix B: Maker PCB component usage details.

description	image	quantity	description	image	quantity
Printed circuit board (v1.0)		1	Tactile button with cap (colour of cap to suit)		1
Red LED		1	Passive buzzer		1
Amber LED		1	1row x 6P male black header		1
Green LED		1	470Ω resistors (carbon or metal film type can be used) bands yellow:purple:brown or yellow:purple:black:black:brown		3
3D printed 'feet' + 6mm M2 self-tap screws		3 + 3	10cm or 20cm female-to-female Dupont jumpers		6
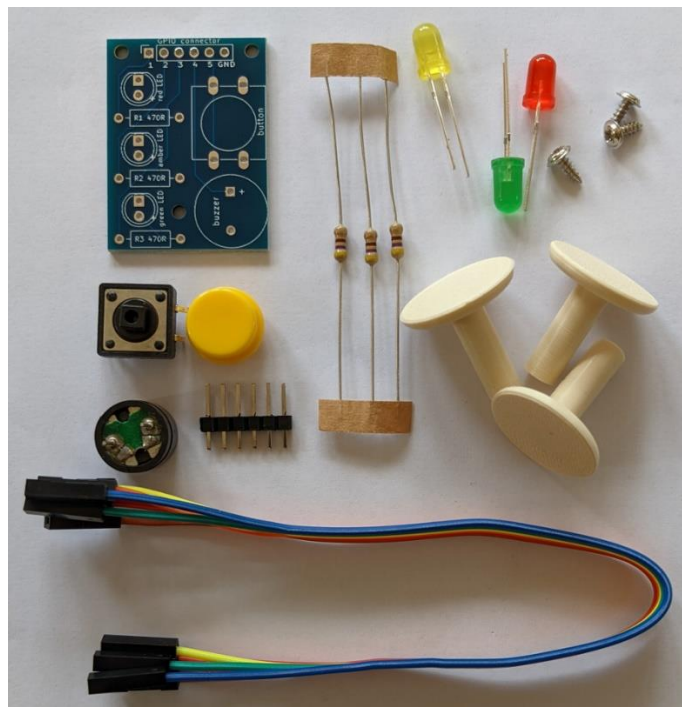
PCB assembly details

Assembling a complete Starter Maker PCB, whilst not a significant soldering task and it is an excellent exercise for a 'beginner maker', does need to be done carefully and the following 'tips' may be found useful.

Some suggested assembly/soldering 'tips'

- There are not a large number of components to be added to the PCB, but as this is a manual soldering process, a suggested component sequence is detailed in the next section that can help avoid previously soldered components 'getting in the way' of the next component being soldered.

- A good quality lead free solder wire with a rosin/flux core should be used. In addition, to ensure that only small amounts of solder can be carefully added to a joint in a controlled way, a very small diameter solder wire, say 0.6mm, is recommended.

- A narrow tipped soldering iron should be used and, as with any soldering activity, you should make sure that the soldering iron is fully up to temperature and the activity is carried out in a well-ventilated area. Whilst a more sophisticated thermostatically controlled soldering iron (set at say 350°C) is always useful, a more basic uncontrolled iron is perfectly adequate for these activities.

- Generally, the use of additional solder paste/flux should not be necessary and should generally be avoided, because whilst it might be 'safely' used with all the Starter Maker PCB discrete components, when soldering female headers in particular, additional flux can cause excess solder to quickly 'wick' up into the header and block the insertion of a male component pin into the female header opening.

- Once a PCB has been fully populated, whilst a final 'cleaning' step may not be necessary, especially if no additional solder paste/flux has been used, washing the complete PCB in isopropyl alcohol/isopropanol and 'solvent brushing' all the soldered joints with a small clean paint brush will ensure any soldering residues are removed which could, for example, create low resistance paths between adjacent pins on the header connector. This solvent use should of course only be carried out in a well-ventilated area observing all necessary safety precautions when using volatile solvents.

- Finally, completing all the needed soldering steps for a complete PCB may take a little time, so take an occasional break if this is your first significant soldering exercise.

Suggested component assembly sequence

Below are a series of images, with associated descriptive text, in the suggested sequence for soldering all of the components onto the PCB.

Resistors x3

All three of the 470Ω resistors are fitted in a similar way by inserting a resistor into the designated area on the front of the PCB – clear labelling is provided on the PCB - and then folding the resistor leads on the underside so that the resistor is held 'snugly' against the PCB front surface as shown in the first image below on the left.

As shown in the second and third images, the soldering process is completed by placing the soldering iron tip firmly on both the PCB opening and the resistor lead, and gently touching the joint with the tip of the solder wire until it is seen to start to melt. A small amount of solder wire is then 'fed' into the joint and the soldering iron tip held in place for a few more seconds until it is seen that the melted solder has 'wetted' the surfaces of the resistor's lead and the PCB opening and some solder has 'wicked' into and completely filled the gap.

The excess length of resistor lead can then be snipped off or 'waggled' until it breaks off.
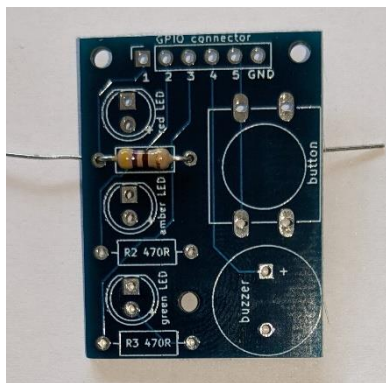
470Ω resistors inserted on front side of PCB and leads folded back on the rear side
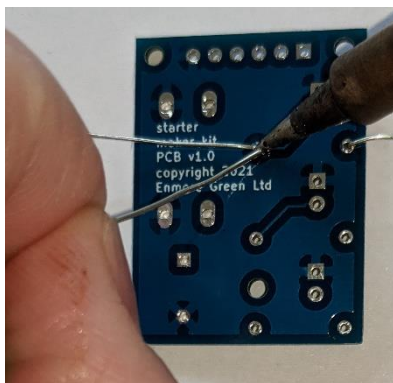
the soldering iron tip held at the junction with the PCB opening
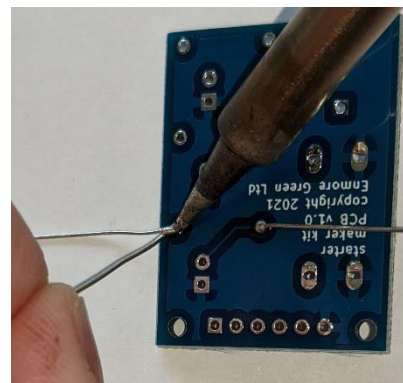
a small amount of solder is 'fed' into the joint as it melts until it just fills the gap

Tactile button

The button is inserted into the front of PCB: each of the four legs may need to be straightened slightly so that they align with the four 'slot' openings on the PCB. The button is firmly pushed into the PCB openings so that it sits evenly and directly on the PCB surface as shown in the first image below left. The next two images then illustrate how all four legs are soldered in place by the soldering iron tip being pressed firmly on the junction between the button leg and the PCB opening with solder being fed into the gap as it melts.

The tactile button is inserted onto the front of the PCB so that it sits evenly and directly on the PCB surface

the soldering iron tip held at the junction of both the button leg and the PCB opening with solder being fed into the gap
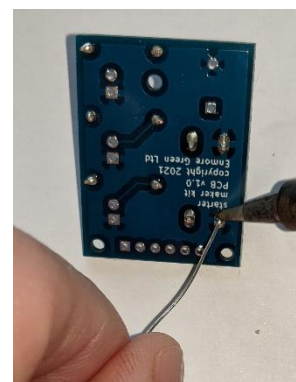
all four legs of the tactile button are soldered in place in a similar way

Passive buzzer

The buzzer is inserted onto the front of the PCB making sure that the +'ve connector is inserted into the correct opening on the PCB. The two connector leads are relatively short and stiff and whilst they can be bent slightly to hold the buzzer in place, to make sure that the buzzer stays fully inserted and flush with the PCB surface, small pieces of Blu Tack can be used to temporarily fix the buzzer in place, as shown in the image below left.

Each of the connectors are then soldered in place in the same way as all the previous components by the soldering iron tip being pressed firmly on the junction between the buzzer connector and the PCB opening with solder being fed into the gap as it melts, as shown in the image below right.

The small excess lengths of buzzer connector can then be snipped off or 'waggled' until they break off.
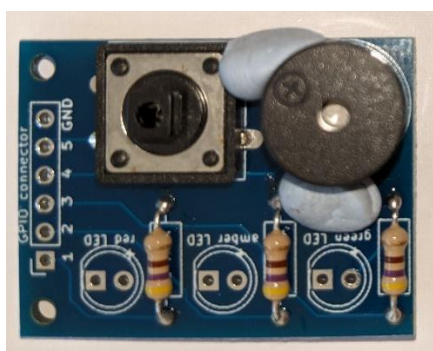
The buzzer inserted into the front of the PCB and temporarily held firmly to the PCB face using some Blu Tack

the soldering iron tip is firmly pressed on both the PCB opening and the buzzer lead with solder being fed into the gap as it melts

Red, Amber and Green LEDs

The LEDs are inserted onto the front of the PCB in their red, amber, green sequence, making sure that the <u>longer</u> +'ve anode connectors are inserted into the correct openings on the PCB, which are clearly labelled with a small +.

Even though both connectors of a LED are quite long, and they can be bent to try to hold the LED in place, because of their height the LEDs do tend to 'wobble' and not stay flush with the surface of the PCB. Small pieces of Blu Tack can therefore be used to fix the LEDs firmly in place, as shown for the red LED in the first image below left.

Each of the connectors are then soldered in place in the same way as all the previous components by the soldering iron tip being pressed firmly on the junction between the LED lead and the PCB opening with solder being fed into the gap as it melts, as shown in the image below centre.

The final image below right shows that more than one LED can be positioned/held in place and soldered at the same time – which will make the installation a little quicker.

Once soldered in place the excess length of LED connector lead can be snipped off or 'waggled' until it breaks off.

the LEDs are inserted onto the front of the PCB and temporarily held firmly to the PCB front face using some Blu Tack
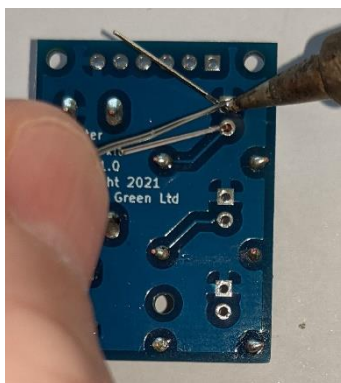
the soldering iron tip held at the junction of the LED leg and the PCB opening with solder being fed into the gap as it melts

More than one component for soldering can be inserted into the PCB at one time !

1 row x 6P male header

The final component to be soldered into the PCB is the 6 pin male header that is used to connect the assembled PCB to its 'controller' using Dupont jumper leads.

As shown in the first image below left, this too can best be held in place on the front of the PCB so that it remains evenly inserted and flush with the PCB surface, by using small pieces of Blu Tack.

The second and third images below then show that the soldering is completed for all six insertion points by the soldering iron tip being successively pressed firmly on the junction between each header insert and its PCB opening with solder being fed into the gap as it melts.

the male header is inserted onto the front of the PCB and temporarily held firmly to the PCB face using some Blu Tack
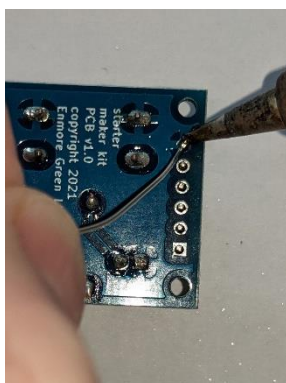
the soldering iron tip held at the junction of the header insert and the PCB opening with solder being fed into the gap
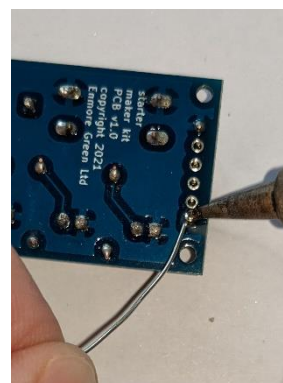
each of the header inserts are progressively soldered in place

Starter Maker PCB software

Using a Virtual Environment for Raspberry Pi Python projects

A virtual environment (*venv*) is a useful, logical way of isolating Python project dependencies from each other on a Raspberry Pi and ensuring things don't break when there are conflicts.

Starting with the Bookworm OS this more structured approach is more strictly enforced, so it should be used when using Python in the various coding examples described next.

To make this easier (using the advice from **this link**) the following is recommended to both create a Virtual Environment (*venv*) and to automatically 'activate' it whenever a CLI window is opened from the Raspberry Pi desktop or the Pi is remotely accessed by SSH.

Using the command: *sudo nano /home/YOURUSERNAME/.bashrc*

- the following should be added at the end of the file:

```
# automatically set up a virtual environment for using python
PY_ENV_DIR=~/my_virtual_env
if [ ! -f $PY_ENV_DIR/bin/activate ]; then
    printf "Creating user Python environment in $PY_ENV_DIR, please wait...\n"
    mkdir -p $PY_ENV_DIR
    python3 -m venv --system-site-packages --prompt myenv $PY_ENV_DIR
fi
printf " ↓ ↓ ↓ ↓   Hello, we've activated a Python venv for you. To exit, type \"deactivate\".\n"
source $PY_ENV_DIR/bin/activate
```

When using the Thonny IDE for Python coding, this should also be configured to use the Virtual Environment set up from the .bashrc edit above. But the current 4.x release of Thonny for the Raspberry Pi with the Bookworm OS does not make the use of a pre-configured *venv* straightforward – so the 'work around' that seems to work OK is to use the Thonny menus to create another new Virtual Environment and then close Thonny and edit its config file at:

/home/YOURUSERNAME/.config/Thonny/configuration.ini

- and change the references in the file to the newly set up *venv* to the pre-existing one.

Electronic basics coding

To support the exploration of some electronic 'basics' with an assembled **Starter Maker PCB**, a number of projects described further in the main *Electronic basics* section below, show how to use:

- both Scratch and Python coding for a Raspberry Pi SBC;

- MicroPython with the Thonny IDE and C/C++ with the Arduino IDE for a Pico microcontroller; and

- C/C++ with the Arduino IDE for both a ESP32 and ESP8266 microcontroller.

Details on how to download all the available electronic 'basics' example code for use on the various 'controller' devices are provided in *Appendix C1: Electronic basics software download*

For the Raspberry Pi, the provided Scratch examples closely match how the corresponding Python examples work and can therefore help digital makers 'transition' from the easier to use graphical user interface of Scratch, to the more extensive and powerful functionality of the text based Python language.

Please note that this document does not provide support for the set up and running of any of the 'controller' devices, nor the general use of Scratch, Python, MicroPython or C/C++, nor the setting up and use of the Thonny and Arduino IDEs - since there are plenty of good books and online materials available to do this.

For Raspberry Pi Scratch coding the examples and descriptions provided are all for the offline version of Scratch 3, available on a Raspberry Pi5 with a minimum of 2GB of memory, which provides custom coding blocks to support the use of the Raspberry Pi's GPIO pins. The download process does however also provide similar code examples for the old/legacy off-line version of Scratch 1.4 in order to support the use of Scratch with much older versions of the Raspberry Pi for which only this version of Scratch is available. Also provided are Scratch 2 examples, but as this version of Scratch uses Adobe Flash which is no longer supported and does not run on a Raspberry Pi running Buster or later, these examples can only run on a Raspberry Pi running Stretch or later (N.B. Bookworm now recommended!). In addition, as Scratch 2 does not allow a GPIO pin's pull up/down status to be set, some additional system configuration is needed if the button is connected to any GPIO pin other than 2 to 8, since these higher numbered pins have their default as pull-down. The provided example code assumes the button is connected to GPIO#26 so this system configuration is needed if the example code is to be used unchanged. This is discussed in more detail in *Appendix F: Raspberry Pi GPIO pin default settings.*

Image Taking coding

To support the exploration of various Image Taking methods, but just on a Raspberry Pi SBC with a USB camera connected to one of its USB ports, a set of example Python code is available.

Details on how to download all the available Image Taking example code for use on a Raspberry Pi SBC are provided in *Appendix C2: Image Taking software download.*

Connecting an assembled PCB to a 'controller'

The Starter Maker PCB is best used with three small 3D printed 'feet' (design downloadable from **here**) and 6mm M2 self-tap screws that allow the PCB assembly to be mounted in a stable and easily handled way, as shown in the two images below.
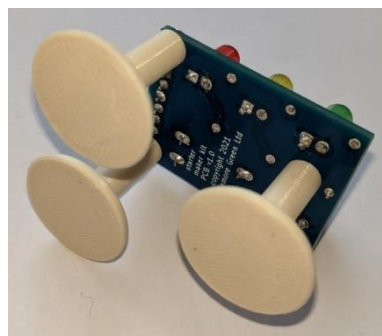
Using readily available 20cm or 10cm female-to-female Dupont jumpers then allow the assembled and mounted PCB to be connected to the various 'controller' devices as shown below.

*Starter Maker PCB connected
direct to the GPIO pins of a
Raspberry Pi SBC*
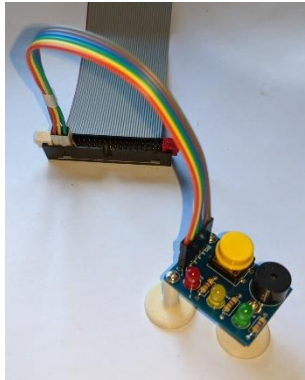
*Starter Maker PCB connected
to a Raspberry Pi SBC GPIO
extension cable*

*Starter Maker PCB connected
to a Raspberry Pi
Pico microcontroller*
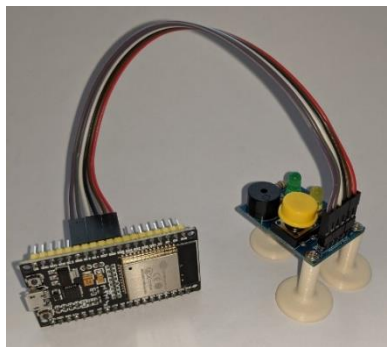
*Starter Maker PCB
connected to a ESP32
microcontroller 38 pin
module*
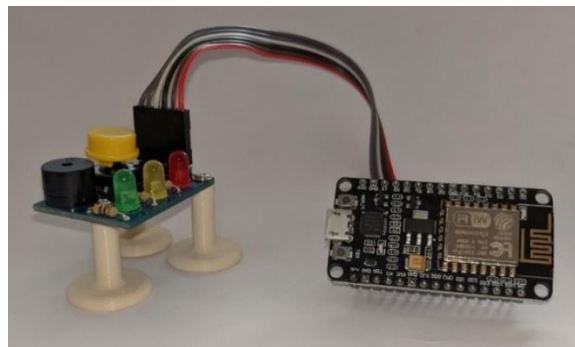
*Starter Maker PCB connected to a
ESP8266 microcontroller NodeMCU v1.0 module*

Electronic basics

A key aim of the **Starter Maker PCB** is to enable a range of simple electronic projects to be explored on a variety of different computing platforms ('controllers') and using several different programming languages.
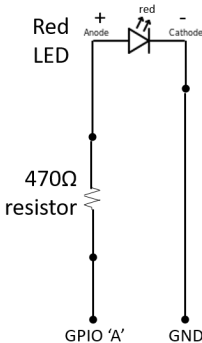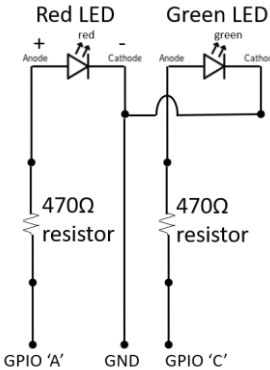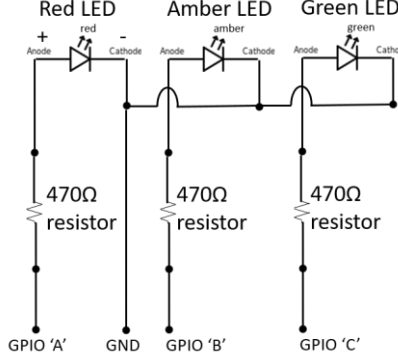
These "electronic basics" are the starting point for any subsequent usage of further Maker PCBs and custom project development by showing how 'output' components like the various LEDs or the buzzer can be operated by an 'input', such as the pressing of a button.
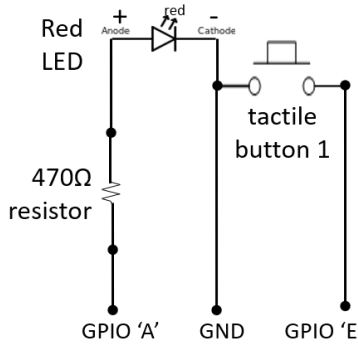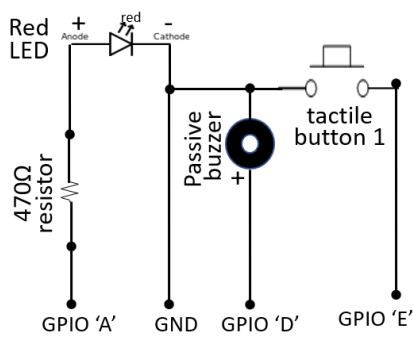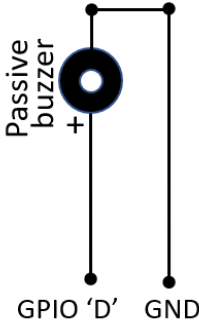
These basic functions introduce a key facet of 'digital' operation where something is 'switched on', not by completing a circuit by the closing of a switch, but instead by sensing something that 'happens' and using logic to decide whether to separately energise/switch something on, or to carry out some other action.

For each of the 'controllers' that are profiled, similar functional example programs have been developed as follows:

- ***Raspberry Pi SBCs*** - Python and Scratch 3 code developed for six example programs, plus a simple Python/Flask based web interface developed to switch the red LED on or off, to flash an LED on/off for a set number of cycles, and to buzz/play tunes.

- ***Raspberry Pi Pico microcontroller*** – MicroPython code for use with the Thonny IDE, and C/C++ code using the Arduino IDE, developed for six example programs.

- ***ESP32 microcontroller*** - C/C++ code developed for six example programs using the Arduino IDE, plus a simple web interface developed to switch the red LED on or off, to flash an LED on/off for a set number of cycles, and to buzz/play tunes.

- ***ESP8266 microcontroller*** - C/C++ code developed for all six example programs using the Arduino IDE, plus a simple web interface developed to switch the red LED on or off, to flash an LED on/off for a set number of cycles, and to buzz/play tunes.

Generic circuit diagrams for the use of the PCB components are shown below for each of the 6 example program arrangements where the 'controller' GPIO connections are labelled A, B, C, D and E. The following sections of the document then provide the A-E GPIO connection values used for each of the 'controller' options.

| **Single flashing LED** | **Red/Green flashing LEDs** | **Red/Amber/Green flashing LEDs** |

| **Button switched LED** | **Button switched LED & buzzer** | **Buzzer tunes** |

Raspberry Pi SBC electronic basics usage

Any of the Raspberry Pi SBCs (Pi5, Pi Zero, etc.) can be used, and the table below shows the circuit diagram (shown above) A-E GPIO connections used in the example software.

Component	6 pin male connector PCB label	circuit diagram label	GPIO pin(s)
Red LED	1	A	**GPIO#21** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	2	B	**GPIO#20** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	3	C	**GPIO#16** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Passive buzzer	4	D	**GPIO#19** connected to the positive terminal of the buzzer
Tactile button	5	E	**GPIO#26** and GND connections across the button

To support the exploration of some electronic basics with an assembled **Starter Maker PCB** connected to a Raspberry Pi SBC, the following projects using both Scratch 3 and Python coding can be used.

The software, when using the downloadable control script, will all have been downloaded to a folder within your own username file space and once this has completed, as shown in *Appendix C1: Electronic basics software download*, the default folder on the Raspberry Pi for all the 'electronic basics' Python software will be:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/

 .. and the default folder for all the 'electronic basics' Scratch 3 software will be:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/scratch3/

The files can of course be moved to somewhere else that may be more convenient, but these are the assumed locations for the descriptions of the individual projects below.

Single flashing LED

The circuit diagram shown earlier illustrates how the red LED on the PCB is connected through to ground from its (negative) cathode lead and to the Raspberry Pi's GPIO#21 pin via a 470Ω resistor from its (positive) anode lead.

The LED will be turned ON when the GPIO pin is set HIGH as this sets the pin at about 3.3V, and to avoid damaging the LED, the 470Ω resistor ensures that the current through it is limited. More details of how LEDs work is provided in

Appendix B: Maker PCB component usage details*.*
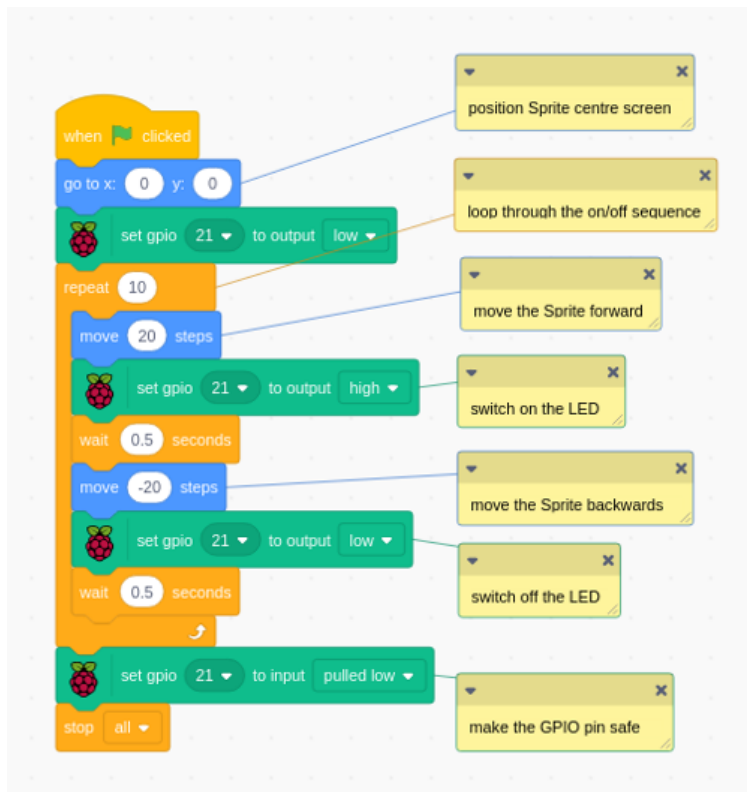
This very simple Scratch project flashes the red LED on/off and the screenshot on the left shows the example code which can be run in Scratch 3 by using the 'File->Load from your computer' menu and loading the ***LED_flash.sb3*** file from the folder it was downloaded to.

To run the Python ***LED_flash.py*** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/LED_flash.py
```

Red/Green flashing LEDs

The circuit diagram shown earlier illustrates how the red and green LEDs on the PCB are connected through to ground from their (negative) cathode leads and to the Raspberry Pi's GPIO#21 and #16 pins via 470Ω resistors from their (positive) anode leads. As discussed above and in *Appendix B: Maker PCB component usage details e*ach LED will be turned ON when its GPIO pin is set HIGH and the 470Ω resistors protect the LEDs from over current damage.

This project alternately flashes the red and green LEDs on/off and the screenshot below shows the example code which can be run in Scratch 3 by using the 'File->Load from your computer' menu and loading the ***LED_red_green_flash.sb3*** file from the folder it was downloaded to.

To run the Python **_LED_red_green_flash.py_** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/LED_red_green_flash.py
```

Red/Amber/Green flashing LEDs

The circuit diagram shown earlier illustrates how the red, amber and green LEDs on the PCB are connected through to ground from their (negative) cathode leads and to the Raspberry Pi's GPIO#21, #20 and #16 pins via 470Ω resistors from their (positive) anode leads.

As discussed above and in

Appendix B: *Maker PCB component usage* details, each LED will be turned ON when its GPIO pin is set HIGH and the 470Ω resistors protect the LEDs from over current damage.

This project alternately flashes the red, amber and green LEDs on/off in sequence.

Only example Python code is made available for this project allowing the digital maker to develop their own version of Scratch code, perhaps building upon the LED_red_green_flash.sb3 example.

To run the Python **LED_red_amber_green_flash.py** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/LED_red_amber_green_flash.py
```

Button switched LED

The circuit diagram shown earlier illustrates how the:

- red LED on the PCB is connected through to ground from its (negative) cathode lead and to the Raspberry Pi's GPIO#21 pin via a 470Ω resistor from its (positive) anode lead, and

- how the button is connected between ground and GPIO #26

This project turns the red LED ON for 3 seconds when the button is pressed. The button is configured to be 'pulled high' and will go low when it is pressed. The screenshot on the left shows the example code which can be run in Scratch 3 by using the 'File->Load from your computer' menu and loading the **LED_button_flash.sb3** file from the folder it was downloaded to.

To run the Python **LED_button_flash.py** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/LED_button_flash.py
```

Button switched LED & buzzer

The circuit diagram shown earlier illustrates how the:

- red LED on the PCB is connected through to ground from its (negative) cathode lead and to the Raspberry Pi's GPIO#21 pin via a 470Ω resistor from its (positive) anode lead,

- how the button is connected between ground and GPIO #26, and how

- the buzzer is connected between ground and GPIO#19

This project turns the red LED ON and sounds the buzzer for 3 seconds when the button is pressed. AS in the previous project the button is configured to be 'pulled high' and will go low when it is pressed. The screenshot below shows the example code which can be run in Scratch 3 by using the 'File->Load from your computer' menu and loading the **LED_button_buzzer.sb3** file from the folder it was downloaded to.
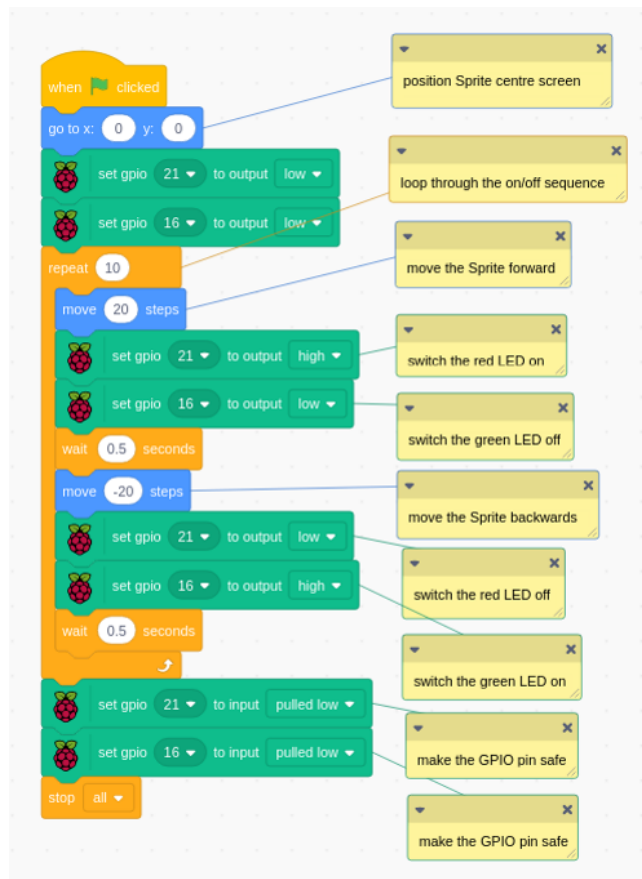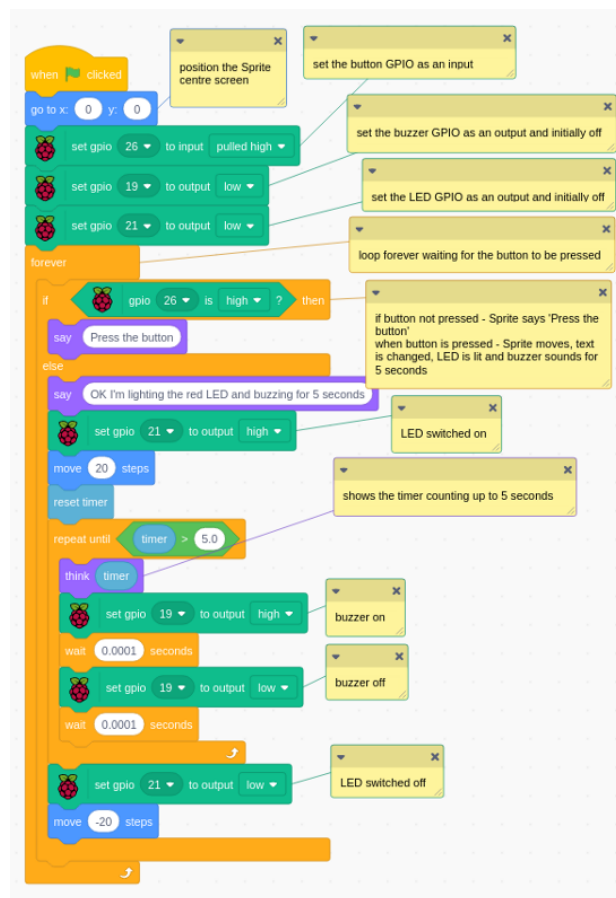

To run the Python **LED_button_buzzer.py** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/LED_button_buzzer.py
```

Buzzer 'tunes'

The circuit diagram shown earlier illustrates how just the buzzer can be used with it being connected between ground and GPIO#19.

This project, only developed using Python software, shows how the very simple passive buzzer installed on the PCB can be programmed to 'play' quite elaborate tunes.

The code:

- defines a set of **notes** by their frequency,

- a series of well-known tunes defined by a **melody** as a short representative sequence of notes, and

- a **tempo** defined for each tune to signify the duration of each note in the melody.

To run the Python **buzzer_player.py** code use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/starter_ebasics/RPi_code/buzzer_player.py
```

LED web interface

This web interface method uses software called 'Flask', which is what is known as a micro web framework. It is written in Python and is based on two other pieces of software called the Werkzeug toolkit and the Jinja2 template engine.

Flask provides a very flexible 'flow' arrangement that allows customised HTML web pages, created with HTML templates, to invoke specific sections of Python code, passing data to it - which in turn can 'call' other HTML web pages, passing data back to the HTML.

Details of the various software elements for this 'electronics' application are provided in *Appendix D: Flask web server details*, which will allow more advanced Python users to use the supplied code as a base for any further developments they may want to do.

What may be considered a quirk of the Flask system is that a browser that accesses the web server can only use the conventional HTTP port 80 if the main software is run by a user with root privileges. Two separate sets of Python software are therefore available for using this web interface coding method, depending upon whether:

- the 'root' user can be used (e.g., using sudo) which can only be done from a terminal window, or whether

- your Raspberry Pi is only set up for use by non-root users and/or the Thonny IDE is to be used

LED web interface: user with root privileges
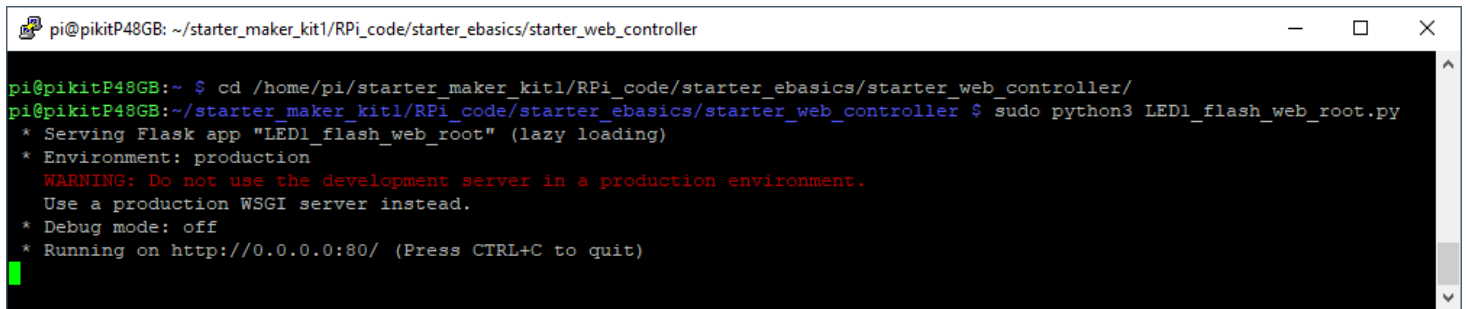
With your Raspberry Pi in 'Desktop' mode and connected to your local network, use a 'Terminal' window to run the **LED1_flash_web_root.py** program that the software download process would have placed in the subdirectory:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/starter_web_controller/

The sequence of commands to move to this directory and then run the code are as follows:

```
$ cd /home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/starter_web_controller/
    $ sudo python3 LED1_flash_web_root.py
```

The screen shot below shows the Terminal window display when running on a Raspberry Pi with a host name of 'pikitP48GB':

Now either open the browser on your Raspberry Pi or use a browser on another computer that is also connected to your local network and go to the URL http://yourpihostname, where *yourpihostname* is whatever host name you have given your Raspberry Pi.

Your local router should then connect the browser to your Pi and will show the screen shot on the right.
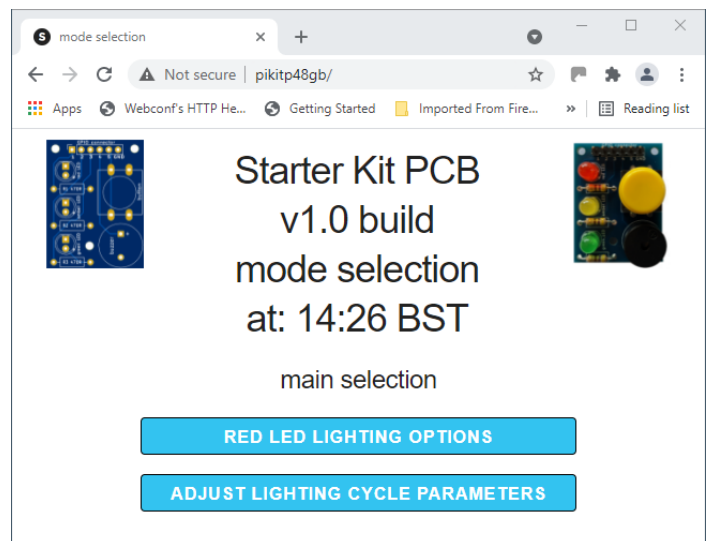
Now you can click the blue 'buttons' on the screen to control the lighting of the red LED in various ways.
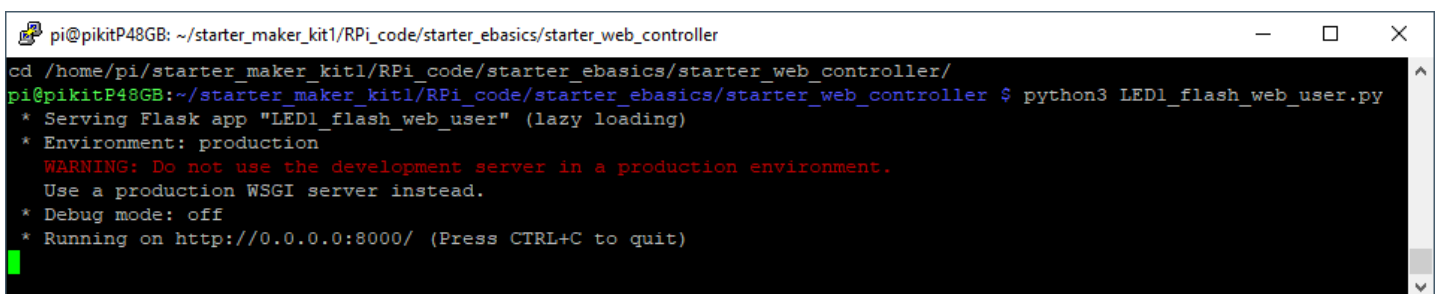
LED web interface: user without root privileges

For non-root privilege user, the Thonny IDE can be used, or the Terminal window commands are as below and as also shown in the screen shot below, i.e., you no longer have to use sudo:

```
$ cd /home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/starter_web_controller/
    $ python3 LED1_flash_web_user.py
```

The other significant difference is that the resultant web pages are now only shown in http port 8000, so the required URL to be typed into your browser is http://yourpihostname:8000, where *yourpihostname* is whatever host name you have given your Raspberry Pi, and the screen shot on the right shows the browser output for this non-root user case.
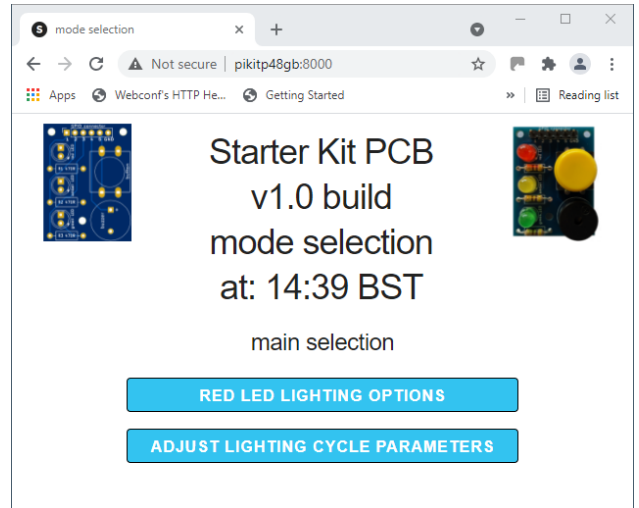
Raspberry Pi Pico microcontroller electronic basics usage

The table below shows the GPIO connections used with the associated circuit diagram A-E labels for a Raspberry Pi Pico microcontroller.

Component	6 pin male connector PCB label	circuit diagram label	GPIO pin(s)
Red LED	1	A	**GPIO#10** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	2	B	**GPIO#11** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	3	C	**GPIO#12** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Passive buzzer	4	D	**GPIO#13** connected to the positive terminal of the buzzer
Tactile button	5	E	**GPIO#14** and GND connections across the button

As each of the available example programs for the Raspberry Pi Pico microcontroller is 'loaded', using either the Thonny or Arduino IDE, the code is made available as a downloadable .ZIP file so that once unzipped the files can be used in whatever computer the IDE is being run.

The details for downloading the Raspberry Pi Pico microcontroller software are provided in *Appendix C1: Electronic basics software download*

ESP32 microcontroller electronic basics usage

The table below shows the GPIO connections used with the associated circuit diagram A-E labels for a ESP32 microcontroller 38 pin module.

Component	6 pin male connector PCB label	circuit diagram label	GPIO pin(s)

Red LED	1	A	**GPIO#25** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	2	B	**GPIO#26** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	3	C	**GPIO#27** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Passive buzzer	4	D	**GPIO#14** connected to the positive terminal of the buzzer
Tactile button	5	E	**GPIO#12** and GND connections across the button

As each of the available example programs for the ESP32 microcontroller is 'loaded', using the Arduino IDE, the code is made available as a downloadable .ZIP file so that once unzipped the files can be used in whatever computer the IDE is being run.

The details for downloading the ESP32 microcontroller software is provided in *Appendix C1: Electronic basics software download*

ESP8266 microcontroller electronic basics usage

The table below shows the GPIO connections used with the associated circuit diagram A-E labels for a ESP8266 microcontroller.

Component	6 pin male connector PCB label	circuit diagram label	GPIO pin(s)
Red LED	1	A	**GPIO#12** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	2	B	**GPIO#13** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	3	C	**GPIO#15** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Passive buzzer	4	D	**GPIO#03** connected to the positive terminal of the buzzer
Tactile button	5	E	**GPIO#01** and GND connections across the button

As each of the available example programs for the ESP8266 microcontroller is 'loaded', using the Arduino IDE, the code is made available as a downloadable .ZIP file so that once unzipped the files can be used in whatever computer the IDE is being run.

The details for downloading the ESP8266 microcontroller software is provided in *Appendix C1: Electronic basics software download*

Image taking methods

The **Starter Maker PCB** allows a number of different Image Taking methods to be explored on a Raspberry Pi SBC that uses a USB camera that is connected to a USB port. Example/demonstration software is available that can use the Maker PCB components listed in the table below for many different image taking control and 'indicator' options.

Component	6 pin header PCB label	circuit diagram label	GPIO pin(s)
Red LED	1	A	**GPIO#21** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	2	B	**GPIO#20** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	3	C	**GPIO#16** connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Tactile button	4	D	**GPIO#26** and GND connections across the button
Passive buzzer	5	E	**GPIO#19** connected to the positive terminal of the buzzer

Whilst many USB cameras will work with a Raspberry Pi, not every camera will necessarily work, especially if the camera needs any special software drivers.

For the various image taking and demonstration software methods discussed below, a simple low cost camera is separately available, as shown above/right attached to a small tripod with a custom 3D printed mounting adaptor – a print file for the adaptor is downloadable **here**.

The illustrated camera only has a maximum resolution of 640x480 pixels, but the demonstration software will allow the use of USB cameras with higher resolutions to be used.

Image streaming

Two separate sets of Python software are available for using this streaming method, depending upon whether the 'root' user can be used (e.g., using sudo) or whether your Raspberry Pi is only set up for use by non-root users. For either method some additional set up is needed and this must be done by the root user. It should also be noted that the software used is relatively advanced and this streaming method has primarily been provided as an initial set up utility. It can therefore be returned to at any time, to check that a connected camera is working correctly for all the other image taking methods described in this document, with images in focus and the camera properly oriented.

It should be noted that most USB cameras need a reasonable amount of power, so if you have an early model of Raspberry Pi which uses a power supply with less than 2.5A you may need to use a powered USB hub for your camera USB connection as you might otherwise get camera errors and unreliable/unstable performance.

As can be seen in the schematics below if a Raspberry Pi Zero is being used or an older version of the Pi which may only have two USB ports, and you are not using a separate powered USB hub, you will need to have a multi-port USB adaptor.

This is illustrated in the schematic below right where the type A sized USB plug on the camera is connected to the micro type B USB socket on the Pi Zero via an adaptor, and so that you have at least one additional port to use a wireless or wired USB keyboard/mouse. An example adaptor shown above right can be sourced from **here**.

Raspberry Pi3

Raspberry Pi Zero
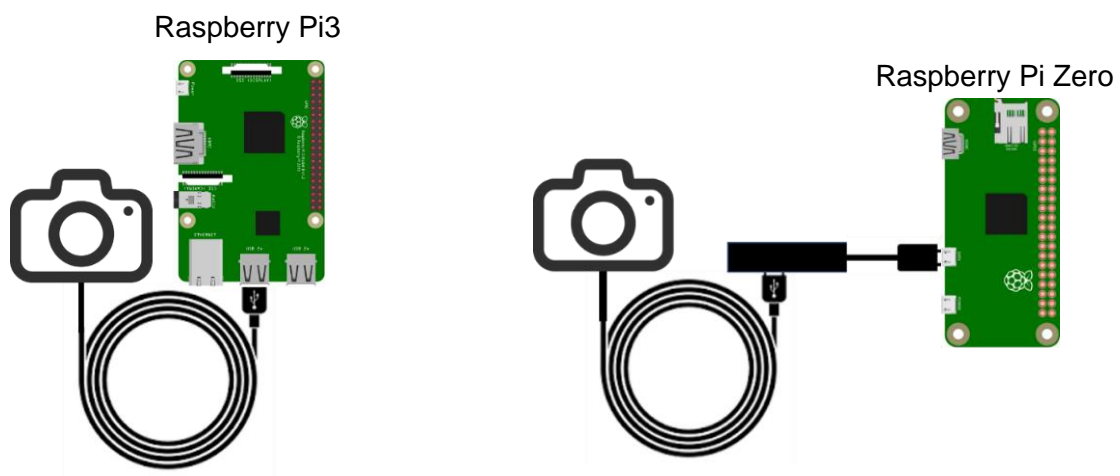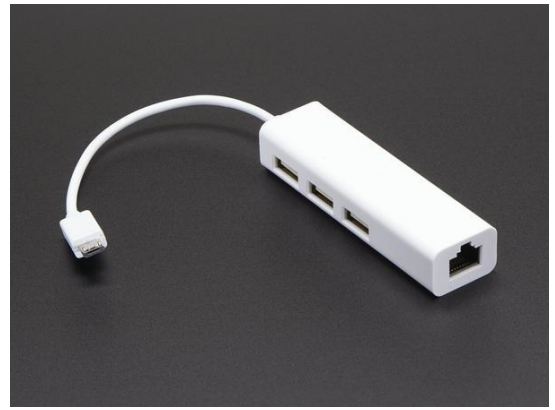
To check that your USB camera has been 'found' by the Pi use the 'lsusb' command in a Terminal window and you would see a GEMBIRD device for the camera shown above, as illustrated in the screen shot below, but obviously a different camera would show up as something else.

```
pi@rpikit01:~ $ lsusb
Bus 001 Device 050: ID 1908:2311 GEMBIRD
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@rpikit01:~ $
```

Then, to check what resolutions are supported by the camera if you are using a USB camera of your own, use the following command:

```
v4l2-ctl --list-formats-ext -d /dev/video0 | grep Size
```

….. and you would see the following output for the GEMBIRD device:

```
pi@rpikit01:~ $ v4l2-ctl --list-formats-ext -d /dev/video0 | grep Size
                Size: Discrete 640x480
                Size: Discrete 352x288
                Size: Discrete 320x240
                Size: Discrete 176x144
                Size: Discrete 160x120
pi@rpikit01:~ $
```

Image streaming: software and setup

These image streaming methods use web software called 'Flask', which has been discussed in the earlier electronic basics section, with details of the various software elements used for streaming also provided in *Appendix D: Flask web server details*. This detail will allow more advanced Python users to use the supplied code as a base for any further developments they may want to do. The overall system approach is that the streamed video consists of a series of individual images taken by the camera, that in real time are first 'manipulated' to add text at the bottom of each image before their rapid 'presentation' to the screen.

Each image does however have to be stored, albeit on a temporary basis. To do this instead of using the Raspberry Pi's normal file system, which is typically a SD card that could rapidly deteriorate with the many thousands of read/write operations involved, a so-called 'ram drive' is used instead.

This technique creates a 'virtual drive' in computer memory that can be written to, and read from, as if it were a normal file system but is not only much faster but will not 'wear out' like a SD card. Details of how the 'ram drive' must be set up on a one-time basis by a user with root privileges are also provided in *Appendix E: Setting up a ram drive*.

As discussed earlier, a quirk of the Flask system is that a browser that accesses the web server can only use the conventional HTTP port 80 if the main software is run by a user with root privileges. Two slightly different image streaming software programs are therefore made available for root and non-root users and for both cases it is recommended that the code is simply run from a 'Terminal' window instead of using the Thonny IDE.

Image streaming: user with root privileges

With your Raspberry Pi in 'Desktop' mode and connected to your local network, use a 'Terminal' window to run the ***image_streaming_app_root_annotate.py*** program that the software download process would have placed in the subdirectory:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/

The sequence of commands to move to this directory and then run the code are as follows:

```
$ cd /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/
    $ sudo python3 image_streaming_app_root_annotate.py
```

The screen shot below shows the Terminal window display when running on a Raspberry Pi with a host name of 'pikitP48GB':

Now either open the browser on your Raspberry Pi or use a browser on another computer that is also connected to your local network and go to the URL http://yourpihostname, where *yourpihostname* is whatever host name you have given your Raspberry Pi. Your local router should then connect the browser to your Pi and will show the screen shot below.
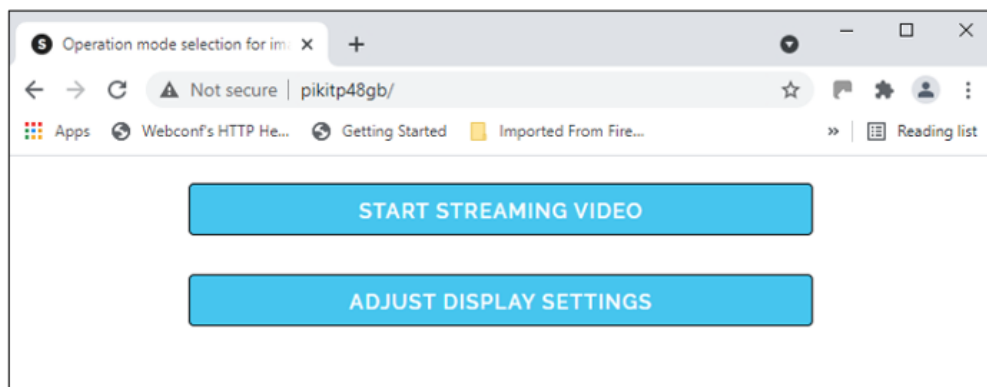
Now you can click the top blue button to start showing a streamed image from the USB camera using all the default camera settings or click the second 'button' to use a series of screens to adjust the camera settings.

Image streaming: user without root privileges

The Terminal window commands for a non-root privilege user are as below and as also shown in the screen shot below, i.e., you no longer have to use sudo:

```
$ cd /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/
    $ python3 image_streaming_app_user_annotate.py
```

The other significant difference is that the resultant streamed images are now only shown in http port 8000, so the required URL to be typed into your browser is http://yourpihostname:8000, where *yourpihostname* is whatever host name you have given your Raspberry Pi, and the screen shot below shows the browser output for this non-root user case.
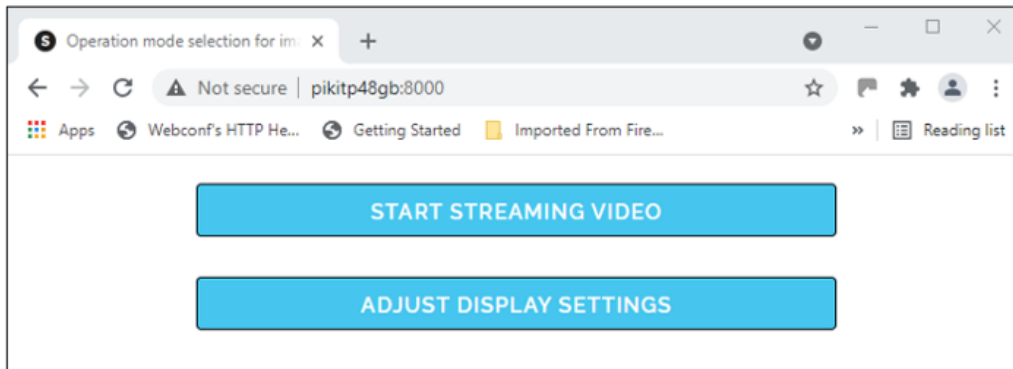
For either usage case (root or non-root), the screen shot below shows what the browser will display when the 'START STREAMING VIDEO' option is selected - where the grey rectangular area will be showing live streamed images from the camera.
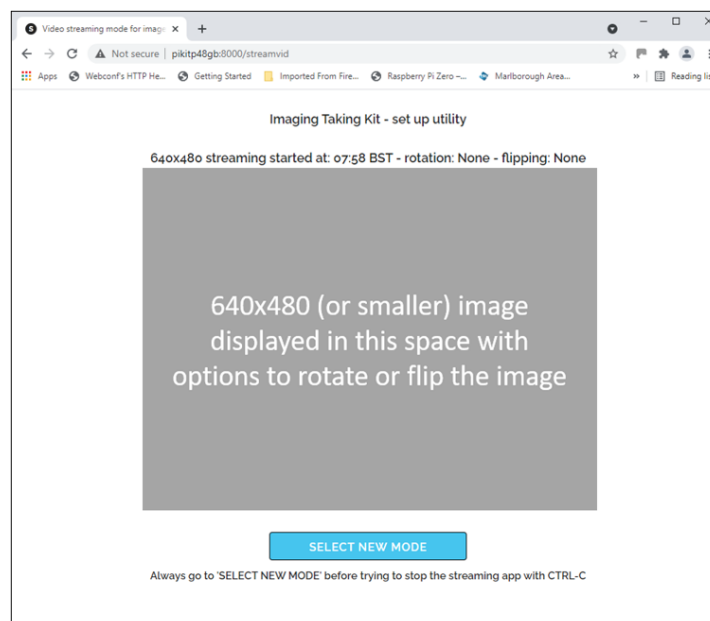
The streamed display should be in focus – if not, for the GEMBIRD camera adjust the front bezel on the camera – it should usually be almost fully clockwise.

Then, depending upon how you have mounted your camera the image may need to be rotated so that it is oriented correctly in the display by either physically remounting the camera the right way round or changing the orientation 'in software'. To do this use the "SELECT NEW MODE" button to return to the main screen, and then click the "ADJUST DISPLAY SETTINGS" to go to this screen where you can set the rotation to 90°, 180° or 270°.

As you will see there are also some 'flip' options on this screen that you might want to use in some circumstances – as well as options to set different camera resolution sizes. The available camera resolutions that can be shown can also be 'set' for the specific camera being used from another administration screen.

Once set up as you require, you should be seeing a reasonably fluid 'real time' streamed display with both the CPU temperature and the current time superimposed on the image at the bottom left and bottom right of the image, although there will be some 'lag' between what is shown and the real world.

You should note however that changing the display settings with this streaming method does not 'fix' the camera in this mode, it just adjusts what is displayed whilst streaming, so that you know what adjustments will be needed when you use the same camera set up with other image taking methods described next most of which could be done using the Thonny IDE to run individual programs or you can continue to use direct commands in a Terminal window.

Button taking image

The circuit diagram shown on the right shows how this basic image taking method uses the button on the PCB, and by examining the **button_take_image.py** Python code you will see that it uses a software command called *fswebcam* to initiate the taking of a single image by the USB camera; the library for this command will have been installed for you with the rest of the software as part of the software download process. The software also has comments to show how to change the camera resolution and orientation.
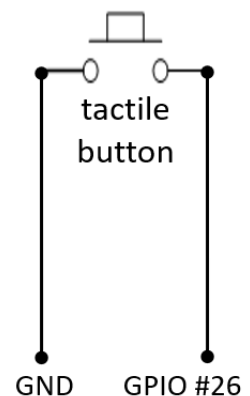
To run this code, use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_take_image.py
```

Once started, you will see a prompt to enter a subfolder name where the images will be stored, and then a second prompt to click the button to take an image or to type CTRL-C to stop the program.

When the button is clicked, the screen will then display a completion notification telling you that the image has been stored at:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/your_input_folder_name/

.. with a file name like *single_image_2021-05-03_10.03.25.jpg* where the numbers in the file name are the date (YYYY-MM-DD) and time (HH.MM.SS) that the image was taken.

You then have an option to briefly view the image on screen before the program repeats the '*button click prompt/take image*' cycle indefinitely until you type CTRL-C to stop it.

At any time, you can also use the Pi's File Manager and Image Viewer utilities to display on your screen any of the images taken.

Button taking image with LED indicator

The circuit diagram shown on the right shows how this image taking method uses the button and the LEDs on the PCB, plus as shown in the **button_led_take_image.py** Python code it also uses *fswebcam* to take an image with the USB camera, with comments showing how to change the camera resolution and orientation.

Whilst the program is starting up, the Red LED is initially 'on', and then once fully started, the Green LED turns on and you will see a prompt to enter a subfolder name where the images will be stored, and then a second prompt to click the button to take an image or to type CTRL-C to stop the program.

When the button is clicked, whilst the image is being taken, the Amber LED is on. When image taking is complete, the Green LED goes back on, and the screen will also display a completion notification telling you that the image has been taken and has been stored at:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/your_input_folder_name/

.. with a file name like *single_image_2021-05-03_10.03.25.jpg* where the numbers in the file name are the date (YYYY-MM-DD) and time (HH.MM.SS) that the image was taken.

You then have an option to briefly view the image on screen before the program repeats the button click prompt/take image cycle indefinitely until you type CTRL-C to stop it.

At any time, you can also use the Pi's File Manager and Image Viewer utilities to display on your screen any of the images taken.

To run this code, you can also use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_led_take_image.py
```

Timer taking image with LED and buzzer indicators

The circuit diagram shown on the right shows how this image taking method uses the button, the LEDs and the buzzer on the PCB, plus as shown in the **button_timer_take_image.py** Python code it also uses *fswebcam* to take an image with the USB camera with the same comments showing how to change the camera resolution and orientation.

Whilst the program is starting up, as with the previous method, the Red LED is initially 'on', then once the program has fully started the Green LED turns on and you are prompted to enter a subfolder name where the images will be stored. You will then see a second prompt to enter a timer duration in seconds (which needs to be more than 5 seconds). Having entered a suitable value, you are then prompted to click the button to take an image or to type CTRL-C to stop the program.
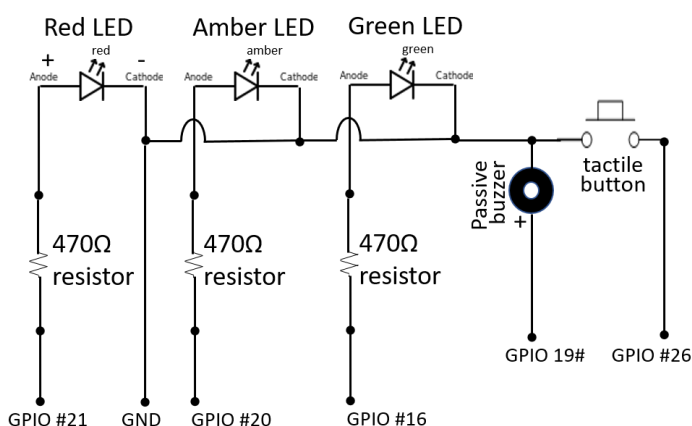
When the button is clicked, the timer starts to count down and the Amber LED flashes on and off, plus for the last 3 seconds of the timer duration the buzzer also beeps; with the Amber LED then steadily on, the image is taken.

On completion of the image taking, the Green LED returns to 'on' and the screen will also display a completion notification telling you that the image has been taken and has been stored at:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/your_input_folder_name/

.. with a file name like *single_image_2021-05-03_10.03.25.jpg* where the numbers in the file name are the date (YYYY-MM-DD) and time (HH.MM.SS) that the image was taken.

You then have an option to briefly view the image on screen before the program repeats the button click prompt/take image cycle indefinitely until you type CTRL-C to stop it.

As always, at any time, you can also use the Pi's File Manager and Image Viewer utilities to display on your screen any of the images taken.

To run this code, you can also use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_timer_take_image.py
```

Video recording

Video clips can be recorded using a set of three Python programs that allow a short video to be captured where the button, the LEDs and buzzer are all used in similar ways to the previous single image capture methods.

For all three of the video capture methods there is an option to view the video on screen immediately after capture before the program repeats the button click prompt/capture video cycle indefinitely until CTRL-C is used to stop it.

The video play back in these Python programs uses 'OMXPlayer', a command line media player that is hardware accelerated on the Raspberry Pi and can therefore efficiently manage what is ordinarily an intensive processing task.

However available storage space on your SD card is something you need to consider for all the image taking methods but especially so for video capture.

Button taking video clip

The Python program ***button_take_video.py*** allows a video clip to be captured when the tactile button is pressed in a similar manner to how ***button_take_image.py*** is used to capture a single image.

To run this code, you can use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_take_video.py
```

Button taking video clip with LED indicator

The Python program **button_led_take_video.py** allows a video clip to be captured using the LEDs to indicate the video taking cycle when the tactile button is pressed in a similar manner to how **button_led_take_image.py** is used to capture a single image.

To run this code, you can use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_led_take_video.py
```

Timer taking video clip with LED and buzzer indicators

The Python program **button_timer_take_video.py** allows a video clip to be captured after a timer countdown when the tactile button is pressed, with the LEDs and the buzzer indicating the cycle progress in a similar manner to how **button_timer_take_image.py** is used to capture a single image.

To run this code, you can use the Thonny IDE or alternatively type or copy/paste the full command shown at the top of the listing into a Terminal window, where a *venv* should have been automatically 'activated', i.e.

```
$ python3 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/button_timer_take_video.py
```

Time lapse video

Time lapse photography is a fun way to present, in a video format, something that changes quite slowly over a period, but is shown 'speeded up' so that, for example, a duration of several days, or much longer, is shown in a few minutes. Some common examples of the uses of time lapse are capturing how plants grow, which can be shown in a fun way with a "cress egg heads" project, or the activity might be during a construction project, or just a 'day in the life' of a busy location.

A normal full speed video would typically be showing more than 30 frames per second (fps), meaning more than 30 individual images are captured and then played back every second to show fluid/jitter-free motion. In comparison, a time lapse video might consist of individual images of the scene being captured perhaps every 5 minutes and then played back at (say) 10fps. So, if for example, individual images of a scene were captured every 5 minutes for as long as 10 days this would result in a total of 2880 images – which would result in a video clip of less than 5 minutes if played back at 10fps.

Production of a time lapse video is therefore composed of the following two steps:

- capturing individual digital images of a scene, on a defined periodic basis, and then

- 'stitching' the individual images together into a digital video file format that can be 'played'

This image taking method uses the USB camera, mounted in a fixed position, and connected to the Raspberry Pi. Various software tools and methods are then used to carry out the two steps described above that are now discussed in more detail.

Individual image capture

To capture an image periodically you could run a Python program, similar to those used in the previous methods, on a continuous basis and wait for defined intervals between image taking cycles. A more reliable and consistent method however is to run a program that takes a single image and then stops – and then to set this program to run at periodic intervals for which there is a standard Linux process called 'cron' which will keep going even if the Raspberry Pi is rebooted.

To start your time lapse image capture process, you should simply connect the USB camera to the Raspberry Pi and power it up. Then set the camera up in a suitable way to 'frame' and capture the required scene. You might want to use video streaming to show a continuous initial view in a browser so that you get the camera positioned correctly and then make sure it is firmly secured since you will need to ensure that neither the camera nor the overall scene context, e.g., the flowerpot or egg cup holding a cress egg head, does not move, perhaps for several hours, or even for days, or weeks!

Two versions of the software used to take single individual images are provided in the downloaded set. The first, called *timelapse_cron_take_image.py,* is the simpler version and this just takes an image, whereas the second version called *timelapse_cron_take_annotated_image.py* adds some annotation text with a time stamp to each image.

Using the Pi's File Manager and Text Editor utility, you should look through the Python code for each of these two versions as they have additional comments throughout to explain what each section of the code does and how the annotation text and other parameters can be edited.

Next, decide whether to use the 'plain' or annotated image software version and run the program in a Terminal window to produce a single <u>test</u> image, by using the command that is shown towards the top of the listing, i.e., either

```
python3
/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_cron_take_image.py
```
or
```
python3
/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_cron_take_annotated_image.py
```

There are no visual indications that an image has been taken other than the screen displaying a completion notification telling you that the image has been taken and has been stored at:

/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_image_folder/

.. with a file name like *single_image_2017-07-03_10.03.25.jpg* where the numbers in the file name are the date (YYYY-MM-DD) and time (HH.MM.SS) that the image was taken.

You can now use the Pi's File Manager and Image Viewer utilities to display the image on your screen to check that the individual captured image is OK.

Assuming you are happy with the camera location/set up, and the test image, we can now use 'cron' to schedule the running of this Python code on a regular basis.

Cron is a tool for configuring scheduled tasks on Unix systems like the Raspberry Pi OS (more info can be found **here**). It is used to schedule commands or scripts like our Python code to run periodically and at fixed intervals.

To add our time lapse routine to any existing scheduled items that may already be on your Pi, we can edit the root user's '*time schedule*' (or cron table shortened to crontab) by typing this command into an opened Terminal window:

crontab -e

This will open up the editor for the schedule but if it has never been edited before you will be asked to select a specific editor (nano is recommended). How to use an editor like nano, is not covered in this document, but there are plenty of online resources to help you with this.

To add a time lapse task just add two lines of text like those shown below (even though a very small font has been used to try to show that these are two continuous single lines, the second line has rolled over into a third line). It should also be noted that because this command will be running in the background on the Pi it is necessary to explicitly set the full path to the Virtual Environment for the command and the full path to the code.

The first line is just some comments (signified by the line starting with a #) to document the second line:

script to run every 5 minutes to take an image with a USB camera (adjust time as necessary)

*/5 * * * * /home/YOURUSERNAME/my_virtual_env/bin/python3
/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_cron_take_image.py >> /dev/null 2>> /dev/null

Or an alternative pair of lines is:

script to run every 5 minutes to take an image between 6:00am and 9:55pm (adjust time as necessary)

*/5 6-21 * * * /home/YOURUSERNAME/my_virtual_env/bin/python3
/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_cron_take_image.py >> /dev/null 2>> /dev/null

For each of these options, the second line is the scheduled task instruction using the format described below: where */5 i.e., the 1st element of the five-element setting for the schedule, means every 5 minutes in the 0-59 minute range; and 6-21 or * sets the hours the command is run i.e., from 6am to 9pm or * for all hours.

The general format for a cron line is therefore:

```
* * * * *   command to execute
│ │ │ │ │
│ │ │ │ └───── day of week (0 - 7) (0 to 6 are Sunday to Saturday,
│ │ │ │        or use names; 7 is Sunday, the same as 0)
│ │ │ └─────── month (1 - 12)
│ │ └───────── day of month (1 - 31)
│ └─────────── hour (0 - 23)
└───────────── min (0 - 59)
```

So, for our example we have the following:

*/5 * * * * means every 5 minutes, in every hour, in every day of the month, in every month, in every day of the week. Please note that if for example you used */9 instead of */5 this would create a schedule that ran on the 0, 9th, 18th, 27th, 36th, 45th, and 54th minutes of the hour, and then this cycle repeats – so the last 'gap' is only 6 minutes. Therefore, to have a uniform gap between all the executed commands you do need to choose a number of minutes that can be multiplied up to 60 exactly.

*/5 6-21 *** then means every 5 minutes if the hour is anywhere from 6 to 21 i.e., 6:00am through to 9:00pm – but as */5 has also been set the last time will be 9:55pm.

The next part of the second line, the command to execute a file part, is an extension of the command used before when testing the code, but now with added in the full paths to the *venv*, i.e. first of all for the python3 command:

/home/YOURUSERNAME/my_virtual_env/bin/python3

- and then the full file path for the code to be executed, i.e.:

/home/YOURUSERNAME/my_virtual_env/bin/starter_maker_kit1/RPi_code/image_taking/timelapse_cron_take_image.py

Then after this command part there is also:

 ">> /dev/null 2>> /dev/null"

This is added at the end because the scheduled tasks will be run in the background i.e., we will not see any output that the code might produce or any errors that might be generated. In fact, the Raspberry Pi does not even need to be connected to a screen with a keyboard/mouse attached, it could be simply plugged in and powered up, which is known as running in "headless server" mode. In this context, this final additional text on the second line, tells the system to just 'dump' the program outputs and any system error outputs. This then avoids these outputs being directed to various standard system log files which could get filled up unnecessarily.

Finally, please note that time lapse files will start being produced immediately after the cron schedule is updated and will continue indefinitely until you update the schedule again to stop the scheduled task. To stop the task, just edit the crontab again and the simplest option is to just add a # to the front of the second line to make it a comment – this will then let you easily re-instate it at any time without having to retype lots of text.

You do therefore need to decide how long you want to capture time lapse images for, to make sure you stop it at the appropriate time, but you also need to make sure that your Raspberry Pi has enough spare storage capacity to store what might be many thousands of reasonably sized .jpg images. Use of the SD card should obviously be avoided, using perhaps a USB storage medium of some sort, updating the code to use this accordingly.

Creating a video from individual images

You now have a series of time lapse images stored in the Raspberry Pi at the */home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_image_folder/* if the SD card has been used! These files will also have an explicit date/timestamp built into their file names.

You may need to check some of the images at the beginning and end of the series in case these need to be deleted if they have captured something incorrectly as part of the initial start/setup or stopping of the time lapse process.

To 'stitch' together the required series of images into a video we can now use the *ffmpeg* system utility which will have been installed, if necessary, as part of the download script. This utility has a set of usage parameters that are exactly for this purpose, but unfortunately the files that *ffmpeg* can stitch together need to be in time sequence order with file names that include a simple sequential number i.e., 0000, 0001, 0002, etc.

To achieve this, we now need to run a small custom utility program that is provided with the downloaded files in a Terminal window with the following command:

>   *python3 sort_number_symlink_files.py*

This custom utility will look at the folder where all your individual time lapse images have been stored and first of all make sure they are sorted in chronological order. Then, one file at a time, it creates in a new folder, equivalent file names that are called symbolic links to each of the real files using a much-simplified file name of NNNN_image.jpg where NNNN is incremented from 0000 onwards to the last file number. In this way, we create a set of file names that satisfy the requirements of using *ffmpeg*, without copying all the existing files, which would double the required storage - nor do we simply rename the existing files, which would lose all their explicit timestamp information.

With this new set of symbolic link files, we can now use *ffmpeg* to do the 'stitching together', and as this is easily done just by executing a single command in a Terminal window, there is not any specific Python software provided to do this process.

An *ffmpeg* command has many different options but the most basic usage for 'stitching together' the file names we are using looks like:

ffmpeg -i '/path_to_symlinks_folder/%04d_image.jpg' /path _to_output_video/yourvideoname.mp4

The small font used above is just to show that this is one contiguous set of text, but let us break the command into its different sections:

-i *is a control parameter that indicates the input stream is defined next*

'/path_to_symlinks_folder/%04d_image.jpg' is the path location and file name 'structure' for the input stream of individual images. So, for this method the path will be something like:

 /home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/timelapse_symlink_folder

… which is where the set of symlinks will have been created with 4-digit sequential numbers i.e., 0000_image.jpg, 0001_image.jpg, etc.

/path_to_output_video/yourvideoname.mp4 *is where the output video should be created, and its file name defined*

Running this command with your paths, filenames, etc., will then create a video file that can be 'played'.

Stop motion video recording

This final described image taking method, stop motion video or stop motion animation, is a very creative and fun way of producing video content that technically is very similar to the time lapse method. In recent years, it has become a very popular cartoon production method e.g., Aardman Animations' Wallace & Grommet characters, and many others.

Production of a stop motion video is composed of the following two steps:

1. capturing a sequence of individual digital images of a scene, that is altered manually by a very small amount, and then
2. 'stitching' the individual images together into a digital video file format that can be 'played'

In this way, controlled motion of individual objects can be 'constructed' so that inanimate objects can be made to look as if they are moving around.

Individual image capture

To capture the individual images, either the "Button taking image" or the "Button taking image with LED indicator" method can be used to systematically collect a series of still images of a scene, storing the images in a dedicated folder for your stop motion video project.

Care needs to be taken to make sure that the camera is firmly secured and not moved when taking each image, that only the objects being 'animated' are moved a tiny amount for each image with the overall scene otherwise kept completely still, and that the lighting for each image also remains the same.

Creating a video from individual images

You now use exactly the same method to 'stitch' together the individual images, as was used for the time lapse video method, i.e., you:

Use the *sort_number_symlink_files.py* software to 'sort' your series of individual still images and to create a new folder of corresponding symbolic link files that have a simplified file name with an ascending number element starting at 0000

Then you use *ffmpeg* to do the 'stitching together' just by running the following command in a Terminal window.

ffmpeg -i '/path_to_symlinks_folder/%04d_image.jpg' /path _to_output_video/yourvideoname.mp4

To illustrate what can be quickly produced using this method, click the image shown on the right to link through to a web page that will show a short video clip – where you will see some of the 'errors' that are all too easily made! i.e., for this test video, the overall scene moved slightly for some individual shots and the lighting differed from shot to shot.

I'm sure you will be able to do much better!
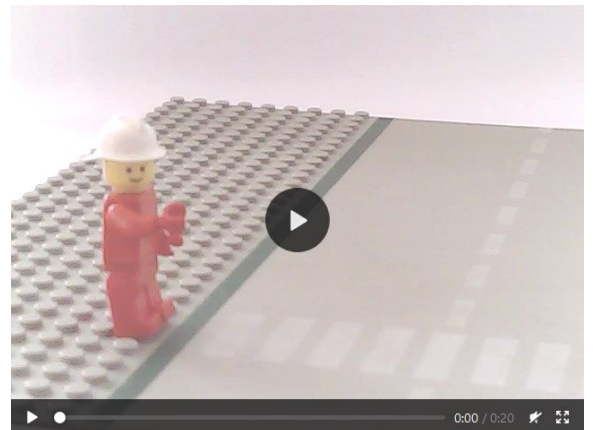
Appendix A: PCB development details

The **Starter Maker PCB** uses a v1.0 design of the PCB and the PCB design process has used the open source KiCAD software from which the images below show some details.
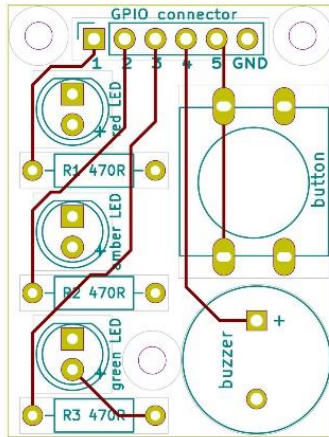
front of the PCB showing the component footprints and the copper power & signal interconnections
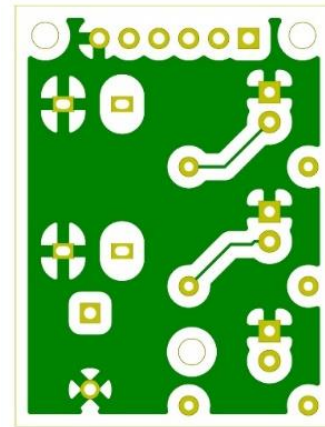
back of the PCB with the 'flooded' ground plane as well as some interconnections on this side

The key principles of the design are as follows:

- The components are laid out so that the overall size of the PCB is minimised.

- Space on the PCB is however allowed for fixing holes, through which 3D printed 'feet' can be affixed.

- The LEDs, buzzer and button are connected through to a 6 pin male connecter so that the assembled PCB can be connected, not only to a Raspberry Pi SBC, but also to a variety of microcontrollers.

The v1.0 design was finalised in April '21 with a detailed front view of a production unit shown on the right.

A .zip file of the 'Gerber' design files can be downloaded from **here** so that anyone can have small quantities of the PCB manufactured using one of the many low cost online suppliers that are now available.
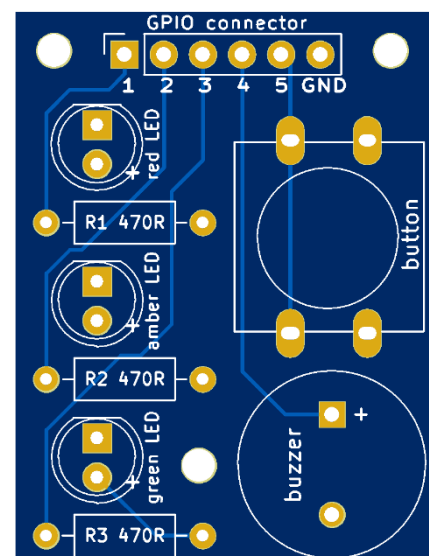
Appendix B: Maker PCB component usage details

This appendix provides more detail for the various components used in an assembled Starter Maker PCB that are to be soldered into their assigned locations on the printed circuit board (PCB).

Tactile buttons

A tactile push button is a switch that makes momentary contact when it is pressed and may also make a 'tactile' click.

The buttons used in the **Starter Maker PCB** are 12mm wide with a coloured push on cap (colours can vary) where the PCB footprint design allows each button to be gently pushed and held in their insert points the correct way round, prior to soldering the connections on the underside of the PCB.
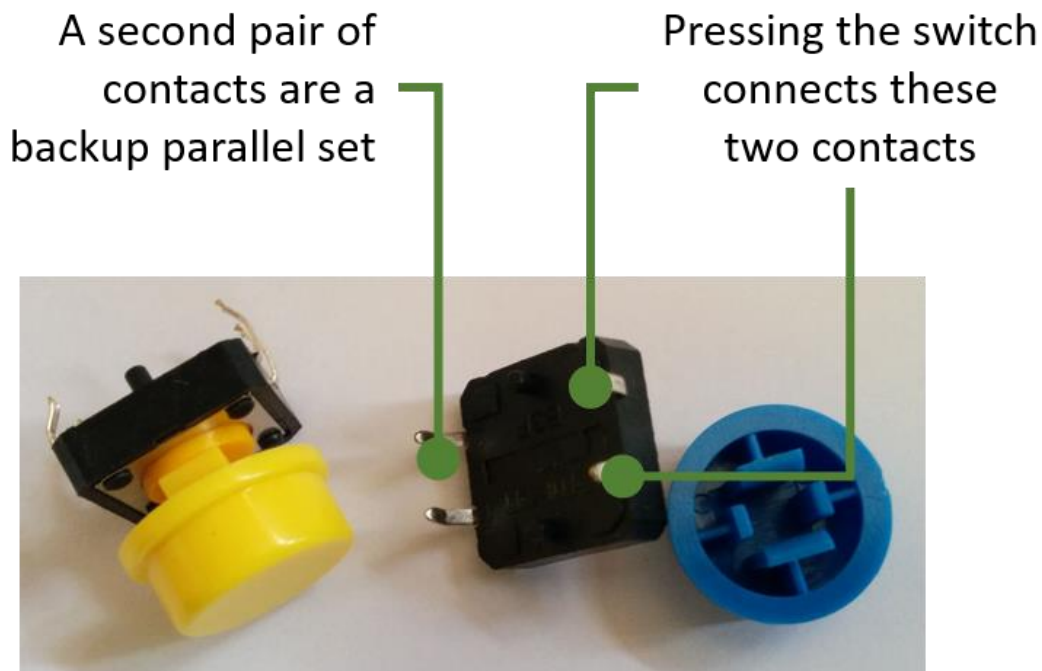
As shown above, it is the two contacts on the same side of the button that are connected when the button is pressed.

Light Emitting Diodes (LEDs)

Light Emitting Diodes, or LEDs, are simple, low energy consumption light sources that are powered by low voltage direct current (DC) and are available in a range of colours.

A very detailed description of their history and technical performance is available in **this Wikipedia link**.
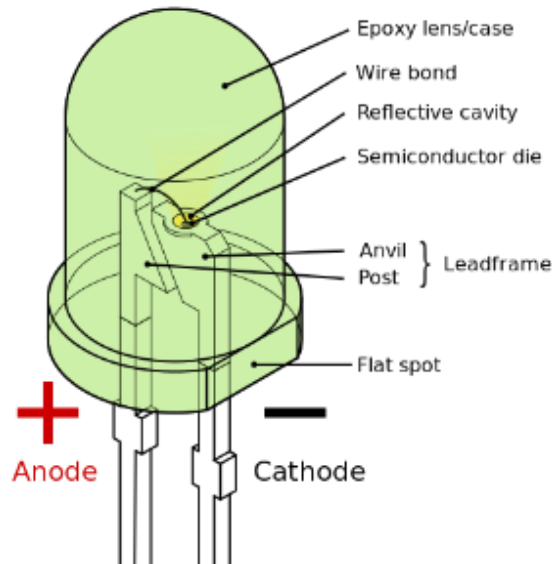
Used in the **Starter Maker PCB** assembly are one each of red, amber and green coloured LEDs that could be packaged in either 5mm or 3mm potted plastic lenses, as shown in the diagram on the left.

For these single colour LEDs, the anode or positive power lead is longer (NB the schematic to the left shows a flat spot that can also be used to denote the cathode side, but this may not always be present).

It is very important that the +'ve DC voltage is correctly applied to the longer anode connection by inserting the LED into the PCB holes so that the longer lead connects through to its associated resistor for each of the LEDs.

The insertion points for the long legs are each labelled on the PCB with a small + to ensure the correct orientation is used.

In operation, each LED type has a slightly different forward voltage drop (V_f) that should not be exceeded, otherwise its maximum current limit could be exceeded which would cause damage.

V_f for red and amber LEDS is typically 1.8V and for blue, green and white it is typically 3.3V.

To avoid over current damage, all the LEDs on the PCB are therefore put in series with 470Ω resistors, as shown right for the red LED example, with the resistor having been sized to achieve a safe level of the applied voltage to any of the LEDs.
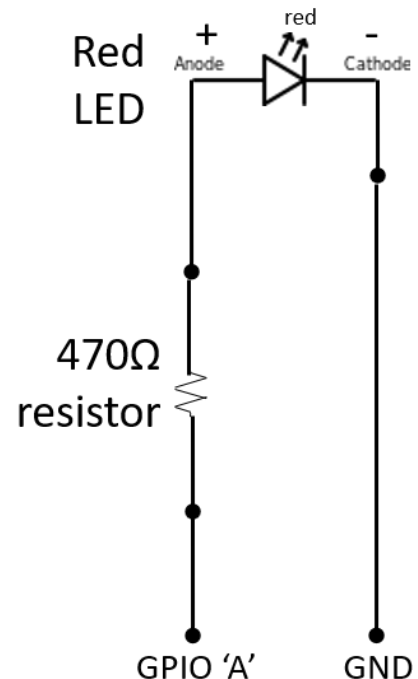
Resistors ───⎓⋀⋁⋀───
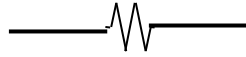
Resistors are small passive electrical devices used to provide electrical resistance measured in ohms (Ω).
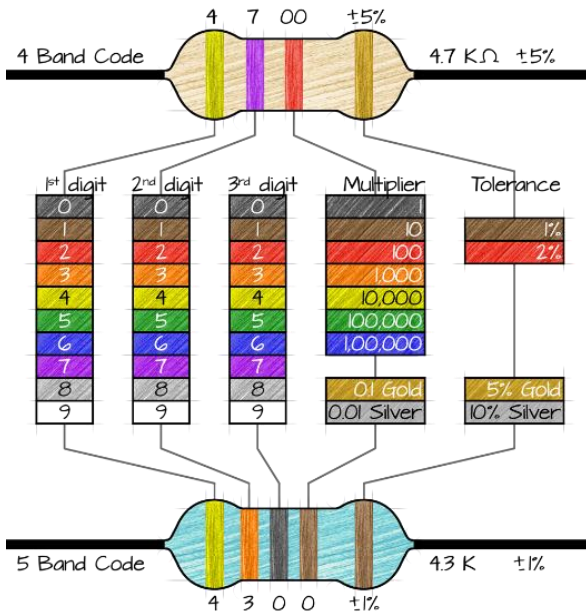
The resistors to be used with the **Starter Maker PCB** are so-called axial-lead devices, meaning the inbound and outbound leads to and from the device are in line with the component.

All axial resistors are marked with either 4 or 5 coloured bands, as shown in the diagram on the left, that indicate their resistance value in ohms (Ω), as well as their manufacturing tolerance as a percentage.

The resistors can be either carbon or metal film with 4 or 5 band coding as follows:

470Ω resistor coloured bands:

4 band: yellow–purple–brown–gold which means:
      4   7   x10   5%   i.e. 470Ω $\pm$5%

5 band: yellow–purple–black-black–brown which means:
      4   7   0   x1   1% i.e. 470Ω $\pm$1%

Buzzers

There are two types of small piezoelectric buzzer or beeper that are commonly available, usually described as either passive or active.

They are called piezoelectric buzzers as they use a small electromechanical element made from a material that exhibits the piezoelectric effect i.e., it will generate electricity when deformed, or as is useful in a buzzer, it will deform when electricity is applied to it.

An <u>active</u> piezo buzzer, as shown left, contains not only the piezo element but also some additional internal electronic components so that it only requires a DC voltage to be applied to it to make a sound, which will be at a single defined pitch. This type of device will usually have the underside 'potted' so that the internals are not visible, they may have a longer pin to designate the +'ve input as well as a + sign being stamped on the top, and as shown above are often shipped with a small sticky label to protect the buzzer's 'sound opening'.

The **Starter Maker PCB** however only uses a <u>passive</u> piezo buzzer type, as shown right. This type of device is simpler than the active type, not having any additional electronic components and it is not potted on the underside. It does however have the +'ve pin indicated on both the top and under sides.
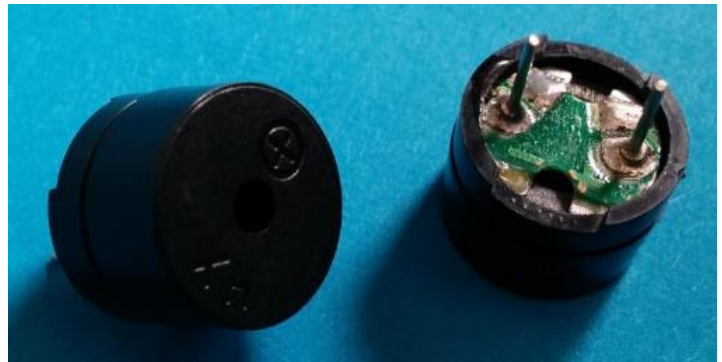
Without the additional electronics a passive buzzer requires an alternating voltage input to generate a sound, and whilst this does require more 'effort' in writing the software to achieve any sound, it means that sounds with a different pitch can be generated from the one device.

Appendix C1: Electronic basics software download

The **Starter Maker PCB** has been designed to connect to any model of Raspberry Pi SBC as well as several different types of microcontroller. The electronic 'basics' example software made available for each of the different 'controller' options is described in the following sections.

Raspberry Pi SBC usage

Whilst any model of Raspberry Pi single board computer (SBC) can be used with the PCB, if a new Pi is to be purchased then a Raspberry Pi 5 with at least 2GB of memory is recommended as well as a SD card that is ideally at least 32GB.

You should also be using the latest Bookworm version of the operating system – if you have an earlier operating system version, you should carry out an update or create/install a new SD card before downloading the software and documentation for this PCB.

For Scratch and Python coding, whilst some knowledge is useful this is not essential since the PCB provides an opportunity to explore and start to understand coding with both Scratch and Python.

With your Raspberry Pi started in 'Desktop' mode and connected to the Internet, this document, plus additional support documents and all the software for the various electronic 'basics' projects can be downloaded to the Raspberry Pi by executing the following commands in an opened 'Terminal' window.

N.B. the $ sign in the command lines below signifies the prompt character in your terminal application, which you do not need to type.

First run the following command to download an initial control script where you substitute your username for YOURUSERNAME:

```
$ wget –O /home/YOURUSERNAME/starter_kit1_ePi.sh https://onlinedevices.org.uk/dl1428
```

(take great care to type this correctly and if you get an error then recheck what you have typed note: -O above is an upper case letter O and the last set of characters are lower case DL1428)

The control script contains all the 'commands' to set up the required folders on the Raspberry Pi's file system, to download the documentation and Scratch + Python exploration scripts and to also install any required system functions/libraries.

You then run the following two commands to prepare and run the control script.

```
$ chmod +x starter_kit1_ePi.sh
$ ./ starter_kit1_ePi.sh
```

At the start of the 2nd command, your username is input and when it has run it will have downloaded all the files and stored them on your Raspberry Pi in a main folder and various subfolders starting at: */home/YOURUSERNAME/starter_maker_kit1/RPi_code*

This process can be repeated at any time, and it will download any updated or new material, but obviously if any customisations have already been carried out these should be stored in new file names/folders since repeat downloads will overwrite the original files.

Finally, please note that the ***starter_kit1_ePi_readme.txt*** file that is downloaded as part of the documentation is a version control file that lists all the Raspberry Pi electronic basics download material for the PCB. Please review this file once downloaded as it will indicate any additional documents/code that may have been added to the set of support materials.

Raspberry Pi Pico microcontroller usage

Both MicroPython code examples using the Thonny IDE, and C/C++ code examples using the Arduino IDE, are available for the Starter Maker PCB to allow code to be developed and to manage the code transfers to the Pico using a USB cable.

However, as both these IDEs can be installed on a range of different computers (Windows PC, Mac, Raspberry Pi etc.,) the two sets of example code are made available by .ZIP files that can be downloaded as follows:

Pico MicroPython electronic basics: **https://onlinedevices.org.uk/dl1421**

… where the last set of characters are lower case DL1421

Pico C/C++ electronic basics: **https://onlinedevices.org.uk/dl1427**

… where the last set of characters are lower case DL1427

Once downloaded you can 'extract' all the available files for use on the computer running either the Thonny or Arduino IDE and connected to your Raspberry Pi Pico microcontroller. The .ZIP files also contains this document and another explanatory software usage PDF.

ESP32 microcontroller usage

At present only C/C++ code examples have been developed for the Starter Maker PCB using the Arduino IDE which can be used to develop and manage the code transfers to the ESP32 using a USB cable.

However, as the Arduino IDE can be installed on a range of different computers (Windows PC, Mac, Raspberry Pi etc.,) the example code is made available by a .ZIP file that can be downloaded from the following URL:

ESP32 C/C++ electronic basics: **https://onlinedevices.org.uk/dl1422**

… where the last set of characters are lower case DL1422

Once downloaded you can 'extract' all the available files for use on the computer running the Arduino IDE and connected to your ESP32 microcontroller. The .ZIP file also contains this document and another explanatory software usage PDF.

Starter Maker PCB ESP8266 microcontroller usage

At present only C/C++ code examples have been developed for the Starter Maker PCB using the Arduino IDE which can be used to develop and manage the code transfers to the ESP8266 using a USB cable.

However, as the Arduino IDE can be installed on a range of different computers (Windows PC, Mac, Raspberry Pi etc.,) the example code is made available by a .ZIP file that can be downloaded from the following URL:

ESP8266 C/C++ electronic basics: **https://onlinedevices.org.uk/dl1423**

… where the last set of characters are lower case DL1423

Once downloaded you can 'extract' all the available files for use on the computer running the Arduino IDE and connected to your ESP8266 microcontroller. The .ZIP file also contains this document and another explanatory software usage PDF.

Appendix C2: Image Taking software download

The Starter Maker PCB can be used with a Raspberry Pi that has a USB camera connected to one of its USB ports.

Only Python software is made available for this Image Taking usage and whilst any model of Raspberry Pi single board computer (SBC) with 2x20 GPIO pins can in principle be used with the PCB, a Raspberry Pi 5 with at least 2GB or memory is recommended as well as a SD card that is ideally at least 32GB.

You should also be using the latest Bookworm version of the operating system – if you have an earlier operating system version, you should carry out an update or create/install a new SD card before downloading the software and documentation for this PCB.

With your Raspberry Pi started in 'Desktop' mode and connected to the Internet, this document, plus additional support documents and all the software for the various image taking methods can be downloaded to the Raspberry Pi by executing the following commands in an opened 'Terminal' window.

N.B. the $ sign in the command lines below signifies the prompt character in your terminal application, which you do not need to type.

First run the following command to download an initial control script where you substitute your username for YOURUSERNAME:

```
$ wget –O /home/YOURUSERNAME/starter_kit1_imgPi.sh https://onlinedevices.org.uk/dl1430
```

(take great care to type this correctly and if you get an error then recheck what you have typed note: -O above is an upper case letter O and the last set of characters are lower case DL1430)

The control script contains all the 'commands' to set up the required folders on the Raspberry Pi's file system, to download the documentation, the Python exploration scripts and to also install the various required system functions/libraries.

You then run the following two commands to prepare and run the control script.

```
$ chmod +x starter_kit1_imgPi.sh
$ ./ starter_kit1_imgPi.sh
```

At the start of the 2nd command, your username is input and when it has run it will have downloaded all the files and stored them on your Raspberry Pi in a main folder and various subfolders starting at: */home/YOURUSERNAME/starter_maker_kit1/RPi_code*

This overall process can be repeated at any time, and it will then download any updated or new material, but obviously if any customisations have already been carried out these should be stored under separate file names/folders since repeat downloads will overwrite the original files.

Finally, please note that the ***starter_kit1_imgPi_readme.txt*** file that is downloaded as part of the documentation is a version control file that lists all the Raspberry Pi image taking download material. Please review this file once downloaded as it will indicate any additional documents/code that may have been added to the set of support materials.

Appendix D: Flask web server details

This detailed information is provided for users that are interested in further developing their Python coding skills with use of the Flask web server system. The information can inform the further development of web interfaces integrated with operations that are carried out on a Raspberry Pi.

Flask is a lightweight web server that integrates with Python to allow a browser based interface to be developed for a Python application.

With the Starter Maker PCB this capability is used in example code for:

- an Electronics demonstration to show how a LED can be controlled through a web interface, and

- an image streaming demonstration, to allow not only a real-time video to be streamed from the USB camera to a browser, but also for a number of camera parameters to be changed through a web interface.

This appendix provides some detail on the Flask set up, but for an in-depth understanding of Flask, the reader is referred to http://flask.pocoo.org/, and also recommended is the excellent book by Miguel Grinberg, "Flask Web Development" O'Reilly ISBN: 978-1-449-37262-0

How it all works

Flask provides a very flexible 'flow' arrangement that allows customised HTML web pages, created with HTML templates, to invoke specific sections of Python code, passing data to it - which in turn can 'call' other HTML web pages, passing data back to the HTML – as illustrated in the schematic below.

IP address/ domain name of server	browser

Main Python application
➢ starts webserver
➢ defines sets of code that are executed whenever a specific URL is requested (routes) and then renders a specific HTML template
    o route 1 → template A
    o route 2 → template B
    o etc.
	➢ HTML of template X with links that can go to various 'routes'
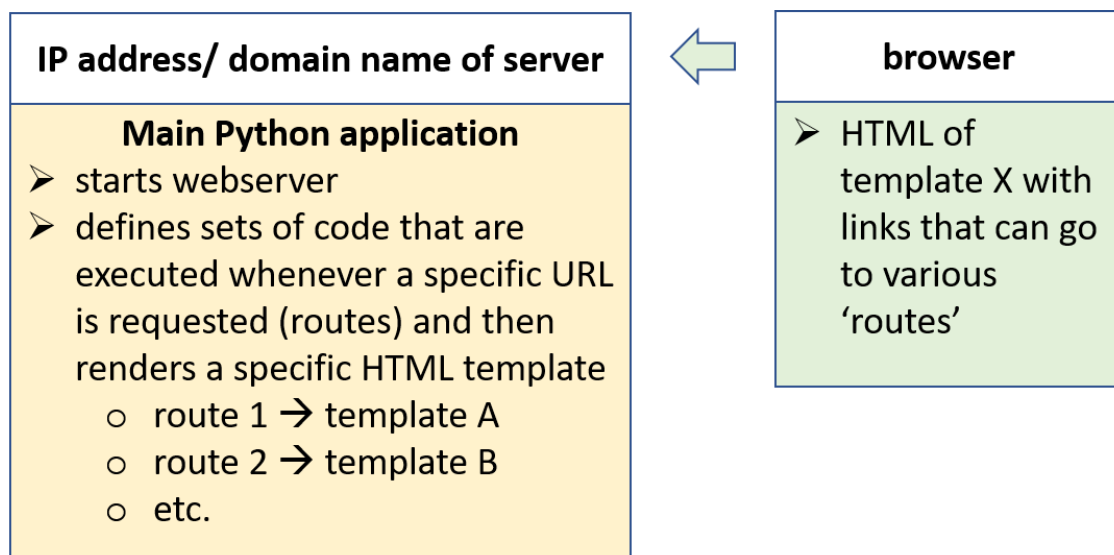
In the context of use on a Raspberry Pi, that is connected to a local network, any browser on another device, also connected to the same local network, can therefore 'connect' to the Raspberry Pi's IP address/host name. This then provides a web interface to the Python application that is running on the Raspberry Pi.

The 'Electronics' Flask software components

File name	Folder path	Brief description/ purpose
LED1_flash_web_root.py	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/starter_web_controller	Main Python application when code is run using 'sudo'
LED1_flash_web_user.py	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/starter_ebasics/starter_web_controller	Main Python application when code is run as YOURUSERNAME
electronics_layout.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a master layout for all the HTML templates
electronics_header_insert.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a common 'included' part of the layout for all the HTML templates
electronics_select_mode1.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a web interface for the main options
led1_setup_mode.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template used to set the usage configuration options
run_led1.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template used to operate the LED in different ways
normalize_advanced.css	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/css	CSS file for 'styling' the HTML
skeleton_advanced.css	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/css	CSS file for 'styling' the HTML
favicon.png	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/image	FAVICON image for the browser tab

LED1_flash_web_root.py

This main Python application code can be considered as three separate sections:

Descriptive text: lines 1-13.

Just some commentary about the application.

Main code – initial set up: lines 14 – 48.

Imports the required modules, initialises a number of parameters and starts Flask as a specific object, *make_image_app01.*

Main code – each of the 'routes': lines 49 – end.

Individual 'routes' that are executed depending upon what URL is requested and then 'returns' with another set of HTML to be rendered by the browser.

- ***@run_electronics_app01.route("/")*** – this is the route that is 'run' when the URL root for the IP address/hostname is requested and simply 'returns' *electronics_select_mode1.html*
- ***@run_electronics_app01.route("/<choice_mode>")***– this is the route that is 'run' when the following set of URLs are requested:
  - ***/ledlight*** – operates the LED according to various set parameters and then 'returns' to *run_led1.html*
  - ***/ledsetup*** – sets some parameters and then 'returns' *led1_setup_mode.html*
- ***@run_electronics_app01.route("/ledsetup/<setup_command>")*** - this is the route that is 'run' when the following set of URLs are requested:
  - ***/update_settings*** – 'receives' a set of parameters from the HTML template to adjust the LED operating parameters and then 'returns' to *led1_setup_mode.html*
  - ***/select_mode*** – goes back to *electronics_select_mode1.html*
- ***@run_electronics_app01.route("/ledlight/<led_command>")*** - this is the route that is 'run' when the following set of URLs are requested:
  - ***/led_cycle*** – flashes the LED on/off according to the duty cycle set and then returns to *run_led1.html*
  - ***/led_on*** – switches the LED on and returns to *run_led1.html*
  - ***/led_off*** – switches the LED off and returns to *run_led1.html*
  - ***/select_mode*** – goes back to *electronics_select_mode1.html*

LED1_flash_web_user.py

This has a similar structure to LED1_flash_web_root.py with some minor changes for the use of HTTP port 8000 since this app is not run with sudo.

electronics_layout.html

This master layout HTML template allows all the web page structure to be laid out in one file and then 'invoked' by all other templates which makes file maintenance more efficient by having all the standard structure information in one place.

electronics_header_insert.html

This 'partial' HTML template is used to define a 'header' section that is common to all the web pages, so it is inserted into the 'layout' template and as it is a separate file it can be edited 'once' to update this element of content for all the web pages.

electronics_select_mode1.html

This 'complete' HTML template provides the web page content to select either of the two main options, i.e., to operate the LEDs in a defined way or to configure the parameters that define the operational regimes.

led1_setup_mode.html

This 'complete' HTML template provides the web page content for a FORM that captures various parameters and returns to **/ledsetup/update_settings** the series of selected parameters as a GET, that the route then 'extracts' using the Flask 'request' function.

run_led1.html

This 'complete' HTML template provides the web page content that selects which operational mode is to be used and returns to a route option in the main 'app' according to the selected /ledlight/<led_command>, where Python code executes the LED on/off functions.

normalize_advanced.css

This is a conventional CSS file, or Cascading Style Sheet, that is used to format the content of the HTML templates and provide a consistent 'look' across all the content.

skeleton_advanced.css

This is another conventional CSS file, or Cascading Style Sheet, that is used to format the content of the HTML templates.

favicon.png

This is just an image used to provide a so-called 'favicon', or an icon that is used to represent the site and is displayed in the browser address bar.

The 'Image Taking' Flask software components

File name	Folder path	Brief description/ purpose
image_streaming_app_root_annotate.py	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller	Main Python application when code is run using 'sudo'
image_streaming_app_user_annotate.py	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller	Main Python application when code is run as YOURUSERNAME
image_camera_usb_opencv_annotate.py	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller	Custom Python class for the USB camera that also adds text annotations to each image
layout.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a master layout for all the HTML templates
header_insert.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a common 'included' part of the layout for all the HTML templates
cam_setup_mode.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a web interface for the camera set up
cam_options_setup.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a web interface for the options that the camera supports
select_mode.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a web interface for the main options
stream_video_mode.html	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/templates	HTML template providing a web interface for the streamed video
normalize_advanced.css	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/css	CSS file for 'styling' the HTML
skeleton_advanced.css	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/css	CSS file for 'styling' the HTML
favicon.png	/home/YOURUSERNAME/starter_maker_kit1/RPi_code/image_taking/image_taking_controller/static/image	FAVICON image for the browser tab

image_streaming_app_root_annotate.py

This main Python application code can be considered as four separate sections:

Descriptive text: lines 1-13.

Just some commentary about the application.

Default/global parameter values: lines 14-62

***Functions*:** lines 63-74.

Only one function, gen(camera, imgwidth), which generates the image stream from the camera using a function from the USB camera class.

Main code – initial set up: lines 75-107.

Imports the required modules, initialises a number of parameters and starts Flask as a specific object, *make_image_app01.*

Main code – each of the 'routes': lines 108 – end.

Individual 'routes' that are executed depending upon what URL is requested and then 'returns' with another set of HTML to be rendered by the browser.

- ***@make_image_app01.route("/")*** – this is the route that is 'run' when the URL root for the IP address/hostname is requested and simply 'returns' *select_mode.html*
- ***@make_image_app01.route('/video_feed/<int:pixwidth>/<int:pixheight> /<int:pbr>/<int:pcon>/<int:psat>/<int:phue>/<int:pgain>/<int:pexp>/ <int:pwbal>/<int:pfocus>/'*** – this special route generates the image stream when used within an IMG tag within the HTML of a web page.
- ***@make_image_app01.route("/<choice_mode>")*** – this is the route that is 'run' when the following set of URLs are requested:
  - ***/streamvid*** – sets some parameters and then 'returns' *stream_video_mode.html*
  - ***/camsetup*** – sets some parameters and then 'returns' *cam_setup_mode.html*
- ***@make_image_app01.route("/camsetup/<setup_command>")*** - this is the route that is 'run' when the following set of URLs are requested:
  - ***/camsetup/update_settings*** – 'receives' a set of parameters from the HTML template to adjust the camera settings and then 'returns' *stream_video_mode.html*
  - ***/camsetup/select*** – goes back to *select_mode.html*
- ***@make_image_app01.route("/camoptions/<setup_command>")*** - this is the route that is 'run' when the following set of URLs are requested:
  - ***/ camoptions /update_resolutions*** – 'receives' a set of parameters from the HTML template to adjust the available camera resolutions and then 'returns' *cam_options_setup.html*
  - ***/ camoptions /settings*** – 'receives' a set of parameters from the HTML template to adjust the more unusual camera settings and then 'returns' *cam_options_setup.html*
  - ***/ camoptions /select*** – goes back to *select_mode.html*
- ***@make_image_app01.route("/streamvid/<streamvid_command>")*** - this is the route that is 'run' when ***/streamvid/select*** is requested, which interrupts the streaming video and goes back to *select_mode.html*

image_camera_usb_opencv_annotate.py

This is a custom Python class that uses OpenCV to capture and manipulate the feed from the USB camera. It assumes the streamed images are in a ram drive mounted at /mnt/ramimage/ and:

- Adjusts the image width/height from the passed parameters, and

- Each image is also annotated with the CPU temperature and a time stamp using the '*imagemagick'* library '*convert*' function.

layout.html

This master layout HTML template allows all the web page structure to be laid out in one file and then 'invoked' by all other templates which makes file maintenance more efficient by having all the standard structure information in one place.

header_insert.html

This 'partial' HTML template is used to define a 'header' section that is common to all the web pages, so it is inserted into the 'layout' template and as it is a separate file it can be edited 'once' to update this element of content for all the web pages.

cam_setup_mode.html

This is an HTML template that is a FORM that returns to **/camsetup/update_settings** a series of selected parameters as a GET, that the route then 'extracts' using the Flask 'request' function to update the camera usage settings.

cam_options_setup.html

This is an HTML template with two FORMs that returns a series of selected parameters to **/camoptions/update_resolutions** or **/camoptions/update_settings** as a GET, that the route then 'extracts' using the Flask 'request' function to update the available camera resolutions or some of the more unusual camera settings.

select_mode.html

This is an HTML template that provides three optional links, where **/streamvid**, **/camsetup**, or **/camoptions** go to the **@make_image_app01.route("/<choice_mode>") route**.

stream_video_mode.html

This is an HTML template that uses the 'passed' parameters of the camera settings to display, by use of an IMG tag, the 'stream' of images generated by the **@make_image_app01.route('/video_feed/<int:pixwidth>/<int:pixheight>/'** route. Logic, that is allowed within the HTML template, is used to set the various parameters of the IMG tag depending upon the values of the 'passed' camera settings.

normalize_advanced.css

This is a conventional CSS file, or Cascading Style Sheet, that is used to format the content of the HTML templates and provide a consistent 'look' across all the content.

skeleton_advanced.css

This is another conventional CSS file, or Cascading Style Sheet, that is used to format the content of the HTML templates.

favicon.png

This is just an image used to provide a so-called 'favicon', or an icon that is used to represent the site and is displayed in the browser address bar.

Appendix E: Setting up a ram drive

Creating a 'ram drive' is a way of using the computer's memory 'as if' it was the type of media that would be used as a 'drive' to read/write files. This type of drive obviously does not create a permanent record of the data in a file since the data is lost whenever the computer is powered down. It is however a way to store/retrieve data quickly on a temporary basis and in usage situations where there is an extremely high/frequent read/write cycle it avoids premature aging/failure of a conventional drive medium such as a SD card used with a Raspberry Pi.

A ram drive is configured by editing the */etc/fstab* file (which needs to be done as sudo) which will then establish the drive whenever the computer is powered up.

As an example, for the image streaming method where a .jpg file is stored/retrieved very rapidly (i.e. many times/second) the following line should be added to the */etc/fstab* file:

```
none    /mnt/ramimage    ramfs    noauto,user,size=2M,mode=0770    0    0
```

where:

- */mnt/ramimage* is the mount point folder, where the *ramfs* filesystem will be mounted and this should have previously been created;

- the `noauto` option prevents the drive being mounted automatically (e.g., at system's boot up);

- the `user` makes this mountable by individual regular users (as well as root);

- the `size` sets this "ramdisk's" size, e.g., just 2M in this example since just a single image file is to be stored; and

- the `mode` is very important: by using the octal code 0770 only root and the user who mounted this filesystem will be able to read and write to the drive, not any other users.

It should also be noted that only one user can use the ram drive at any one time!

Appendix F: Raspberry Pi GPIO pin default settings

PLEASE NOTE: *this information is only relevant for older versions of the Raspberry Pi OS that can support Scratch 2 and will generally be irrelevant to most users today.*

On power up/reboot all the Raspberry Pi GPIO pins default to being INPUTs and:

- Pins 2-8 have their pull resistor set to PULL UP, with

- All the other pins having their pull resistors set to PULL DOWN.

When programming with Python, Scratch 1.4 and Scratch 3 the 'pull' status of a Raspberry Pi GPIO pin can be explicitly set by the code, and all the provided example code does this appropriately.

However, Scratch 2 on the Pi does not have an option to set the pull up/down status for a GPIO pin, so the tactile button should either be connected to one of the pins 2 through to 8, or the default status of pins can be changed by editing the /boot/config.txt file as shown below.

Since it is more convenient to group the Starter Maker PCB's 6 GPIO pin connections together, and to also connect them to a Raspberry Pi in an easily defined position, all the example code has assumed that GPIO pins 21, 20, 16, 19, 26 and GND are used, i.e., the 6 pins grouped together at the end of the connector furthest away from the power connection pins. Therefore, if these pins are used with Scratch 2 the following reconfiguration of the Raspberry Pi can be carried out on a one-time basis:

For older version of the Raspberry Pi OS use the following command in Terminal window to edit the config.txt file

sudo nano /boot/config.txt

… and add the following two lines to the file

 # change the pull on (input) pin 26
gpio=26=pu

END OF DOCUMENT