

## BUILD GUIDE

### LOFT STAIRS LED STRIP PROJECT

This is the build guide for the project. This guide assumes some basic understanding of electronics, how to solder, and use an IDE with an ESP32 to edit and upload code.

I'd say it's around mid-difficulty – so whilst I've tried to be as comprehensive as possible in this guide, if you haven't got the knowledge mentioned above, it might be a bit of a slog.

It features a parts list and instructions on how to assemble the project.

For now, the current version will focus only on the vero/strip board version as that's the only one I've built so far.

As I build and improve the PCBs, I will update the build guide with PCB specific info.

### Parts List

This project will require the following components:



**Figure 1:**ESP32 WROOM dev board



**Figure 2:** USB plug and cable



**Figure 3:** RGB LED STRIP  
- Non-Addressable



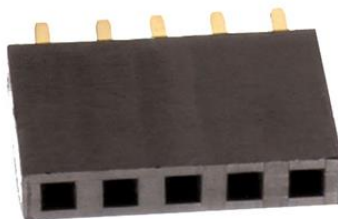
**Figure 4:** 3x 220R RESISTOR



**Figure 5:** 3x 10K RESISTOR



**Figure 6:** 3x MOSFET IRFZ44N



**Figure 7:** 6x 5-WAY SOCKET HEADER (2.54mm PITCH)



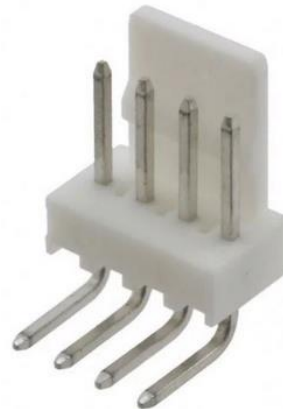
**Figure 8:** 2x 3-way MOLEX KK CONNECTORS



**Figure 9:** 2x 3-way MOLEX RIGHT ANGLE PCB HEADER



**Figure 10:** 1x 4-way MOLEX KK CONNECTOR



**Figure 11:** 1x 4-way MOLEX RIGHT ANGLE PCB HEADER



**Figure 12:** 10x crimp wire pin terminals



**Figure 13:** 1x 2.1mm DC SOCKET



**Figure 14:** HOOK-UP WIRE



**Figure 15:** VEROBOARD:  
X-Axis: 30 holes (across) – Y-Axis: 23 rows (down)



**Figure 16:** 2x HC-SR501 PIR sensors



**Figure 17:** 2x PIR sensor enclosures



**Figure 18:** LED strip mounting kit with diffuser

## TOOLS & ACCESSORIES

### You will also require the following tools:

- Soldering iron and solder
- Wire strippers
- Wire cutters
- Crimping tool
- Drill and drill bit
- Screwdriver
- Strip board cutting tool
- Blu-Tack / Sticky Tabs / Sticky Backed Velcro – to mount the sensors and PCB.
- Arduino IDE and relevant libraries
- Laptop/PC
- Cable ties/clips and/or cable tie mounts

### Optional Accessories:

- **De-soldering tool:** To remove solder when re-doing a connection.
- **Magnifier:** Really helpful for checking part values and soldering work.
- **Extractor fan with filter:** Or at least open a window so the solder fumes can escape!
- **Breadboard:** Use to prototype the circuit beforehand if you're not feeling brave enough to commit to PCB right away.
- **Helping Hands:** To hold the components or PCB in place as you solder them.
- **Jumper & DuPont cables:** To connect components on the breadboard to external components such as the PIR sensors.

## PROGRAMMING THE ESP32

First thing's first, before we build the PCB we need to add some code to the ESP32 so it can control the LEDs once it's integrated into the circuit.

See the **README file** in the GitHub repository for details on how the code works.

Simply put, the PIR sensors activate the LEDs when triggered, and the LEDs will light up with a different colour and brightness depending on the time of day.

I wanted to have enough light to see, but not blast my eyeballs out late at night when it's dark and I'm all bleary eyed, so some variation is required.

Some examples I tried:

Late night: Red, 40%

Early morning: Green, 70%

Day time: White, 100%

Evening: Green, 85%

Late Evening: Orange, 70%

I set time ranges for these settings using the logic "if the time was between "X" and "Y" hours, then activate LEDs using "Z" settings."

The code also accounts for daylight savings time changes between UK BST and GMT, and will activate the LEDs according to a different set of time ranges and colour/brightness settings.

Finally, it also runs a sort of POST routine on start up where it will cycle through Red, Green, and Blue, to confirm each colour channel is working, and then cycles through all the preset colour and brightness values in the code to confirm each setting is working too.

### Installing the code:

- Go to the code section of the repository on GitHub.
- Open the \*.ino file that contains the code for the project.
- Copy the code from the repo into your Arduino IDE or other suitable environment.
- Connect your ESP32 board to your PC with a suitable data capable USB cable.
- Ensure you have the correct board and port settings in your IDE.
- Click "Upload".

### Confirmation:

When updating the code on the assembled system, keep an eye on the LED strip when finishing an upload. When the IDE confirms a successful upload and execution of the code, it will execute its POST routine and cycle the LED strip through all the different settings. This will also be detailed in the serial monitor window so you can check to make sure the colours on display match the colours listed in the monitor.

This is a sure fire way of confirming the code is working and LEDs are wired correctly.

You can then double check by triggering one of the PIR sensors and checking the colour against the time of day.

### Pro Tip:

On start-up, if you realise you've mis-wired the sensors or LED strip to the wrong pins, you can simply re-define the pins in the code and save yourself the hassle of de-soldering and re-soldering the connections. Assuming you've got them wired to a suitable pin in the first place, that is.

### Wi-Fi Updates:

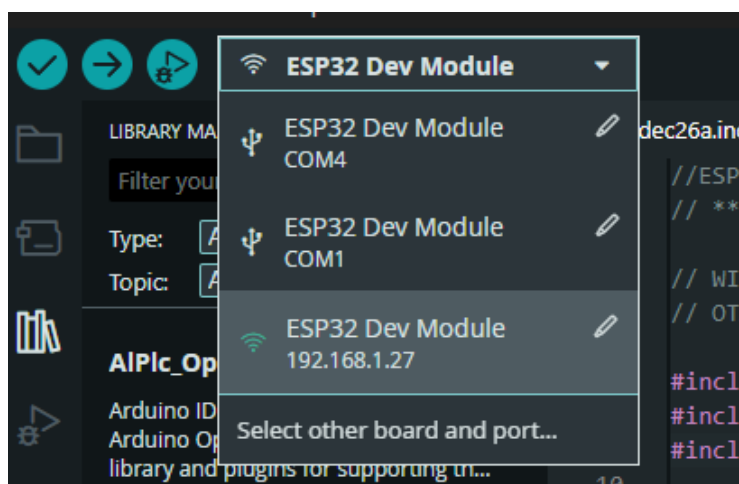
The code also includes **OTA (Over The Air)** update functionality.

Once you have uploaded the functioning code via USB for the first time, you can update the code via your Wi-Fi network and dispense with the USB cable for future updates.

Great for discreetly mounted/hidden enclosures that would be a pain to run a cable to.

Simply find the line in the code that says something like "OTA password", and change that to a password of your choice. You'll need to enter that password when updating the code over Wi-Fi.

Then select the Wi-Fi option from the drop down menu in your IDE, and click Upload.



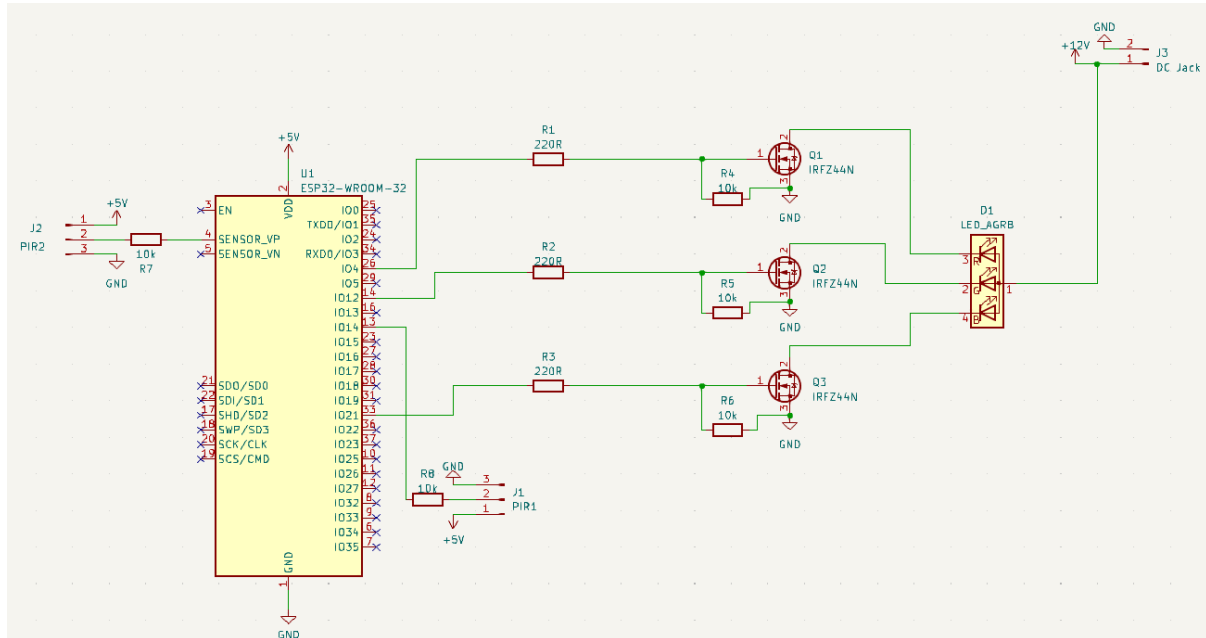
# THE CIRCUIT

Now we've got the code installed on the ESP32, it's time to start work on the circuit design.

## SCHEMATIC:

Firstly we need to establish the correct connections for the circuit to work.

We will do this by creating a schematic diagram so we have something to work from and guide us when we're putting the physical circuit together later.



**Figure16:** Schematic diagram of the circuit

The above diagram shows the components involved and how they will be connected to make the circuit work how we want it to. I matched the pin connections in the schematic to the pin definitions in the code to make sure there were no discrepancies that could trip me up later.

**NOTE:** See the GitHub repository for an image of the schematic as well as a KiCad file you can edit to your own ends.

## **Components and their functions:**

- **ESP32 Dev Board:** The brains of the operation. The chip is programmed with code to control the LEDs. Components will connect to relevant pins on the board to allow them to receive input from sensors, and send commands to the LEDs. Powered using a suitable USB cable and plug.
- **LED Strip:** A basic, non-addressable RGB LED strip with its own power supply. You *can* use a buck converter or third party power supply but using the supplied PSU makes it a bit simpler as you don't have to do any maths!
- **2.1mm DC Socket:** Used to supply power to the circuit from the LED strip's PSU, which we will modify later so it can connect to the socket.

- **PIR Sensors:** I'm using HC-SR501 PIR sensor units. These feature a PIR sensor mounted on a PCB with some controls for on time and range/sensitivity. I have them both set to low as the ESP32 will be handling the timings. The central pin (data) goes to the relevant pin on the ESP32, the other pins go to the PWR and GND rails.
  - **10k Resistors (1):** The first set of 10k resistors go between pins 1 and 3 of the MOSFETs. These act as current limiters to protect the MOSFET from excessive current draw.
  - **10k Resistors (2):** The second set of 10k resistors go between the data pins of the PIR sensors and the relevant input pins on the ESP32. They act as a pull down resistor, stabilising the sensors' output signal and stop them from being triggered by small signals/background noise. I omitted them in the veroboard version as they didn't feel necessary for the setup I was using.
  - **220 R Resistors:** The 220R resistors go between pin 1 (Gate) of the MOSFETs, and the relevant pins on the ESP32, and limit the current coming into the Gate. They work with the 10k pulldown resistors to create a simple gate driver circuit.
  - **IRFZ44 MOSFETs:** N-channel MOSFET in a TO-220 package. The ESP32 can only output signals up to 3.3 Volts, but the LED strip requires a substantially higher voltage. The MOSFETs act as a signal booster to send the required signal level to power the strip. These are especially handy as they have a pin spacing of 2,54mm so they fit perfectly onto vero board without having to bend the legs.
- 1) Gate (middle pin)** connects to relevant control pin on ESP32,  
**2) Drain** connects to relevant LED colour negative lead,  
**3) Source** connects to GND. Also connected to Gate via 10k resistor.

One MOSFET per colour channel; one each for the red, green, and blue LEDs. If we were building a simple "white LED only" system, we could get away with one, but as this is an RGB system, we will need 3 of them.

Now you have all the parts, code, and schematic together, you can take this opportunity to prototype the system on your breadboard if you want to see how everything goes together "in real life".

You will need to make use of your DuPont cables for this so you can connect the HC-SR501 PIR modules to the bread board. The jumper cables mentioned in the parts list will also come in handy as you can use these to link components and breadboard rails, saving you the extra work of cutting, stripping, and bending a load of hook-up wire.



## Vero Board Layout:

OK, so you've bread boarded your prototype version of the circuit using the schematic, and got all the bits connected and working as they should. Nice one!

Now it's time to commit the circuit to a PCB, in this case we will be using a veroboard layout to confirm functionality before we go through designing a PCB we can send to a fab house like PCB Way and spending money we don't need to.

First we need to design a layout, I've used DIY Layout creator for this as it's dead simple and really great for vero layouts.

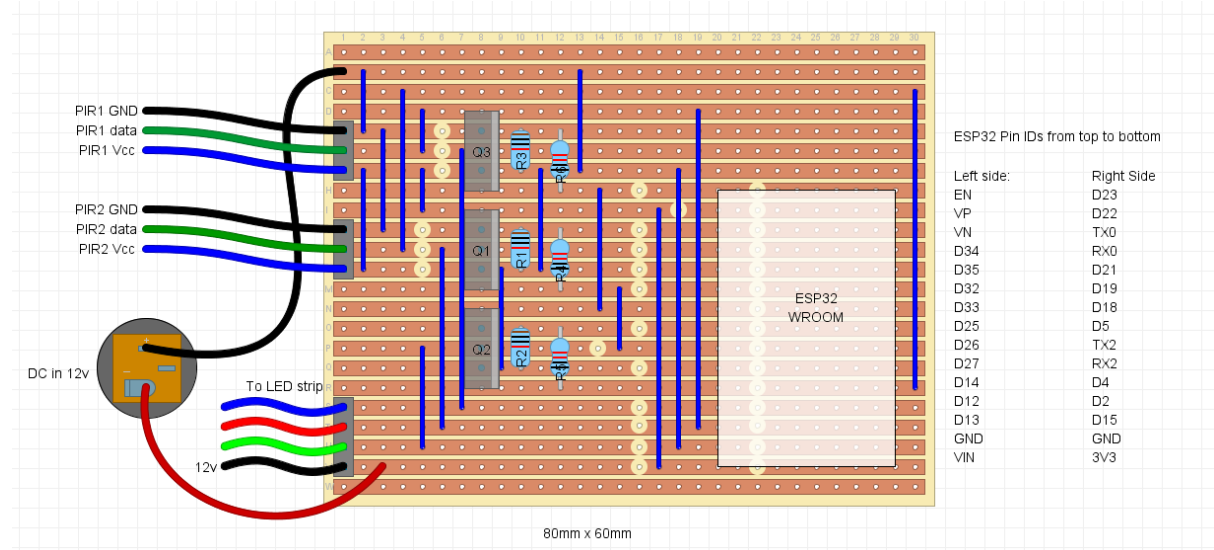


Figure 8: I'm sure I could get this down to a smaller size but it'll do.

### **Note how the PWR and GND wires are reversed on the DC socket.**

This is because the PSU on this LED strip is wired "centre positive", meaning the pin in the DC socket will receive the +ve signal from the PSU, and the sleeve goes to GND. This is the opposite of how these sockets are usually wired. Be sure to confirm the polarity of your PSU before wiring up.

- Take the veroboard and cut to size – 30 holes across and 23 rows down.
- Add a couple of extra rows and holes at each end if you want to use mounting standoffs to secure your board in place.
- Use a stripboard cutting tool to cut the tracks in the locations specified.
- Install the wires and components in order of how tall they are, so:  
Jumper wires first, then...  
Resistors, female PCB sockets, Molex pin headers, MOSFETs,
- Mount the ESP32 on the board using the female PCB sockets.
- Solder the DC socket to the PWR and GND rails.
- Measure, cut and strip wires for the Molex connectors.
- Use the crimping tool to terminate the wires with a pin connector.
- Install terminated wires into the Molex connectors.
- Solder the other end of the wires to the PIR sensors and LED strip per the schematic and layout diagrams.
- Connect the Molex connectors to the relevant pin headers on the board.

Before you can connect the power to the board, you will need to modify the LED strip and PSU a little.

- The PSU has a barrel connector which connects to the integrated controller unit that it comes with from the factory. We don't need that control unit so snip the cable and remove that altogether.
- Strip the unconnected ends of the wire where the controller used to be.
- Add heat shrink to the wire.
- Snip off the barrel connector from the wire, leaving enough that you can comfortably strip the ends and have plenty of bare wire ready to solder.
- Twist and solder the two sets of stripped wire together being careful to maintain polarity.
- Slide the heat shrink over the joint and apply heat to seal the connection.

You now have a fully assembled LED stair lighting control system that is wired, coded, powered, and ready to go. Fire it up and give it a test to ensure the wiring is correct and it functions as expected.

Adjust connections and reset the board as needed to check correct functionality.

Once this is all assembled and confirmed functional, you can source or design a suitable enclosure using some CAD software and a 3D printer to house your project.

For now I'm going to leave mine bare bones as I'll be creating a proper PCB for it down the line, and will create a suitable enclosure for that when the time comes. No sense in doing the work twice.

## MONITORING:

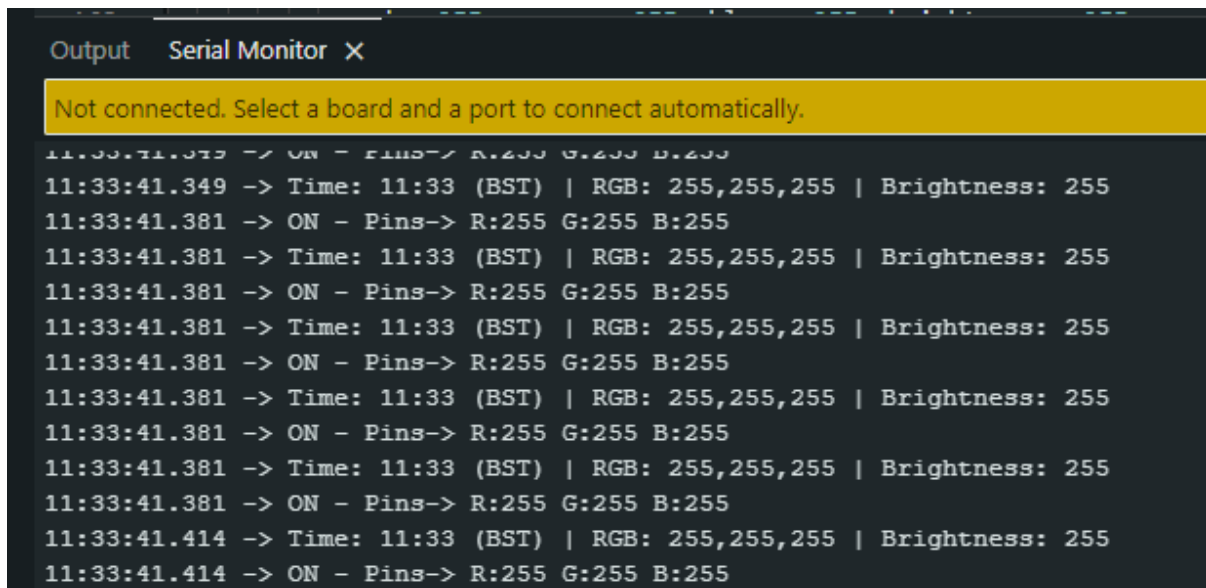
Monitoring allows us to keep an eye on the system to see what it is doing, and make sure it's working correctly. Thanks to the debugging lines added into the code in the \*.ino file, each time something an event occurs, it will be logged and can be viewed using a suitable monitoring system.

There are 2 main ways to monitor the system; wired connection and Wi-Fi.

### USB Connection:

With a USB cable connected to your computer, you can use the Serial Monitor window in your IDE to see a log of the actions occurring when the system is active. It will show you the following:

- Event time stamp
- Time registered on device
- Time zone
- Event type
- RGB values
- Brightness values
- etc



The screenshot shows the 'Serial Monitor' window in an IDE. At the top, there is a yellow status bar that reads 'Not connected. Select a board and a port to connect automatically.' Below this, the serial output is displayed in a monospaced font. The log consists of alternating lines of event types and system status reports. The event types are 'ON - Pins->' and 'Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255'. The 'ON - Pins->' lines show RGB values of R:255 G:255 B:255. The 'Time' lines show the current time as 11:33 (BST) and the system status as RGB: 255,255,255 | Brightness: 255. The log ends with the time 11:33:41.414.

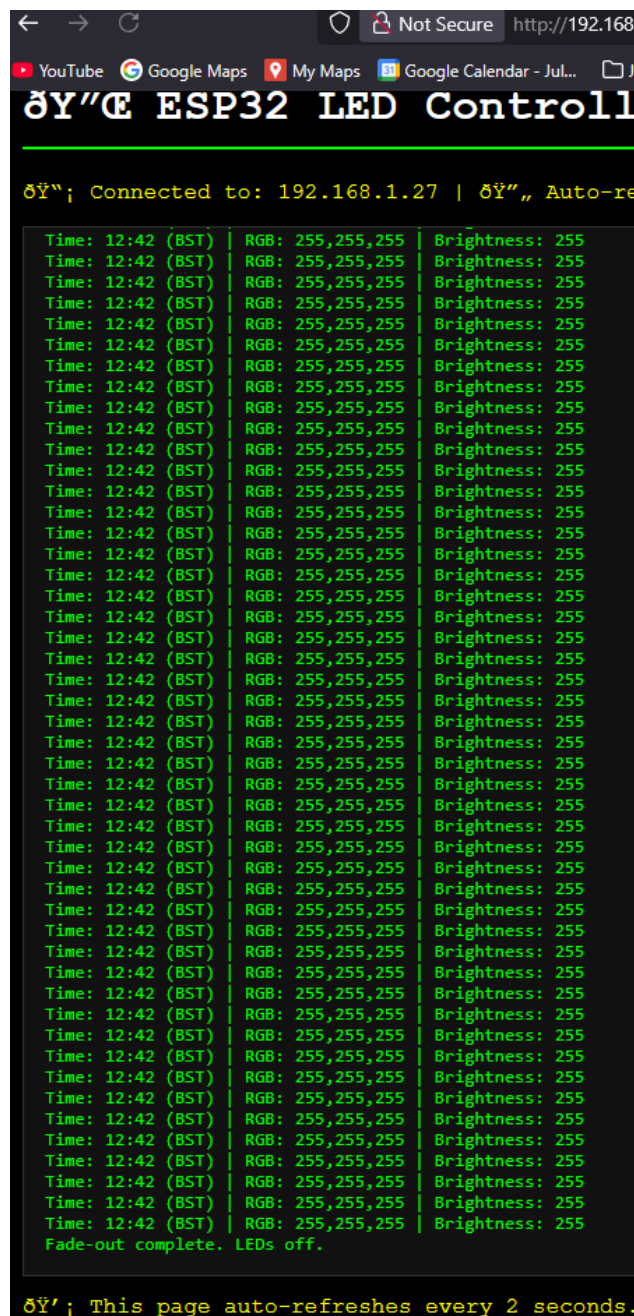
```
Output  Serial Monitor X
Not connected. Select a board and a port to connect automatically.
11:33:41.375 -> ON - Pins-> R:255 G:255 B:255
11:33:41.349 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.381 -> ON - Pins-> R:255 G:255 B:255
11:33:41.381 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.381 -> ON - Pins-> R:255 G:255 B:255
11:33:41.381 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.381 -> ON - Pins-> R:255 G:255 B:255
11:33:41.381 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.381 -> ON - Pins-> R:255 G:255 B:255
11:33:41.381 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.381 -> ON - Pins-> R:255 G:255 B:255
11:33:41.414 -> Time: 11:33 (BST) | RGB: 255,255,255 | Brightness: 255
11:33:41.414 -> ON - Pins-> R:255 G:255 B:255
```

## Wi-Fi Connection:

If you want to mount your device in a hidden or discrete location that isn't suitable for a wired connection, or just to simply free up another USB socket on your PC; you can get rid of the USB cable after you've run the first upload and just use the OTA functionality from there on out. As you won't have a wired connection to your IDE any more it won't be possible to monitor the device in your IDE's Serial Monitor window, but you can use a web browser and the ESP32's IP address to monitor the readouts over the WiFi network instead.

Simply put <https://> followed by the IP address of the device you want to monitor into the address bar and hit Enter. You'll get something like this when it records some events.

I found it quite handy having the window up in a browser on my phone whilst testing.



## INSTALLATION

All that's left is to install the LED strip on your stairs.

You'll need your mounting kit, drill and drill bit for this part.

Decide where you're going to place the LED mount. On my stairs, I had three options:

- Along the wall just above the stringer (left side)
- Same as above but on the right side
- On the underside of the bannister.

I decided to go for the left side as I'm looking down from the top of the stairs. This was for two main reasons really:

- The left side is shorter as the staircase is curved at the bottom, this means I can use less LED strip and less of the mounting kit, leaving more spares for future builds.
- The curve on the right side of the stairs would also make it difficult to run an unbroken LED strip tidily along the full length of it.

Decide what works best for your particular stair case.

### **Mounting and Wiring:**

- Drill some suitable holes in the wall at suitable intervals just above the stringer at the side of the stairs.
- Use the brackets to guide how far above the stringer to drill. I drilled 3-4 holes per mounting strip should be fine.
- Insert the wall plugs that come with the mounting kit into the holes you've drilled.
- Secure the brackets to the wall with the screws in your mounting kit.
- Run the cable connecting the board to the PIR sensor at the far end of the stairs through the mounting brackets so it will sit tidily behind the mounting strips when installed.
- Clip the mounting strips into the brackets, measuring and cutting the strips to size as needed for your particular staircase, and making sure they're all properly aligned.
- Peel off the backing tape from the LED strip, exposing the adhesive on the underside.
- Put the LED strip into the mounting strips, pressing down along the length of the strip to help the adhesive stick it in place.
- Once you reach the end of the mounting strips, remove excess LED strip by cutting the strip across the copper connectors you'll see spaced regularly along the strip.

- Install the diffuser strip by clipping it into the mounting strip, measuring and cutting to fit your particular staircase.
- Mount the PIR sensors at either end of your stairs using either blu-tack, sticky pads, or sticky back Velcro. Have them mounted on the same side of the stairs that you approach from, facing away from the direction you approach.  
That way they will not be triggered unless you are right next to the stairs.  
Having them mounted facing toward where you approach from risks them being triggered unnecessarily from a distance when you don't want them activated.
- Tidy up the wiring. Tape up the trailing wires and use some cable ties and anchor mounts, or cable clips with pin nails to route the wires from the sensors and LED strip to the board.

**NOTE:** Make sure to locate your board in such a way that you have enough wire length for all sensor and LED connections, and that is reasonably close to a power socket so you can connect the LED PSU and the plug and USB lead for the ESP32.

## Further Modifications

You can expand on this project in many ways. Some I have in mind for future builds are:

- THT PCB – smaller footprint than vero layout.
- SMT PCB – potentially even smaller footprint; much easier to 3D print an enclosure.
- Enclosure – pre-formed enclosure drilled accordingly, or bespoke 3D printed version.
- Integration with Google Assistant – voice commands etc
- Web based control panel – change/control settings from a remote device
- Adding solar charging/power capability