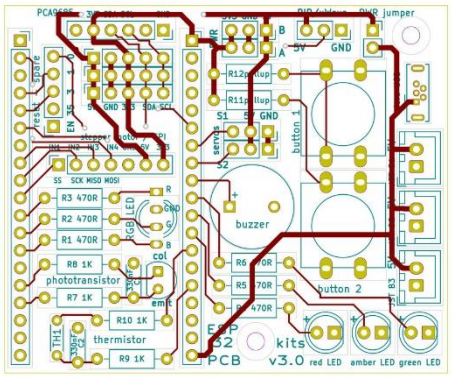
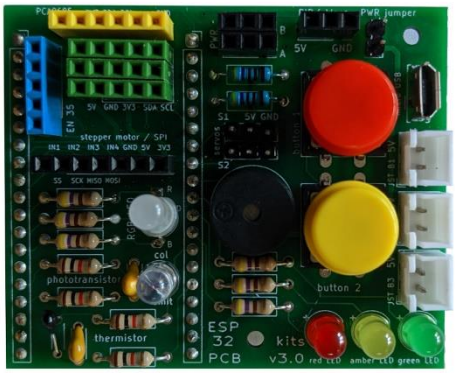
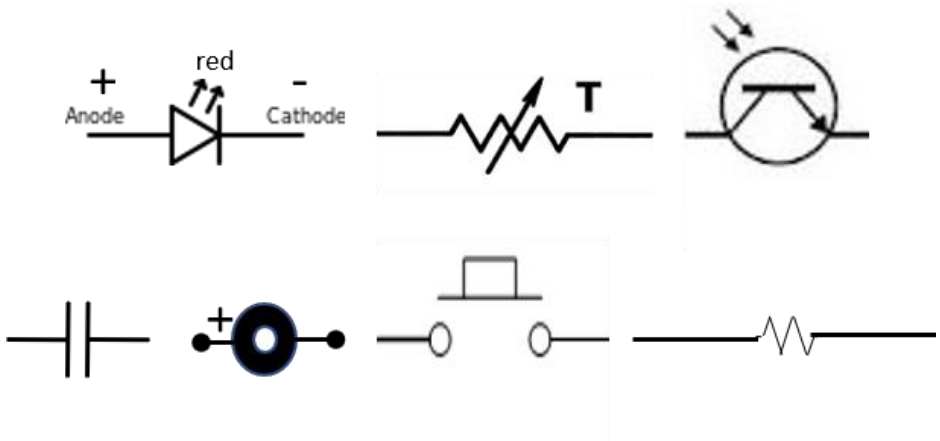
ESP32
Maker Kit







Usage Documentation

version 2.0

Table of contents

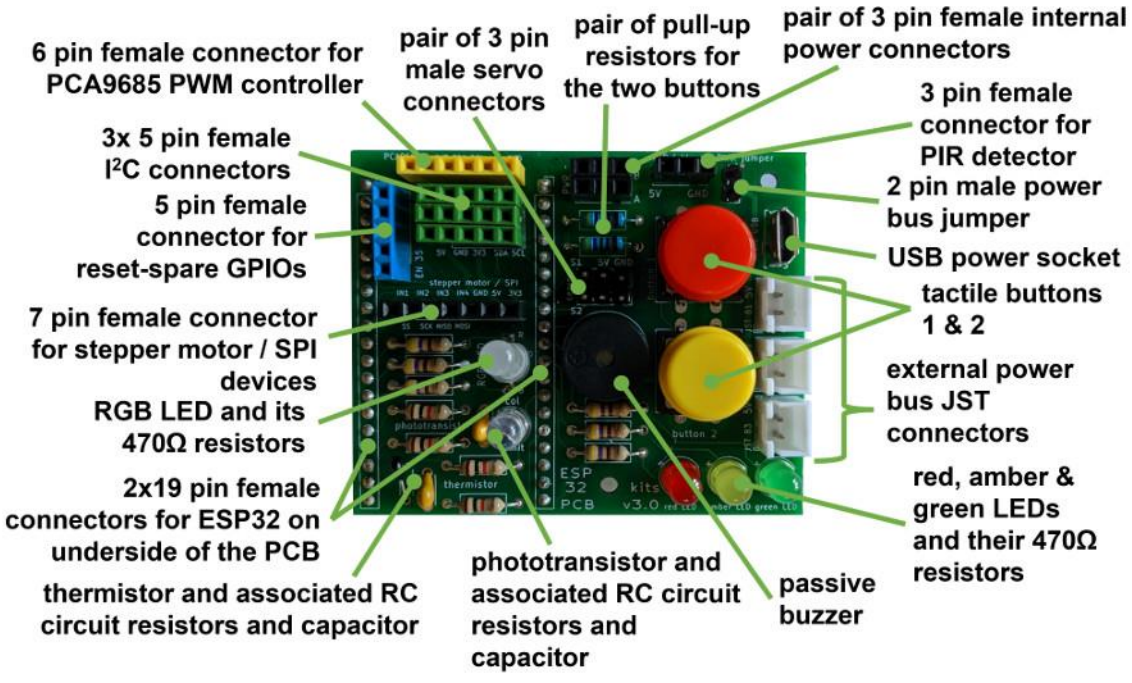
Introduction

The **ESP32 Maker Kit** enables digital makers to explore a wide range of 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 microcontrollers like the ESP32 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' 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 all new jobs will require digital skills to some degree.

The **ESP32 Maker Kit** provides a custom printed circuit board (PCB) onto which a set of electronic components and connectors can be soldered. This allows a digital maker to create a 'module' that connects to a 38 pin ESP32 module and enables a wide variety of projects and methods, all involving software control by this microcontroller.

The aim of this project development is to provide access to an expanding library of example code to allow a digital maker to explore and control many different components and devices. The project builds upon the development of a Raspberry Pi Maker Kit resulting in this microcontroller Maker Kit that allows a robust, permanent assembly to be built as shown in the schematic below which illustrates a populated printed circuit board that connects to an ESP32.



Building a more permanent assembly from a kit of components that includes 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.

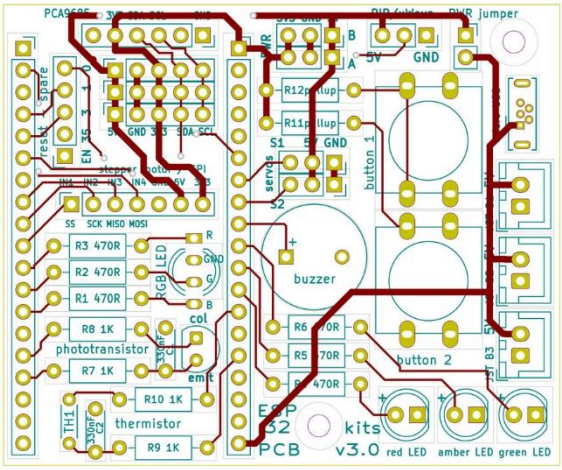
Example software has been developed to provide 'get going' capabilities that digital makers can use immediately and then build upon to extend their software development skills.

The Arduino IDE with its extensive range of libraries has been used to develop code for all the various example projects and can be used to manage the code transfers to the ESP32 using a USB cable.

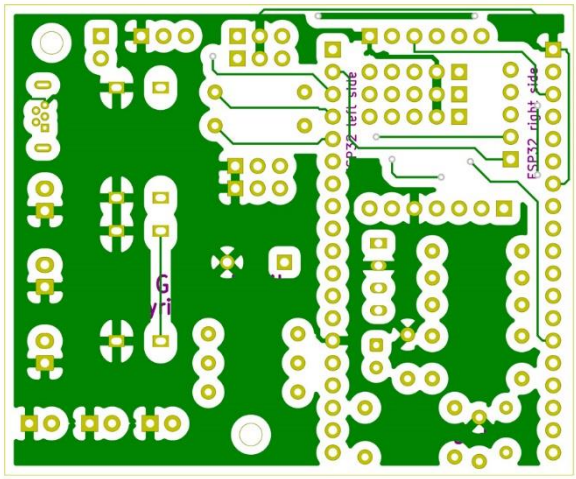
Kit development and contents

The **Maker Kit** now uses a v3.0 of the PCB which reflects several iterations of the design to implement refinements and functional extensions from the start of the project in March'21.

The PCB design process has used the opensource KiCAD software and the images below show some details of the v3.0 design generated by the software.



front of the PCB showing the component footprints and the copper power & signal interconnections

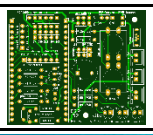
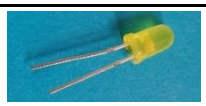
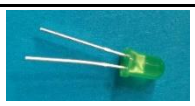
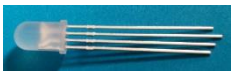
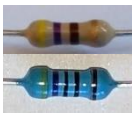
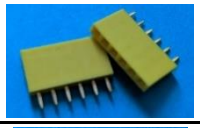
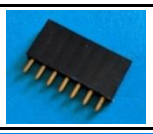
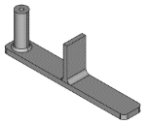
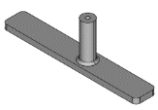


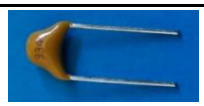
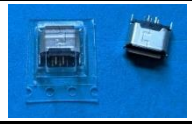
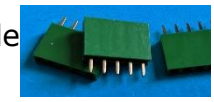
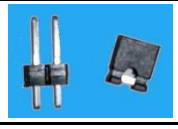
back of the PCB with the 'flooded' ground plane as well as some interconnections on this side

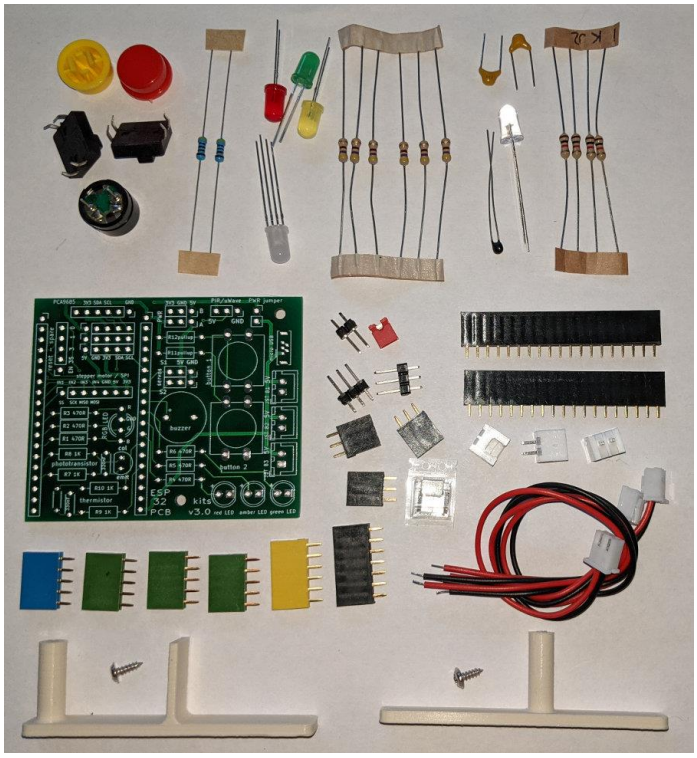
Kit component details

Each **Maker Kit** provides all the components in the quantities set out in the table below and shown all together in the following image. A complete assembled PCB can be produced by soldering all the components in place with the *PCB assembly details* section of this document providing further build description information.

Further technical information on each of the components is provided in *Appendix B: Maker Kit component details*.

description	image	quantity
Printed circuit board (v3.0)		1
Red LED		1
Amber LED		1
Green LED		1
RGB LED		1
470Ω resistors (carbon or metal film type may be supplied) bands: yellow-purple-brown or yellow-purple-black-black-brown		6
JST sockets + plugs with lead		3
1row x 6P female yellow header		1
1row x 4P female blue header		1
1row x 7P female black header		1
1row x 3P male black header		2
3D printed PCB 'stand-1' (colour may vary)+ 1x 6mm M2 self-tap screw		1 + 1
3D printed PCB 'stand-2' (colour may vary)+ 1x 6mm M2 self-tap screw		1 + 1

description	image	quantity
Passive buzzer		1
330nF capacitor		2
Thermistor 10kΩ NTC		1
Phototransistor		1
Tactile buttons (colour of caps may vary)		2
1kΩ resistors (carbon or metal film may be supplied) bands: brown-black-red or brown-black-black-brown-brown		4
USB type B micro socket		1
1row x 5P female green headers		3
1row x 3P female black headers		3
1row x 19P female black header		2
1row x 2P male black header + jumper		1 + 1
4.7kΩ resistors (metal film type) bands: yellow-purple-black-black-brown		2



Kit usage summary

The **ESP32 Maker Kit** has been designed to allow a very wide (and expanding!) range of projects and methods to be explored, as summarised below, with example/starter code provided that a digital maker can develop further themselves.

The aim of all the discussed projects and methods is, not only to show how a particular component/device operates, but to also show by using 'operational logic' set out in code how individual physical actions can be carried out under the control of the ESP32.

Individual sections of this document are provided later for each of these project/method areas summarised below, providing the detail for connecting various devices and using the example/starter software.

Electronic basics projects:

This set of electronic projects are the starting point for all the subsequent usage of the ESP32 Maker Kit. These show 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.

The Maker Kit supplies a set of components that when assembled on the PCB allow a range of 'basic' electronic projects to be explored.

Please note that this document does not provide support for the set up and running of an Arduino IDE which can run on a Windows PC, Mac or a Raspberry Pi, nor the general use of the C/C+ programming language used by the IDE.

Motor control:

Servo, stepper and drive motors can all be controlled using the Maker Kit by using additional commonly available motors and other components (that are not supplied with the Kit). Each of these motor types can be used in different ways to develop projects for 'things that move', which could be anything from a robot that runs around, to a complex, static mechatronic construction.

Each motor type can be controlled with code that runs on the ESP32 and example Arduino IDE developed code is provided to demonstrate a range of control methods for each motor type.

Display projects:

Various types of display (LCD, OLED, 7 segment LED, etc.) can be explored using the Maker Kit where an I^2C or a SPI interface might be required. No displays are provided with the Kit, but details of commonly available low-cost displays are given, and example Arduino IDE developed code for them is made available.

Sensor projects:

The ESP32 is an excellent platform for measuring a wide variety of device performance and environmental parameters with many different types of low cost sensors being readily available. No sensors are provided with the Kit, but it provides a means to easily connect many different types of sensor to an ESP32 with example Arduino IDE developed software provided to help a digital maker explore different ways of 'measuring the world'.

PCB assembly details

Assembling a complete ESP32 Maker Kit PCB is a significant soldering task for which some experience at soldering is needed.

If you are just a 'beginner' when it comes to soldering then the **Starter Maker Kit** will provide a less challenging project to practice soldering before tackling this more extensive **ESP32 Maker Kit**.

Some suggested assembly/soldering 'tips'

- As there are a large number of components to be added to the PCB, and this is a manual soldering process, a suggested component sequence is detailed below that can help avoid previously soldered components 'getting in the way' of the next component being soldered.

- A particular danger to avoid is part of the soldering iron inadvertently touching a previous component, particularly the plastic surround of a female header. This may cause sufficient damage that the header cannot be used and removing and replacing a previously soldered component like a header with multiple connection points is extremely difficult! It is therefore particularly important to avoid damaging the 1x19 GPIO headers once they are soldered onto the underside of the PCB as any damage to these could prevent the ESP32 module from inserting properly, making the entire PCB assembly unusable!
- 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, the activity is carried out in a well ventilated area and all appropriate safety precautions are undertaken. Whilst a more sophisticated thermostatically controlled soldering iron is always useful, a more basic uncontrolled iron is perfectly adequate for these activities.
- Generally, the use of additional solder paste/flux should be avoided, and whilst it might be 'safely' used on say a discrete component like a resistor, it should be avoided when soldering female headers in particular, since additional flux can cause excess solder to quickly 'wick' up into the header and block the subsequent 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 a 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 will take some time, so take an occasional break!

Suggested component assembly sequence

Below is a suggested sequence for soldering all of the components onto the PCB.

As previously mentioned this is an extensive soldering exercise for which some soldering experienced is needed – but the **Starter Kit Usage Documentation** provides additional tips and illustrations that may be found useful for soldering these components onto the ESP32 Maker Kit PCB.

Resistors x12

Soldering discrete components like resistors is perhaps the most straightforward of the assembly tasks since there are no problems if too much solder is applied since it can 'safely' wick along the wires and it is very visibly obvious if too little solder is applied.

All twelve of the supplied resistors, 6x 470Ω, 4x 1kΩ, and 2x 4.7kΩ are fitted in a similar way by inserting the resistor into the designated area on the front of the PCB –

clear labelling is provided on the PCB - and then folding the leads on the underside so that the resistor is held 'snugly' against the PCB front surface. As these are simple resistive devices it does not matter which way round the resistors are inserted into the PCB.

The soldering process is completed by placing the soldering iron tip firmly on the PCB's underside 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.

Capacitors x2

The two ceramic capacitors are installed in exactly the same way as the resistors, inserting them into their designated areas on the front of the PCB. Just like the resistors, it also does not matter which way round these simple ceramic capacitors are inserted into the PCB.

All the components on the front of the PCB between the 19P ESP32 headers in the following order:

Phototransistor and Thermistor

These discrete components can be inserted into the front of the PCB, initially held in place with Blu Tack if necessary and soldered in place. Please note that the long lead of the phototransistor is the emitter.

It should be noted that all these components have PCB connections that are spaced at 2.54/5.08mm so that, instead of directly inserting them into the PCB, a female header could be soldered in place (not supplied with the Kit) and the components connected on a long multi-wire leads to allow them to be used 'remotely'.

1 row x 6P female yellow header x1 and *1 row x 5P female green header x3*

This 'group' is the PCA9685 connector and the three I^2C connectors, which can all be Blu Tacked in place together before soldering the connections.

1 row x 7P female black header x1

This female header is for the SPI/motor control connections, and it can be individually Blu Tacked in place before soldering the connections.

1 row x 5P female blue header x1

This header is for the 'spare' GPIO, and it can also be individually Blu Tacked in place before soldering the connections.

RGB LED

This discrete component is inserted into the front of the PCB and can initially be held in place with Blu Tack if necessary before soldering in place.

It should be noted that this component also has PCB connections that are spaced at 2.54/5.08mm so that it could also be used 'remotely'.

2 off 1x 19 female GPIO headers

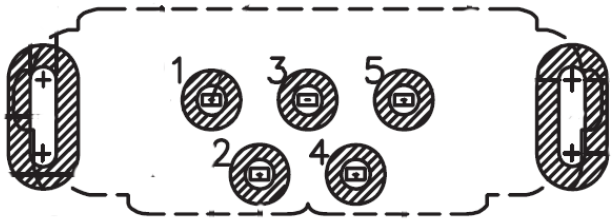
The greatest of care should be taken with the installation of the two 1x19 female headers, which are fitted to the underside of the PCB. It is essential that all 38 connections are electrically sound whilst no excess of solder is allowed to 'wick' up into the headers which would prevent the header from fully inserting on the ESP32's GPIO pins.

A simple suggestion to help with this installation is to use some Blu Tack to hold each header, one at a time, firmly in place and to first solder 2 or 3 well separated connections not too near the Blu Tack to initially secure the header in place. The Blu Tack can then be easily removed (as it will not have been heated up to any extent) and the rest of the connections completed.

For each connection the same soldering technique as used for a discrete component like a resistor is used i.e., the soldering iron tip is held firmly on the PCB opening and the header connection tip whilst gently touching the joint with the solder wire until it is seen to start to melt. Only allow the most minimum amount of solder to flow into the joint whilst ensuring that the header connector and the PCB opening are properly 'wetted' by the solder.

USB micro connector

This component can also be initially held in place on the front of the PCB using some Blu Tack and secured in place by soldering the two outside metal tabs (as shown right) into their 'slots' on the PCB.



With the unit now physically secured, the Blu Tack can be removed and just the two outside small connectors (1 and 5 as shown above) can be soldered. These are just very small openings, so the soldering iron tip just needs to be held onto each opening and a very small amount of solder used to 'wet' the connection.

JST female sockets x3

All three JST sockets can be inserted into the front of the PCB and held in place with Blu Tack whilst the socket pins are soldered into the PCB openings.

Tactile buttons x2

The two buttons are inserted into the front of the PCB: the four legs of each button may need to be straightened slightly so that they align with the four 'slot' openings on the PCB. Each button is firmly pushed into the PCB openings so that it 'clicks in' and sits evenly and directly on the PCB surface and is held in place. All four legs of each button are then 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.

Passive buzzer

Make sure this is inserted correctly with the + pin of the buzzer inserted into the PCB opening also marked with a +, then use some Blu Tack to hold the buzzer in place whilst the buzzer pins are soldered into the PCB openings.

More headers

Several additional female headers plus some male headers can now be fitted to the front of the PCB in specific 'groups' using a similar technique as used for the previous female headers i.e., Blu Tack is used to initially hold the headers in place while 1 or 2 connections are soldered, the Blu Tack is then removed, and the rest of the connections completed. In all instances only a minimal amount of solder should be used to avoid excess solder 'wicking' up into the header. Extra care should also be taken to not let the soldering iron 'touch' the plastic sides of the 1x19 headers as this 'group' of headers are in close proximity to the ESP32 header.

1 row x 3P male black header x2

These headers are for the two servo connections.

1 row x 3P female black header x3 and *1 row x 3P male black header x2*

This 'group' is the two internal power connections, the PIR/μwave connector and the male 2P 'jumper' connector.

Red, amber, and green LEDs

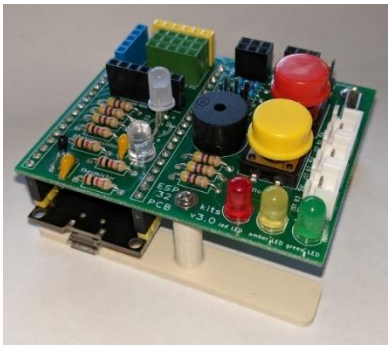
The LEDs are inserted onto the front of the PCB in their red, amber, green sequence, making sure that the longer +'ve anode connectors are inserted into the correct openings on the PCB, which are clearly labelled with a small +.

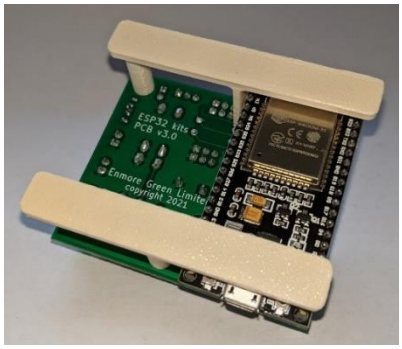
Even though both connectors of a LED are quite long, and they can be bent to try to hold the LED in place, because of their height until they are soldered in place the LEDs do tend to 'wobble' and not stay flush with the surface of the PCB. Small pieces of Blu Tack can therefore be used to temporarily hold the LEDs firmly in place. Each of the connectors are then soldered in place in the same way as all the previous discrete components by the soldering iron tip being pressed firmly on the junction between the LED lead and the PCB opening with a small amount of solder being fed into the gap as it melts.

All the LEDs can be positioned/held in place and soldered at the same time – which will make the installation a little quicker. Then once soldered in place the Blu Tack can be removed and the excess length of LED connector lead can be snipped off or 'waggled' until it breaks off.

The complete ESP32 module/PCB assembly

To create a stable overall assembly, with the ESP32 module 'underslung' from the PCB, a pair of 3D printed 'stands' have been designed that fit the two fixing holes for the PCB, as shown on the right.





Arduino IDE coding

To support the exploration of an assembled Maker Kit, the projects described below use coding developed with the Arduino IDE, i.e., so-called "sketches" that are based upon the C/C++ programming languages.

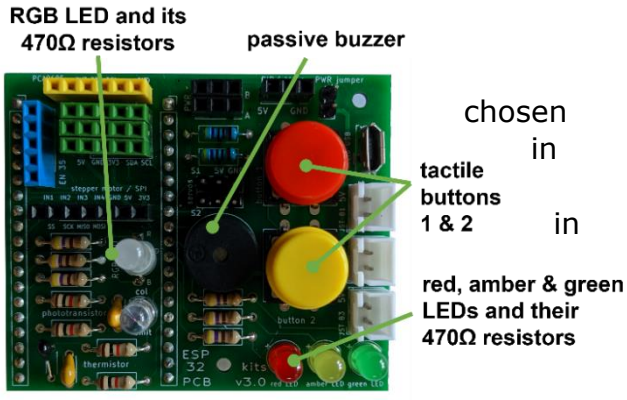
Details on how to download all the example code are provided in *Appendix C: Software & documentation download* and *Appendix A: Kit development details* provides more detailed information on how the ESP32 GPIO pins are 'exposed' on the 38-pin module and the various constraints that the ESP32 firmware imposes upon their usage.

Electronic basics

As previously discussed, one of the main aims of the ***ESP32 Maker Kit*** project has been to enable a range of simple electronic projects to be explored.

These "electronic basics" are the starting point for all the subsequent usage of the Maker Kit and show how 'output' components like the various LEDs or the buzzer can be operated by an 'input', such as the pressing of a button.

This main aim determined the need for the 'electronic basics' components to be included in the Kit and interconnected on a populated PCB as shown on the right, and in more detail in the table below.



Component	ESP32 GPIO pin connections
Red LED	GPIO#14 connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Amber LED	GPIO#27 connected through a 470Ω resistor to the LED's +'ve anode (long leg)
Green LED	GPIO#26 connected through a 470Ω resistor to the LED's +'ve anode (long leg)
RGB LED	GPIOs #17, #16 & #04 connected to the RGB LED's red, green and blue legs respectively
Tactile button 1	GPIO#39 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB
Tactile button 2	GPIO#34 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB
Passive buzzer	GPIO#25 connected to the positive terminal of the buzzer

To support the exploration of some electronic basics with an assembled ESP32 Maker Kit, the following projects can be used.

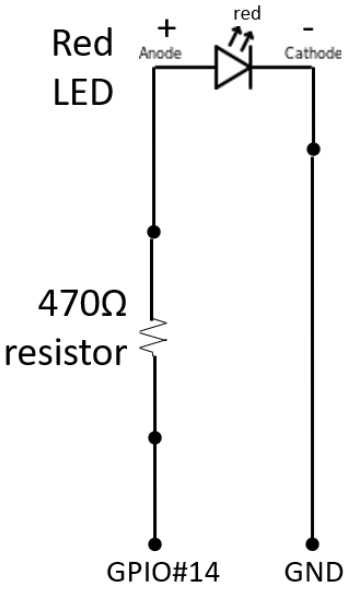
Once the example software has been downloaded, as shown in *Appendix C: Software &* documentation download, a suitable folder can be used to store the code on the machine where the IDE is running.

Single flashing LED

The circuit diagram shown on the right shows how the red LED on the PCB is connected through to ground from its (negative) cathode lead and to the ESP32's GPIO#14 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 Kit component details.*

To run the ***LED_flash.ino*** software, it should be loaded into the ESP32 from the IDE host machine in the usual way.

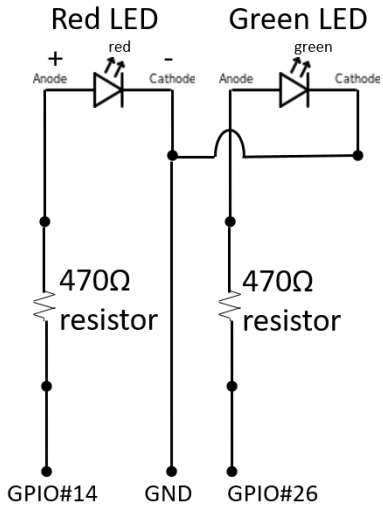


Red/Green flashing LEDs

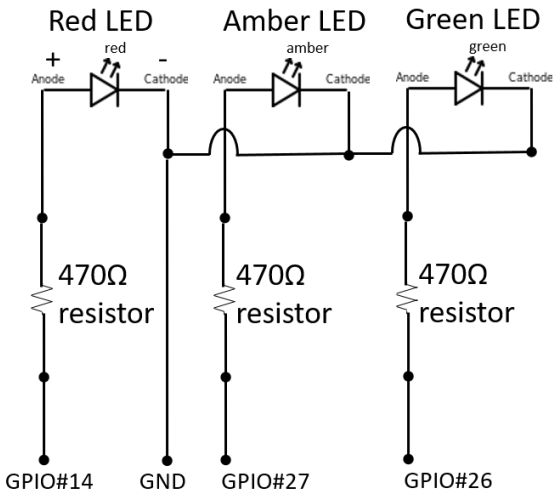
The circuit diagram shown on the right shows how the red and green LEDs on the PCB are connected through to ground from their (negative) cathode leads and to the ESP32's GPIO#14 and #26 pins via 470Ω resistors from their (positive) anode leads.

As discussed above and in *Appendix B: Maker Kit component 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 and green LEDs on/off, and to run the ***LED_red_green_flash.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.



Red/Amber/Green flashing LEDs

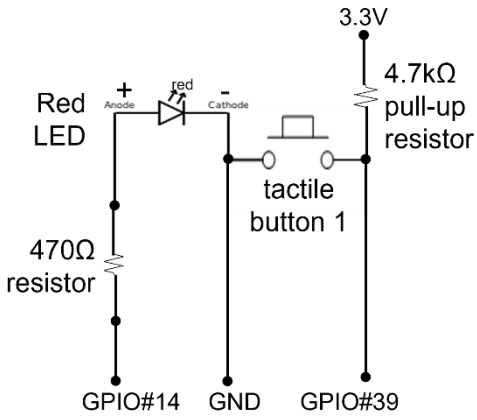


The circuit diagram shown on the right shows how the red, amber and green LEDs on the PCB are connected through to ground from their (negative) cathode leads and to the ESP32's GPIO#14, #27 and #26 pins via 470Ω resistors from their (positive) anode leads.

As discussed above and in *Appendix B: Maker Kit component 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 and to run the ***LED_red_amber_green_flash.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Button switched LED

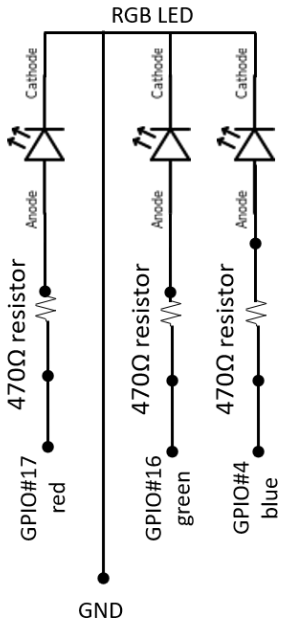


The circuit diagram shown on the right shows how the:

- red LED on the PCB is connected through to ground from its (negative) cathode lead and to the ESP32's GPIO#14 pin via a 470Ω resistor from its (positive) anode lead, and
- how button 1 is connected between ground and GPIO #39 with a 4.7kΩ pull-up resistor on the GPIO side of the button.

This project turns the red LED ON for 3 seconds when button 1 is pressed and to run the ***LED_button_flash.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

RGB flashing LED

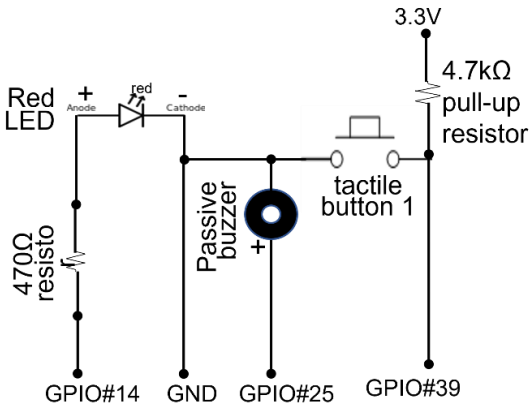


The circuit diagram shown on the right shows how the red, green and blue LEDs within the combined RGB LED on the PCB are connected through to ground from their common (negative) cathode lead and to the ESP32's GPIO#17, #16 and #4 pins via 470Ω resistors from their (positive) anode leads.

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, green and blue LEDs on/off and to run the ***LED_RGB_flash.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Button switched LED & buzzer

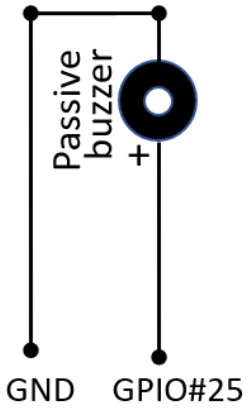


The circuit diagram shown on the right shows how the:

- red LED on the PCB is connected through to ground from its (negative) cathode lead and to the ESP32's GPIO#14 pin via a 470Ω resistor from its (positive) anode lead,
- how button 1 is connected between ground and GPIO #39 with a 4.7kΩ pull-up resistor on the GPIO side of the button, and how
- the buzzer is connected between ground and GPIO#25

This project turns the red LED ON and sounds the buzzer for 5 seconds when button 1 is pressed and to run the ***LED_button_buzzer.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Buzzer 'tunes'



The circuit diagram shown on the right just shows how the buzzer is connected between ground and GPIO#25.

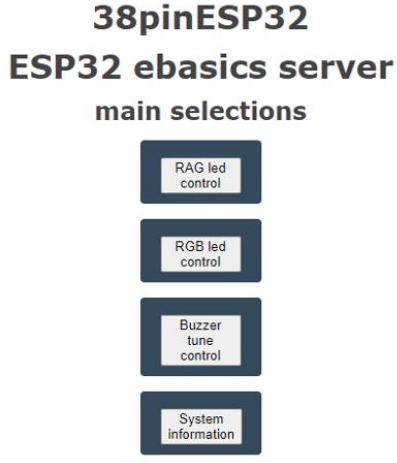
This project shows how the very simple passive buzzer installed on the PCB can be programmed to 'play' quite elaborate tunes.

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 ***buzzer_player.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Web control

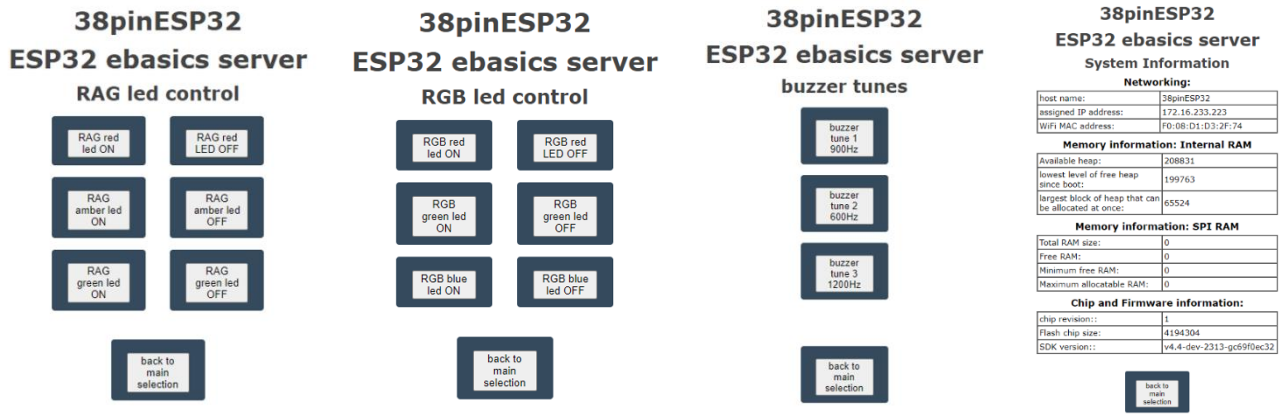


The ESP32 supports WiFi access, so a number of standard library functions allow network access to a web server that runs on the ESP32.

To demonstrate this capability the ***ESP32_HelloServer.ino*** code shows how an extremely simple web interface can be built.

Building upon this code, ***ESP32_ebasicsServer.ino*** then provides a more 'complete' web interface example that allows the three individual red, amber and green LEDs, the RGB LED, and the buzzer to be controlled from individual web pages.

The screen shot shown above right shows the main 'selection' web page and the screen shots below show the individual pages that are shown for each of the 'selections'.



For both these 'web' examples the code needs to be edited so that the WiFi credentials for the local WiFi are used. When the code runs, information about the connection is then shown on the IDE's Serial Monitor, which includes the local IP address of the Web server that is established – this address can then be used with a browser running on another device connected to the same local network.

It should be noted that the ESP32 also supports functions that 'make' the ESP32 a WiFi access point to which the web server is also connected so that web control functions can be used in situations where there is no local WiFi, or its credentials are unknown. The available code examples however do not use this capability at present.

Motor control

The ESP32 Maker Kit can support many different methods for controlling three main types of electric motor, i.e., servo, stepper and drive motors.

The simple control examples profiled in the following sections can be used to help develop more advanced projects.

Servo motor control

The ESP32 Maker Kit can support two main methods for controlling servo motors and the components and GPIO connections used in a populated PCB are set out in the table below.

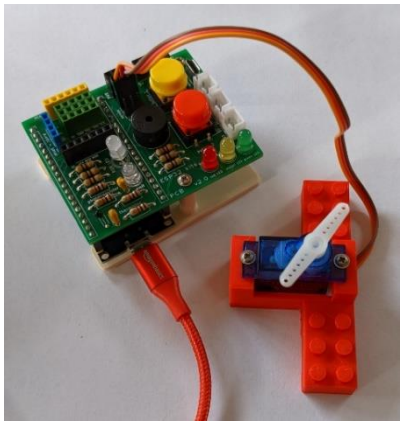
In all the worked examples the low cost, and readily available, SG90 servo as shown on the right has been used and more detail about servos are provided in *Appendix D: Control methods and 'plug in' component details*.

Component	GPIO pin(s)
directly connected servo on the S1 servo connector	GPIO #32 plus 5V and GND connections all provided on the 3 pin male S1 connector
if a 2nd directly connected servo is needed the S2 servo connector can be used	GPIO #33 plus 5V and GND connections all provided on the 3 pin male S2 connector
Tactile button (1)	GPIO#39 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB
Tactile button (2)	GPIO#34 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB
PCA9685 PWM control board	GPIOs SDA & SCL plus 3V3 and GND connections all provided on the PCB's dedicated 6 pin yellow female connector that 'matches' the 6 pins on the PCA9685 board

The set of components tabulated above allows many different types of movement control to be programmed for which example 'starter' Arduino IDE developed code is made available. The details below summarise just some of the control options that are possible for which the starter code enables more complex projects to be developed.

Simple single servo control

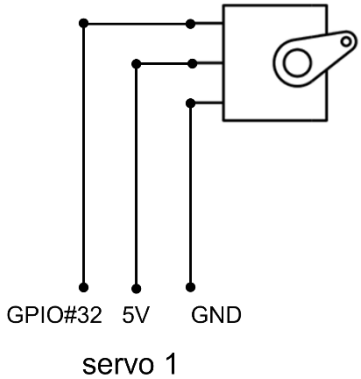
This first method shows how a servo can be controlled by directly connecting it to the ESP32, although whilst this is OK with a single, or at most two, low powered servos, it is generally not recommended since voltage fluctuations due to multiple servo operations can affect the stability of the operation of the ESP32 itself.



The image on the right shows a SG90 servo mounted on custom 3D printed Lego compatible blocks and connected to the dedicated S1 (servo 1) 3-pin male connector which has its 3 pins aligned to the standard SG90 servo cable female connector, i.e.,

- orange signal wire – servo1 pin
- red supply voltage +'ve wire – 5V pin
- brown supply –'ve wire – GND pin

In the image shown the servo has one of its standard 'horns' fitted to the splined gear box drive shaft so that the servo movement is more clearly visible.



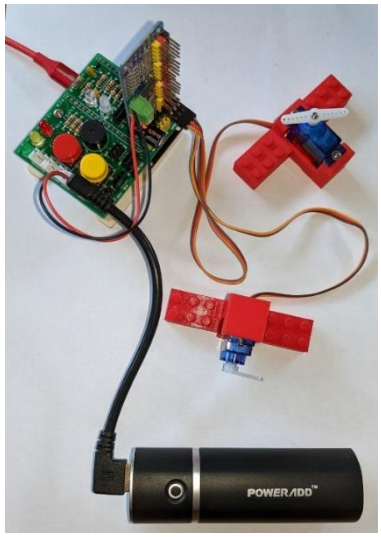
servo 1

The circuit diagram shown on the right shows this basic servo control method, and by examining the ***ESP32_servo_sweep.ino*** Arduino IDE developed code you will see that successive Pulse Width Modulation (PWM) signals are applied to GPIO#32 to produce a continuous 'sweeping' action. More details on the control of servos are explained in *Appendix D: Control methods and 'plug in' component details*.

The ***ESP32_servo_sweep.ino*** code for this simple test routine is part of the main software download and to run this code it should be loaded into the ESP32 from the IDE host machine in the usual way.

I²C button controlled multiple servos

This second method uses a PCA9685 I²C Pulse Width Modulation (PWM) control board as shown left.

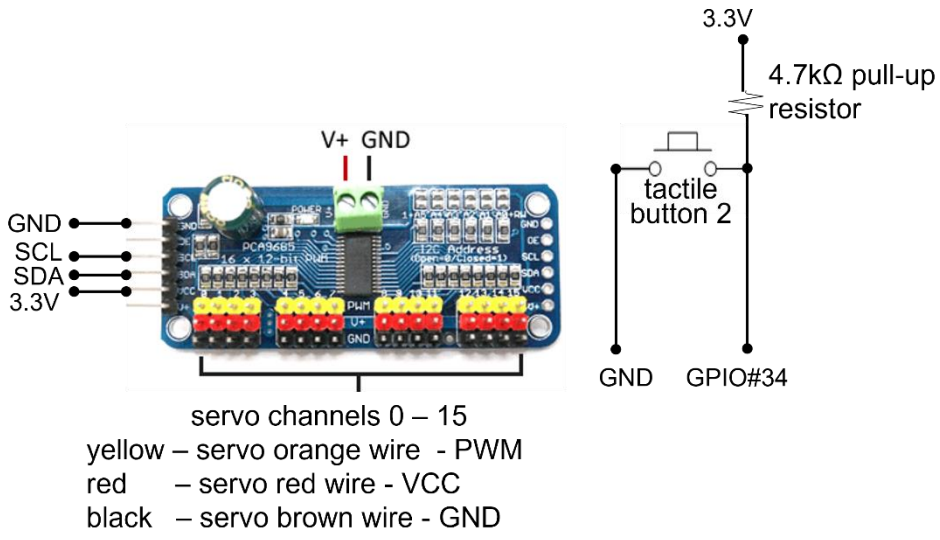


These boards are readily available and more details about this type of board are provided in *Appendix D: Control methods and 'plug in' component details*, but in summary the board takes care of the PWM in a much more electrically stable way using hardware and a separate power supply, than can be produced by the ESP32 using only software and its 'internal' power lines, and it can also separately manage up to 16 individual servos.

The image on the right shows how the board can be inserted into its dedicated 6 pin yellow female connector on the Kit's assembled PCB.

Servos can then be connected to the PCA9685 board (two shown in this example) and using JST connectors, a power supply can then be connected to the 'external' power bus on the Kit's PCB (a 5V rechargeable battery bank in this example). The PCA9685 board's external power connection is then cross-connected to the Kit's PCB's power bus.

The schematic/circuit diagram below shows in some more detail how this more sophisticated servo control method uses button2 on the PCB, and by examining the ***ESP32_PCA9686_btn_servo.ino*** Arduino IDE developed code you will see that button presses initiate Pulse Width Modulation (PWM) being applied to the servos by sending I²C control signals to the PCA9685 board.



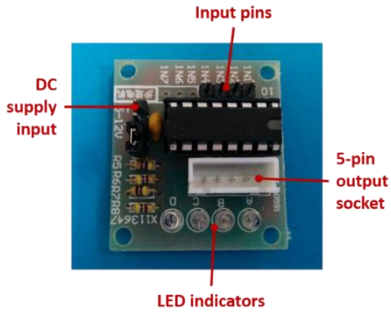
The PCA9685 board has 6 pins, which with its usual set of male connectors, can be directly inserted into the dedicated 6-pin yellow female connector on the Kit's assembled PCB, making the connections through to the ESP32 as shown in the table below:

GND	Any GPIO GND pin
OE (Output Enable)	Not used
SCL (serial clock line)	SCL GPIO pin (GPIO pin#22)
SDA (serial data line)	SDA GPIO pin (GPIO pin#21)
VCC (3.3V supply for the control board)	3.3V GPIO pin
V+ (5V supply for the devices)	Not used because an external power supply is used

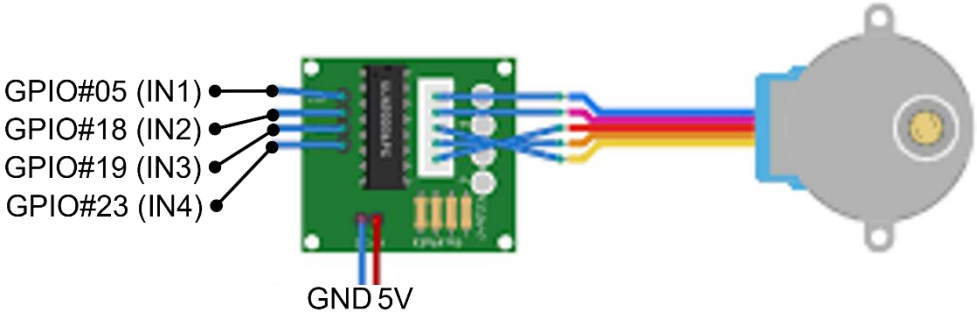
The ***ESP32_PCA9686_btn_servo.ino*** test routine is part of the main software download and to run the code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Stepper motor control

This motor control method uses the low cost 28BYJ-48 stepper motor (not supplied with the Kit) with its associated ULN2005 control board that are shown below and for which more details of these readily available components are provide in *Appendix D: Control methods and 'plug in' component details*.



The schematic/circuit diagram below shows how this 28BYJ-48 stepper control method uses the INI-IN4 connections that are on the 7-pin black female connector on the Kit's assembled PCB, and by examining the Arduino IDE developed code ***ESP32_stepper_basic.ino*** you will see how the connected GPIO pins are sequenced HIGH/LOW in the ways needed to energise/de-energise the stepper's coils using the so-called *full-step* method.

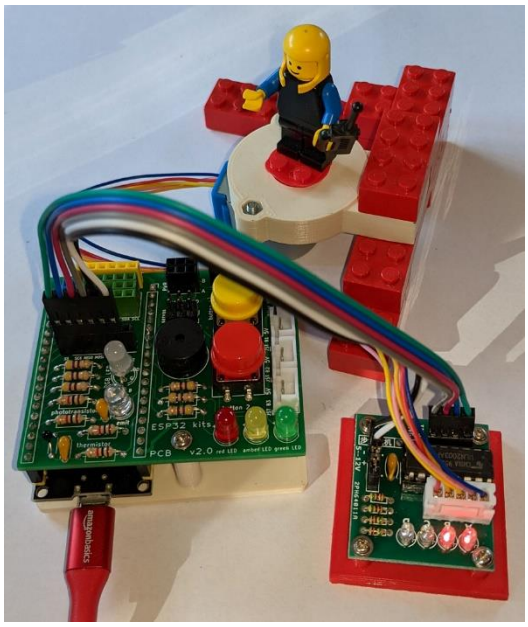


The stepper motor can usually be powered direct from the ESP32 power supply, assuming it is supplied by an adequate 5V DC power supply with at least a 2A capacity.

For this direct powered arrangement, the image on the right shows how the GND-5V connections for the ULN2005 control board can also be fed from the 7-pin black female connector on the PCB.

However, if there is any doubt about this, or problems are experienced, then the ULN2005 can be separately powered from the PCB's 'external' power bus using either 4xAA batteries or a 5V rechargeable battery bank.

The ***ESP32_stepper_basic.ino*** code for these stepper motion test routines is part of the main software download and to run the code it should be loaded into the ESP32 from the IDE host machine in the usual way.



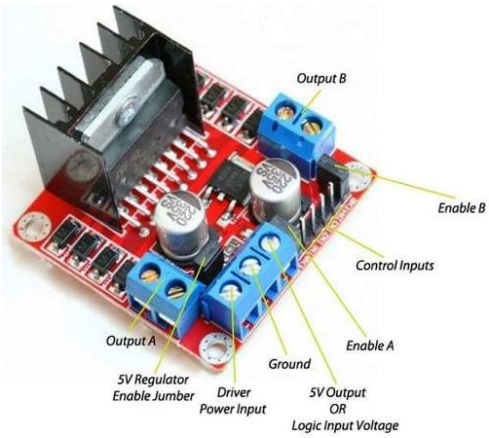
Drive motor control

Conventional 'drive' motors, controlled by a ESP32, are very commonly used in many projects e.g., various types of 'wheeled' robot.

The Maker Kit can be used to prototype/demonstrate 'drive' motor control and the example below describes the use of very commonly used 5-6V geared motors controlled by a readily available motor controller.

Arduino IDE developed software is available to demonstrate all the usual motor control methods that would be needed for a 'wheeled' robotics project.

L298N motor controller

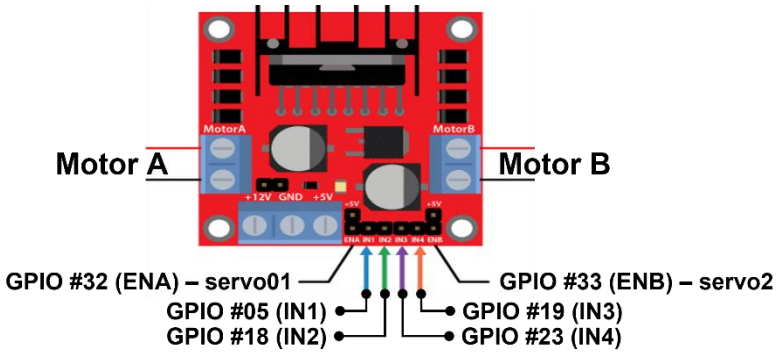


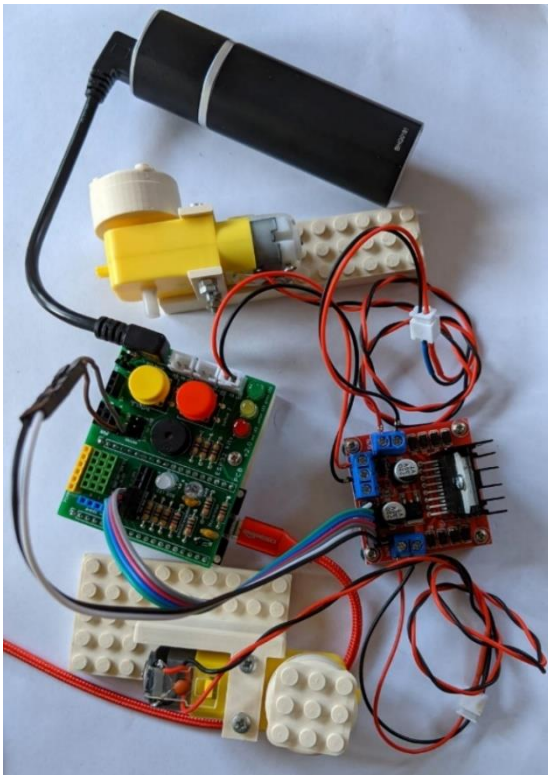
The L298N motor controller board, as shown on the left, provides a wide range of capabilities for controlling two electric drive motors.

The motors can be powered from a source that can supply from 5 to 35 volts DC and each motor can be supplied with 2A continuously. The board itself is powered from the same supply source with the voltage automatically stepped down to 5 volts DC and when powered correctly the L298N board has a red LED switched on.

A very useful capability is that if the input power comes from anything that supplies no more than 12 volts DC, then a board 'jumper' enables an output feed of 5 volts DC which can be used to power the controller e.g., a ESP32, Arduino board, etc.

The schematic/circuit diagram on the right shows how this test L298N motor control method uses the INI-IN4 connections that are on the 7-pin black female connector plus the two GPIOs usually used for servo connections (#32 and #33).

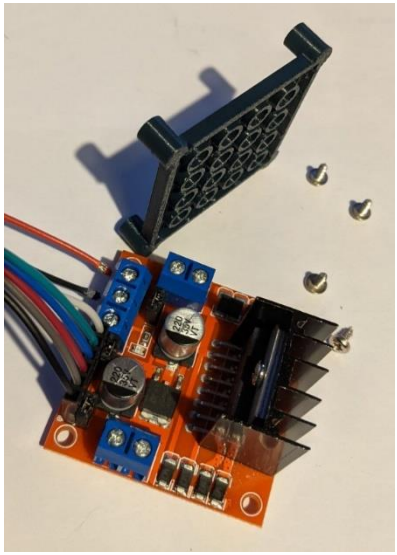




By examining the Arduino IDE developed code ***ESP32_L298N_drive_1motor_test.ino, ESP32_L298N_drive_2motors_test.ino,*** and ***ESP32_L298N_motor_functions.ino*** you will see how motor control functions have been developed that set the IN1-IN4 and the additional GPIO pins appropriately.

As shown in the image on the left, a L298N motor controller can be used to independently control two drive motors when connected to the Maker Kit PCB, with power for the motor controller and the two drive motors supplied from the Maker Kit PCB power bus either by a 4xAA battery pack, or as shown, from a 5V USB battery bank.

The drive motors shown here have been mounted on custom 3D printed LEGO compatible blocks so that they can be incorporated into an overall LEGO 'mechatronic' construction where parts of the build can be moved by the motors.



The L298N has also been mounted on a 3D printed custom designed LEGO compatible 'tile', as shown on the left, so this too can be securely incorporated into an overall LEGO construct.

The code for these motor controller test routines is part of the main software download and to run the code, and to run each code example it should be loaded into the ESP32 from the IDE host machine in the usual way.

Display projects

The ESP32 Maker Kit can be used to explore I²C, and SPI (Serial Peripheral Interface) controlled display devices, as well as simpler 7 segment LED displays as described in more detail in the following sections.

I²C managed displays

The ESP32 Maker Kit has been specifically designed to support multiple devices more easily when managed through the microcontroller's I²C interface. The assembled PCB has three 'general' 5-pin green connectors that provide access to 5V-GND-3V3-SDA-SCL, in addition to the dedicated PCA9685 6-pin yellow connector.

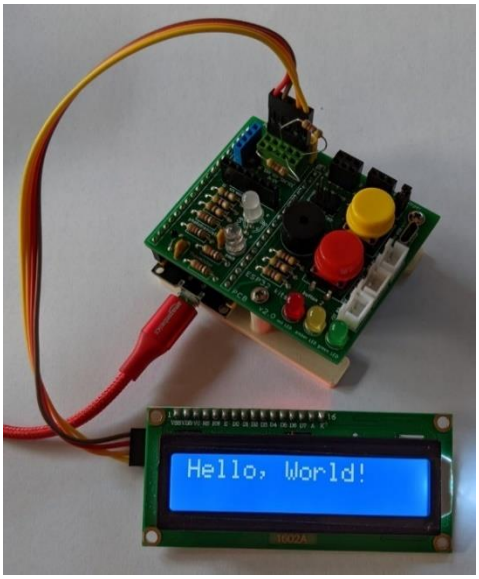
Both 5V and 3V3 connections have been made available on these 5-pin connectors so that devices that need either 5V or 3V3 to power them can be used - but as discussed below, some consideration and care must be taken to ensure that any device with its own pull-up resistors between its SDA and SCL connections and its 5V supply line are disabled - and if several devices are being managed on the I²C bus simultaneously, then the 'accumulative' pull-up resistance needs to be judged/calculated to ensure sufficient 'pull-up' is provided to the combined SDA and SCL lines into the ESP32.

This section describes the use of two different types of display that each use the I²C and addresses some of the pull-up issues discussed above.

16 character x 2 line LCD

A liquid crystal display (LCD) uses the light-modulating properties of liquid crystals combined with polarizers. Liquid crystals do not emit light directly, instead they use a backlight or reflector to produce images in colour or monochrome.

The commonly available monochrome 16 character x 2 line LCD, with an I²C 'backpack' for control, is powered by 5V, so some care is needed to ensure that the voltages on the SDA and SCL control lines do not exceed 3.3V as this would potentially damage the ESP32 to which it is connected.



These LCD devices have internal 4.7kΩ pull-up resistors between both the SDA and SCL control lines and their supply voltage of 5V - so the safest way to use these displays when connected to and controlled by a ESP32, is to remove these pull-up resistors as illustrated in the image above right.

The arrangement shown on the left then shows the four I²C connection leads (GND, VCC, SDA and SCL) plugged into one of the 5-pin I²C (green) female connectors on the PCB using the 5V port for the supply voltage - with two 4.7kΩ resistors inserted (just on a temporary, test basis!) between SDA - 3V3 and SCL - 3V3 connections on the other I²C (green) female connectors on the PCB.

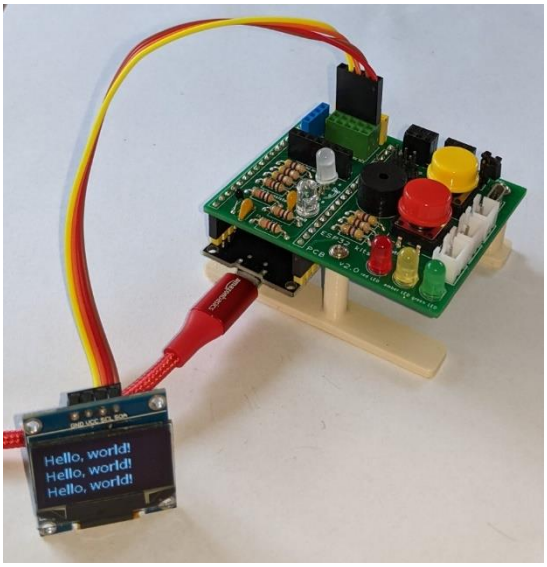
Once connected to the I²C bus the address for these 16 character x 2 line LCD devices is usually 0x27 and this is the assumed default in the Arduino IDE developed code used to demonstrate these devices.

The ***ESP32_I2C_16x2_LCD.ino*** code for this display demonstration is part of the main software download and to run the code example it should be loaded into the ESP32 from the IDE host machine in the usual way.

64x128 pixel OLED

An organic light-emitting diode (OLED or organic LED), also known as organic electroluminescent (organic EL) diode, is a light-emitting diode (LED) in which the emissive electroluminescent layer is a film of organic compound that emits light in response to an electric current. These displays are very attractive since, as they emit visible light, they can operate without a backlight and allow deep black levels to be displayed. They can also be made much thinner and lighter than Liquid Crystal Displays (LCDs).

The commonly available, small 128x64 pixel OLED I²C managed display, shown on both sides in the image on the right, is powered by 3V3 so does not cause any concerns about too high a voltage on the ESP32's SDA or SCL control lines.



It therefore works OK directly connected to any of the 5-pin I²C (green) female connectors on the PCB as shown in the image above right - using only the 3V3 port of course!

The display can be used for text or images, or a combination of both, with Arduino IDE developed software used to 'compose' a 'bit image' of whatever content is required that is then displayed as shown in the example in the image on the left.

Once connected to the I²C bus the default I²C address for these displays is usually 0x3c (and this may be the address despite what may be printed on your device!!)

The ***ESP32_128x64_OLED.ino*** code along with a set of fonts, for this display demonstration is part of the main software download and to run the code example it should be loaded into the ESP32 from the IDE host machine in the usual way.

SPI managed displays

The Serial Peripheral Interface (SPI) is a synchronous serial communication interface specification used for short-distance communication that can be used to connect devices to a ESP32 using the Maker Kit's PCB. Compared to I²C, which is a relatively low speed bus, designed primarily for transferring small amounts of data, e.g., for sensors etc., SPI is a higher speed interface which requires more wires but can be better for supporting some types of display that require more data to be transmitted such as larger pixel sizes or multi-colour devices such as those described below.

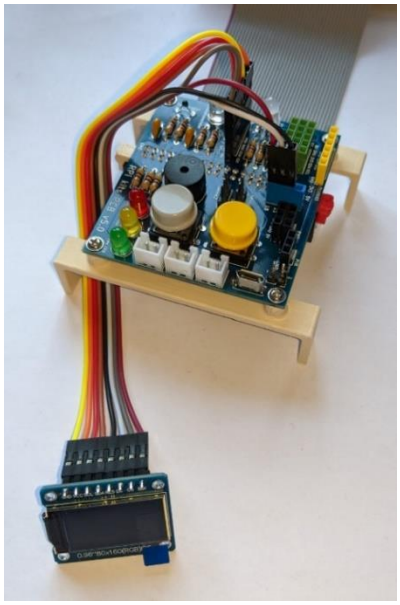
The devices below are described as IPS TFT, where:

- Thin Film Transistor (TFT) is a variant of a liquid crystal display (LCD) that uses thin-film-transistor technology to improve image qualities such as addressability and contrast, and
- In-Plane-Switching (IPS) is an additional improvement that increases the viewing angle as well as colour quality.

80x160 IPS display

This commonly available 0.96-inch and 65k colour display with an 80x160 pixel resolution, as shown front and back on the right, uses the ST7735S integrated circuit chip and can be connected to the Maker Kit PCB using both the 7P black female connector and three additional GPIO pins.

Display pin	Maker Kit PCB connection
GND	7P connector GND
VCC	7P connector 3V3
SCL (SPI clock)	7P connector SCK (GPIO #18)
SDA (SPI write data)	7P connector MOSI (GPIO #23)
RES (reset)	GPIO #1 (spare 1)
DC (display control)	GPIO #33 (servo 2 connection)
CS (SPI control)	7P connector SS (GPIO #5)
BLK (backlight)	GPIO #32 (servo 1 connection)



The PCB's 7P connector provides access to the SPI interface on the ESP32 and all of the display connections are as set out in the table above with a test set up shown in the image above right.

The ***80x160_ST7735_Colour_Test.ino***, ***80x160_ST7735_drawXBitmap.ino***, and ***80x160_ST7735_Hello_World.ino*** code 'sketches' for these display demonstrations are part of the main software download. They all use the readily available *TFT_eSPI*

Arduino IDE Library which uses customised 'templates' to define all the various parameters of the display and its connections.

The 'template' for a 80x160 ST7735 managed display connected to the Maker Kit PCB is provided as *Setup1_ST7735_80x160.h* in a *Custom_Setups* folder. The location of the *Custom_Setups* folder is then defined in a *User_Setup_Select.h* file which should be placed in the *TFT_eSPI* Library folder for access by the Arduino IDE.

Then to run each of the code 'sketches' they should be loaded into the ESP32 from the IDE host machine in the usual way.

240x240 IPS display

This commonly available 1.3-inch and 65k colour display with a 240x240 pixel resolution, as shown front and back on the right, uses the ST7735S integrated circuit chip and can be connected to the Maker Kit PCB using both the 7P connector and three additional GPIO pins.

Display pin	Maker Kit PCB connection
GND	7P connector GND
VCC	7P connector 3V3
SCL (SPI clock)	7P connector SCK (GPIO #18)
SDA (SPI write data)	7P connector MOSI (GPIO #23)
RES (reset)	GPIO #1 (spare1)
DC (display control)	GPIO #33 (servo2)
BLK (backlight)	GPIO #32 (servo1)

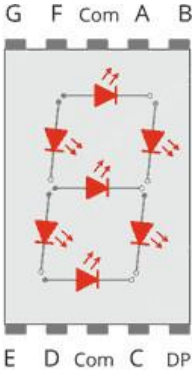
The PCB's 7P connector provides access to the SPI0 interface on the ESP32 and all of the display connections are as set out in the table above with a test set up shown in the image on the right.

The ***1.3_IPS_LCD.ino*** code for this display demonstration is part of the main software download. They all use the readily available *TFT_eSPI* Arduino IDE Library which uses customised 'templates' to define all the various parameters of the display and its connections.

The 'template' for a 240x240 ST7789 managed display connected to the Maker Kit PCB is provided as *Setup1_ST7789_240x240.h* in a *Custom_Setups* folder. The location of the *Custom_Setups* folder is then defined in a *User_Setup_Select.h* file which should be placed in the *TFT_eSPI* Library folder for access by the Arduino IDE.

Then to run the code 'sketch' it should be loaded into the ESP32 from the IDE host machine in the usual way.

7 segment LED displays



7 segment LED displays are one of the simplest types of display that are commonly used, and they have been in use for many decades in digital clocks, basic calculators and any devices that need a basic numerical display.

Each digit of a display is made up of 7 LEDs, as shown in the image on the left, where an eighth LED may be used for a decimal point or for a separate colon in a multi-digit construct, and all the LED's anodes (or the cathodes depending upon the display type) are grouped together to a common connector.

An individual digit segment can be 'lit' by turning its supply pin HIGH just as is shown in the previous *Electronic basics* section of this document, but obviously this becomes quite complex when forming individual characters by lighting sets of segments and this complexity multiplies when multiple sets of 7 segment digits are constructed.

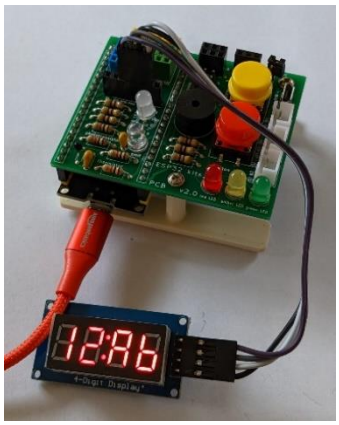
To simplify this complexity challenge, sets of 7 segment digits are therefore usually packaged with their own dedicated driver/control chip and there are two commonly used integrated circuits (ICs) that are used for this, namely the Titan Micro Electronics TM1637 and the Maxim Integrated MAX7219 which are discussed in the sections below.

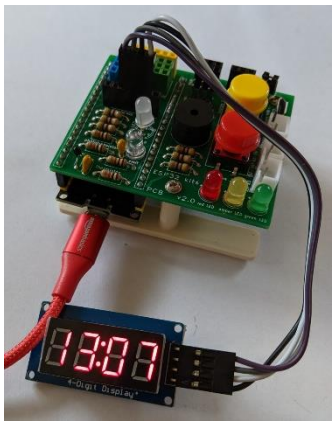
TM1637 LED display

The images on the right show a front and back view of a 4 digit 7 segment LED display, with an additional 'colon' LED, that is packaged with and controlled by a TM1637 IC.

The four pins of the display can be connected to the Maker Kit PCB as shown in the table below and in the images below right: a custom Arduino IDE library is available to simplify the coding for displaying text in many different and useful ways.

Display pin	Maker Kit PCB connection
CLK	IN3/MISO GPIO #19
DIO	IN4/MOSI GPIO #23
VCC	5V from the 7P connector
GND	GND from the 7P connector





Details about the Arduino IDE library used with the demonstration code can be found **here** and it can be installed in the IDE in the normal way.

Three test programs are available:

- ***ESP32_TM1637_7seg_LED_basic.ino*** which just displays some simple test digits as shown in the left-hand image above right;
- ***ESP32_TM1637_7seg_LED_clock.ino*** which continuously gets the current time from a remote time server and displays it in a usual clock format, as shown in the right-hand image above; and
- ***ESP32_TM1637_7seg_LED_clock2.ino*** which initially gets the current time from a remote time server to set the internal ESP32 clock and then updates the clock formatted display continuously.

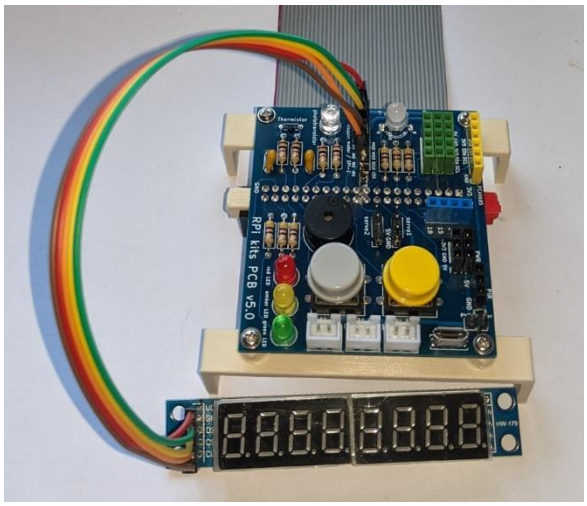
These display demonstration programs are part of the main software download and to run the code examples they should be loaded into the ESP32 from the IDE host machine in the usual way.

MAX7219 LED display

The images on the right show a front and back view of an 8 digit 7 segment LED display, with additional 'decimal point' LEDs, that are packaged with and controlled by a MAX7219 IC.

The display uses a SPI protocol for connection to a ESP32 and the five pins of the display can be connected to the Maker Kit PCB as shown in the table and image below: a custom Arduino IDE library is available to simplify the coding for displaying text in many different and useful ways.

Display pin	Maker Kit PCB connection
VCC	7P connector 5V
GND	7P connector GND
DIN	7P connector IN4/MOSI (GPIO #23)
CS	7P connector IN1/SS (GPIO #5)
CLK	7P connector IN2/SCK (GPIO #19)



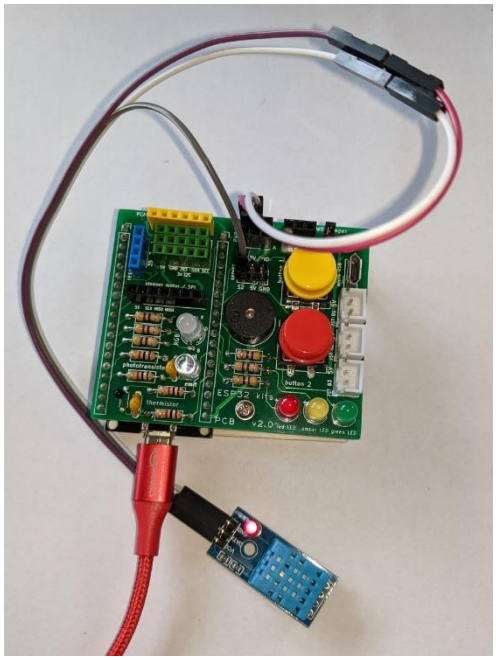
Details about the Arduino IDE library used to demonstrate this display, and how it can be installed on a ESP32, can be found **here**.

The ***ESP32_MAX7219_7seg_basic.ino*** code for this display demonstration is part of the main software download and to run the code examples they should be loaded into the ESP32 from the IDE host machine in the usual way.

Sensor projects

The Maker Kit supports the connection of a wide range of sensors that can be managed by the ESP32. The following sections describe just some of the many ways that the ESP32 can be used to measure the performance of devices and the general environment.

Temperature & humidity sensing

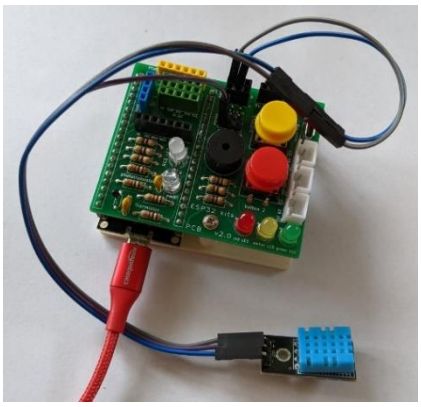


The DHT11 sensor is a commonly available and low-cost sensor of both temperature and relative humidity (RH).

The sensor can be obtained in a number of different 'packages' with one incorporating a LED to indicate it is operating shown in the image on the right and a more basic one shown below.

Whilst not shown in these images, as the Maker Kit supports the connection of an I²C controlled 16x2 LCD, any test code could, not only continuously collects the sensor data, but also displays the output on the LCD.

The DHT11 sensors all work in much the same way, providing a 40-bit (5 byte) digital data output when it is triggered by an external 'start signal', with the humidity and temperature data encoded in this output data. Whilst not providing very high accuracy, i.e., ± 5% within a 20-80% RH range and ± 2°C in a 0-50°C range, this simple to use sensor is quite reliable and provides fast responses from a resistive-type humidity measurement component and an NTC (negative temperature coefficient) temperature measurement component connected to an 8-bit microcontroller.



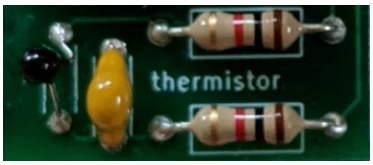
The units shown here both have the sensor and processing elements mounted on small module boards with just 3 connectors labelled: *VCC*, *DATA* and *GND* or +, *out* and –

The sensor module can be powered by 3V3 (therefore ensuring a safe output voltage level for the ESP32 on the *DATA/out* line), so for the test arrangements shown here the '*DATA*' or '*out*' pins are connected to one of the 'servo' GPIOs on the PCB (servo 2/GPIO #33) and the 3V3/GND connections are provided from the nearby PWR-B 3-pin female connectors.

Importantly, built-in to the 'LED board' is a 1kΩ pull up resistor between the 'VCC' and 'DATA' connectors, and on the 'non-LED board' there is a 5kΩ pull up resistor between the '+' and 'out' connectors. Both 'levels of pull-up' satisfactorily ensure that the '*DATA*' or '*out*' signals operate at a stable voltage level and do not 'float'.

The ***DHT11_simple.ino*** code for this temperature and relative humidity measurement demonstration is part of the main software download and to run the code example it should be loaded into the ESP32 from the IDE host machine in the usual way.

Thermistor temperature sensing



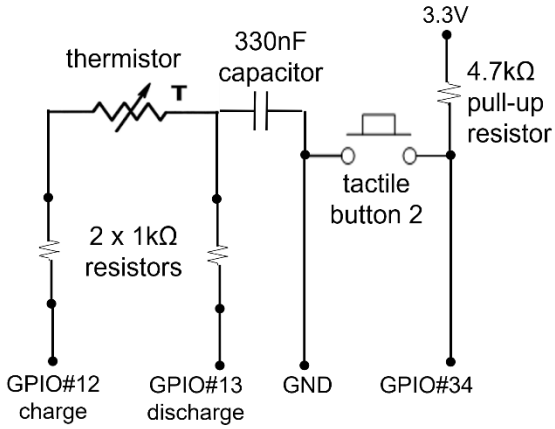
The supplied Maker Kit electronic components includes a thermistor, packaged as a small black ethoxyline resin coated ceramic bead with a pair of uninsulated tinned copper wire leads.

The thermistor used in the PCB assembly has a reference resistance of 10kΩ at 25◦C and is suitable for measuring temperature over a range of -30◦C to +125◦C, although obviously the overall PCB assembly should not be subjected to such a temperature range!! The thermistor with the (also) supplied pair of 1kΩ resistors and the 330nF ceramic capacitor are configured on the PCB (as shown above left) as a "RC charging" circuit that can be used to 'deduce' the thermistor resistance from which the surrounding temperature can then also be 'deduced'.

The electronic component interconnections on a populated ESP32 Maker Kit PCB are as summarised below:

Component	GPIO pin(s)
Tactile button 2	GPIO#34 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB - this button is used to initiate the temperature measurement cycle
Red, Amber and Green LEDs used to indicate the progress of the measurement process	GPIO#14 connected through a 470Ω resistor to the Red LED's +'ve anode (long leg)
	GPIO#27 connected through a 470Ω resistor to the Amber LED's +'ve anode (long leg)
	GPIO#26 connected through a 470Ω resistor to the Green LED's +'ve anode (long leg)

The circuit diagram shown on the right shows how pressing button1 on the PCB can be used to initiate an 'RC charging circuit' cycle which consists of:



- Discharging the capacitor by setting GPIO#19 as an OUTPUT that is LOW
- Measuring the recharge time by setting GPIO#19 as an INPUT, GPIO#13 as an OUTPUT that is HIGH, and then 'clocking' how long it takes for GPIO #19 to go HIGH
- This time is then used to deduce the thermistor resistance, from which the surrounding temperature can then be deduced.

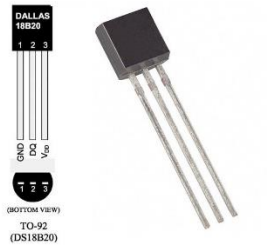
A more detailed description of this 'deduction analysis' is provided in *Appendix E: RC charging circuit analysis*.

Arduino IDE developed code is available as part of the main software download to support the use of this 'RC charging' circuit and to run the ***temperature_gauge.ino*** code it should be loaded into the ESP32 from the IDE host machine in the usual way.

1-wire temperature sensor

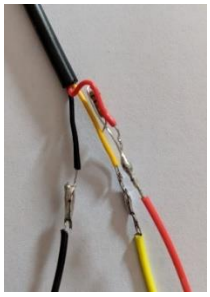
As shown below in the first two images, the DS18B20 temperature sensor is commonly available either as a simple 'chip package' or as a waterproof assembly in a sealed metal tube. For either packaging type the sensor connects to a ESP32 using the 1-wire interface, although there are usually 3 wires, i.e., power and ground as well as the single data line.

For the Maker Kit PCB test set up described here, a waterproof tube sensor is used, and the 3rd and 4th images below show that in order to ensure the data line voltage does not 'float', a 10kΩ pull-up resistor has been connected across the power and data lines when soldering male jumper leads to the wires.

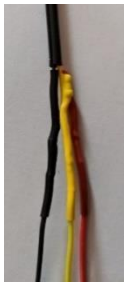


The DS18B20 device in a TO-92 'transistor-like' chip package

Waterproof version of the DS18B20 embedded in a metal tube with a long connection lead

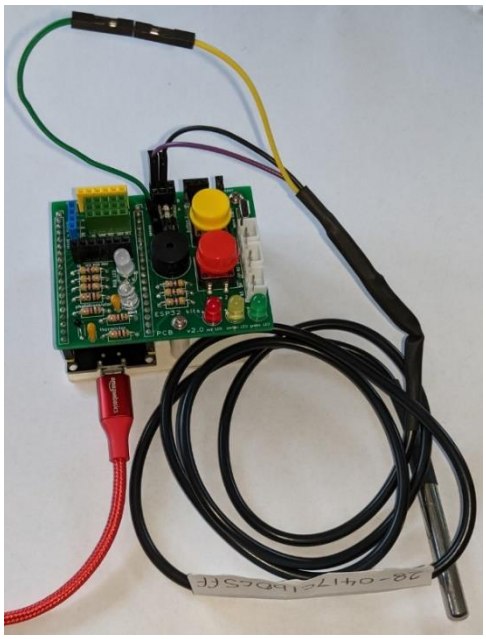


10kΩ pull-up resistor soldered between the power and data lines



heat shrink tubing used to insulate the soldered joints

The sensor is powered by 3.3V (therefore ensuring a safe output voltage level for the ESP32), so for the test arrangement shown in the image on the right, the sensor's (yellow) data line is connected to GPIO#33 on the Maker Kit PCB - which is usually used to control a servo (servo2). The 3V3 and GND connections are then provided from the nearby PWR-B 3-pin female connectors on the PCB.



The DS18B20 is a very functional sensor, providing configurable ◦C temperature measurements with 9-12 bit resolution that defaults to the full 12 bit resolution. It also has an alarm function with non-volatile user-programmable upper and lower trigger points and a measurement accuracy of +0.5◦C over a -10◦C to +85◦C range.

The ***1-wire*** interface is a device communications bus system designed by Dallas Semiconductor Corp. that provides low-speed (16.3kbps) data with associated signalling and it can also power devices over the single data conductor so long as there is also a ground connection; this is called parasitic supply, although as we have used in this test set up, providing a separate power connection is usually more satisfactory.

As it is a bus system there is always a 1-wire master that initiates and controls the communication with one or more 1-wire slave devices on the bus which on a general basis may carry out a variety of functions other than temperature measurement.

Each 1-wire slave device has a unique, unalterable, factory-programmed, 64-bit ID (identification number), which serves as a device address on the 1-wire bus. The 8-bit family code, that is a subset of the 64-bit ID, identifies the device type and functionality. Typically, 1-wire slave devices operate over the voltage range of 2.8V (min) to 5.25V (max). More detailed information on the 1-wire bus system is available at this link.

The Arduino IDE developed test software takes care of the data collection from any connected slave devices with the ESP32 acting as the master. The demonstration code assumes the DS18B20's data line is connected to the servo2 signal pin (GPIP #33) with the sensor's power and ground lines connected to a 3V3 and GND connection on one of the PWR connectors on the PCB. The code then uses the PCB's tactile button 2 to 'trigger' a data collection cycle and outputs the 'interpreted' data to the serial monitor.

The ***DS18B20_simple.ino*** code for this temperature measurement demonstration is part of the main software download and to run the code it should be loaded into the ESP32 from the IDE host machine in the usual way.

Object detection

An important 'class' of sensor is one that can be used to detect objects and the following series of sections describe the use of various sensors that use heat, a radar doppler effect or sound to indicate whether an object is within the 'range' of a sensor.

Each sensor type can connect to the Maker Kit PCB so that it can be controlled by a ESP32.

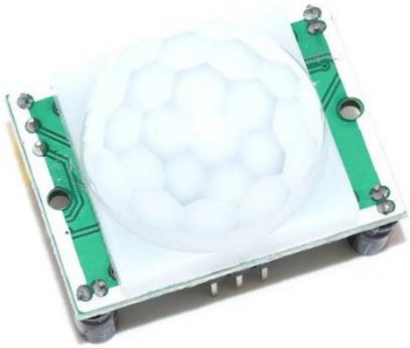
Passive Infrared object detection

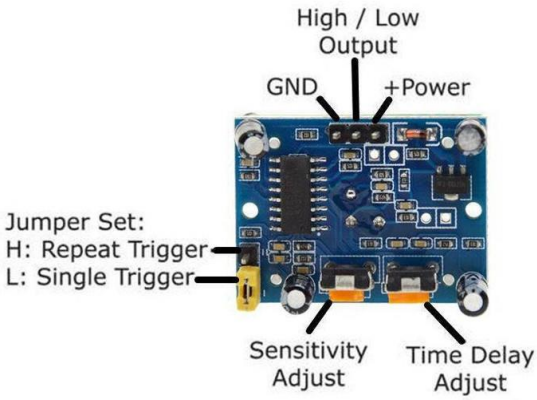
A passive infrared sensor (PIR sensor) is a sensor that measures infrared (IR) radiation, i.e., heat that radiates from objects in the sensor's field of view.

These types of sensor are readily available in a number of different 'packages' and the following details a few of the available types.

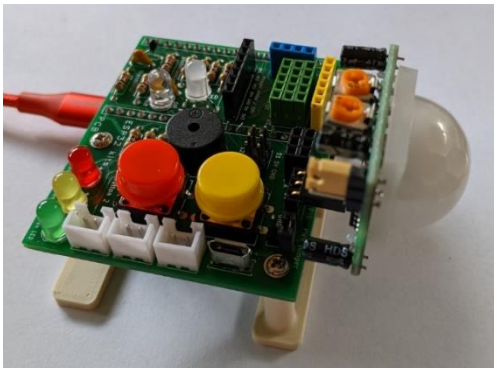
HC-SR501, HC-SR312 and HC-SR602 PIR sensors

The **HC-SR501**, shown on the right, is the most commonly available PIR sensor and as the schematic below of the underside shows there are a number of configuration options that can be set.

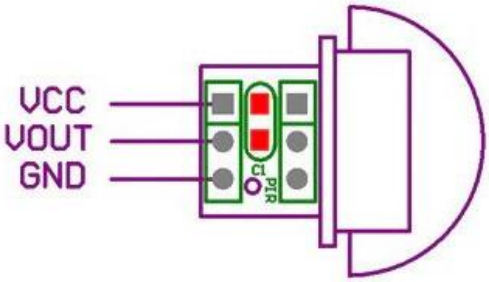


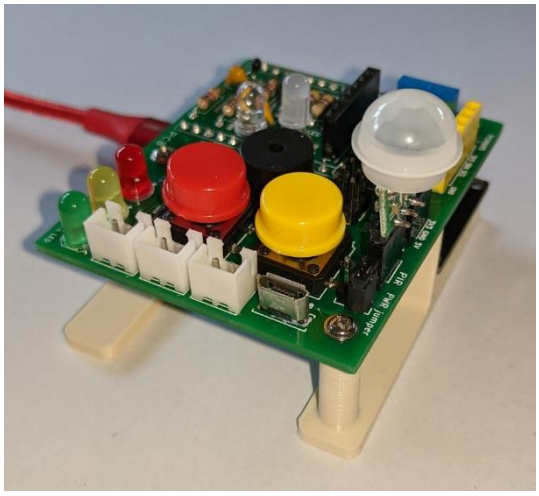


Using a right-angled adaptor, this sensor can be directly inserted into the Maker Kit PCB, as shown on the right, and Arduino IDE developed software is then used to monitor the 'Output' pin which connects through to GPIO #36.



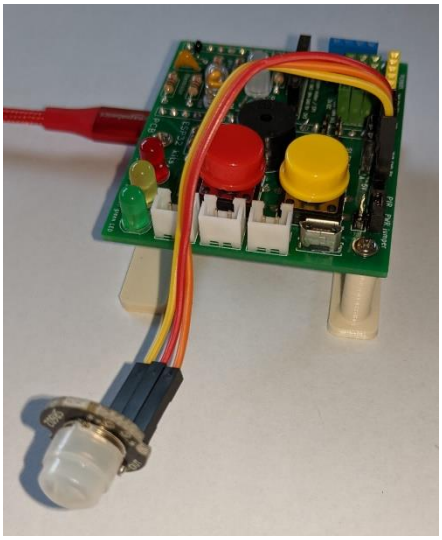
The **HC-SR312**, shown on the right, is a less commonly available 'mini' PIR that has exactly the same pin configuration as the HC-SR501, as shown in the schematic below.





This means it can also be inserted direct into the Maker Kit PCB as shown left, taking care to align the pins correctly (!!) and exactly the same Arduino IDE developed software can be used to collect object detection data.

The **HC-SR602**, shown on the right, is an even less commonly available 'mini' PIR, but whilst it still has 3 pins (OUT, +, -) they are not in the same order as the HC-SR501 or the HC-SR312.



Therefore, as shown on the left, jumper leads must be used to correctly connect it to the 3 pin female PIR on the Maker Kit PCB.

As all the above PIRs behave in the same way, ***PIR_detection.ino*** is a general set of Arduino IDE developed code used for this PIR object detection demonstration that is part of the main software download.

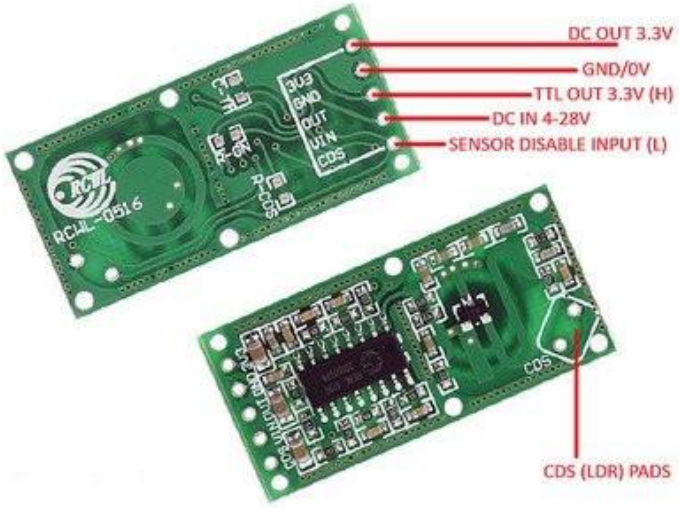
To run the code, it should be loaded into the ESP32 from the IDE host machine in the usual way.

Microwave doppler radar moving object detection

A Doppler radar is a specialised radar that uses the Doppler effect to produce velocity data about moving objects at a distance. It does this by bouncing a microwave signal off the object and analysing how the object's motion has altered the frequency of the reflected return signal.

Typically, this measurement technique is used for high accuracy measurements in applications such as 'radar guns' for vehicle speed assessment.

The image on the right however shows a low cost (and obviously less accurate) but now widely available RCWL-0516 module.



This can be used in many different digital making projects - with an input voltage range of 4-28VDC and an output signal voltage of 3.3VDC, the board can easily be connected to a ESP32 for moving object detection.

In addition, whilst the RCWL-0516 has 5 pins, the main 3 middle pins are in the same order as required by the PIR connector on the Maker Kit PCB. This means it can be directly inserted into this connector as shown in the image above left.

The unused 'outer' pins are a regulated 3.3V output and sensor 'disable' input pin, neither of which are needed for the exploratory testing described here.

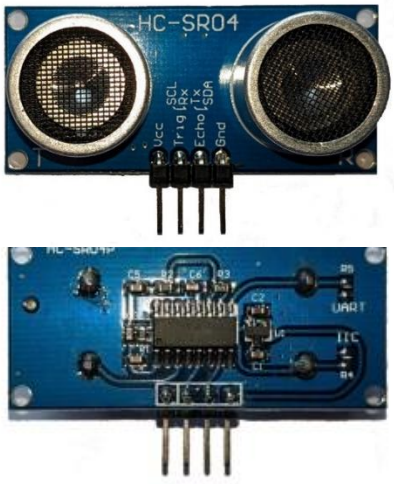
Some other points to note about using the RCWL-0516 are:

- Its operating frequency is at or about 3.181GHz, so non-metallic structures will NOT block the broadcast signal i.e., it will detect through walls!
- The 'forward' side of the board (the side with all the components) is the main object detection direction and should not be obstructed by anything metallic.
- The default detection range is about 7m but adding a 1MΩ resistor across the R-GN pads on the 'back' side of the module reduces it to about 5m. Lower resistance values might further reduce the range, perhaps down to less than 1m.
- The 'back' side of the module should have clearance of more than 1cm from anything metallic.
- Operating multiple modules in close proximity is not recommended as their broadcast signals will interfere with each other.
- The default 'repeat trigger time', i.e., the period of time that the object detection pin will remain HIGH, is 2 seconds which can be adjusted by adding a capacitor across the C-TM pads on the 'back' side of the module.
- The module includes an option to fit a photoresistor that will enable/disable the sensor when the background light level is either high or low. The area on the module where this is fitted is labelled CDS because inexpensive CdS or Cadmium Sulphide devices have often been used – BUT this is a material that is now severely restricted throughout Europe due to the RoHS ban on cadmium. The use of a phototransistor as a light sensor is a better substitute.

As this device produces a similar detection output as the PIR devices, 'derivative' Arduino IDE developed code ***microwave_detection.ino*** is available as a basic object detection demonstration and is part of the main software download.

To run the code, it should be loaded into the ESP32 from the IDE host machine in the usual way.

Ultrasonic sound object detection

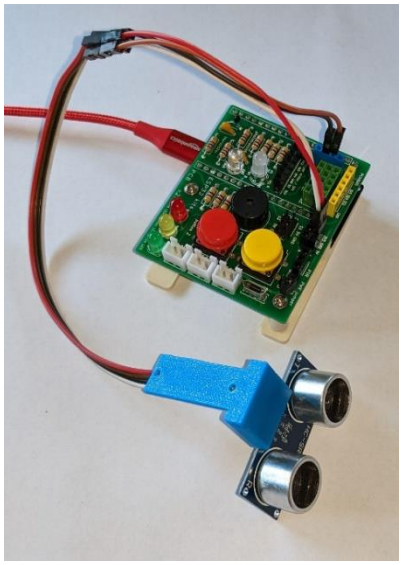


The HC-SR04**P** ultrasonic sensor shown front and back on the right (wrongly labelled on the front face!) is a newer version of the original HC-SR04. This newer version can be powered with a wider range of input voltage, i.e., from 3.0V to 5.5V – but in all other respects its operation is the same as the original.

This type of sensor can be programmed to emit a high frequency sound wave (c. 40,000 Hz which is well above the highest frequency that a human can hear) from its 'transmission speaker' (labelled T).

This sound will 'bounce back' to the module if it hits a solid object in its path and can be 'heard' by the module's 'receiving microphone' (Labelled R).

As the speed of sound in air is known, the time interval between the sound being emitted and its reflection being 'heard' by the sensor allows the object's distance to be calculated.



The sensor module has a 4 pin male connector labelled: VCC, Trig, Echo, GND - and the image on the left shows how the pins are connected to the Maker Kit's PCB using male-female jumper leads

One of the PCB's 3 pin black female PWR connectors is used to provide the 3V3 and GND connections.

Then:

- Trig is connected to the 'spare' GPIO#3 and configured as an OUTPUT pin, and
- Echo is connected to the 'spare' GPIO#35 and configured as an INPUT pin.

The continuous measurement cycle developed in the Arduino IDE then consists of the following steps:

- Trig is set LOW for 0.5 seconds and then set HIGH for 10 µseconds which initiates the sending of the ultrasound signal.
- A timer is started and stopped when the Echo pin goes HIGH, which indicates that the reflected sound has been detected – but this timing cycle is curtailed if it exceeds 0.038 seconds as that is 'deemed' to show that the 'reflection' was missed as it came back too quickly for the electronics/software to 'catch' it, i.e., an object was too near the sensor.
- Each successful timed response is then used to calculate the distance to the detected object.

The Arduino IDE developed code ***ultrasonic_detection.ino*** is available as a basic object detection demonstration and is part of the main software download.

To run the code, it should be loaded into the ESP32 from the IDE host machine in the usual way.

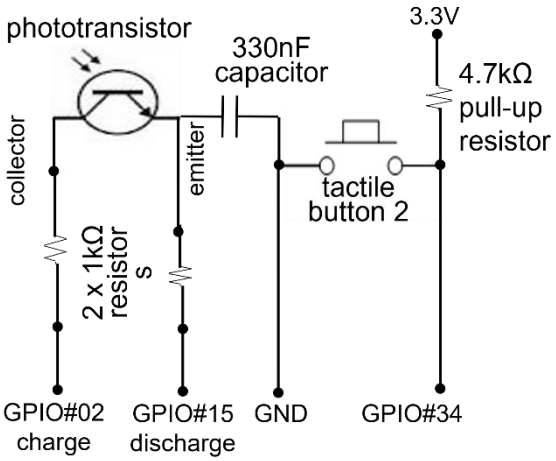
Phototransistor light sensing

The supplied Maker Kit electronic components includes a phototransistor of the type that has its output tuned to the visible light range.

The phototransistor with the (also) supplied pair of 1kΩ resistors and the 330nF ceramic capacitor are configured on the PCB (as shown above left) as a "RC charging" circuit that can be used to 'deduce' the phototransistor's resistance which can then be used as a simple 'proxy' for the prevailing light level.

The electronic component interconnections on a populated ESP32 Maker Kit PCB are as summarised below:

Component	GPIO pin(s)
Tactile button 2	GPIO#34 and GND connections across the button with a 4.7kΩ pull-up resistor included on the PCB - this button is used to initiate the light sensing cycle.
Red, Amber and Green LEDs used to indicate the progress of the measurement process	GPIO#14 connected through a 470Ω resistor to the Red LED's +'ve anode (long leg)
	GPIO#27 connected through a 470Ω resistor to the Amber LED's +'ve anode (long leg)
	GPIO#26 connected through a 470Ω resistor to the Green LED's +'ve anode (long leg)
Phototransistor	GPIO connections: collector #02 and emitter #15, each through a 1kΩ resistor with a 330nF ceramic capacitor used to form a charging/discharging 'RC circuit' triggered by a tactile button press.



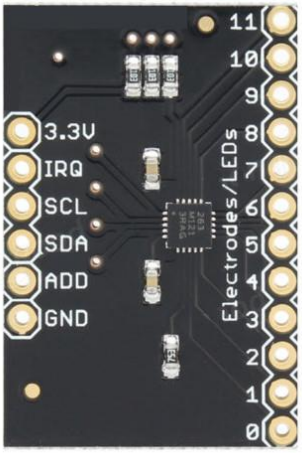
The circuit diagram shown on the right shows how pressing button 2 on the PCB can be used to initiate an 'RC charging circuit' cycle which consists of:

- Discharging the capacitor by setting GPIO#15 as an OUTPUT that is LOW
- Measuring the recharge time by setting GPIO#15 as an INPUT, GPIO#02 as an OUTPUT that is HIGH, and then 'clocking' how long it takes for GPIO #15 to go HIGH
- As this time depends on the voltage at the phototransistor's collector, which moves from high to low when the light level increases, this then increases the current flow, which decreases the capacitor charging time. A short capacitor charging time is therefore a 'proxy' for a high light level and a long time for low light level. These times could then be calibrated on a one-time basis against an accurate light meter.

A more detailed description of this 'deduction analysis' is provided in *Appendix E: RC charging circuit analysis*.

Arduino IDE developed code is available as part of the main software download to support the use of this 'RC charging' circuit and to run the ***phototransistor_button.ino*** code, it should be loaded into the ESP32 from the IDE host machine in the usual way.

Capacitive touch sensing



An MPR121 module, as shown on the left, is a breakout board for Freescale's MPR121QR2 chip providing a capacitive touch sensor controller driven by an I²C interface.

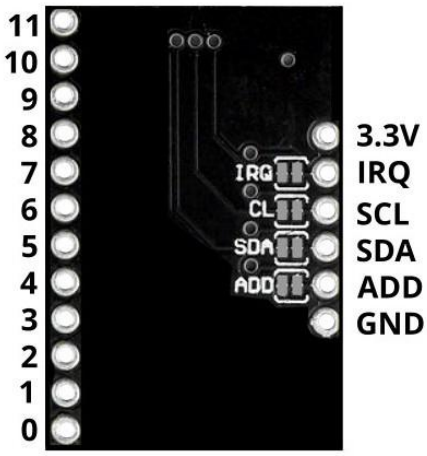
The chip and module can control up to twelve individual electrodes, as well as a simulated thirteenth electrode, and the module provides twelve input pins (0-11) which can each be used as a 'capacitive touch' input (or indeed to control up to 8 LEDs).

The module is powered by 3V3 and has internal 10kΩ pull-up resistors for the SDA, SCL and IRQ lines, so generally there are no concerns about too high a voltage level on these I²C control lines when connected to a ESP32. However, consideration needs to be given as to whether these module pull-up resistors should be 'disconnected' (see later) depending upon how many pull-up resistors are 'on the bus' at the same time and may therefore prevent sufficient 'pull' to occur to set the SDA and SCL pins high enough.

The module is connected to the ESP32 Maker Kit PCB via one of the I²C (green) 5-pin female connectors - but obviously only using the 3V3 port on the connector to power the device.

When connected to the ESP32 the default I²C address for the device is 0x5a which conveniently does not conflict with any of the other common I²C devices that may be used with the ESP32 Maker Kit, so the use of a single MPR121 device may not need to have its I²C address changed.

The device I²C address can however be changed by a process that cuts the small 'trace' between the two 'pads' next to the ADD connector on the underside of the module (see the image on the right). This removes the internal connection from the ADD pin to GND (which can obviously be restored by reapplying a 'solder bridge' across the pads).



To assign an alternative I²C address, the ADD pin is 'shorted' to a control pin as follows:

ADD's default connection to GND	address = 0x5A
ADD tied to 3V	address = 0x5B
ADD tied to SDA	address = 0x5C
ADD tied to SCL	address = 0x5D

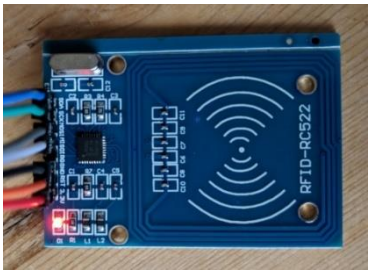
With four different addresses available this means that up to 48 capacitive touch inputs could be enabled using four separate modules.

To disconnect the module's internal pull-up resistors on any of the control pins, the small 'trace' between the two pads next to the IRQ, SCL or SDA connectors on the underside of the module should be cut (see the image above right).

Some simple Arduino IDE developed code is available as part of the main software download to support the use of this capacitive touch sensor and to run ***MPR121_simple.ino*** it should be loaded into the ESP32 from the IDE host machine in the usual way.

Radio Frequency Identification (RFID)

Radio **F**requency **ID**entification uses short range electromagnetic fields, so-called Near Field Communication (NFC), to automatically identify and communicate with 'tags' attached to objects such as key ring fobs and plastic cards. The tags contain electronically-stored information, including an individual/unique ID (UID) . Importantly the tags can be what are called 'passive', meaning that they can 'collect' the energy needed to communicate from a RFID reader's interrogating radio waves.

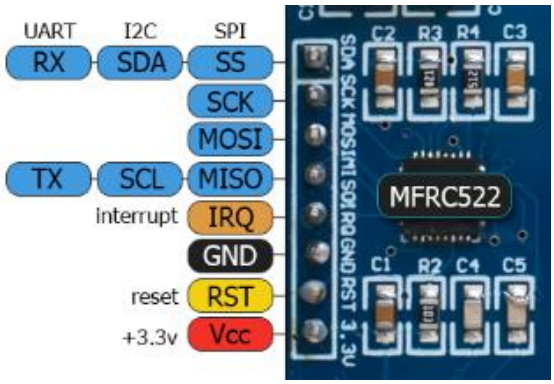


RC522 RFID reader

Key ring fob and credit card sized RFID tags

The images above show a typical RFID reader module that can be used with the Maker Kit along with a key ring and credit card sized plastic card that each have a RFID 'tag' embedded within them that can be 'read' by the RC522 RFID reader – the reader and 'tags' are all readily available but are not supplied with the Maker Kit.

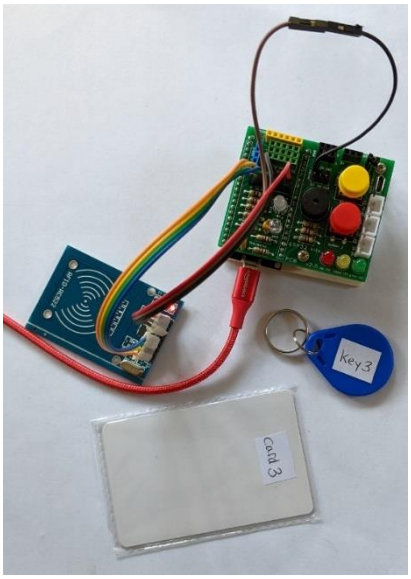
The RC522 RFID reader module has 8 connection pins, as shown on the right, labelled to imply it might be able to use a variety of protocols to connect and exchange data, i.e. simple Serial UART, I^2C or SPI.



However, these overall modules are usually hard-wired to connect to the on-board MFRC522 chip (that does all the work) so that only the SPI protocol can be used without cutting and 'messing with' the board's trace connections, which is possible, but is not covered here.

The reader is therefore connected to the ESP32 Maker Kit PCB to use the SPI protocol, as shown in the table below and in the adjacent image.

RFID reader pin	Maker Kit PCB connection
SDA	7P connector SS (GPIO #5)
SCK	7P connector SCK (GPIO #18)
MOSI	7P connector MOSI (GPIO #23)
MISO	7P connector MISO (GPIO #19)
IRQ	not used
GND	7P connector GND
RST	GPIO #32 (servo1)
3.3V	7P connector 3V3



File storage

For many projects it is useful to be able to store data files in a way that makes them readily accessible to the code running on the ESP32 with standard file management and read/write actions being possible.

The following describes some projects/methods that make this very easy.

SPIFFS file system

SPI Flash File Storage, or **SPIFFS**, is a system available on the ESP32 that uses ESP32 module memory as if it were a conventional read/write storage medium, albeit with several limitations.

More detail can be found here, but the most significant points to note are:

- Directories are not supported; only a flat file structure is used. So, a file with a path like */tmp/myfile.txt* will create a file called */tmp/myfile.txt* instead of a file called *myfile.txt* in a directory called */tmp.*
- SPIFFS can only reliably use about 75% of the assigned partition space (use the Arduino IDE Board settings to assign a fraction of the available memory to SPIFFS).
- Deleting a file does not always remove the whole file, which can leave unusable sections throughout the file system.

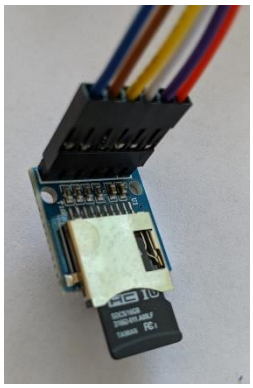
It should be noted that to initially 'upload' a set of files to SPIFFS, these can be stored in a folder called *data* that is in the same folder as the Arduino sketch, and these can be uploaded to SPIFFS using the custom 'uploader' for the Arduino IDE (e.g., see **here** for more details for a Windows installation).

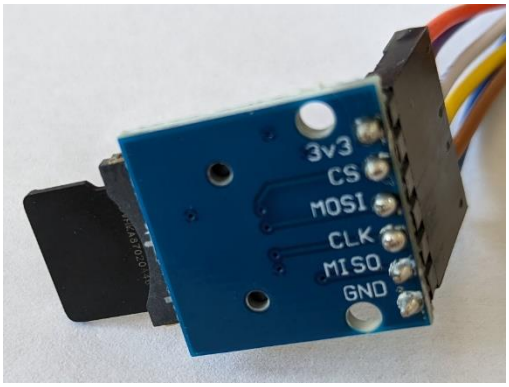
The code for several example projects uses SPIFFS so coding methods can be seen for example for the **Buzzer 'tunes'** project.

SD card reader projects

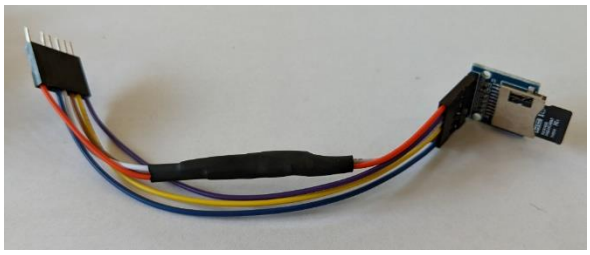
Low cost micro SD card readers are readily available and provide a simple way of providing an ESP32 project with a file storage capability.

The card reader shown on the right uses the SPI protocol for its connection so will connect to the 7-pin (black) female header on the PCB using the 3V3 power option.

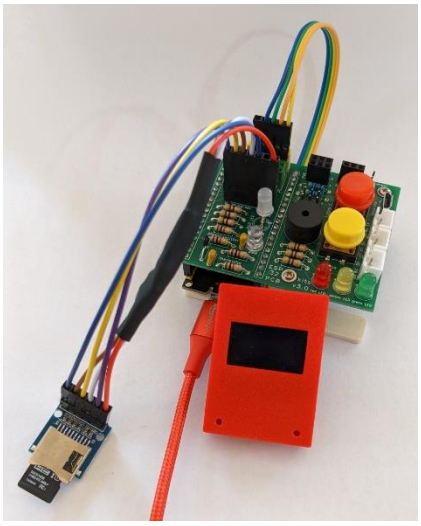




It should be noted however that more repeatable/stable usage of the SD card reader with the ESP32 was only achieved once the MOSI and MISO lines were *pulled-up* with 10kΩ resistors that were 'built-in' to the harness wiring, as illustrated in the mage on the right.



The image on the right shows a test arrangement with the SD card reader connected to the SPI header, as shown in detail in the table below, and an I^2C OLED also connected to display the various steps through the test process.



SD card reader pin	**Maker Kit PCB connection**
3V3	7P connector 3V3
CS	7P connector SS (GPIO #5)
MOSI	7P connector MOSI (GPIO #23)
CLK	7P connector SCK (GPIO #18)
MISO	7P connector MISO (GPIO #19)
GND	7P connector GND

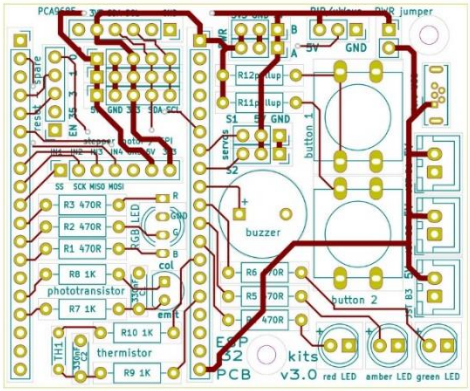
Arduino IDE developed code is available as part of the main software download to support the use of this micro SD card reader, and to run the ***SD_reader_OLED_SPI_test01.ino*** code or the simpler (non-display) ***SD_reader_SPI_test01.ino***, it should be loaded into the ESP32 from the IDE host machine in the usual way.

Appendix A: Kit development details

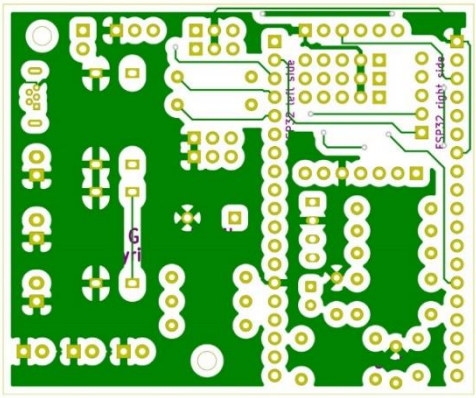
Development cycle

The **Maker Kit** now uses a v3.0 of the PCB which reflects several iterations of the design to implement refinements and functional extensions (and corrections!) from the start of the development.

The PCB design process has used the KiCAD software, and the images below show some details of the v3.0 design generated by the software.



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:

- a pair of 1x19 female headers on the underside of the PCB allows the fully assembled unit to be directly inserted onto a 38 pin ESP32 module's GPIO pins;
- to make the '*component to GPIO pin*' connections as straightforward and non-circuitous as possible, the components needed to be laid out on the front of the PCB using both the space between the 1x19 female headers and the space to one side, and as far as possible to use GPIO pins that were nearest the component and grouped together appropriately;
- a small number of GPIO pins were not needed to interconnect defined components, and these were grouped together so that they could still be used and connections on the PCB provided for them (labelled as spare pins);
- to provide external power for a PCA9685 PWM module to manage servos and to provide power for other uses, a separate 'power bus', using JST connectors plus a USB type B micro connector with a common ground to the rest of the PCB, was added so that a 4xAA battery pack, 5V rechargeable battery bank or a 5V mains power supply could be connected as 'input' to this 'power bus', and additional connectors used for outputs to various devices;
- inclusion of a 'jumper' connection between the separate 'power bus', fed by the JST and micro USB connectors, with the main 5V power lines from the ESP32 allowing the unit to be powered in a stand-alone mode - although this usage mode must take care to not use a voltage supply greater than the 5V required by the ESP32!!

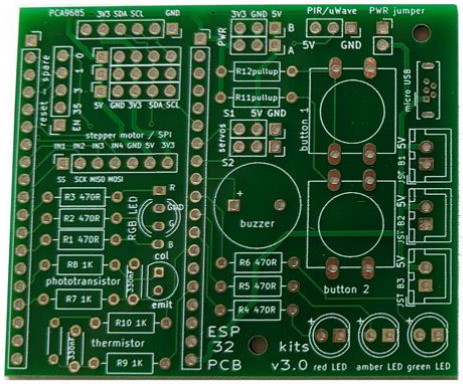
- to provide power output options from the internal ESP32 power lines, two 3-pin female connectors (PWR-A and PWR-B) were added to provide 3V3-GND-5V outputs;
- to make the connection of a pair of servos, directly powered by the ESP32 (two servos probably being a reasonable limit from the ESP's internal power) as easy as possible, two 3 pin male connectors have been added labelled S1-5V-GND and S2-5V-GND;
- whilst a dedicated 6-pin yellow female connector is used to provide I^2C control for a PCA9685 PWM control board for servo management, three additional green 5-pin I^2C female connectors are provided to allow further components to be connected to the ESP32's I^2C bus, so that yet more project developments are enabled;
- a black 7-pin connector, using various pin combinations is optionally used as:
  - a connector for a stepper motor;
  - a connector for direct drive motor controllers; and
  - as a general purpose SPI connector
- in addition, the ESP32 has a number of constraints/limitation in the way specific GPIO pins can be used - the table below summarises these and how the Maker Kit pin usage has been 'allocated' to work within these constraints.

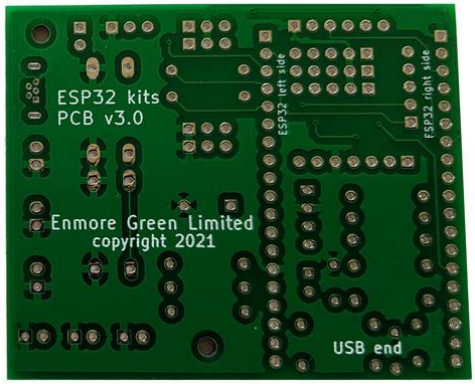
ESP32 GPIO	use as an Input	use as an Output	Notes for individual pins	Notes for ESP32 Maker Kit v3.0 usage
EN (RESET)	n/a	n/a	Enable (EN) is the 3.3V regulator's enable pin and is pulled up by default but could be grounded to disable the 3.3V regulator - which means that this pin could either be connected to a pushbutton or could be 'programmed', to restart the ESP32.	These GPIOs are only used as 'spare' pins since their general use needs 'special' attention.
0	pulled up	OK	outputs PWM signal at boot	
1	TX pin	OK	debug output at boot	
3	OK	RX pin	HIGH at boot	
2	OK	OK	connected to the on-board LED	Phototransistor RC circuit 'charging' pin
4	OK	OK		Blue colour ON/OFF pin in combined RGB LED
5	OK	OK	outputs PWM signal at boot	IN1/SS pin on 7P stepper/SPI female header
6-11	x	x	All these pins are connected to the internal integrated SPI flash	These GPIOs are not used at all and are not 'exposed' on the Maker Kit PCB
12	OK	OK	boot fails if pulled high	Thermistor RC circuit 'charging' pin
13	OK	OK		Thermistor RC circuit 'discharging' pin

ESP32 GPIO	use as an Input	use as an Output	Notes for individual pins	Notes for ESP32 Maker Kit v3.0 usage
14	OK	OK	outputs PWM signal at boot	Red LED ON/OFF pin
15	OK	OK	outputs PWM signal at boot	Phototransistor RC circuit 'discharging' pin
16	OK	OK		Green colour ON/OFF pin in combined RGB LED
17	OK	OK		Red colour ON/OFF pin in combined RGB LED
18	OK	OK		IN2/SCK pin on 7P stepper/SPI female header
19	OK	OK		IN3/MISO pin on 7P stepper/SPI female header
21	OK	OK		I^2C SDA pin
22	OK	OK		I^2C SCL pin
23	OK	OK		IN4/MOSI pin on 7P stepper/SPI female header
25	OK	OK		Buzzer control pin
26	OK	OK		Green LED ON/OFF pin
27	OK	OK		Amber LED ON/OFF pin
32	OK	OK		Servo 1 control pin
33	OK	OK		Servo 2 control pin
34	OK	x	These pins are input only	Depending upon usage these pins may need a physical pull up/down resistor since internal pull up/down is not supported. Buttons 1 and 2 (pins 39, 34) have pull-up resistor locations on the PCB that must be used. PIR (pin 36) does not need pull up/down. Spare (pin 35) is assumed to use a pull up/down in its external circuitry if it is needed.
35	OK	x		
36	OK	x		

Finalised design

The v3.0 design was finalised in November '21 with back and front views of a production unit shown below.

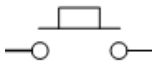




Appendix B: Maker Kit component details

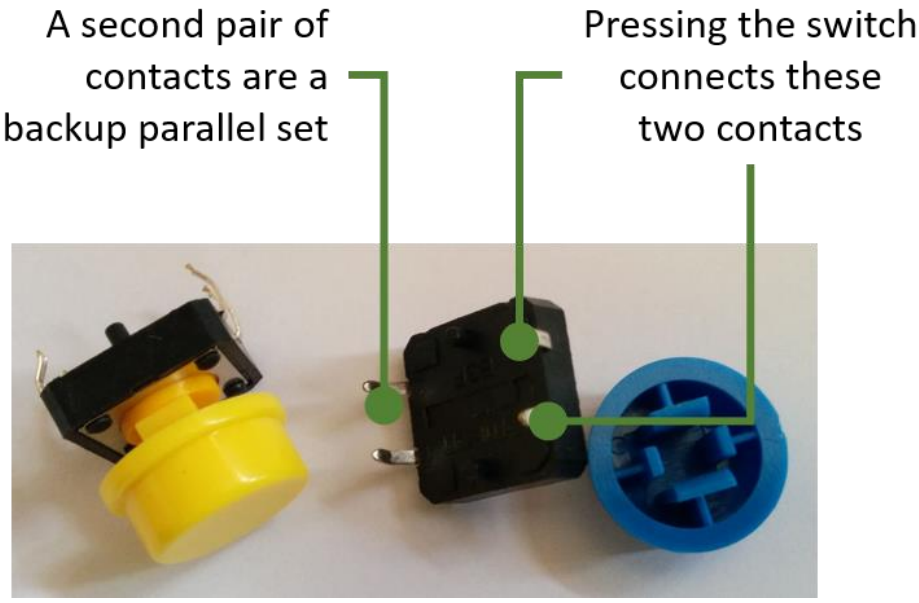
This appendix provides more detail for the various components included in the Maker Kit that are to be soldered into the printed circuit board (PCB).

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 supplied in the **ESP32 Maker Kit** 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.

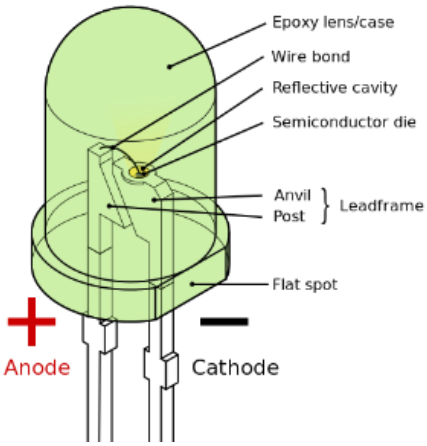


As shown above, it is the two contacts on the same side of the button that are connected when the button is pressed.

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.

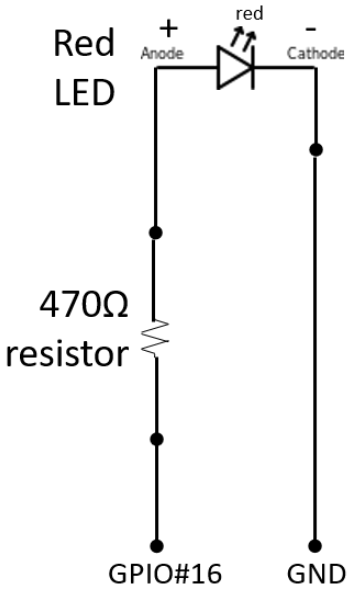


Included in the **ESP32 Maker Kit** are one each of red, amber and green coloured LEDs packaged in either a 5mm or 3mm potted plastic lenses, as shown in the diagram on the left.

There is also one so-called RGB LED which has separate red (R), green (G) and blue (B) light emitting diodes packaged all together in a single plastic lens. This allows each of the RGB colours to be independently set on or off, or to different light intensities and by 'mixing' light in this way, a range of different coloured light can be produced.

For the supplied single colour LEDs, the anode or positive power lead is longer (NB the schematic above shows a flat spot can also be used to denote the cathode side but this may not be present on the supplied LEDs). 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 is nearest the resistors for each of the LEDs. The insertion points for the long legs on the PCB are each labelled with a small + to ensure the correct orientation is used.

For the RGB LED, the negative (cathode) leads for the three colours are all interconnected within the plastic lens to a single 'common' ground lead that is longer than all the separate anode leads. The sequence of the four leads from a RGB LED is: RED, common, GREEN, BLUE so it is important that the RGB LED is inserted into the PCB holes correctly with the insertion points labelled R, GND, G and B on the PCB to ensure the correct orientation is used.



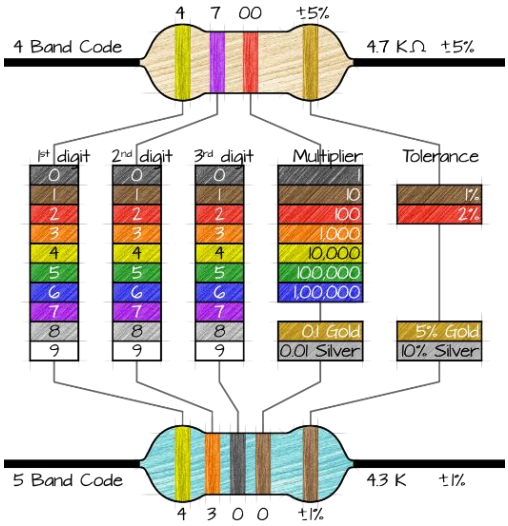
In operation, each LED type has a slightly different forward voltage drop (V_f) that should not be exceeded, otherwise its maximum current limit could be exceeded which would cause damage.

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, each of the LEDs on the PCB are therefore put in series with supplied 470Ω resistors, as illustrated in the circuit diagram on the right, with the resistor having been sized to achieve a safe level of the applied voltage for all the LED types.

Resistors

Resistors are small passive electrical devices used to provide electrical resistance measured in ohms (Ω).



The resistors provided with the **ESP32 Maker Kit** 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 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 provided with the kit are either carbon or metal film with 4 or 5 band colour coding as follows:

1kΩ resistor coloured bands:

4 band: brown–black–red–gold which means:

1 0 x100 5% i.e. 1000Ω or 1kΩ ±5%

5 band: brown-black-black-brown-brown

1 0 0 x10 1% i.e. 1000Ω or 1kΩ ±1%

470Ω resistor coloured bands:

4 band: yellow–purple–brown–gold which means:

4 7 x10 5% i.e. 470Ω ±5%

5 band: yellow–purple–black-black–brown which means:

4 7 0 x1 1% i.e. 470Ω ±1%

4.7kΩ resistor coloured bands:

5 band: yellow–purple–black-brown–brown which means:

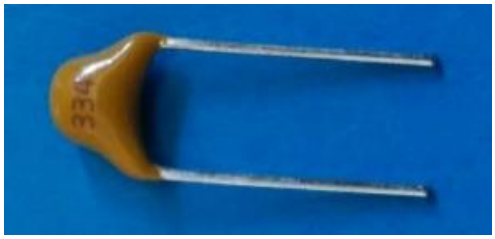
4 7 0 x10 1% i.e. 4.7kΩ ±1%

Capacitors

A capacitor is a passive electrical component, available in a range of shapes and sizes, as shown right, that is designed to store electrical energy.

As these devices store energy, if a capacitor has a high value of capacitance i.e., it can store a lot of energy, it does need to be treated with some caution since the abrupt discharging of all the stored energy can cause damage or even personal injury!

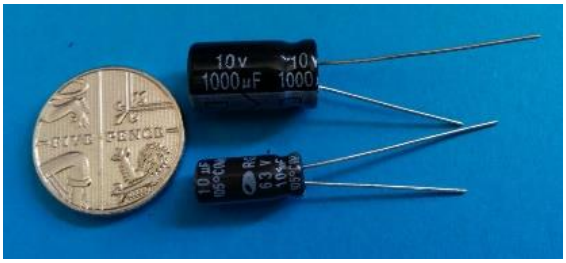
The unit of capacitance is called a farad (F) which is the number of coulombs per volt (C/V), where a coulomb is a measure of electrical charge. Capacitors used in general electronics have typical vales from 1 pF (10^{-12} F) to about 1 mF (10^{-3} F).



The type of capacitor provided in the **ESP32 Maker Kit**, as shown left, is a ceramic disc capacitor with a nominal value of 330nF (330×10^{-9} F) and it can operate up to 50V.

This type of capacitor is quite common as they are generally small and relatively cheap although they are not particularly robust.

Another common type is an electrolytic capacitor as shown in the image on the right. These capacitors must be connected with the right polarity and the longer lead will usually be the +'ve and the –'ve lead may be indicated by a stripe on the main body.



Electrolytic capacitors are generally physically larger than ceramics, as can be seen above compared to the 5p coin. They can also have higher capacitance values i.e., the two shown in the image are 1000µF and 10µF respectively, so as mentioned before they should be used carefully when being discharged.

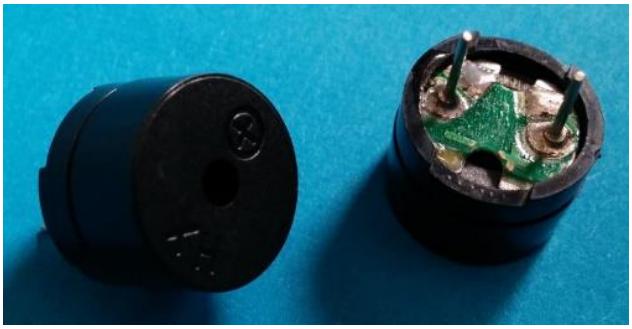
Buzzers

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

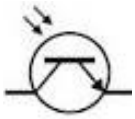
They are called piezoelectric buzzers as they use a small electromechanical element made from a material that exhibits the piezoelectric effect i.e., it generates electricity when deformed, or for a buzzer, it will deform when electricity is applied to it.

An active 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 also have a longer pin to designate the +'ve input as well as a + sign stamped on the top, and as shown above are often shipped with a small sticky label to protect the buzzer's 'sound opening'.

The **ESP32 Maker Kit** however supplies a passive 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. But does have the +'ve pin indicated on both the top and under sides.



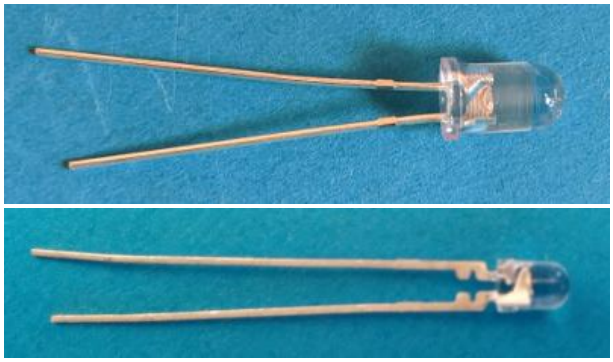
Without the additional electronics a passive buzzer requires an alternating voltage input to generate a sound, and whilst this does require more 'effort' in writing the software to achieve any sound, it means that sounds with a different pitch can be generated from the one device.

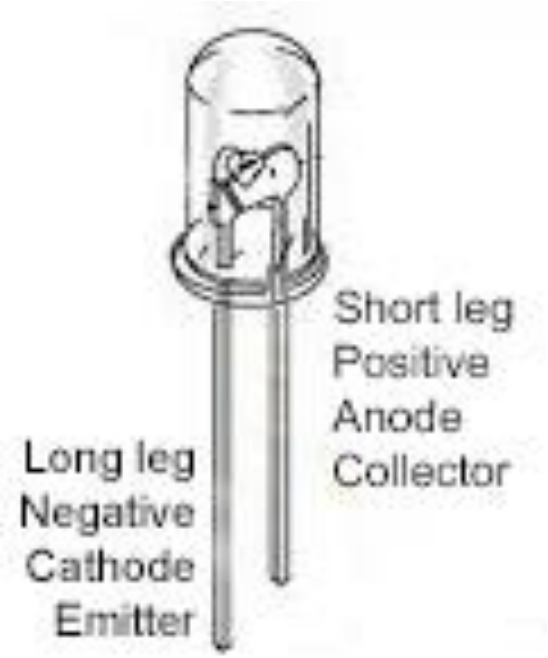
Phototransistors 

A phototransistor is a semiconductor light sensor formed from a transistor set inside a transparent cover. It can be thought of as the opposite of a light emitting diode (LED) i.e., instead of lighting up when some current flows through it, it will generate a current when light shines on the device.

In the same way that LEDs have been 'tuned' to emit light at different wavelengths (colours), phototransistors are designed to be sensitive to particular wavelengths.

The phototransistor supplied with the **ESP32 Maker Kit** will be one of the types shown right and has its output tuned to the visible light range.





Phototransistors can be easily confused with LEDs as they can be of very similar construction, so take care to correctly identify the supplied kit component which will have a transparent casing.

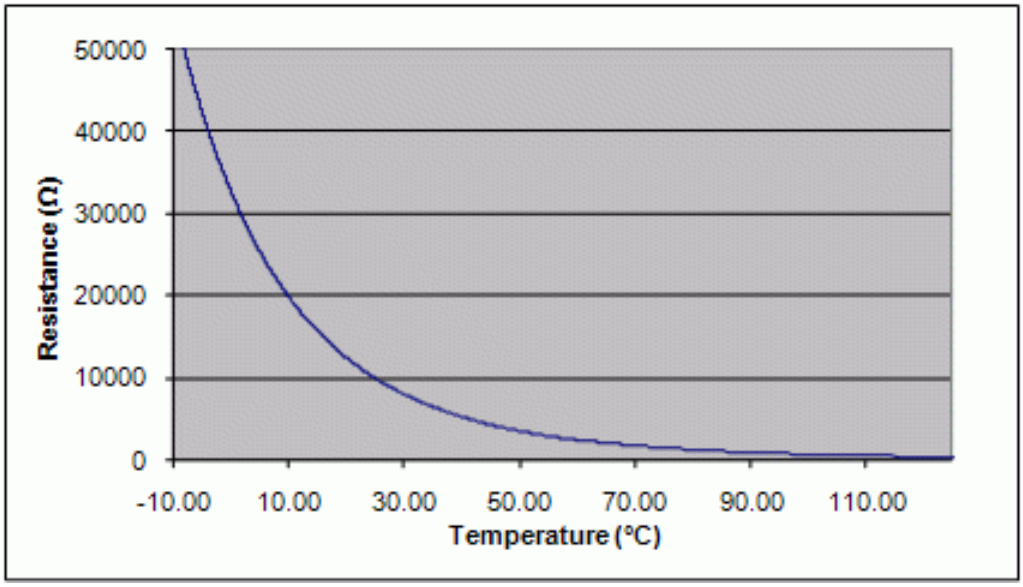
Also, as shown left, it should be noted that the longer lead is the emitter i.e., the equivalent of a –'ve or cathode connection.

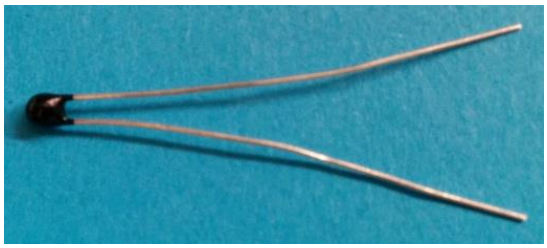
Phototransistors are not usually precise devices, so they cannot be used for any application that requires an exact measurement of light level, but with some empirical calibration they are perfectly adequate for detecting different degrees of it 'getting dark' and can be successfully used in, say, a home automation project to decide when to switch lights on or off.

Thermistors

A thermistor is a passive electronic component that, with temperature, changes its electrical resistance over a wide range and is very commonly used as: a temperature sensor, a current limiter; or a self-regulating heater element.

For a temperature sensor, a thermistor with a Negative Temperature Coefficient (NTC type) would usually be employed. This means that the resistance of the device decreases (usually quite substantially) with an increase in temperature as illustrated in the graph below.





The thermistor provided with the **ESP32 Maker Kit**, as shown left, is a small black ethoxyline resin coated ceramic bead with a pair of uninsulated tinned copper wire leads.

This thermistor has a reference resistance of 10kΩ at 25°C and is suitable for measuring temperature over a range of -30°C to +125°C.

Appendix C: Software & documentation download

ESP32_Maker_Kit_v3_support_material.zip contains all the software and documentation and can be downloaded from:

https://onlinedevices.co.uk/dl1919

- where the last set of characters are lower case DL1919

This document (*ESP32 Maker Kit Usage Documentation.pdf*) is in the .ZIP file along with other documentation, plus a separate ***ESP32_code*** folder that contains the example software.

All the software is C/C++ for use with the Arduino IDE with individual code 'sketches' available to control the components directly fitted to the Maker Kit PCB as well as a wide variety of additional components that can be connected to the PCB.

To manage code changes and to upload individual code 'sketches' to the ESP32 microcontroller, a separate computer is needed to run the Arduino IDE which can be installed on a variety of machines (Windows PC, Mac, Raspberry Pi, etc). Each sketch is provided in the folder/file structure for direct use in the Arduino IDE.

Finally, please note that the ***maker_kit5_readme.txt*** file that is downloaded as part of the documentation is a version control file that lists all the 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: Control methods and 'plug in' component details

This appendix provides information about various control methods and further details on various components that are not included in the Maker Kit but can be 'plugged in' to the assembled PCB, and for which example software is made available for their use.

Passive Infra-Red sensors (PIRs)

A Passive InfraRed sensor, or PIR sensor, is an electronic module that measures infrared (IR) light radiating from objects in its field of view. They are commonly used in PIR-based motion sensors packaged as part of alarm or automated lighting systems and have become a mass produced, commodity component in recent years.

They are described as 'passive' because they do not generate or radiate any energy of their own for detection purposes, but instead simply rely upon detection of energy given off by other bodies. Therefore, they cannot detect the movement of bodies that do not radiate heat energy, unlike an 'active' detection method such as radar or sonar which generates a signal and interprets a reflected return signal.

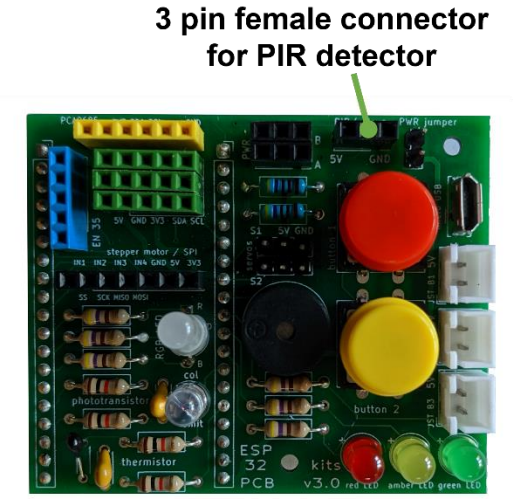
All objects that are at a temperature above absolute zero will emit heat energy i.e., they will be emitting radiation in the infrared spectrum, which is not visible to the human eye, but is easily detected by appropriate electronics. These normally comprise a solid state pyroelectric sensing element and electronics that detect a change in the overall sensed level of radiation, e.g., being caused by a warm body moving in and out of the field of view. The sensor will convert this change in radiation level to an output voltage level that can be used to trigger further processes.

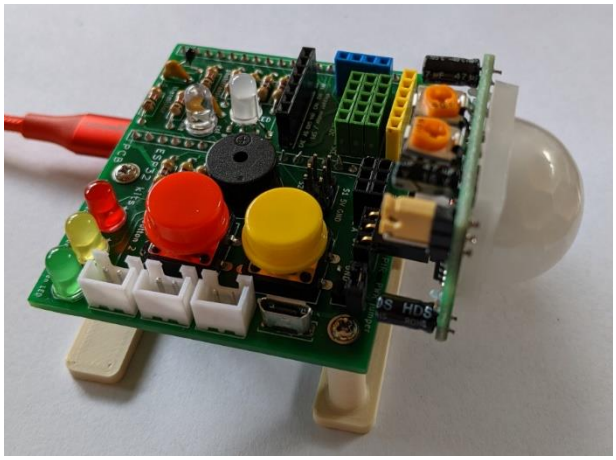
More details on PIRs can be found at this **Wikipedia link**.

A very commonly available PIR sensor module, often described as HC-SR501 but may be called something else, is shown below along with a schematic showing its various connections and ways of adjusting its performance.

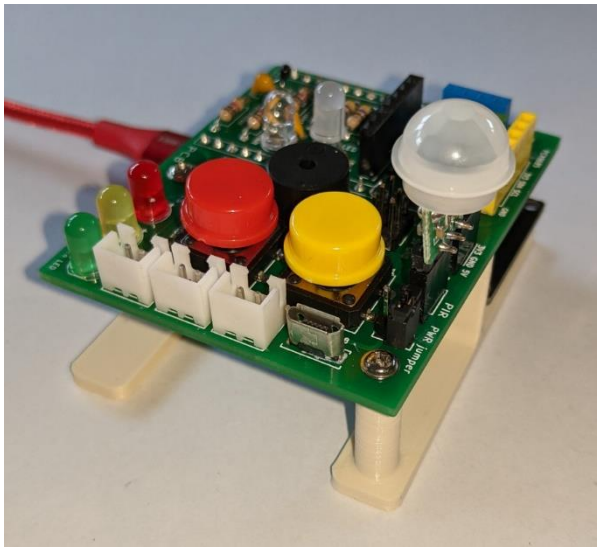
The +Power and GND pins are used to provide a 5V DC power supply to the electronics and the High / Low Output pin will provide a 3V3 'detection' signal, i.e. whenever there is a variation in the infrared radiation this pin will go HIGH.

The Maker Kit PCB has a dedicated 3-pin female connector, highlighted in the schematic below left with its connections aligned to suit the HC-SR501 PIR. Therefore, using a right angled 3 pin male-female adaptor it can simply be inserted into the PCB as shown below right (ensuring it is connected the right way round! - the PCB PIR 1st and 3rd connection points are labelled 5V and GND to ensure the correct orientation is used)





Other types of PIR are also commonly available and if they have a similarly oriented set of three conectors then they can also be directly inserted into the PCB as illustrated with the mini HC-SR312 shown below.



Servo motors

A servo motor is a packaged motor and gearbox with control circuitry that acts as a rotary or linear actuator with control of angular or linear position. Servos are used extensively in industrial control applications.

Importantly the motor has an in-built sensor that gives position feedback to its controller, so the actuator is a so-called *closed-loop* servomechanism i.e., the current position of the actuator is continuously compared to the command position and any difference is fed back to the controller so that the motor is made to move further towards its required position. A servo motor will consume power as it moves to its command position but once there it stops, unlike a stepper motor as discussed later.

The low cost SG90 micro servos, shown right, are simple servos commonly used in radio controlled models and only provide position feedback with so-called bang-bang control, i.e., the motor is either at full speed or stopped.

More sophisticated servos used in industrial control will often use both position and speed feedback and can vary both the motor speed and direction to provide more accurate and smooth control.

The SG90 servos can rotate approximately 180°, (90° in either direction) and are supplied with a selection of plastic 'horns' that fit on to the splined gear box shaft. Control of the servo is provided by applying a Pulse Width Modulation (PWM) signal to the orange signal wire, where the PWM Frequency is set to 50Hz and the middle, full left and full right positions are set by the PWM Duty Cycle being set between 1 and 2 milliseconds (ms). A fuller description of Pulse Width Modulation and its various parameters is provided next.

The starter code available for the Maker Kit provides some basic methods of controlling servos to enable you to go on and use them in projects of your own. Additional interesting ways of using servos are illustrated below (click each image to link through to online videos that are available on the web):

hexabot robots

automated marionettes

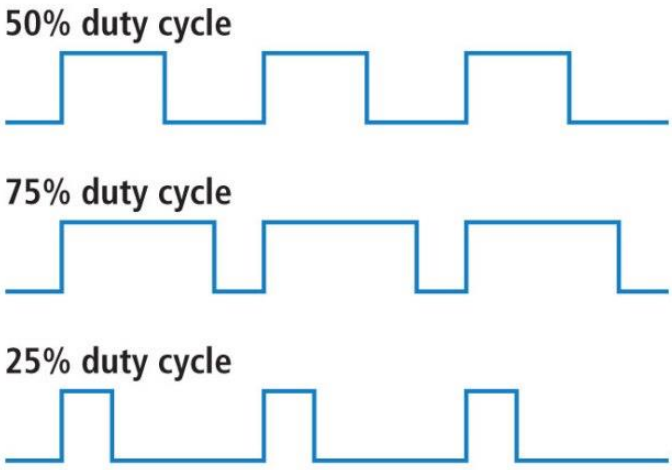
automated xylophones

Pulse width modulation

Pulse-width modulation (PWM) simply means a digital signal that is on/off (pulsed) for a period of time, where the **frequency** at which the on/off switching occurs can be set, and where the amount of time spent on, versus off, known as the **duty cycle** and expressed either as a time or as a percentage, can also be set.

PWM can be used to encode a message into a pulsing signal, but it is also commonly used to control the amount of power supplied to electrical devices, such as electric drive motors and LEDs.

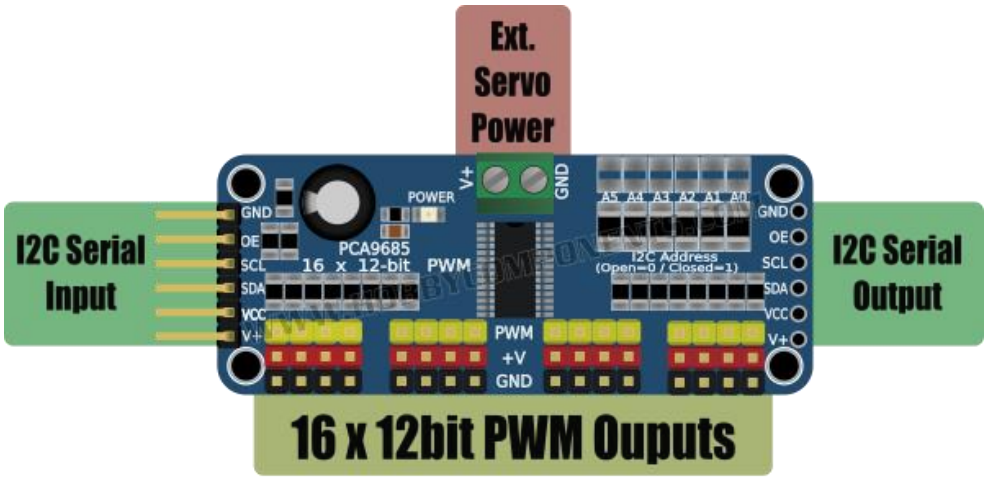
By switching the applied voltage on/off, the average voltage (and therefore current) that is applied to the electrical device is controlled by using different duty cycles, e.g., a duty cycle of 0% means no power and a duty cycle of 100% means full power. The schematic on the right illustrates some other duty cycles where the frequency is exactly the same for all three examples.



The frequency at which the switching occurs needs to be appropriate for the electrical device being powered, so that there are no negative effects i.e., the resultant wave form that is applied to the device is 'perceived' as something that is very smooth. For example, if the frequency used to control a LED was too low you might see the light varying in intensity, or for a motor it might not rotate smoothly. A typical frequency for use with a drive motor might be 50Hz, whereas you would need about 500Hz for a LED to avoid visible 'flicker'.

PCA9685 control module

The PCA9685 board is a I^2C controlled device that can apply Pulse Width Modulated power to 16 devices such as LEDs or servo motors.



As shown in the schematic above, there are six input pins, GND, OE, SCL, SDA, VCC and V+, and these are mirrored as outputs (but no pins inserted) so that additional I²C slave devices could be added downstream on the bus. GND and VCC are a 5V DC input to power the board and V+ is power input for the PWM devices if this can be supplied down the bus. The SCL and SDA pins are the two main I²C control lines and the OE (Output Enable) pin is effectively an overall on/off switch which has a default LOW setting for 'on'. For the Kit's servo methods, the V+ is not used as the microcontroller cannot support sufficient power for multiple servos, nor the OE as this is unnecessary.

There is also a pair of screw terminals to provide a separate power source for the 16 PWM controlled devices and 16 sets of 3 pins for the PWM devices, designated as channels 0 to 15 (left to right on the diagram above).

I²C bus

The I²C bus [1] was originally designed by Philips in the early 1980s to allow easy communication between components which reside on the same circuit board, and the name stood for "Inter IC" i.e., Inter Integrated Circuit, sometimes shown as IIC but more commonly as I²C.

I²C is now, not only used on single boards, but also to connect components which are linked via cable, and it is its simplicity and flexibility that make this bus attractive for many applications.

The most significant features of I²C include:

- Only two bus lines are required for the data transport and signalling:
  - SDA: serial data (I²C data line)
  - SCL : serial clock (I²C clock line)
- A simple master/slave relationship can be defined between all components on the bus with multiple masters being possible.
- Each device connected to the bus is software-addressable by a unique address
- There are no strict data exchange speeds (baud rate) requirements like for instance with RS232, as the master generates a bus clock speed
- I²C is a true multi-master bus providing arbitration and collision detection

Stepper motors

Stepper motors have many industrial uses and come in many sizes and shapes.

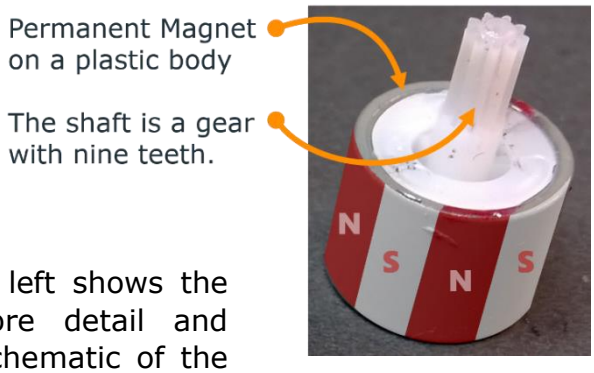
The image on the right shows a dis-assembled stepper commonly used by digital makers that is low cost and widely available.

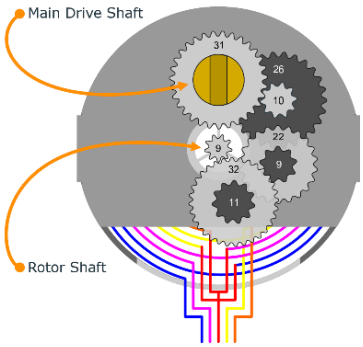
[1] *a bus is a shared digital pathway between multiple resources and devices in an electronic system where signals and data can be exchanged between everything connected to the bus using defined protocols.*

This 28BYJ-48 stepper motor's separate components consist of:

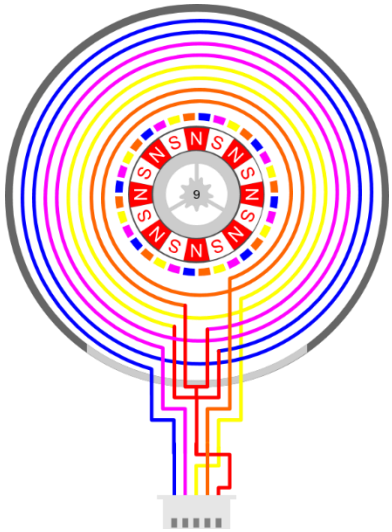
- An outer case with 8 teeth at the base and a centre post for the rotor
- Two wire coils (windings) in white plastic rings with 3 plates that also have 8 teeth
- Each coil acts as an electromagnet but they also have a common 'centre tap' so each coil has three ends or is effectively two coils and there are therefore 5 leads:
  - Blue/yellow are the ends of one coil,
  - Pink/orange are for the other, and
  - The red lead is for the common centre tap
- A rotor body with magnet wrapped around outside
- Gears that finally attach to a motor shaft that goes through a top plate which produces a 64:1 gear ratio

The image on the right illustrates the rotor's permanent magnet poles and it also shows the attached 9-tooth gear that is part of the overall 64:1 gear train.





The image on the left shows the gear train in more detail and below right is a schematic of the four coils that magnetise the 32 teeth which will align with the rotor's magnetic poles.



Rotor movement is therefore achieved by energising the coils in a specific sequence which will magnetise the metal teeth that are set around the coils, causing them to align to the rotor's magnetic poles.

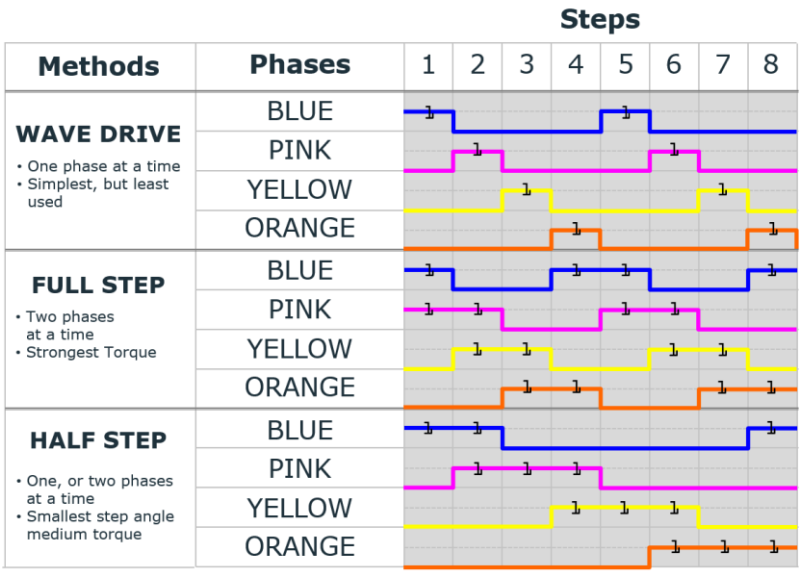
Each energising of a coil therefore produces a 'step' movement of the rotor. With 32 teeth, 32 steps will rotate the rotor one full revolution, and with a 64:1 gear train, 2048 steps will rotate the drive shaft once.

Therefore, depending upon the sequencing of the energisation of the coils, and the frequency that this is carried out, this will determine the overall drive shaft rotation amount and direction, as well as its speed of movement.

The 28BYJ-48 stepper motor plugs directly into its control board, which uses a ULN2003 integrated circuit (described in more detail in the next section), and a set of four GPIO pins on the ESP32 can then 'apply' the energisation sequencing to achieve the required stepper motor movement.

With this overall physical arrangement, several 'stepping' methods are possible which are summarised below.

A **Wave Drive** approach is the simplest method as this just sets each GPIO pin HIGH for a period of time, which therefore energises and de-energises each individual coil one at a time producing a single 'step', and after every 4-step 'sequence' the pattern is repeated to achieve continuous motion. This method, although simple, is however not used very often since the next two methods have superior characteristics.



A **Full Step** approach energises two coils at a time which means that twice the power is applied to the rotor to achieve a single step, producing a stronger drive torque. As with the Wave Drive, after every 4-step 'sequence' the pattern is repeated to achieve continuous motion.

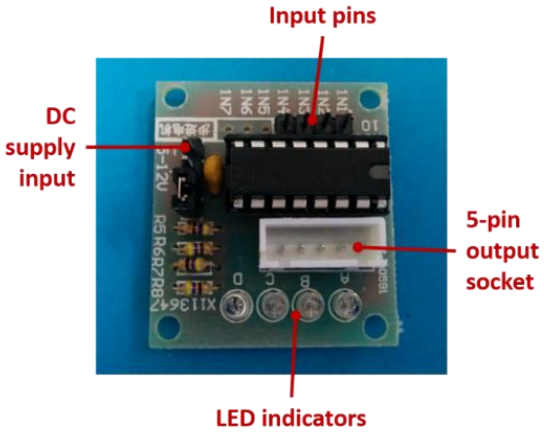
A **Half-Step** approach energises either one or two coils at one time which has the effect of creating a different single step that is half the amount moved in the previous two methods i.e., 4096 'half' steps are needed to achieve one full drive shaft rotation. This does not produce as much torque as the Full Step method, but it obviously achieves a finer resolution of movement.

ULN2003 control board and Darlington arrays

The primary component of the ULN2005 control board is a ULN2003 integrated circuit (IC) with seven high voltage and high current Darlington transistor pairs in an array.

Each Darlington pair is essentially a current amplifier and each one can be used independently to switch and amplify an input signal current to a higher level so that it can be used to drive a variety of devices such as relays, LEDs and of course stepper motors.

The control board, as illustrated on the right, packages the IC with input pins (IN1 – IN4), an output socket that the stepper motor can plug in to, LED indicators for the four input/output streams, and a pair of DC supply input pins.



Only four of the Darlington pairs are exposed with input pins – as this is all that are needed for driving the 28BYJ-48 stepper motor. These input pins are used to amplify input electrical pulses applied to the IN1 – IN4 pins, by connecting them to ESP32 GPIO pins which are set HIGH/LOW in a set sequence and frequency.

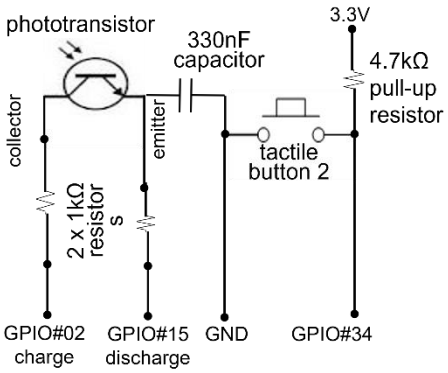
These input signals are also used to light the four LEDS (A – D). It should be noted however that for stepper motor control, the electrical pulses are usually at such a high frequency that the LEDs will appear to be permanently lit.

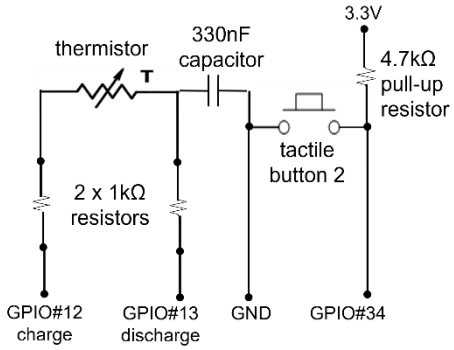
Appendix E: RC charging circuit analysis

A so-called RC charging circuit is used with both a phototransistor and a thermistor in the Maker Kit to provide approximate measurement techniques for light level and temperature.

The phototransistor circuit shown on the right connects the phototransistor to two GPIO pins via 1kΩ resistors, along with a 330nF ceramic capacitor connected between the phototransistor's emitter and ground.

This arrangement allows a simple measurement technique by successively discharging the capacitor and timing how long it takes to recharge.

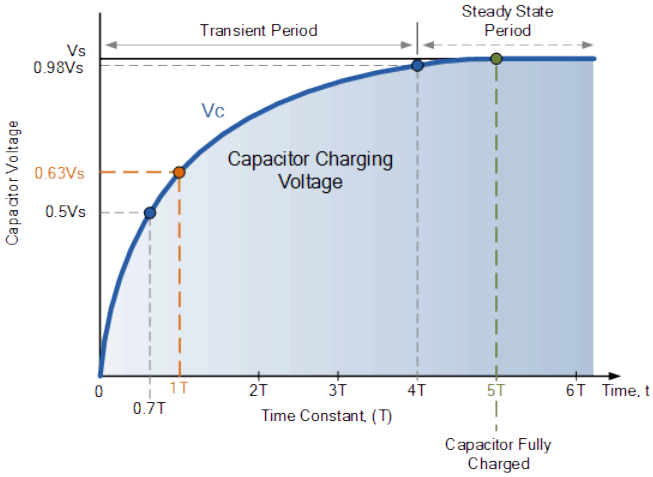




In a similar way, the thermistor circuit shown on the left connects a thermistor to two GPIO pins via 1kΩ resistors, along with a 330nF ceramic capacitor connected between one side of the thermistor and ground.

This arrangement also uses the technique of successively discharging the capacitor and timing how long it takes to recharge.

Both measurement methods rely on the capacitor charging rate being of the form as shown on the right where the time constant T (tau) equals R*C, and R is the combined resistance of the 1kΩ resistor and either the phototransistor or the thermistor, and C is the capacitance of the ceramic capacitor.



A next assumption is that when charging at 3.3V the sensing GPIO pin will go HIGH at about 1.25V which is when the capacitor will stop being charged and is therefore the time that is measured t_m.

A further assumption is that the initial portion of the curve up to 0.7 T can be approximated by a straight line so T (tau) can therefore be calculated as a simple ratio:

$$T = (t_m * 0.5 * 3.3) / (1.25 * 0.7)$$

So, the combined resistance is therefore T/C from which the individual phototransistor or thermistor resistance can be calculated.

For a light level measurement the phototransistor resistance can be used as a simple proxy but once the thermistor resistance is known the temperature can be deduced from a relationship such as the Steinhart-Hart equation (see **this ref**).

END OF DOCUMENT