Grab an attendance sheet

6.900

Lecture 8

March 5, 2026

THIS WEEK

What happened in the past…and what can we learn from it?

- Retrospective on prior year designs
- Enclosure design
- **Design for success** ← HERE

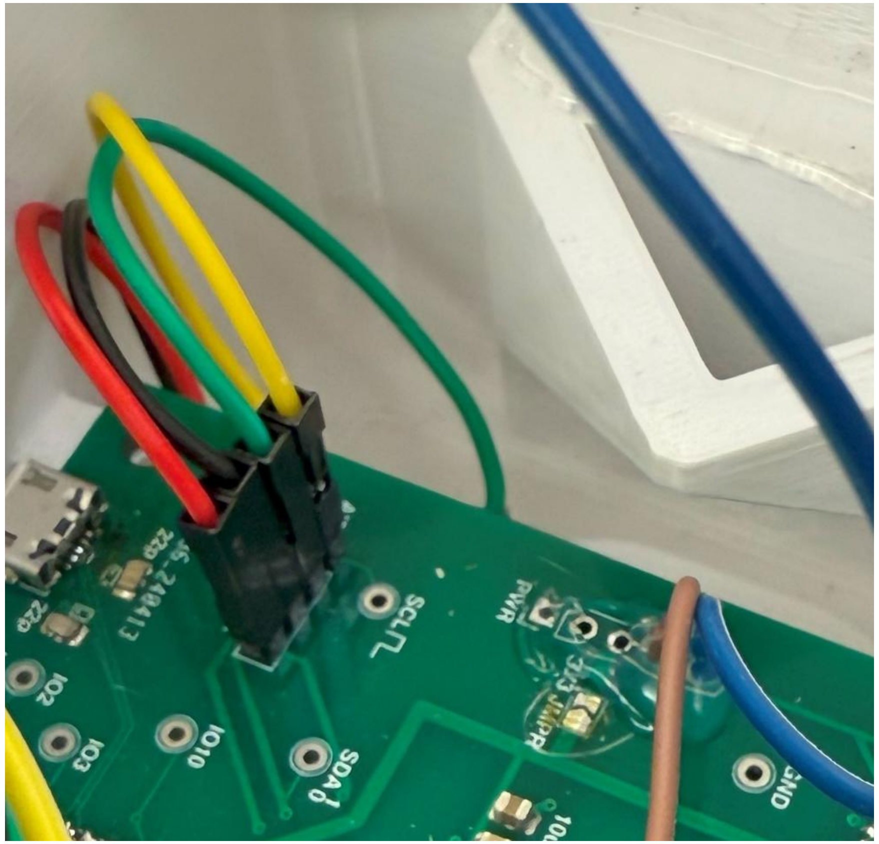
We are going to be discussing things that didn't work well last year. This isn't meant to insult to last year's teams…

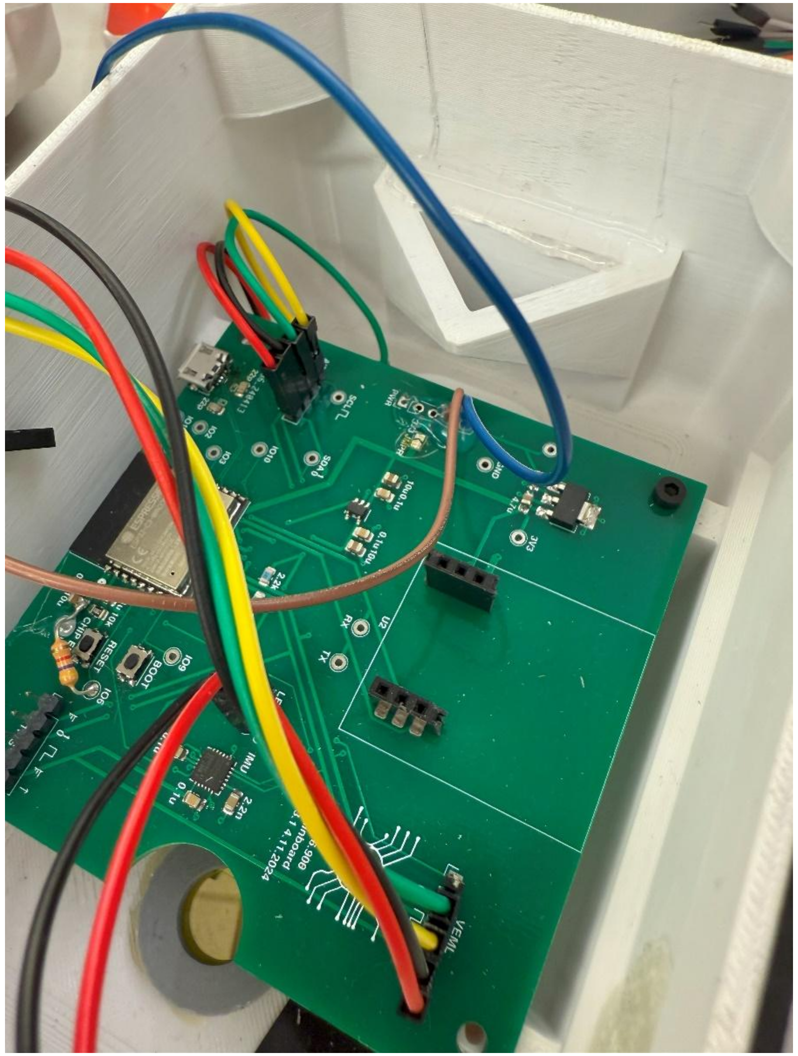
…but we want to build on their work

Next year we'll be doing the same for your teams…

Electrical connections

Don’t use hookup wires + header pins





Electrical connections

Use cables + connectors instead

More robust

Easier/faster/more reliable assembly/disassembly

Electrical connections

Common cabling/connector specs

- Number of conductors
- Voltage & current ratings
- Frequency
- Temperature
- Pitch
- Plating
- Insertion force
- SMT, thru-hole, panel mount, etc.
- Locking, polarized, keyed

connectorbook.com
Amazing website

Some cable approaches need special tooling to create Beware!

Nothing in our projects is too crazy. Take the win!

Disambiguating connectors

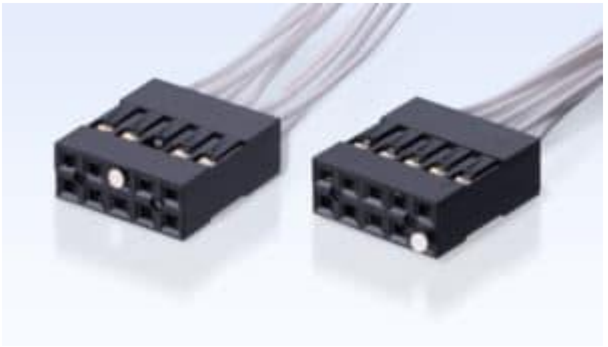
If you have 12 2x2 connectors in your system

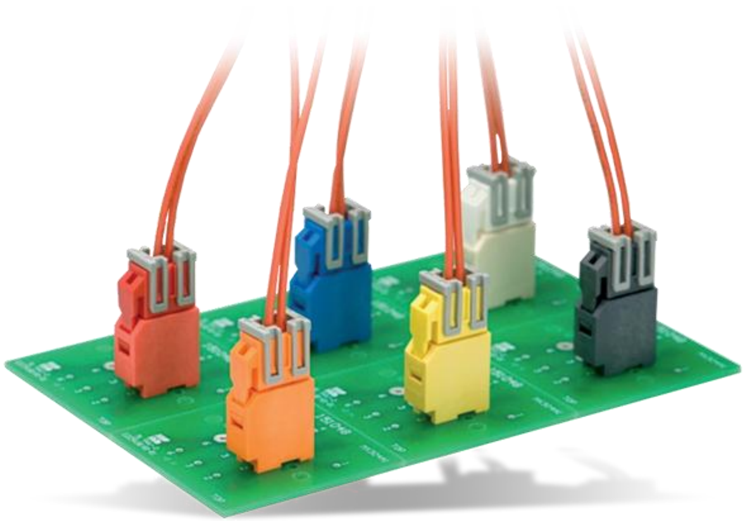
- (**Note, don’t do this!**)
- What cable goes to what?

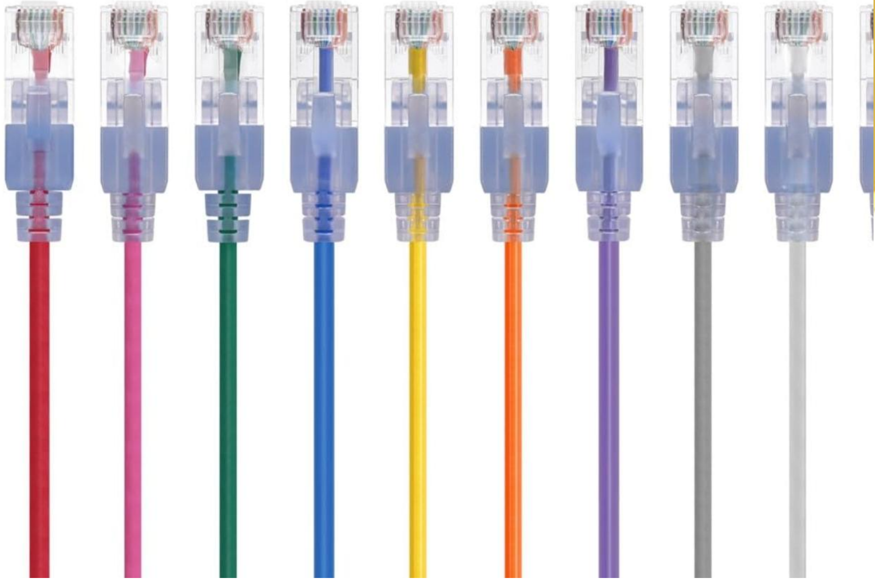
You can use

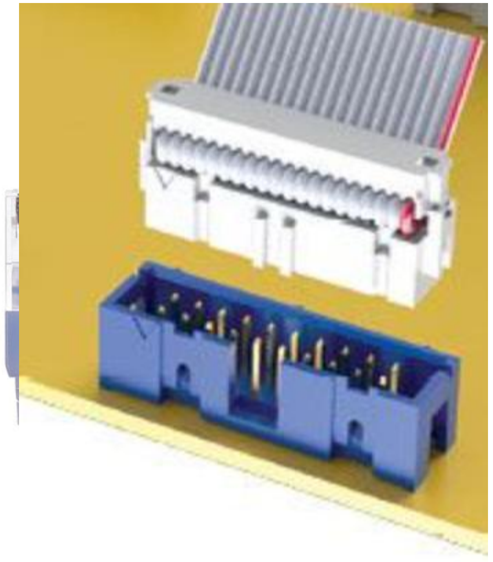
- Keying
- Polarization
- Color

Keyed connectors



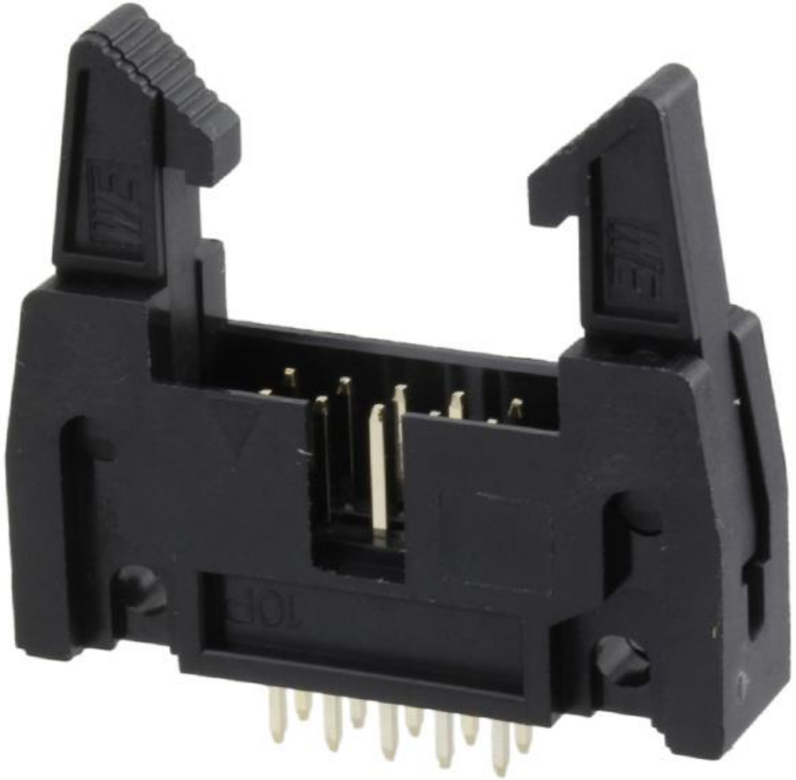


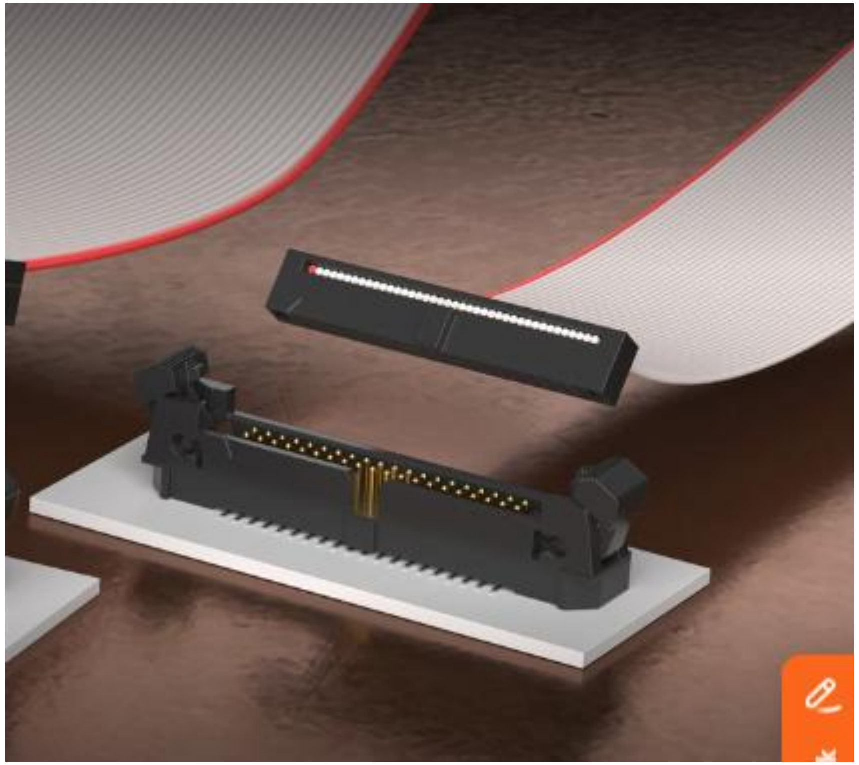




Locking connectors

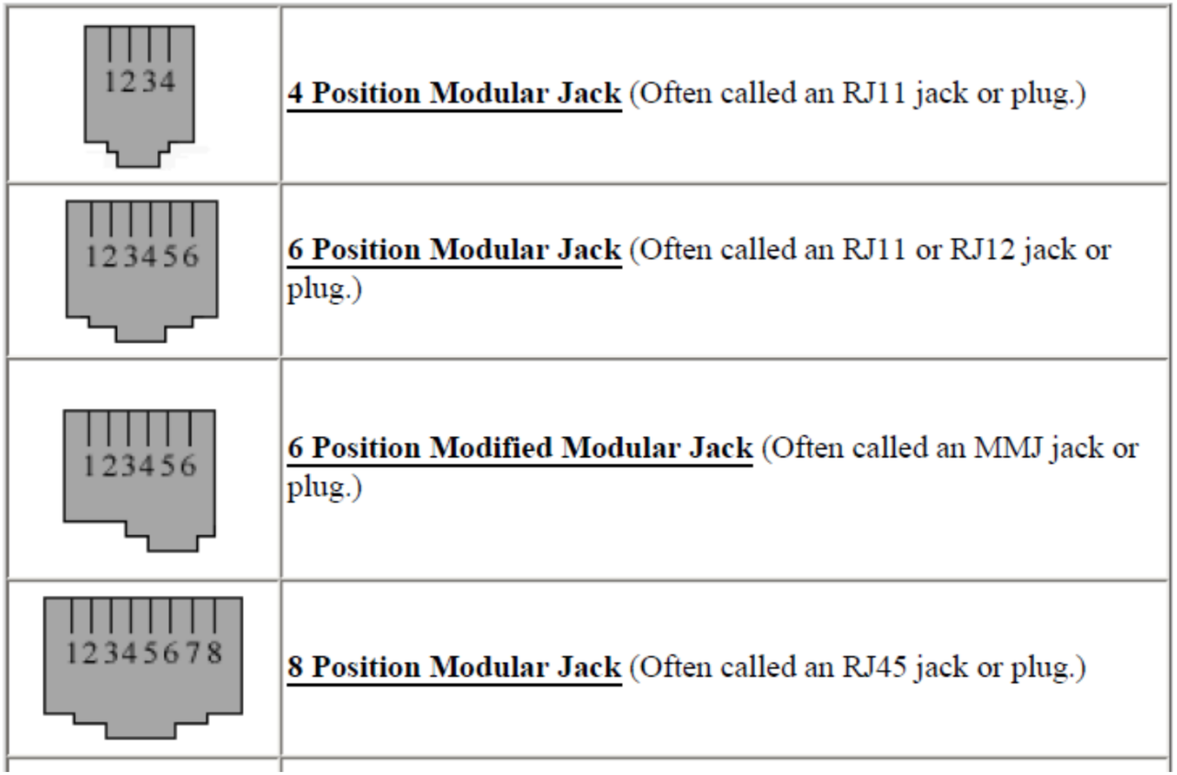
- Vibration happens





Locking connectors

- There are plenty of other types of locking connectors

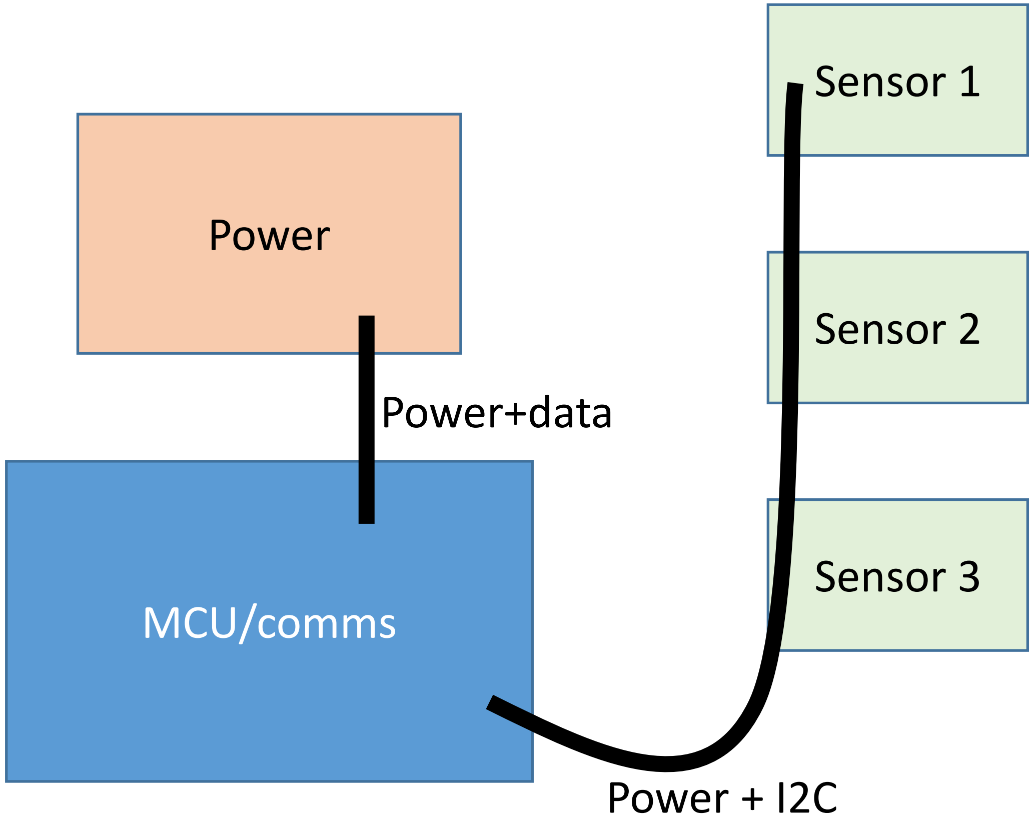


1234	**4 Position Modular Jack** (Often called an RJ11 jack or plug.)
123456	**6 Position Modular Jack** (Often called an RJ11 or RJ12 jack or plug.)
123456	**6 Position Modified Modular Jack** (Often called an MMJ jack or plug.)
12345678	**8 Position Modular Jack** (Often called an RJ45 jack or plug.)

RJ11 jack
$0.109@150

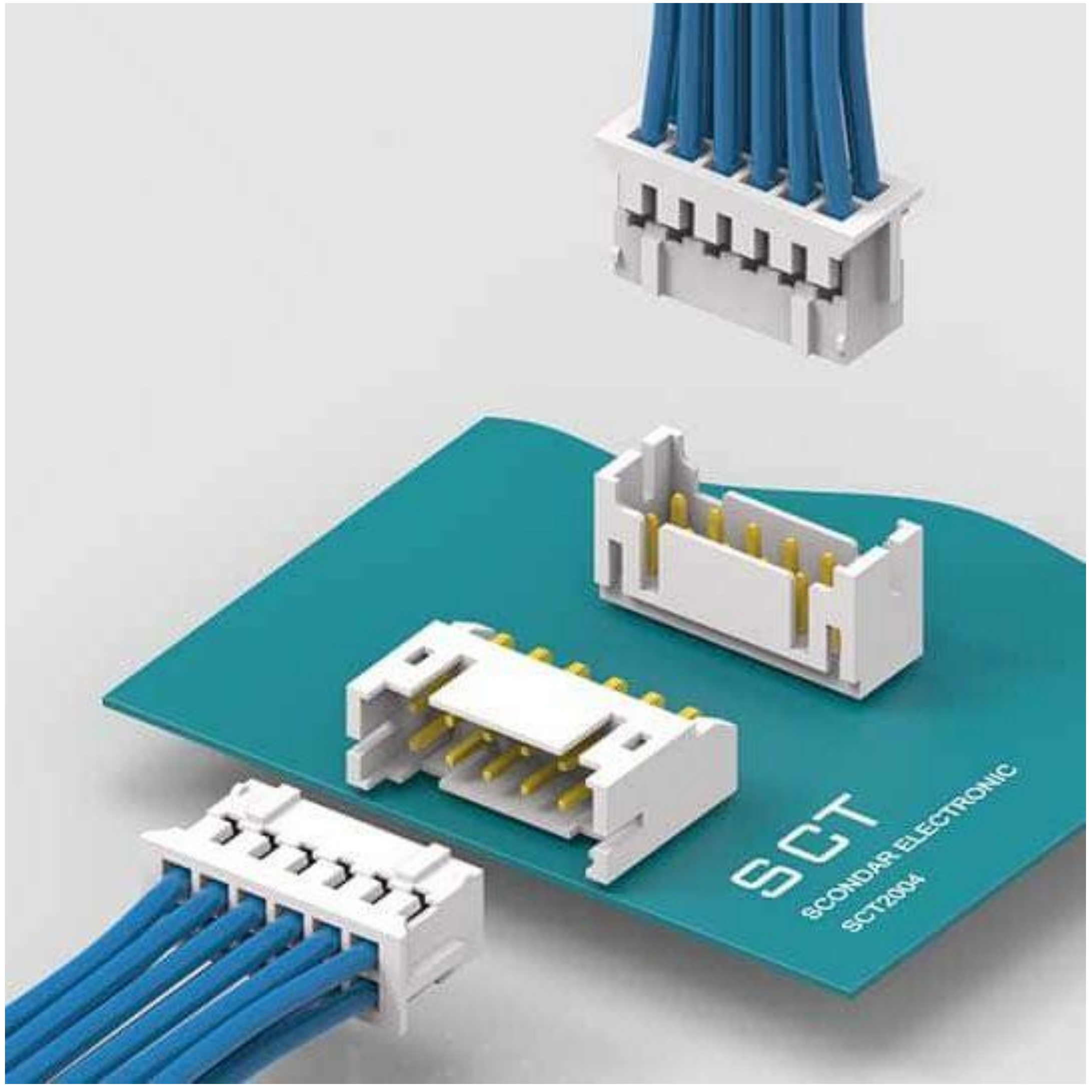
One cable many connectors

- IDC ribbon cable can do multiple connectors with one cable
- Why have a bunch of I2C cables when they all share same SDA, SCL, PWR, GND

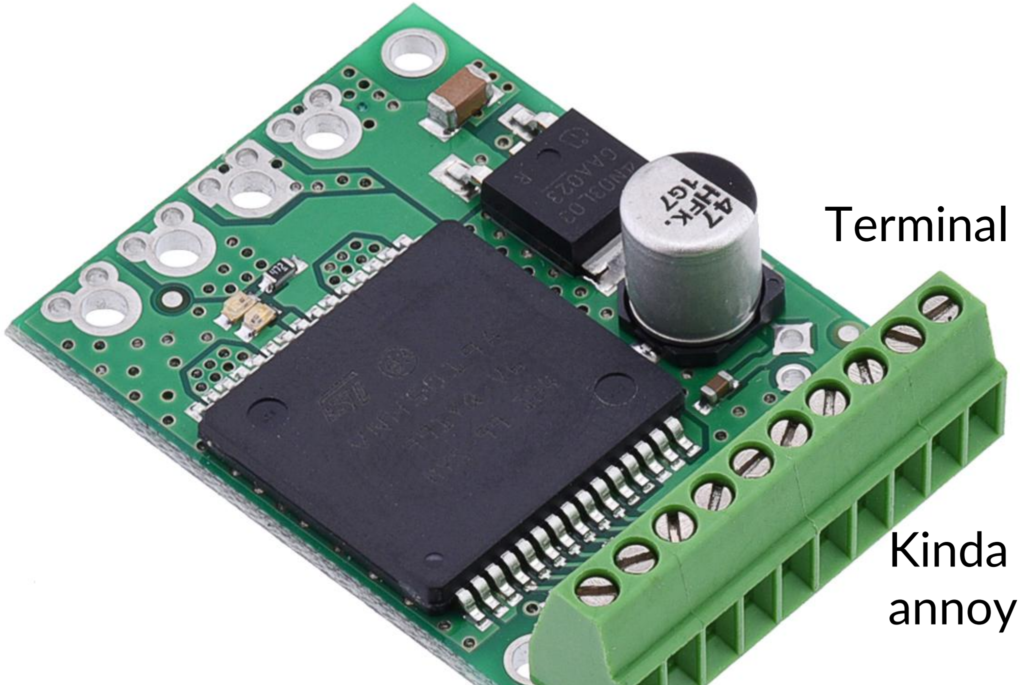


Connectors up or side

- Connectors can “vertical” (point up) or be “right angle”
- Almost any connector

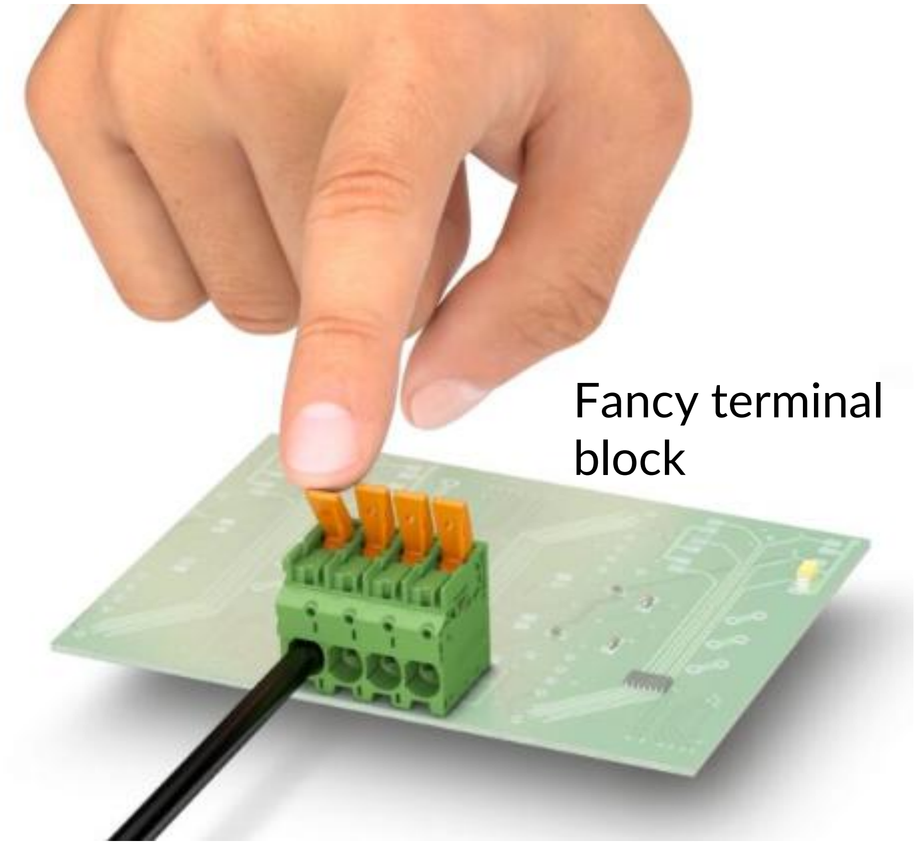


More connectors



Terminal block

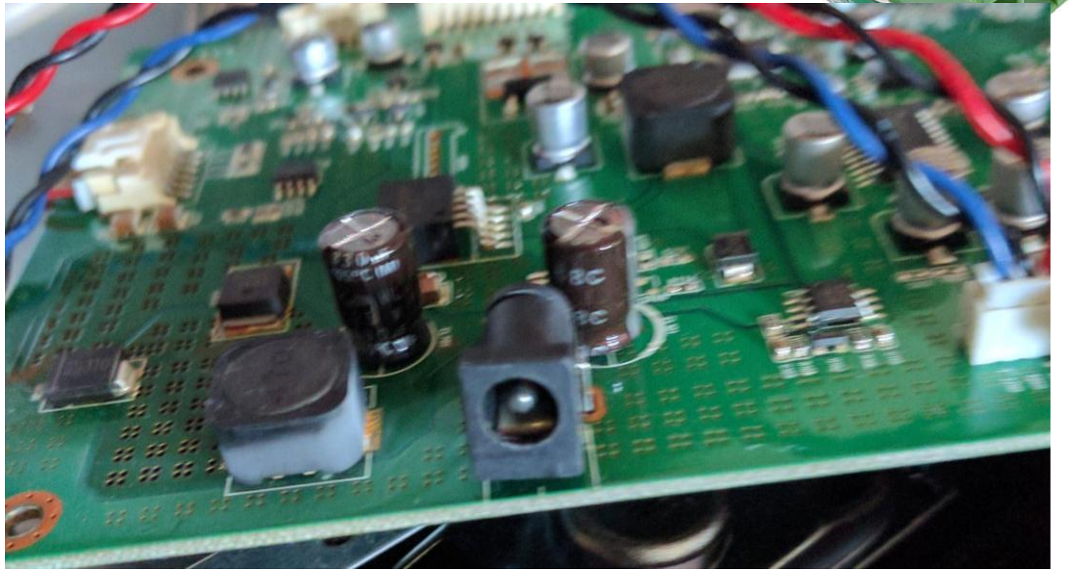
Kinda annoying



Fancy terminal block

Cute, pricey

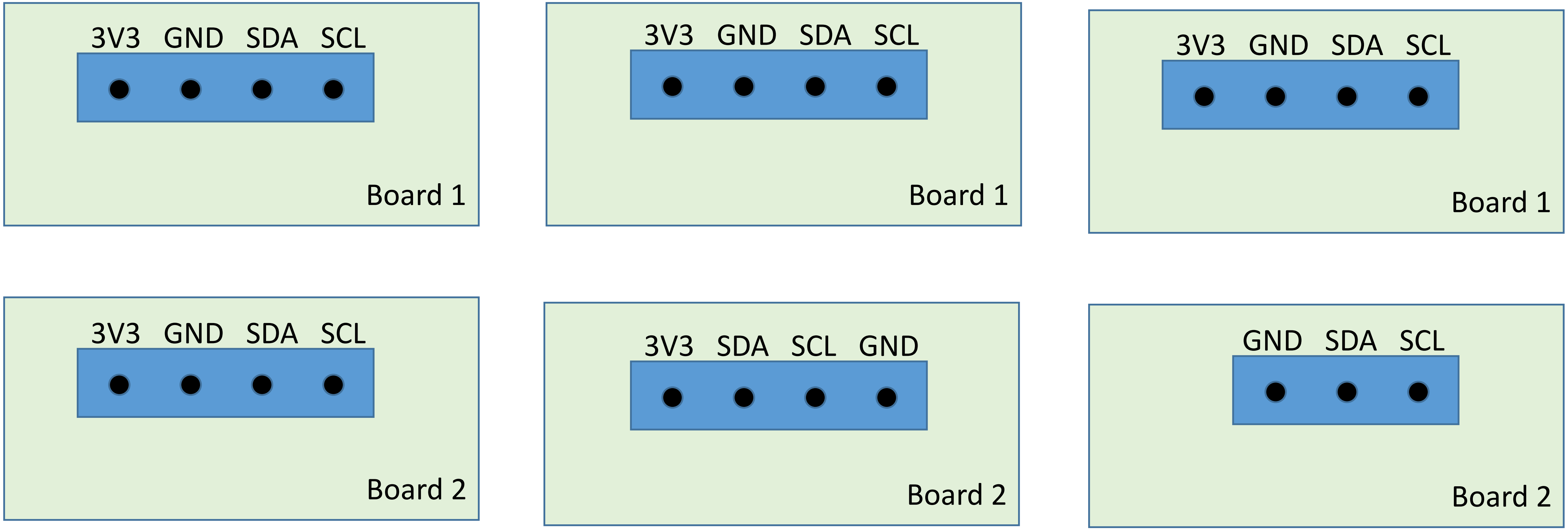
Barrel jack



Common for power, a bit bulky

Define the physical/electrical interface

- Cabling will ONLY work if connectors on different boards correspond **exactly**



Where does happiness lie?

Make your own boards

- You get exactly what you want
- And nothing else (to eat your power)
- With correct size and attachment for your enclosure

It’s fine (preferable, actually) to use breakout boards for prototyping

But final product should use your own boards*

*Unless there’s some crazy part that is too hard to deal with

Make your own boards

- Be careful in choosing the ESP32-C3 chip

NO

ESP32-C3 chip

NO

ESP32-C3 MINI-1 module

YES

ESP32-C3-WROOM-02 module

These have same exposed pins, so no win on GPIOs in either case

Make your own boards

- Be careful in choosing the ESP32-C3-WROOM pins

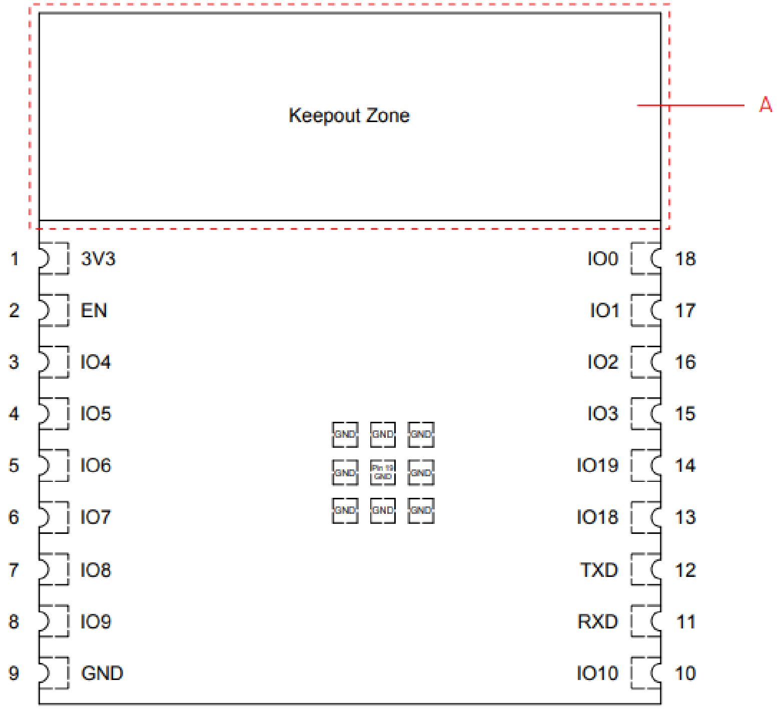
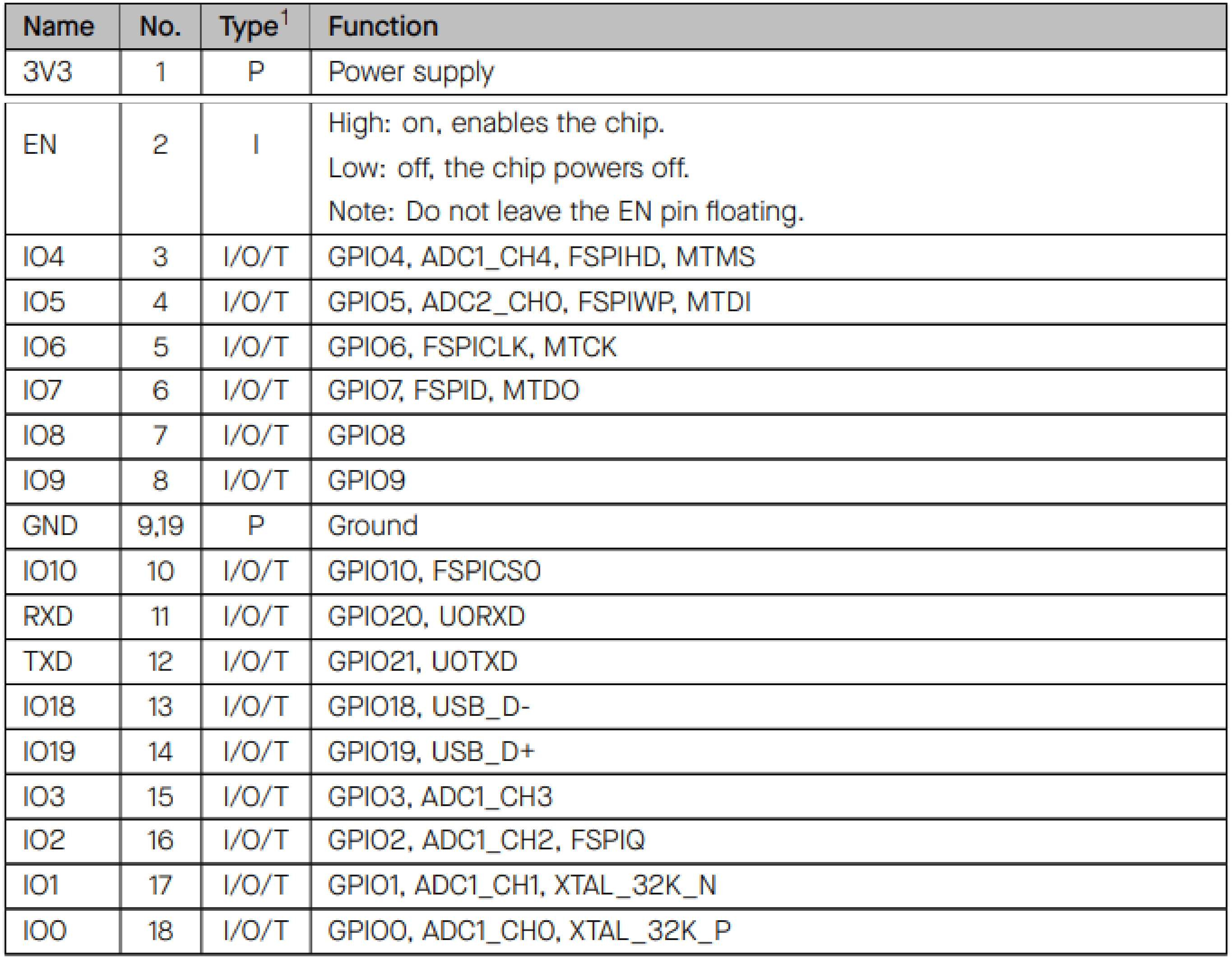


Table 3: Pin Definitions



Name	No.	Type[1]	Function
3V3	1	P	Power supply
EN	2	I	High: on, enables the chip. Low: off, the chip powers off. Note: Do not leave the EN pin floating.
IO4	3	I/O/T	GPIO4, ADC1_CH4, FSPIHD, MTMS
IO5	4	I/O/T	GPIO5, ADC2_CH0, FSPIWP, MTDI
IO6	5	I/O/T	GPIO6, FSPICLK, MTCK
IO7	6	I/O/T	GPIO7, FSPID, MTDO
IO8	7	I/O/T	GPIO8
IO9	8	I/O/T	GPIO9
GND	9,19	P	Ground
IO10	10	I/O/T	GPIO10, FSPICS0
RXD	11	I/O/T	GPIO20, U0RXD
TXD	12	I/O/T	GPIO21, U0TXD
IO18	13	I/O/T	GPIO18, USB_D-
IO19	14	I/O/T	GPIO19, USB_D+
IO3	15	I/O/T	GPIO3, ADC1_CH3
IO2	16	I/O/T	GPIO2, ADC1_CH2, FSPIQ
IO1	17	I/O/T	GPIO1, ADC1_CH1, XTAL_32K_N
IO0	18	I/O/T	GPIO0, ADC1_CH0, XTAL_32K_P

[1] P: power supply; I: input; O: output; T: high impedance.

Make your own boards

- Be careful in choosing the ESP32-C3 pins
- Figure out what pins/peripherals you need
  - USB, I2C, SPI, UART, ADCs, GPIOs, etc.
- Watch out for strapping pins
  - Can be used with care
- Map pins early!
- Make sure you have enough pins!

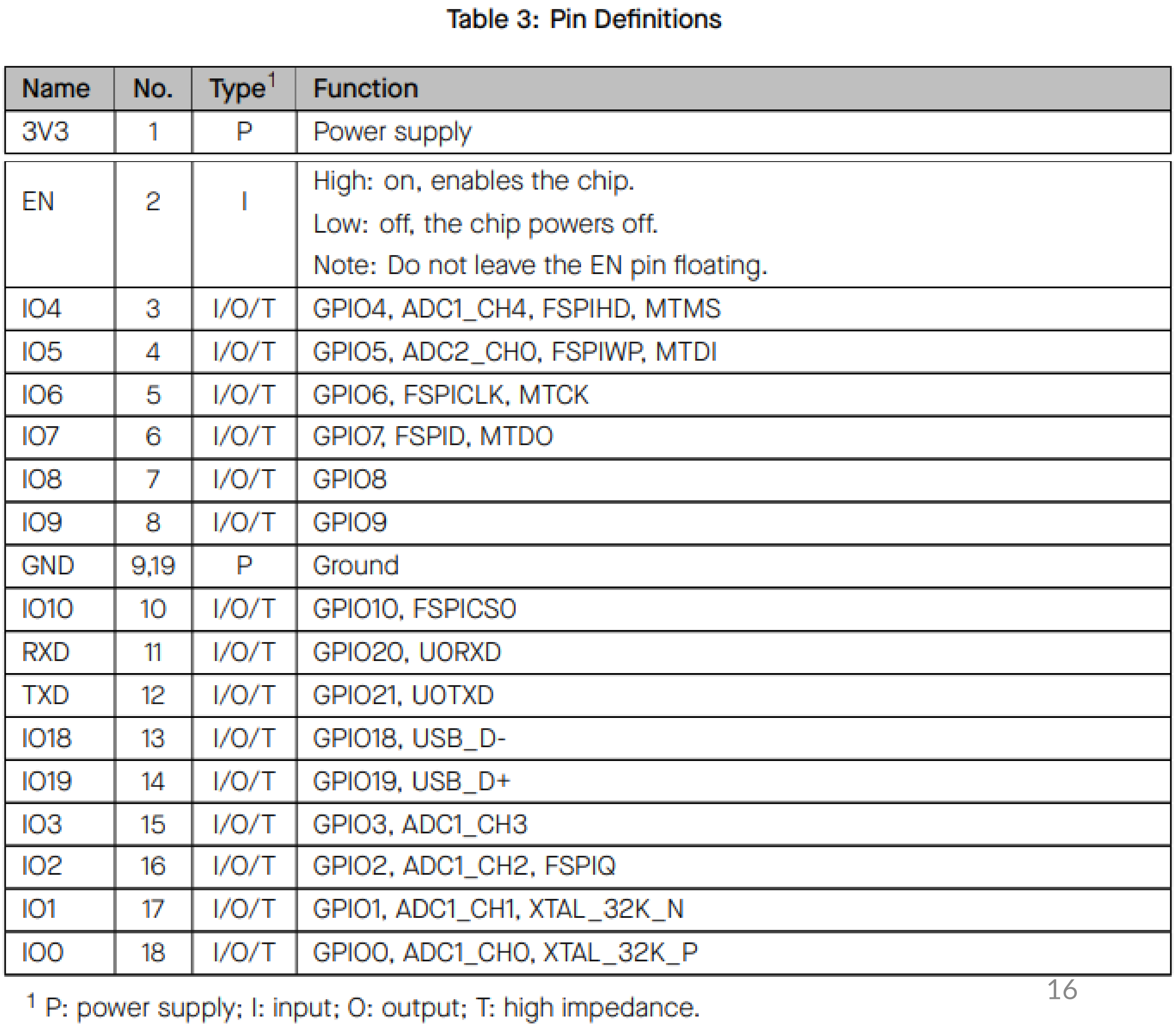


Table 3: Pin Definitions

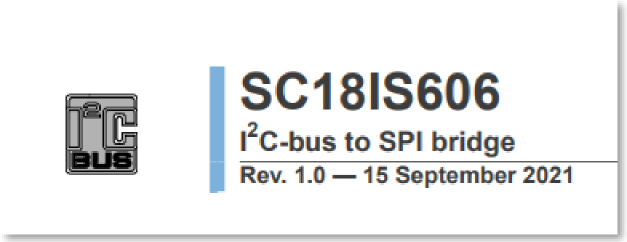
Name	No.	Type[1]	Function
3V3	1	P	Power supply
EN	2	I	High: on, enables the chip. Low: off, the chip powers off. Note: Do not leave the EN pin floating.
IO4	3	I/O/T	GPIO4, ADC1_CH4, FSPIHD, MTMS
IO5	4	I/O/T	GPIO5, ADC2_CH0, FSPIWP, MTDI
IO6	5	I/O/T	GPIO6, FSPICLK, MTCK
IO7	6	I/O/T	GPIO7, FSPID, MTDO
IO8	7	I/O/T	GPIO8
IO9	8	I/O/T	GPIO9
GND	9,19	P	Ground
IO10	10	I/O/T	GPIO10, FSPICS0
RXD	11	I/O/T	GPIO20, U0RXD
TXD	12	I/O/T	GPIO21, U0TXD
IO18	13	I/O/T	GPIO18, USB_D-
IO19	14	I/O/T	GPIO19, USB_D+
IO3	15	I/O/T	GPIO3, ADC1_CH3
IO2	16	I/O/T	GPIO2, ADC1_CH2, FSPIQ
IO1	17	I/O/T	GPIO1, ADC1_CH1, XTAL_32K_N
IO0	18	I/O/T	GPIO0, ADC1_CH0, XTAL_32K_P

[1] P: power supply; I: input; O: output; T: high impedance.

Realizing on Apr 15 that you don't have enough pins

What if we need more pins?

- Use a different ESP32 module
  - ESP32-S3-WROOM-1 has 36 GPIO pins exposed
  - Versus 15 for ESP32-C3-WROOM
- I2C
  - Should not run out of I2C addresses
- SPI
  - Can purchase I2C to SPI bridge
- UART
  - Can purchase I2C to UART bridge
- GPIO
  - I2C GPIO expander
- ADC
  - ADC ICs

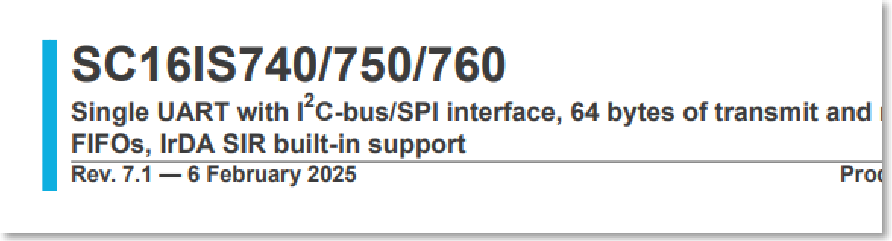


SC18IS606

I^2C-bus to SPI bridge

Rev. 1.0 — 15 September 2021

$3.59@1



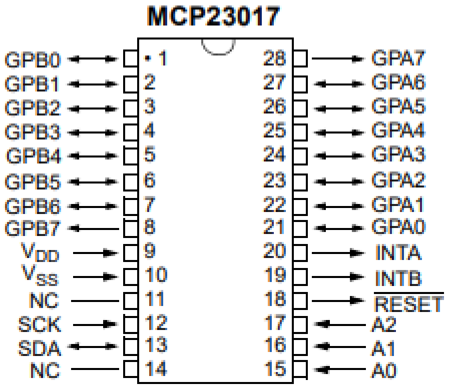
SC16IS740/750/760

Single UART with I^2C-bus/SPI interface, 64 bytes of transmit and

FIFOs, IrDA SIR built-in support

Rev. 7.1 — 6 February 2025 Proc

$3.53@1



$1.62@1

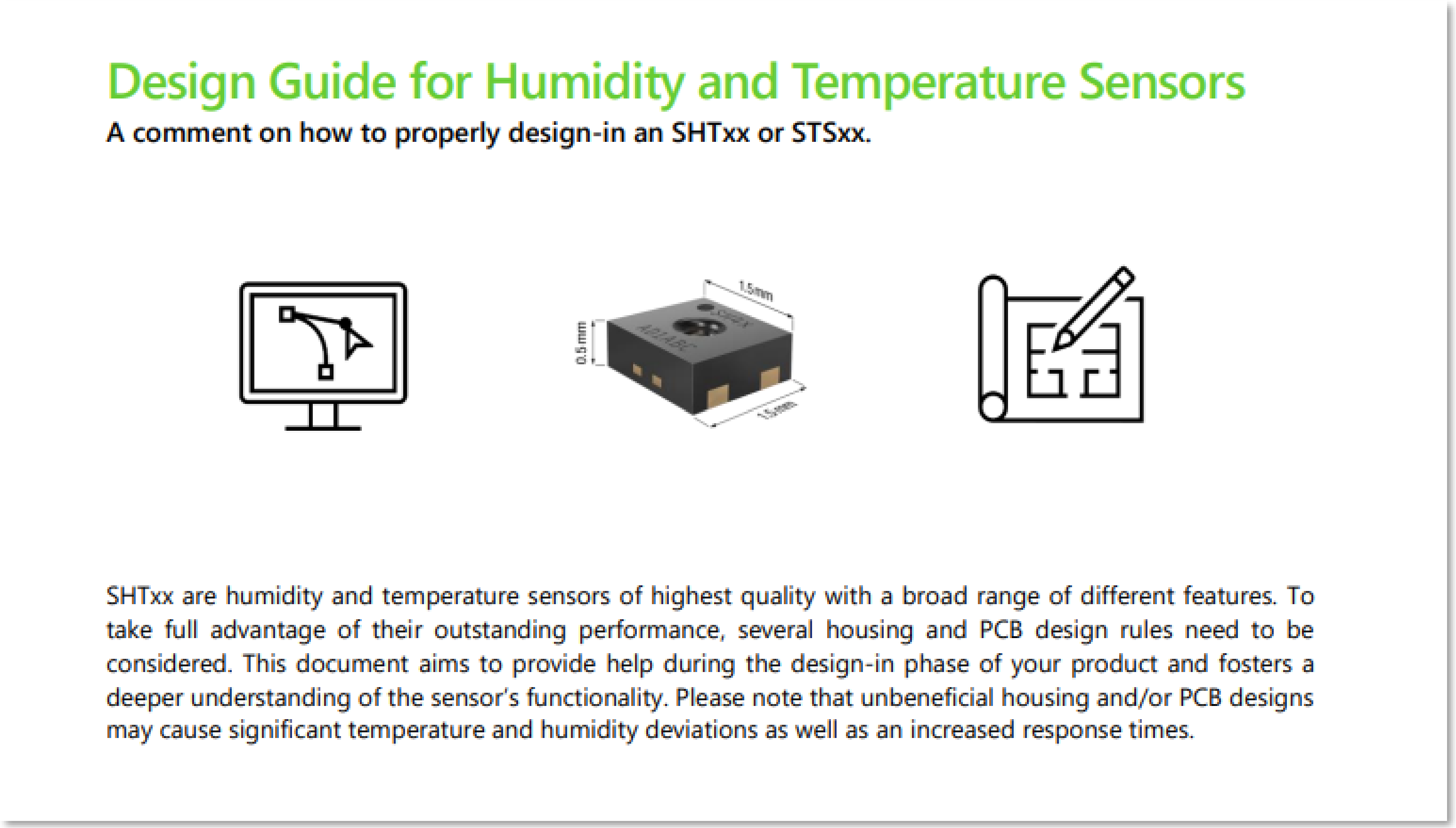
Make your own boards

- We’re building in time for two revisions
  - First boards due right before spring break
  - Second rev in April
- It takes ~5-10 days to go from Gerbers to boards in hand
  - Plus time needed to do schematic design, layout, reviews, assembly, test
- It **is possible** to have JLC do assembly in addition to fab
  - They will do better soldering than you…
  - Will save you assembly + test time

  BUT
  - Takes ~1 extra week [they quote 24-48h…]
  - You must design to the parts they have on hand
    - Or will need to assemble those parts
    - Teams 1-2 will need to design using their constraints…even though they’ll be assembling their own boards during the term

Make your own boards

- Manufacturers have many resources to help you succeed



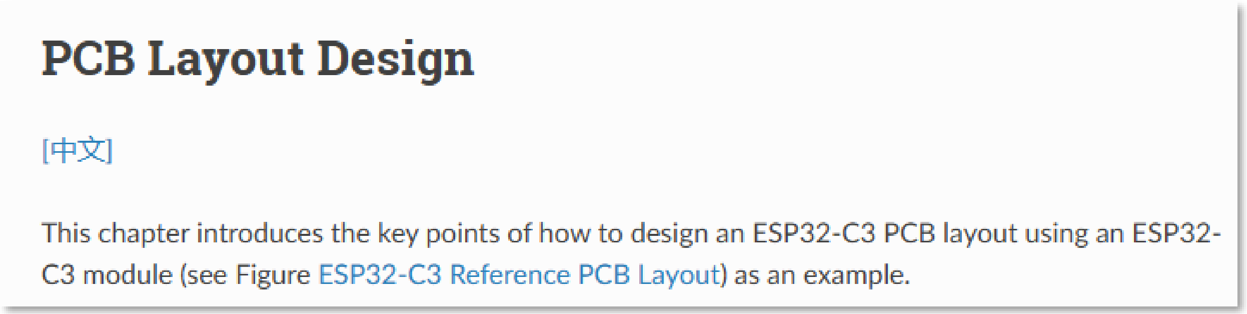
Design Guide for Humidity and Temperature Sensors

A comment on how to properly design-in an SHTxx or STSxx.

SHTxx are humidity and temperature sensors of highest quality with a broad range of different features. To take full advantage of their outstanding performance, several housing and PCB design rules need to be considered. This document aims to provide help during the design-in phase of your product and fosters a deeper understanding of the sensor's functionality. Please note that unbeneficial housing and/or PCB designs may cause significant temperature and humidity deviations as well as an increased response times.

Make your own boards

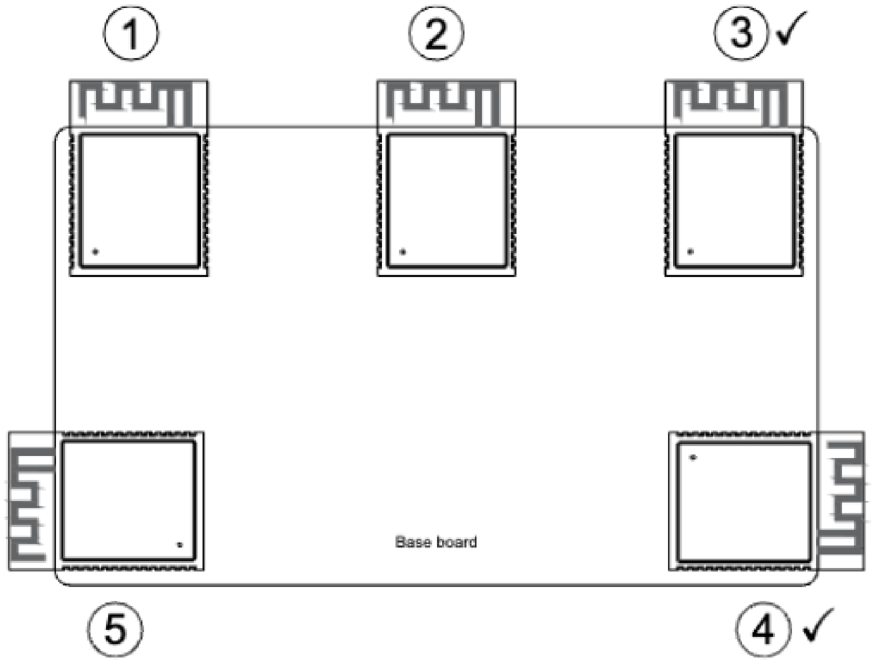
- Manufacturers have many resources to help you succeed



PCB Layout Design

[中文]

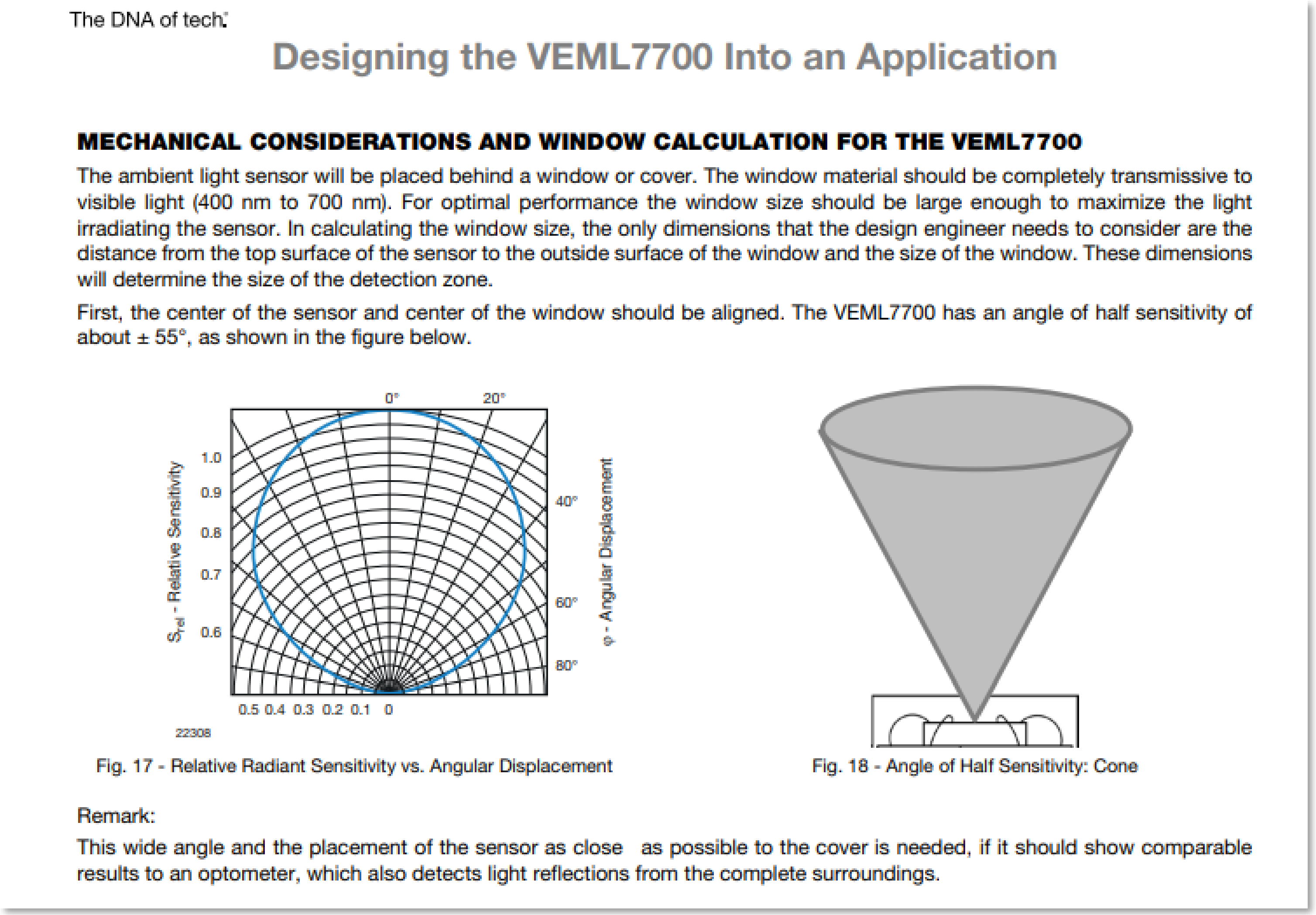
This chapter introduces the key points of how to design an ESP32-C3 PCB layout using an ESP32-C3 module (see Figure ESP32-C3 Reference PCB Layout) as an example.



Placement of ESP32-C3 Modules on Base Board (antenna feed point on the right)

Make your own boards

- Manufacturers have many resources to help you succeed



The DNA of tech.

Designing the VEML7700 Into an Application

MECHANICAL CONSIDERATIONS AND WINDOW CALCULATION FOR THE VEML7700

The ambient light sensor will be placed behind a window or cover. The window material should be completely transmissive to visible light (400 nm to 700 nm). For optimal performance the window size should be large enough to maximize the light irradiating the sensor. In calculating the window size, the only dimensions that the design engineer needs to consider are the distance from the top surface of the sensor to the outside surface of the window and the size of the window. These dimensions will determine the size of the detection zone.

First, the center of the sensor and center of the window should be aligned. The VEML7700 has an angle of half sensitivity of about ± 55°, as shown in the figure below.

Fig. 17 - Relative Radiant Sensitivity vs. Angular Displacement

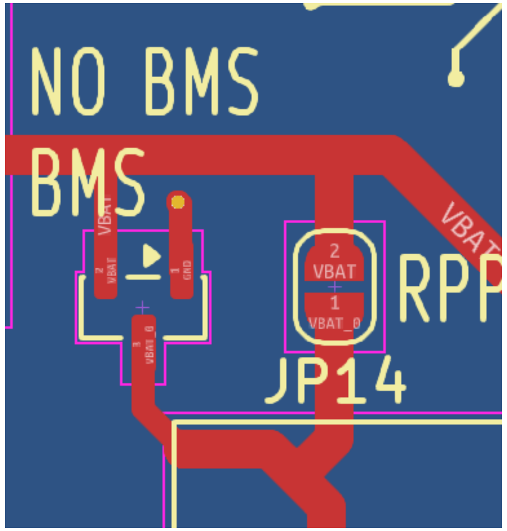
Fig. 18 - Angle of Half Sensitivity: Cone

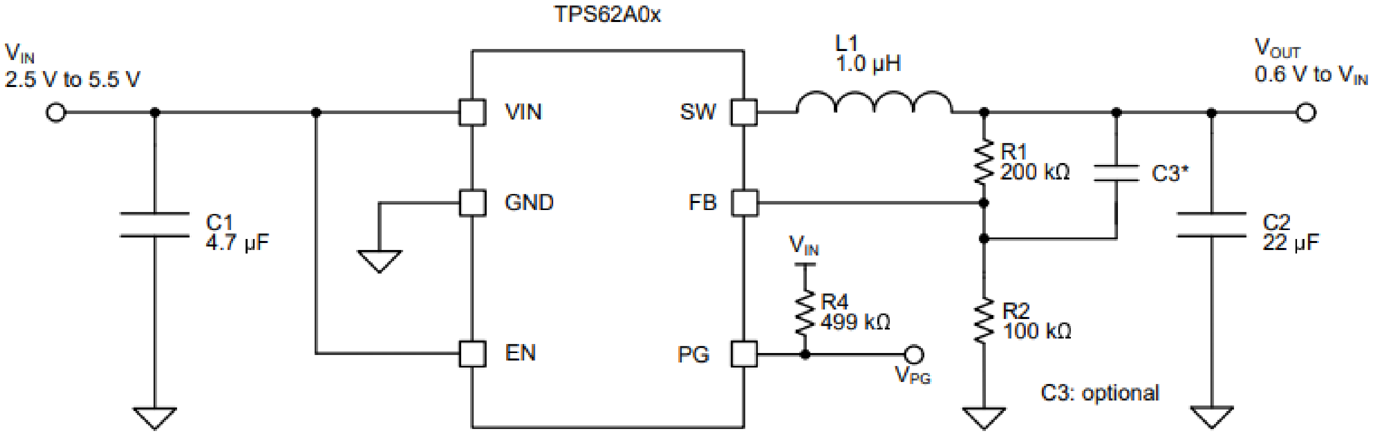
Remark:

This wide angle and the placement of the sensor as close as possible to the cover is needed, if it should show comparable results to an optometer, which also detects light reflections from the complete surroundings.

Design for test & assembly & debug

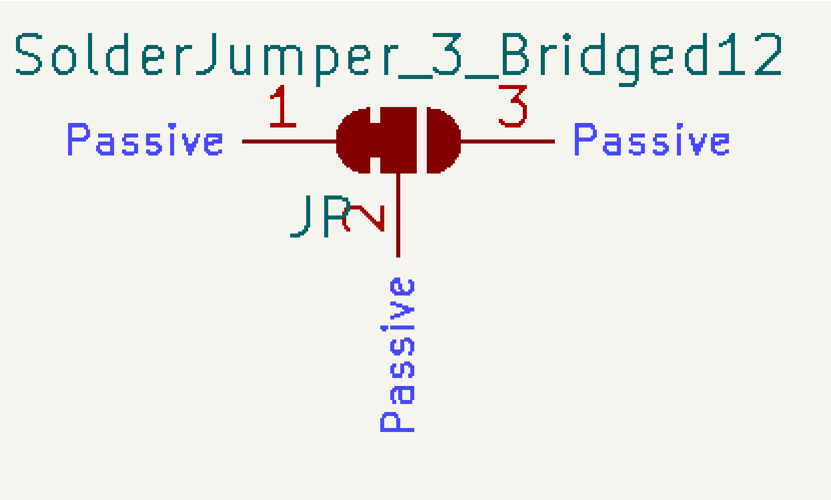
- **Put testpoints everywhere**
  - Especially on unused GPIOs
- **Give yourself alternative connection options**
  - Pins you have >1% chance of wanting to connect differently
    - EN pins, for example
- **Allow for ability to isolate or bypass subsystems**
  - Solder jumpers, etc.

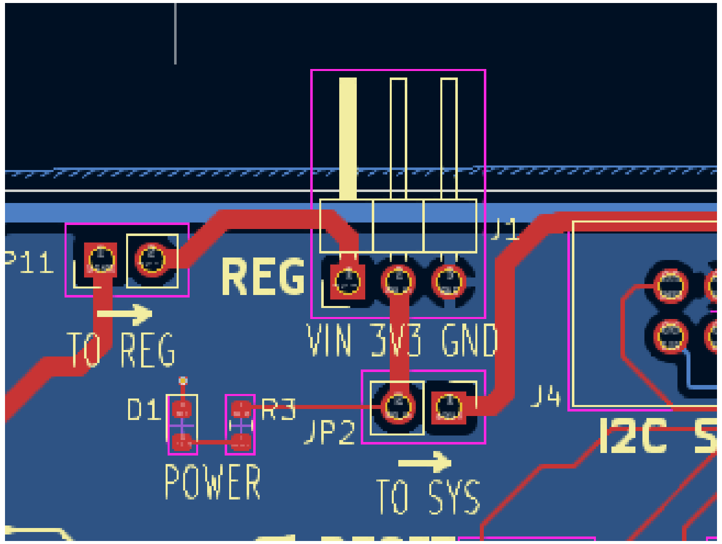


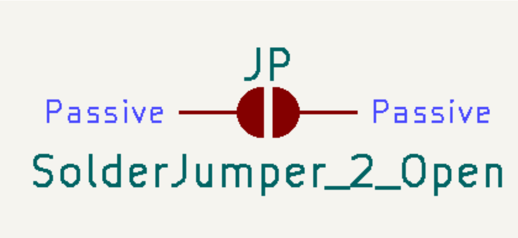


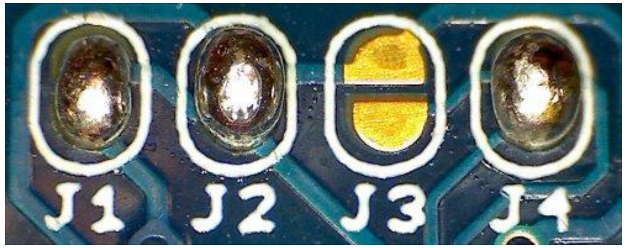
Design for test & assembly & debug

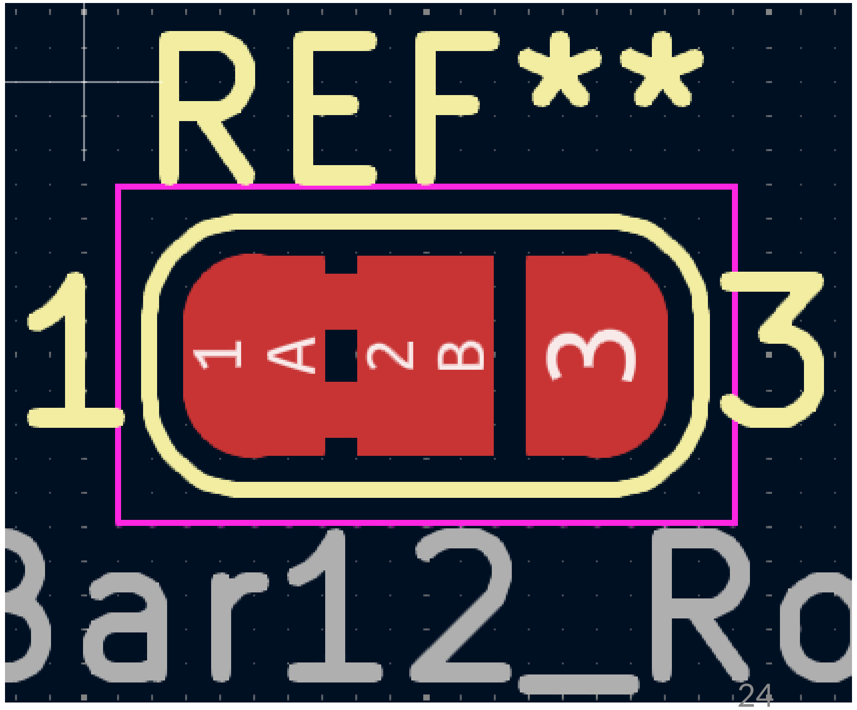
- Jumpers can be NO, NC, 3-way
  - Want to specify later on
  - Might want to change
  - Joulescope access





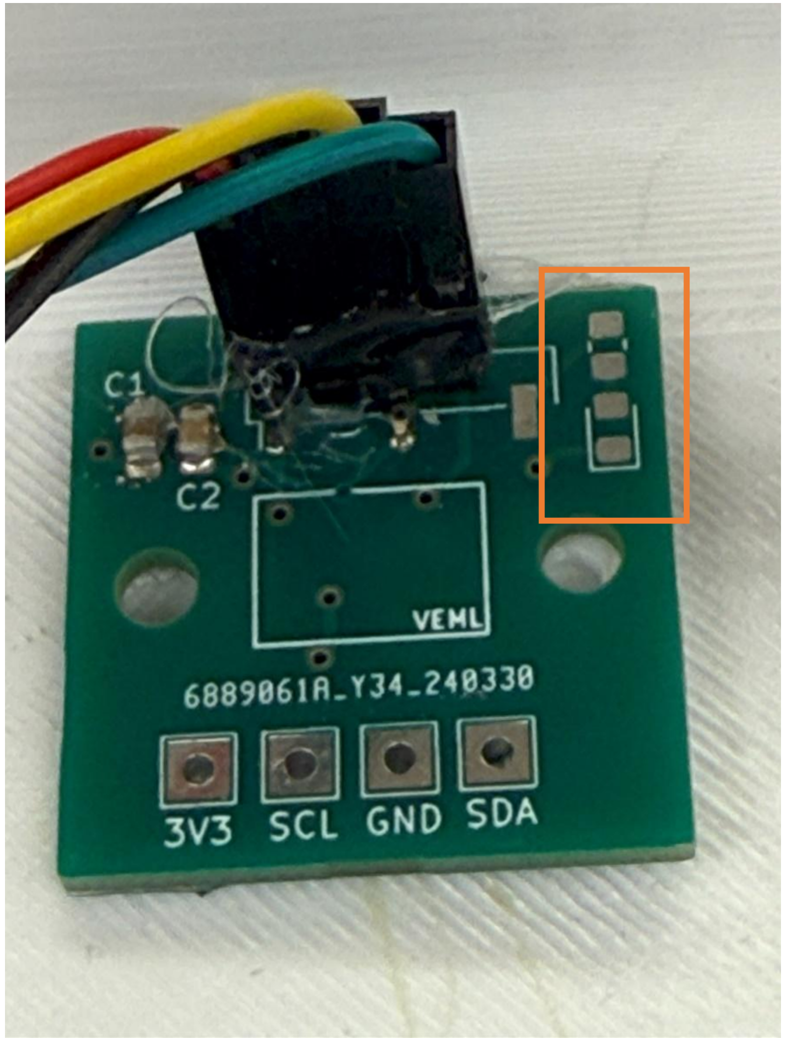






Unassembled parts

- Components in parallel can be placed without being assembled
  - Put in I2C pull-ups on each sensor board, only assemble on one



Unassembled pull-ups

When things break

- When something goes wrong…what do you do?

- LEDs
- Displays
- SD cards
- Connectors

No data on dashboard

Is battery dead?
Sensor disconnected?
Not getting on WiFi?
MCU crash?
Some other issue?

When things break

- When something goes wrong…what do you do?

- LEDs
- Displays
- SD cards
- Connectors

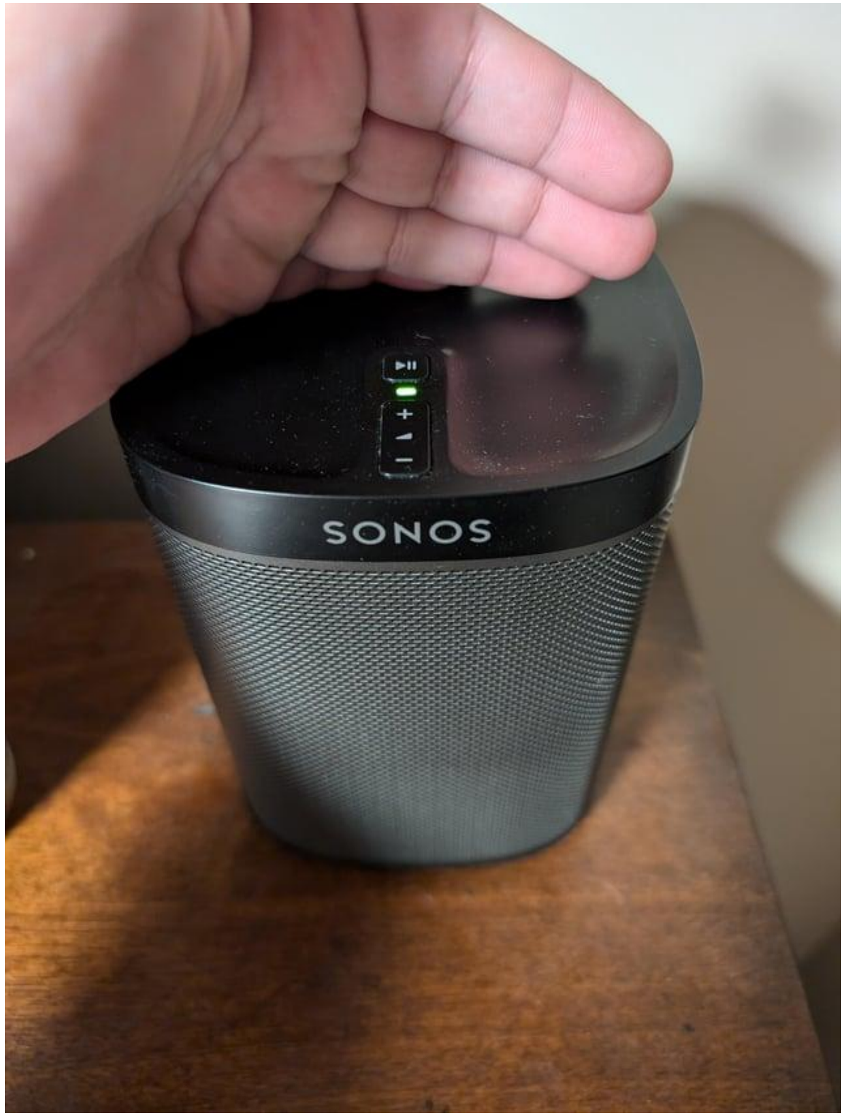
No data on dashboard

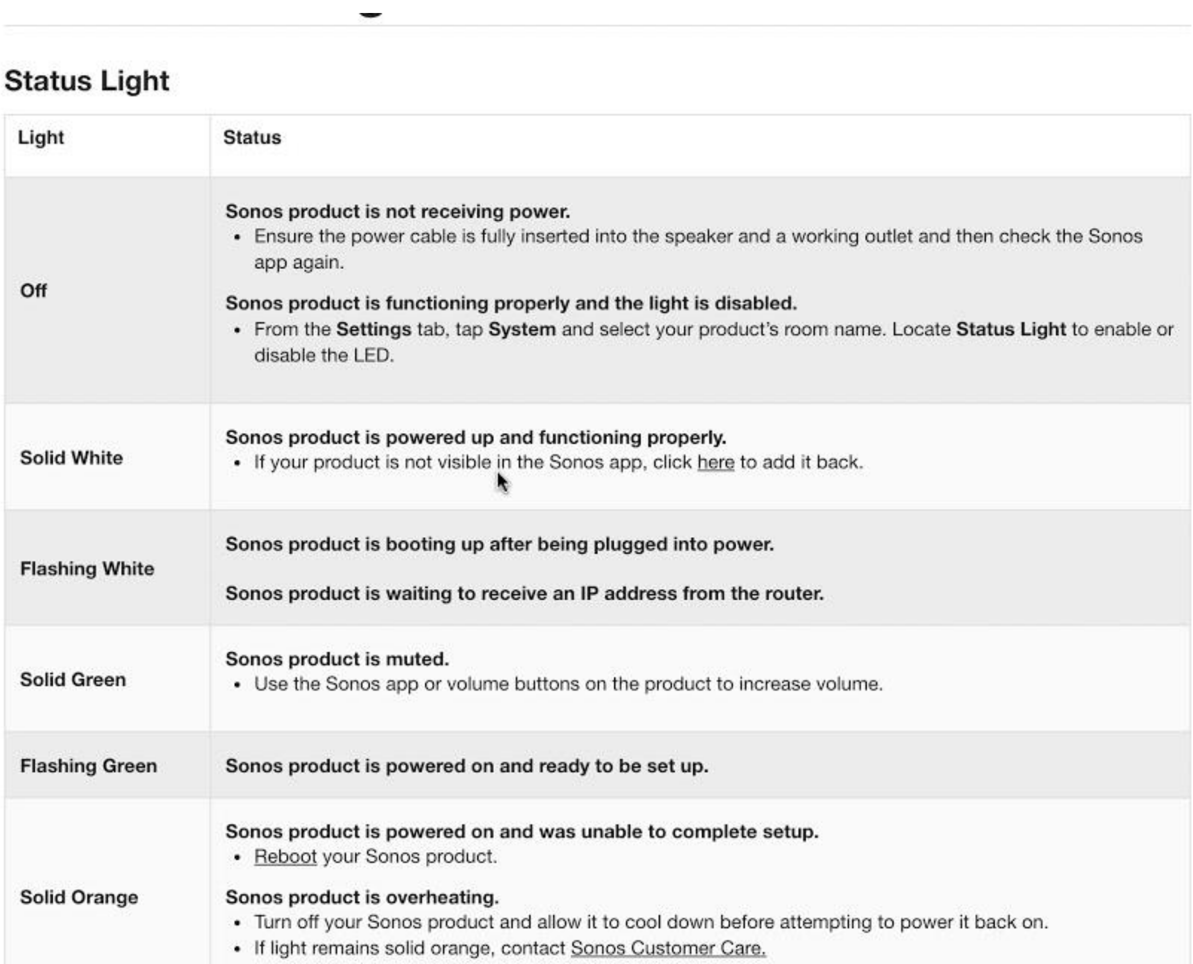
Is battery dead?
Sensor disconnected?
Not getting on WiFi?
MCU crash?
Some other issue?

When things break

- When something goes wrong…what do you do?

- LEDs





Status Light

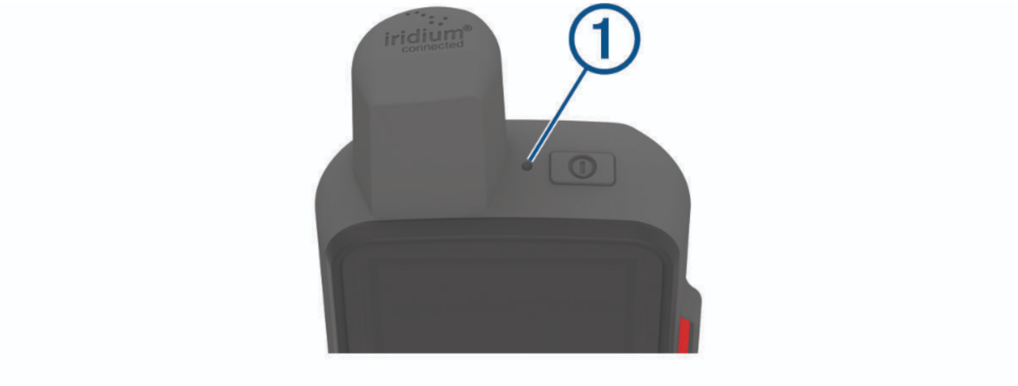
Light	Status
Off	**Sonos product is not receiving power.** • Ensure the power cable is fully inserted into the speaker and a working outlet and then check the Sonos app again. **Sonos product is functioning properly and the light is disabled.** • From the **Settings** tab, tap **System** and select your product's room name. Locate **Status Light** to enable or disable the LED.
Solid White	**Sonos product is powered up and functioning properly.** • If your product is not visible in the Sonos app, click here to add it back.
Flashing White	**Sonos product is booting up after being plugged into power.** **Sonos product is waiting to receive an IP address from the router.**
Solid Green	**Sonos product is muted.** • Use the Sonos app or volume buttons on the product to increase volume.
Flashing Green	**Sonos product is powered on and ready to be set up.**
Solid Orange	**Sonos product is powered on and was unable to complete setup.** • Reboot your Sonos product. **Sonos product is overheating.** • Turn off your Sonos product and allow it to cool down before attempting to power it back on. • If light remains solid orange, contact Sonos Customer Care.

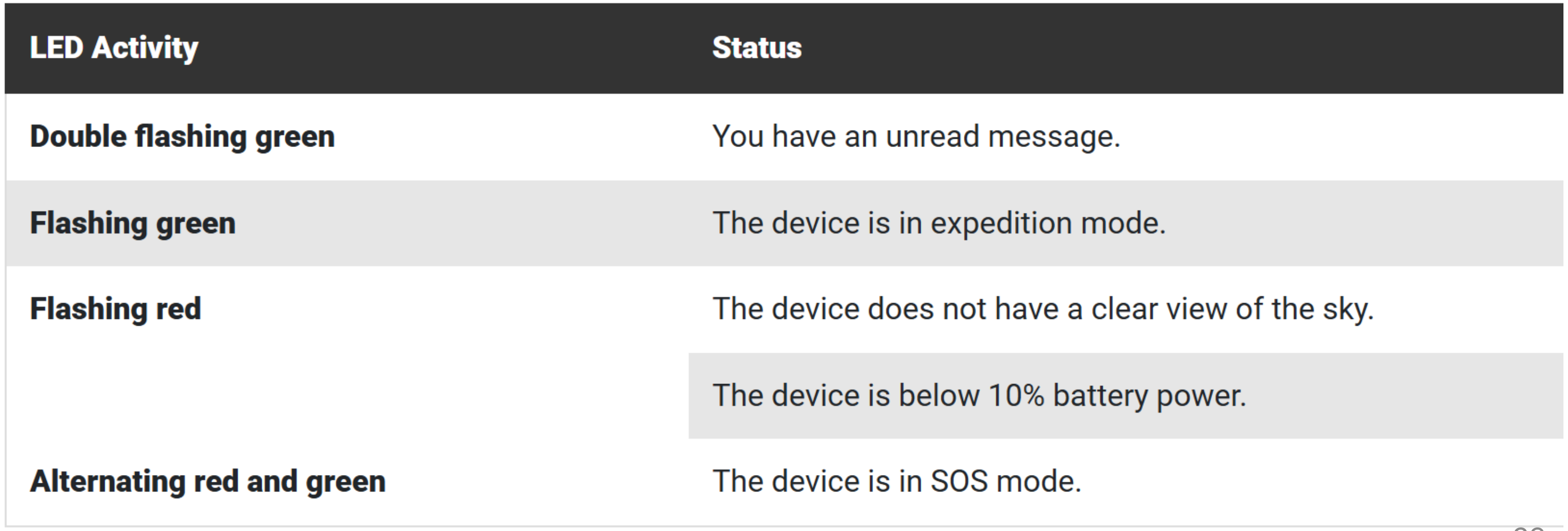
One RGB LED with color+blinking

When things break

- When something goes wrong…what do you do?

- LEDs





LED Activity	Status
Double flashing green	You have an unread message.
Flashing green	The device is in expedition mode.
Flashing red	The device does not have a clear view of the sky.
	The device is below 10% battery power.
Alternating red and green	The device is in SOS mode.

When things break

- When something goes wrong…what do you do?

- LEDs

Never have an LED on all the time…blinking is what you want

Blink frequently enough that it’s not annoying (<30 sec? <15 sec?)

Blink **patterns** are easier to see outdoors than blink **color**

Each LED requires a hole in your enclosure…

When things break

- When something goes wrong…what do you do?

- Displays

E-paper uses no static power

But non-trivial power to write/update

When things break

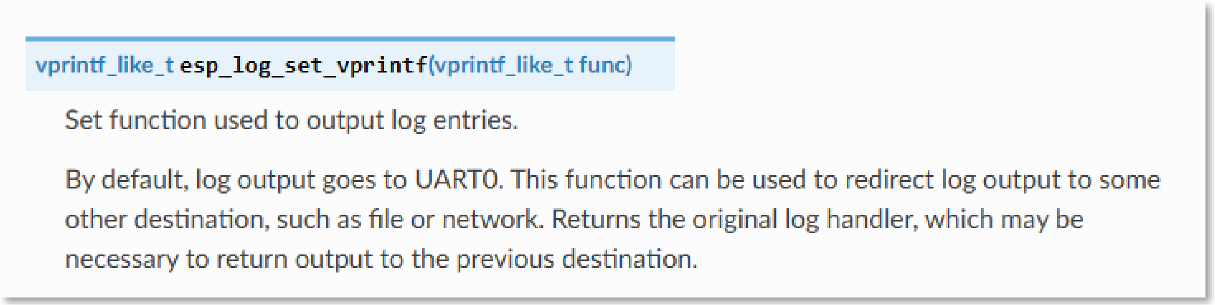
- When something goes wrong…what do you do?

- SD cards
  - Simplest ones are basically just SPI interface to ESP32
    - SD_MMC also exists (faster but more pins)

When things break

- When something goes wrong…what do you do?

- SD cards
  - How will you log?
  - Redirect ESP_LOG messages
    - Basically overwrite the existing vprintf function that ESP_LOG expands into
  - Probably want to build up buffer to avoid constantly writing
  - May choose different amounts of logging for testing vs. deployment



```
vprintf_like_t esp_log_set_vprintf(vprintf_like_t func)
```

Set function used to output log entries.

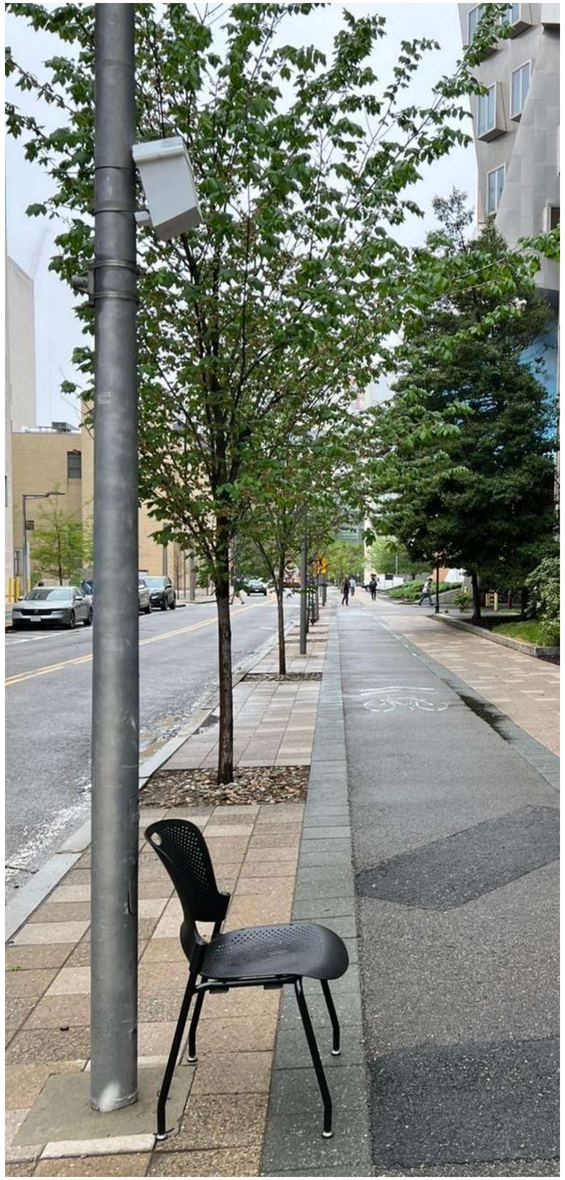
By default, log output goes to UART0. This function can be used to redirect log output to some other destination, such as file or network. Returns the original log handler, which may be necessary to return output to the previous destination.

When things really go wrong

- Sometimes even with careful design, code fails
- How to reset device in the field?

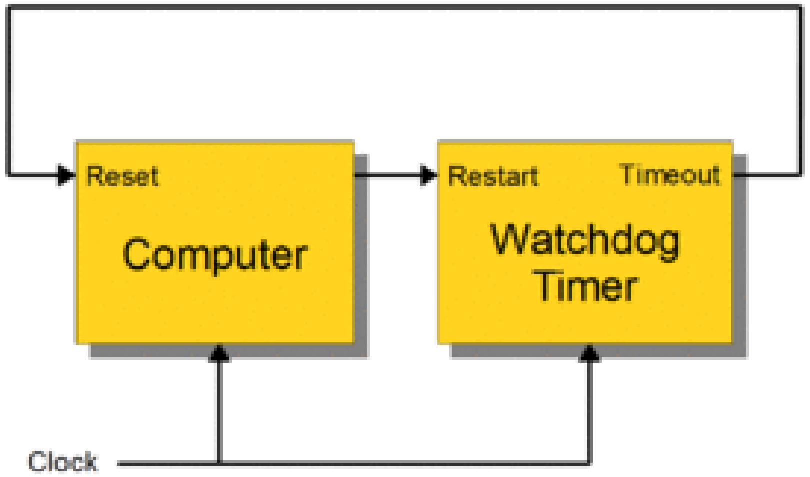
When things really go wrong

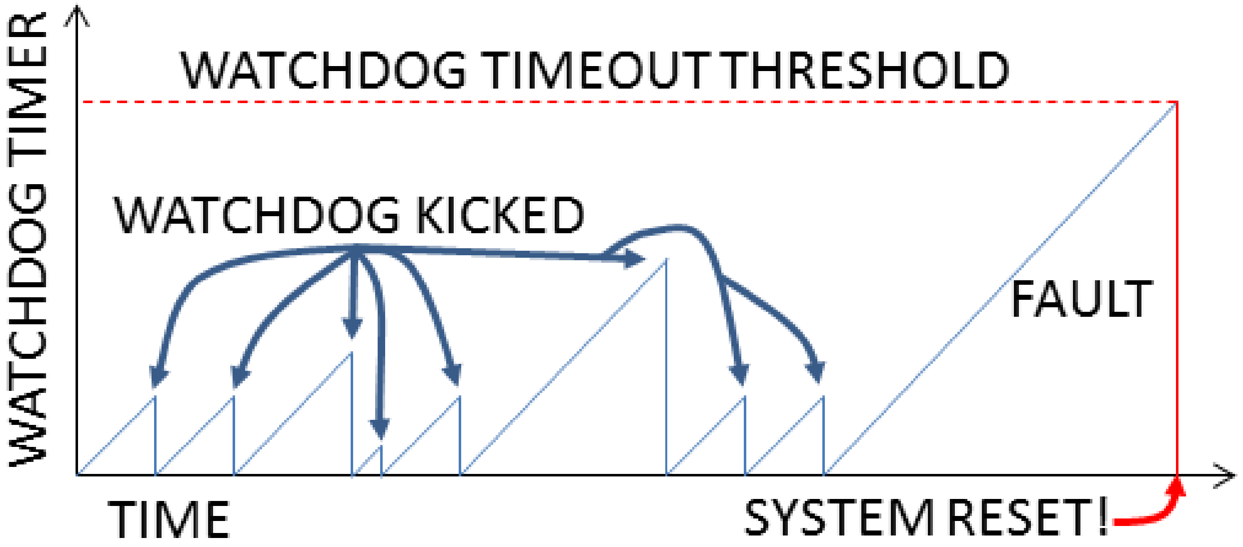
- Sometimes even with careful design, code fails
- How to reset device in the field?



Watchdog timers

- A sentinel that monitors your MCU
- A timer that counts up
- If the timer reaches threshold, it reboots the MCU*
- The MCU code periodically resets the watchdog when it is sure that everything is OK

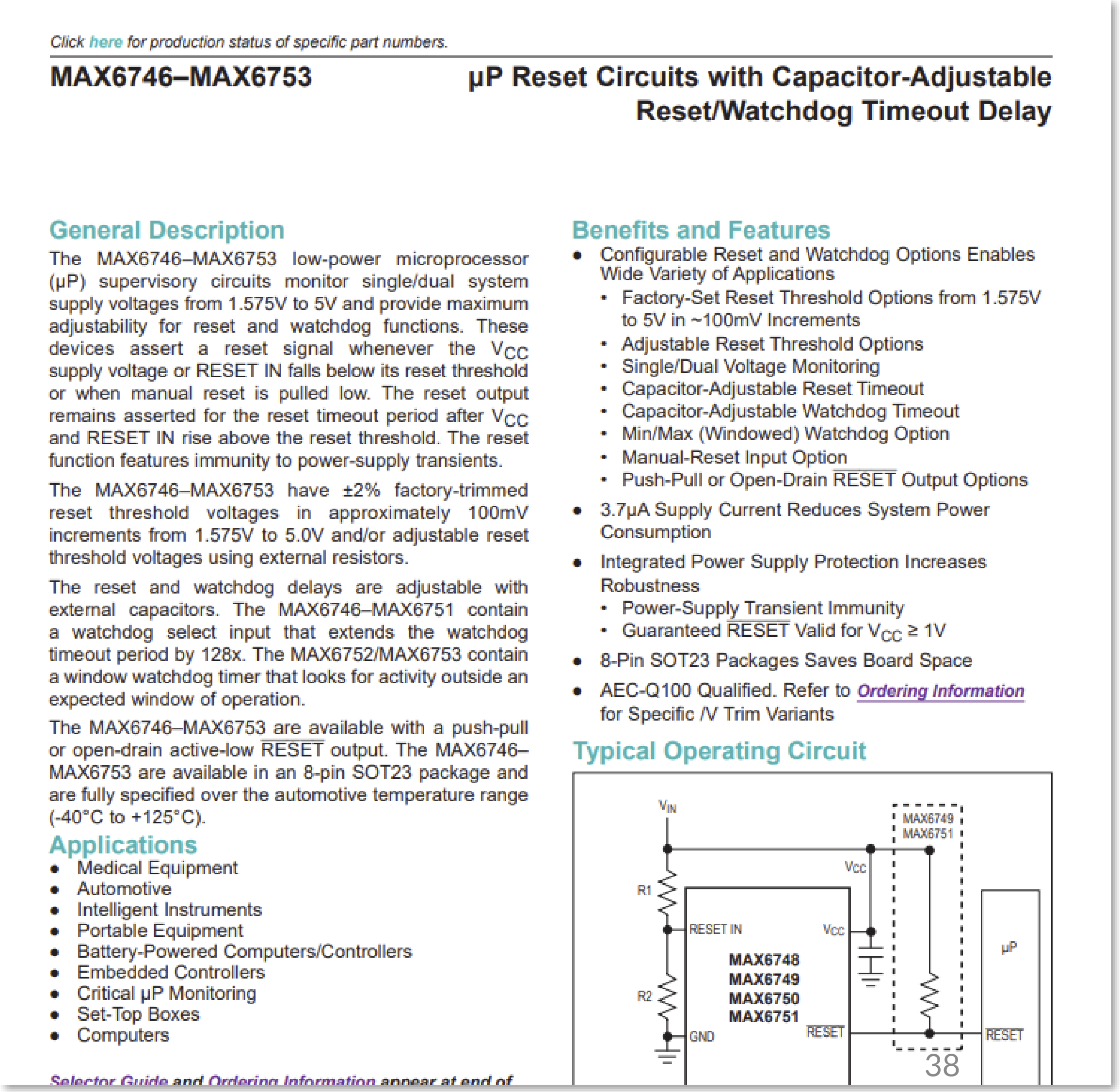




*or does something else specified

Watchdog timers

- Software watchdog
  - A function with ESP.restart() inside it
  - Called when a timer variable goes off
  - Basically useless
  - **Don’t do (only) this**
- Hardware watchdog
  - A distinct internal HW block
  - This is what we’ll use
- A separate HW watchdog chip →
  - If you **really** want to be safe



Click here for production status of specific part numbers.

MAX6746–MAX6753 **µP Reset Circuits with Capacitor-Adjustable Reset/Watchdog Timeout Delay**

General Description

The MAX6746–MAX6753 low-power microprocessor (µP) supervisory circuits monitor single/dual system supply voltages from 1.575V to 5V and provide maximum adjustability for reset and watchdog functions. These devices assert a reset signal whenever the V_{CC} supply voltage or RESET IN falls below its reset threshold or when manual reset is pulled low. The reset output remains asserted for the reset timeout period after V_{CC} and RESET IN rise above the reset threshold. The reset function features immunity to power-supply transients.

The MAX6746–MAX6753 have ±2% factory-trimmed reset threshold voltages in approximately 100mV increments from 1.575V to 5.0V and/or adjustable reset threshold voltages using external resistors.

The reset and watchdog delays are adjustable with external capacitors. The MAX6746–MAX6751 contain a watchdog select input that extends the watchdog timeout period by 128x. The MAX6752/MAX6753 contain a window watchdog timer that looks for activity outside an expected window of operation.

The MAX6746–MAX6753 are available with a push-pull or open-drain active-low RESET output. The MAX6746–MAX6753 are available in an 8-pin SOT23 package and are fully specified over the automotive temperature range (-40°C to +125°C).

Applications

- Medical Equipment
- Automotive
- Intelligent Instruments
- Portable Equipment
- Battery-Powered Computers/Controllers
- Embedded Controllers
- Critical µP Monitoring
- Set-Top Boxes
- Computers

Selector Guide and Ordering Information appear at end of

Benefits and Features

- Configurable Reset and Watchdog Options Enables Wide Variety of Applications
  - Factory-Set Reset Threshold Options from 1.575V to 5V in ~100mV Increments
  - Adjustable Reset Threshold Options
  - Single/Dual Voltage Monitoring
  - Capacitor-Adjustable Reset Timeout
  - Capacitor-Adjustable Watchdog Timeout
  - Min/Max (Windowed) Watchdog Option
  - Manual-Reset Input Option
  - Push-Pull or Open-Drain RESET Output Options
- 3.7µA Supply Current Reduces System Power Consumption
- Integrated Power Supply Protection Increases Robustness
  - Power-Supply Transient Immunity
  - Guaranteed RESET Valid for $V_{CC} \geq 1V$
- 8-Pin SOT23 Packages Saves Board Space
- AEC-Q100 Qualified. Refer to *Ordering Information* for Specific /V Trim Variants

Typical Operating Circuit

ESP32 watchdog timers

- Four on-board watchdog timers
  - Two main system watchdog timers
    - This is what you'll be using, as the "task watchdog timer" monitoring (on MWDT0)
      - When it fires, will cause core reset
    - There is also an interrupt watchdog timer (on MWDT1) to make sure interrupts don't get blocked
      - If something disables interrupts for too long
  - RTC Watchdog
    - Used during boot to make sure boot occurs quickly enough
  - One low-power "super watchdog" timer
    - Operates independently, looking for feed from CPU every second

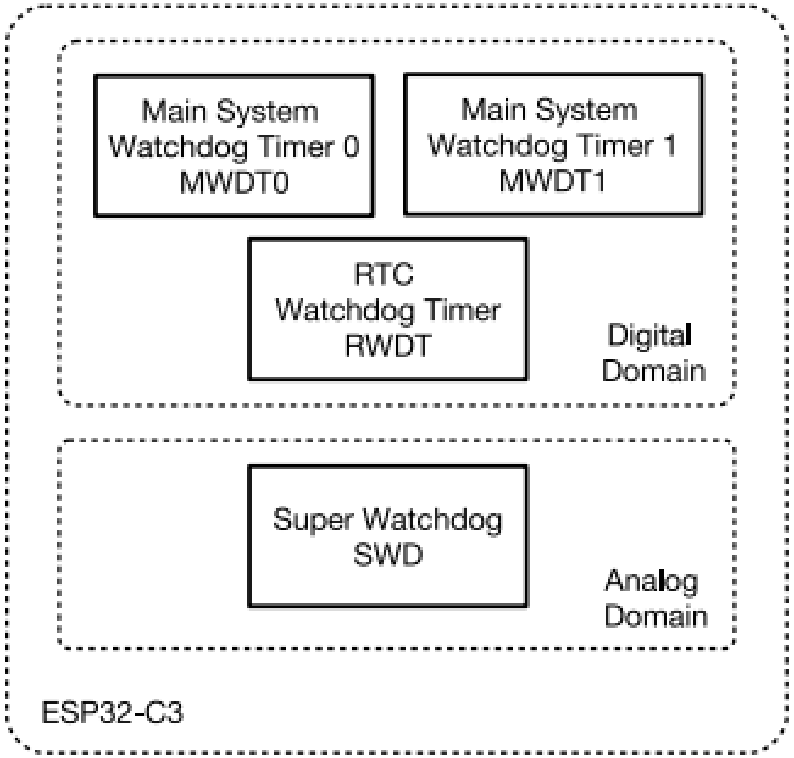


Figure 12.1-1. Watchdog Timers Overview

Watchdog timers

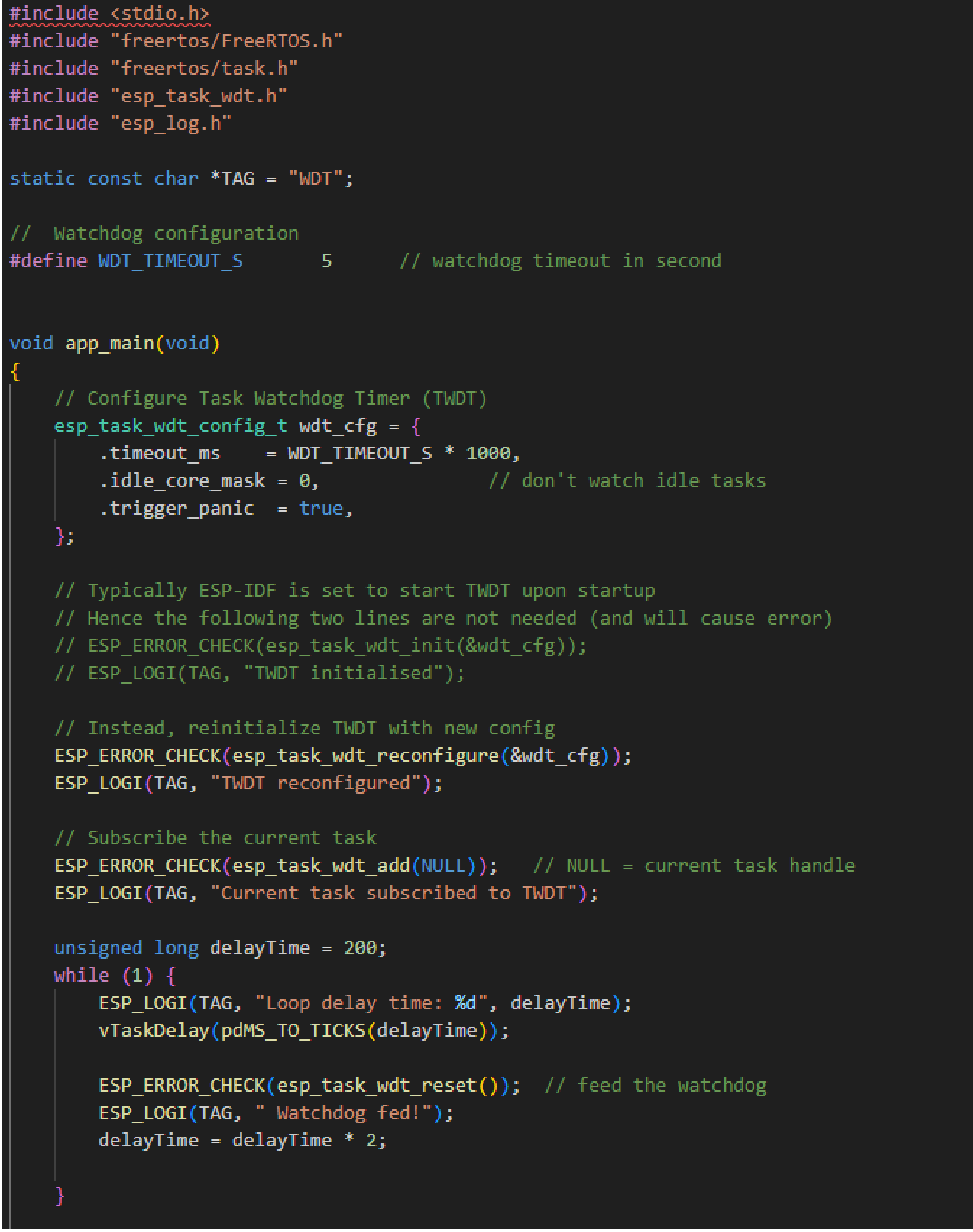
- **Not for general error handling**
- For catching unanticipated errors
  - ESD, electrical noise, etc.
  - Memory leak causing overflow
- Or things that really shouldn't happen
  - Up to you to decide!

Watchdog timers

- Setting the WDT timeout
- **Longer than the longest you expect to go thru your loop**
- This can get tricky with
  - Comms: WiFi, cellular, etc.
  - SD card writes
  - etc.

Your projects are not life-threatening or safety critical…

…err on side of too long

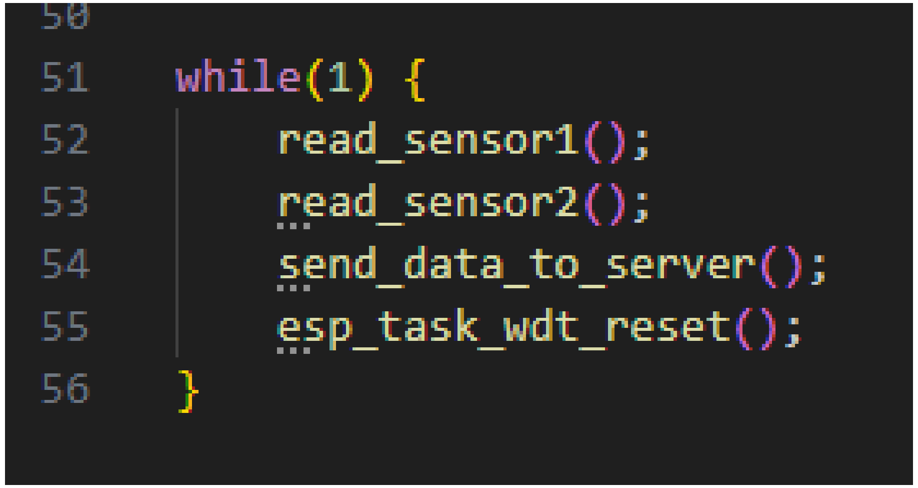


```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_task_wdt.h"
#include "esp_log.h"

static const char *TAG = "WDT";

//  Watchdog configuration
#define WDT_TIMEOUT_S       5       // watchdog timeout in second


void app_main(void)
{
    // Configure Task Watchdog Timer (TWDT)
    esp_task_wdt_config_t wdt_cfg = {
        .timeout_ms     = WDT_TIMEOUT_S * 1000,
        .idle_core_mask = 0,                    // don't watch idle tasks
        .trigger_panic  = true,
    };

    // Typically ESP-IDF is set to start TWDT upon startup
    // Hence the following two lines are not needed (and will cause error)
    // ESP_ERROR_CHECK(esp_task_wdt_init(&wdt_cfg));
    // ESP_LOGI(TAG, "TWDT initialised");

    // Instead, reinitialize TWDT with new config
    ESP_ERROR_CHECK(esp_task_wdt_reconfigure(&wdt_cfg));
    ESP_LOGI(TAG, "TWDT reconfigured");

    // Subscribe the current task
    ESP_ERROR_CHECK(esp_task_wdt_add(NULL));   // NULL = current task handle
    ESP_LOGI(TAG, "Current task subscribed to TWDT");

    unsigned long delayTime = 200;
    while (1) {
        ESP_LOGI(TAG, "Loop delay time: %d", delayTime);
        vTaskDelay(pdMS_TO_TICKS(delayTime));

        ESP_ERROR_CHECK(esp_task_wdt_reset());  // feed the watchdog
        ESP_LOGI(TAG, " Watchdog fed!");
        delayTime = delayTime * 2;

    }
```

Watchdog timers

- Simplest architecture
- Execute each time you go thru your loop

- Simpler, a bit less robust
- Will this catch all the faults you might expect to occur?

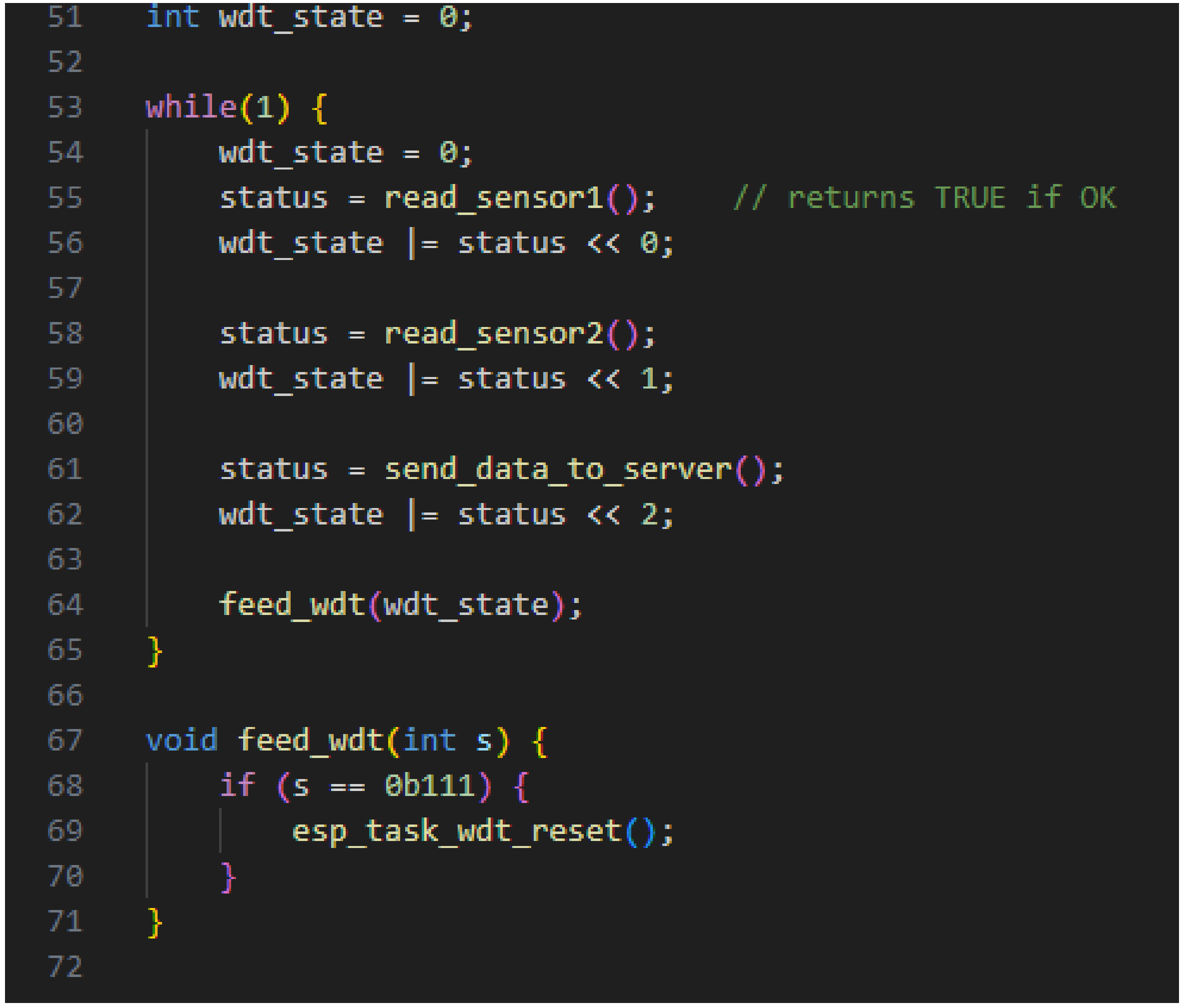


```
while(1) {
    read_sensor1();
    read_sensor2();
    send_data_to_server();
    esp_task_wdt_reset();
}
```

Watchdog timers

- A bit more complex
- Run various checks during your loop
- Only feed WTD if all checks are good

- Make sure this is what you want!
  - Do you really want to reboot if any of these checks are not ok?



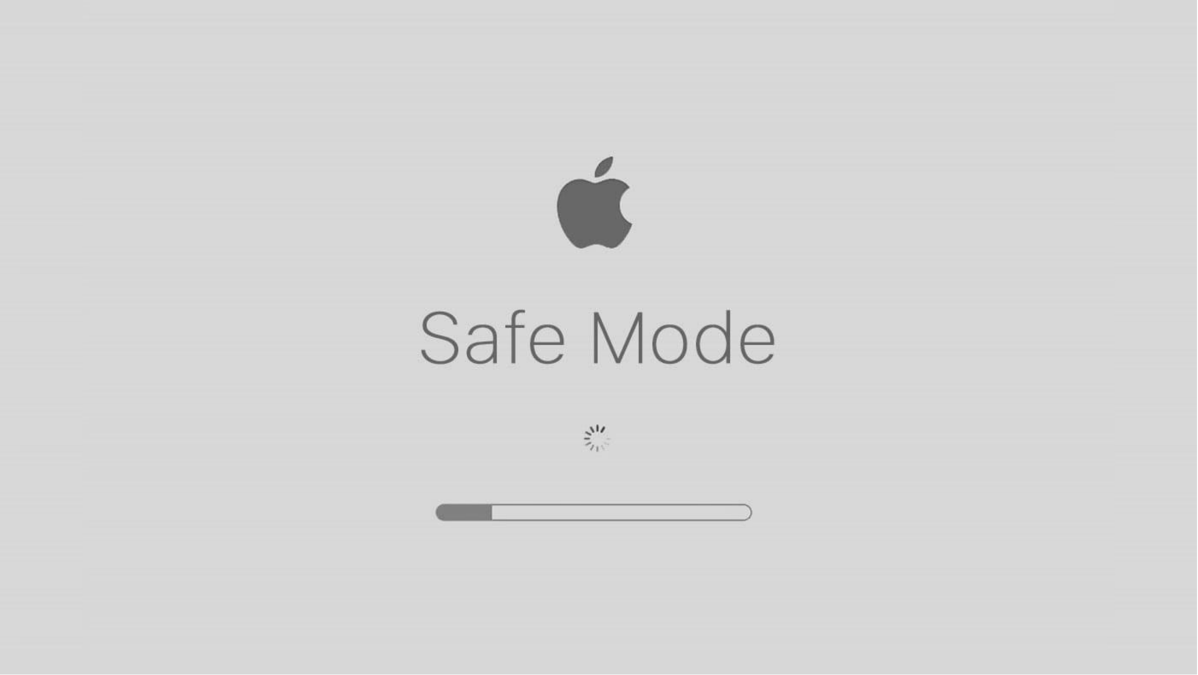
```
int wdt_state = 0;

while(1) {
    wdt_state = 0;
    status = read_sensor1();    // returns TRUE if OK
    wdt_state |= status << 0;

    status = read_sensor2();
    wdt_state |= status << 1;

    status = send_data_to_server();
    wdt_state |= status << 2;

    feed_wdt(wdt_state);
}

void feed_wdt(int s) {
    if (s == 0b111) {
        esp_task_wdt_reset();
    }
}
```

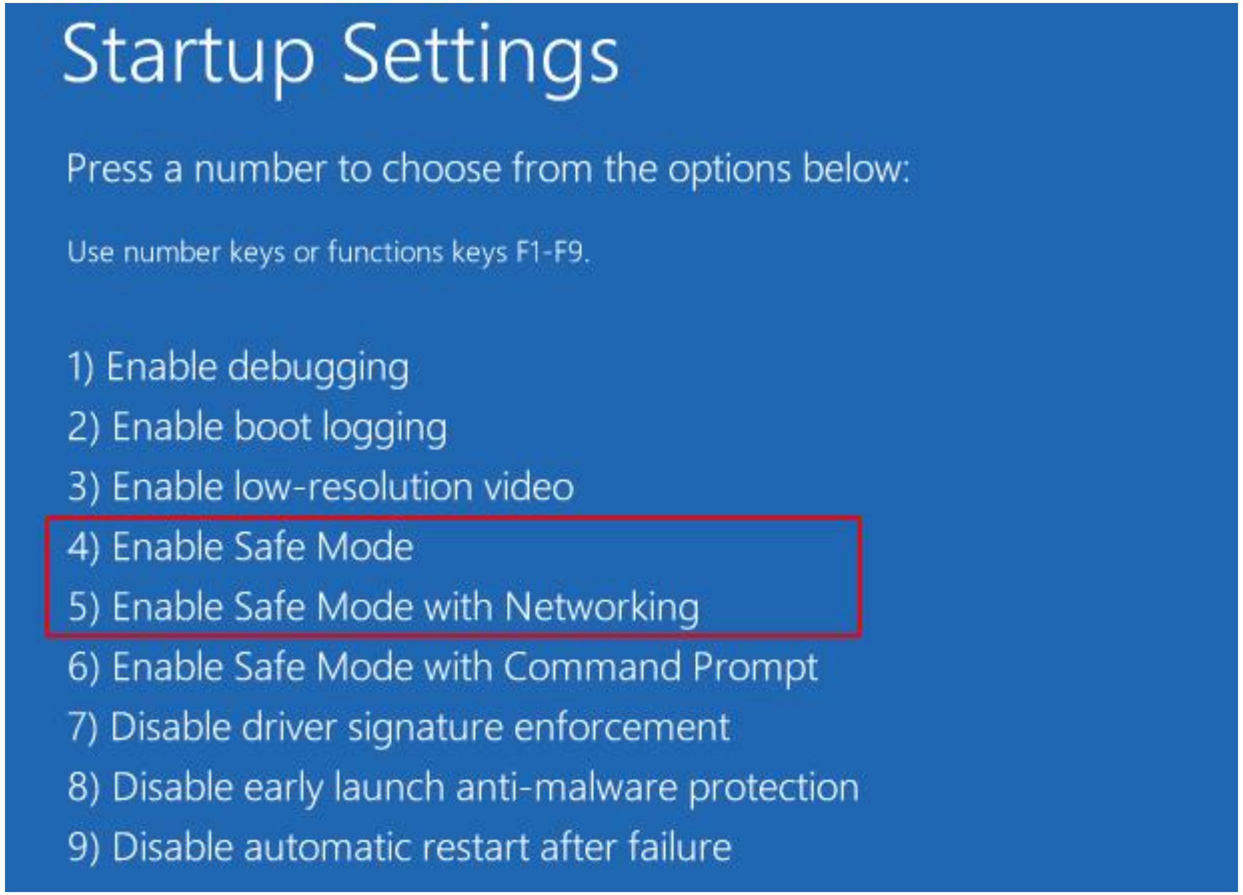
Watchdog timers

- What to do after WTD timeout?

Watchdog timers

- What to do after WTD timeout?





Startup Settings

Press a number to choose from the options below:

Use number keys or functions keys F1-F9.

1) Enable debugging
2) Enable boot logging
3) Enable low-resolution video
4) Enable Safe Mode
5) Enable Safe Mode with Networking
6) Enable Safe Mode with Command Prompt
7) Disable driver signature enforcement
8) Disable early launch anti-malware protection
9) Disable automatic restart after failure

Watchdog timers

- What to do after WTD timeout?
- Definitely capture your reboot reason

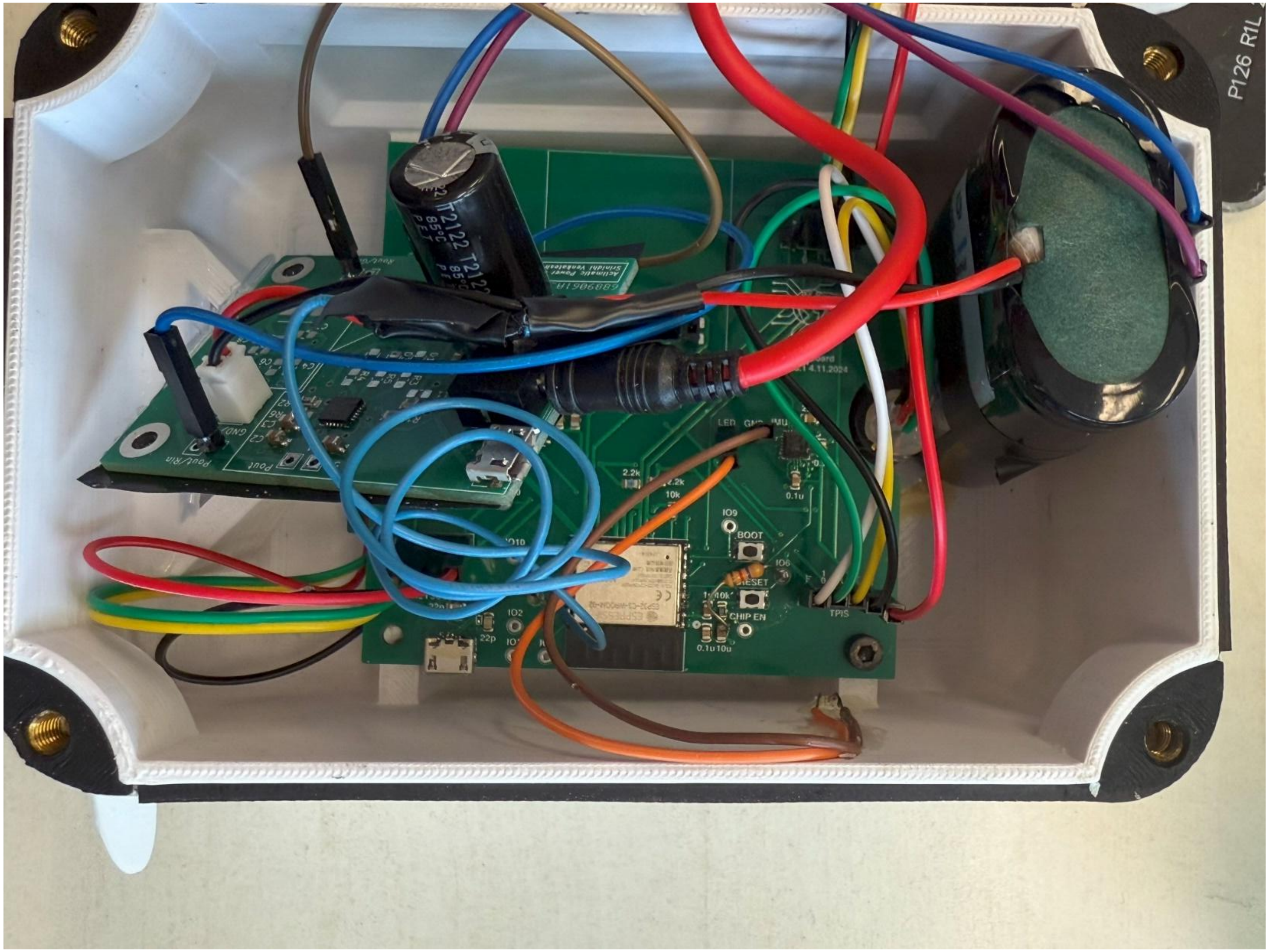
  esp_reset_reason()

  - This is an enum that will tell you whether reboot was due to WTD, deepsleep, brownout, power-on, etc.

- Then what?
- Some options
  - Act as if nothing bad happened
  - Go into “safe mode”
    - E.g., wait for firmware update

Rebooting

- Sometimes you need to really power cycle
- Make sure that is doable



How to power cycle this?

System variables you may want to track

- Raw and processed sensor data
  - All with timestamps
- Communications parameters
  - SSID, RSSI, etc.
  - Number of failed connection attempts
  - Number of failed POSTs, GETs
- Time
  - Can get time from internet
    - From your server
    - SNTP: Simple Network Time Protocol
  - Can keep time on ESP32
    - RTC timer works thru deep sleep
      - 150 kHz RC oscillator
      - Drift is temperature dependent ← *Anywhere from 1 sec/min to 1 sec/day*

Especially early on in testing/validation, keep raw-er data

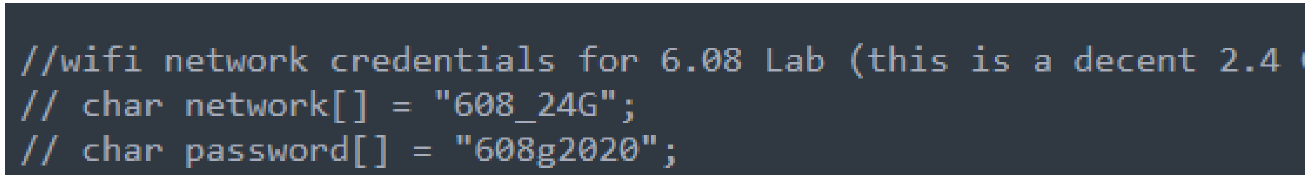
System variables you may want to track

- System data
    - System temperature & RH
        - ESP32C3 has an internal temperature sensor
    - Last reset reason
    - System memory utilization
        - Memory leak?

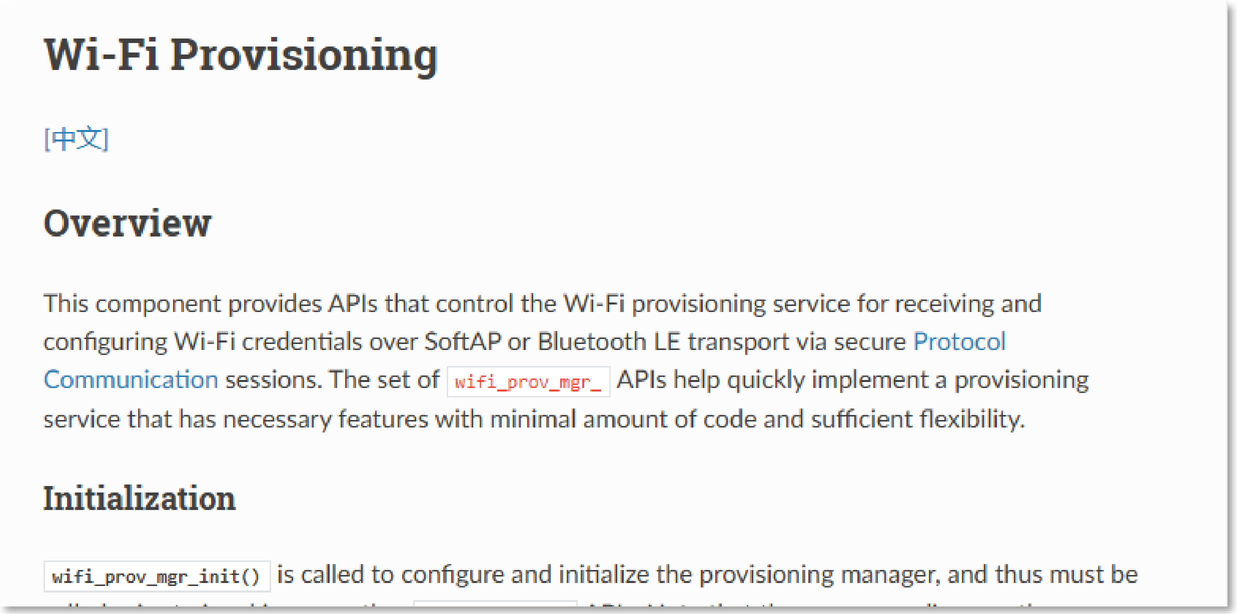
        ```
        heap_caps_get_free_size(MALLOC_CAP_DEFAULT);
        ```
    - Battery state
        - In lab01 we measured battery voltage
        - You can also use fuel gauge if desired

WiFI provisioning

- If using WiFi, can hardcode credentials
    - Good idea to have a set of possible SSIDs that you can connect to
- Or can add code to send credentials to ESP32 over BLE or WiFi



```
//wifi network credentials for 6.08 Lab (this is a decent 2.4
// char network[] = "608_24G";
// char password[] = "608g2020";
```



Wi-Fi Provisioning

[中文]

Overview

This component provides APIs that control the Wi-Fi provisioning service for receiving and configuring Wi-Fi credentials over SoftAP or Bluetooth LE transport via secure Protocol Communication sessions. The set of `wifi_prov_mgr_` APIs help quickly implement a provisioning service that has necessary features with minimal amount of code and sufficient flexibility.

Initialization

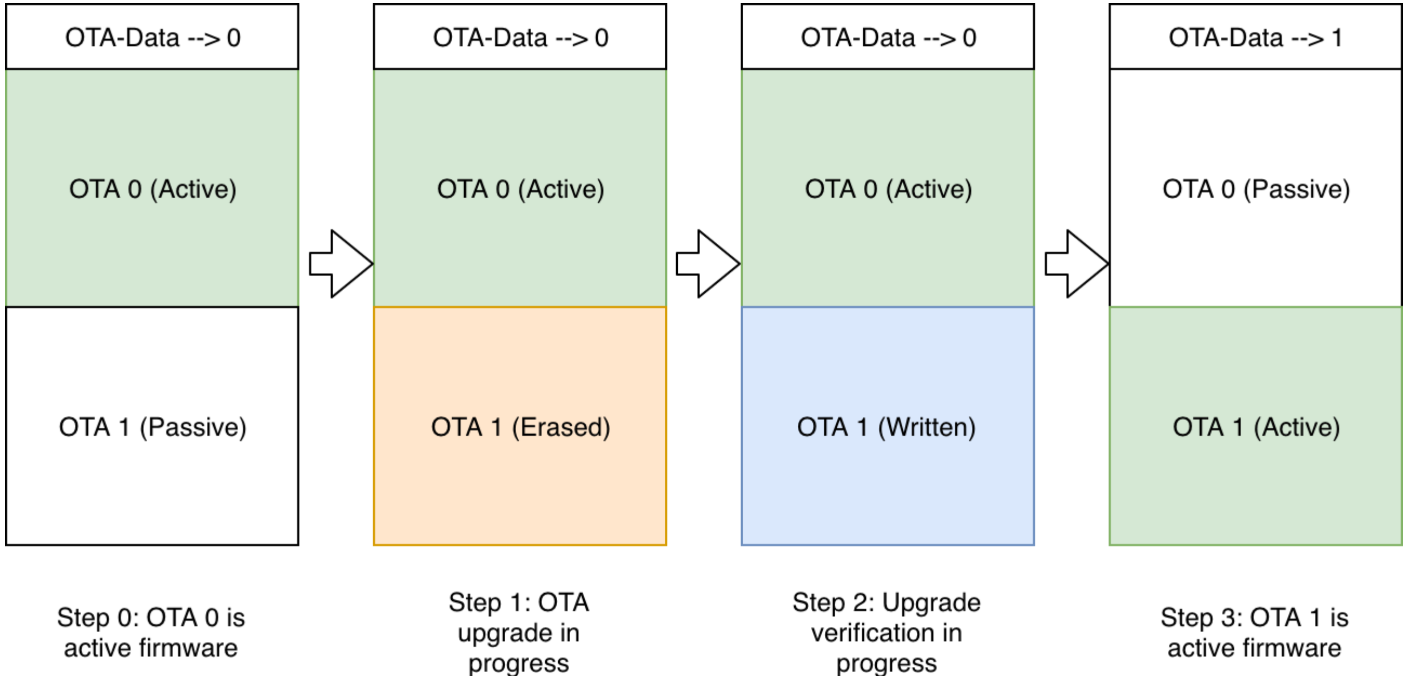
`wifi_prov_mgr_init()` is called to configure and initialize the provisioning manager, and thus must be

OTA updates

- It’s *possible* that your FW will not be perfect
    - It might have bugs
    - It might need new features
    - It might have a testing variant and a release variant
- ESP32 can do over-the-air (OTA) FW updates

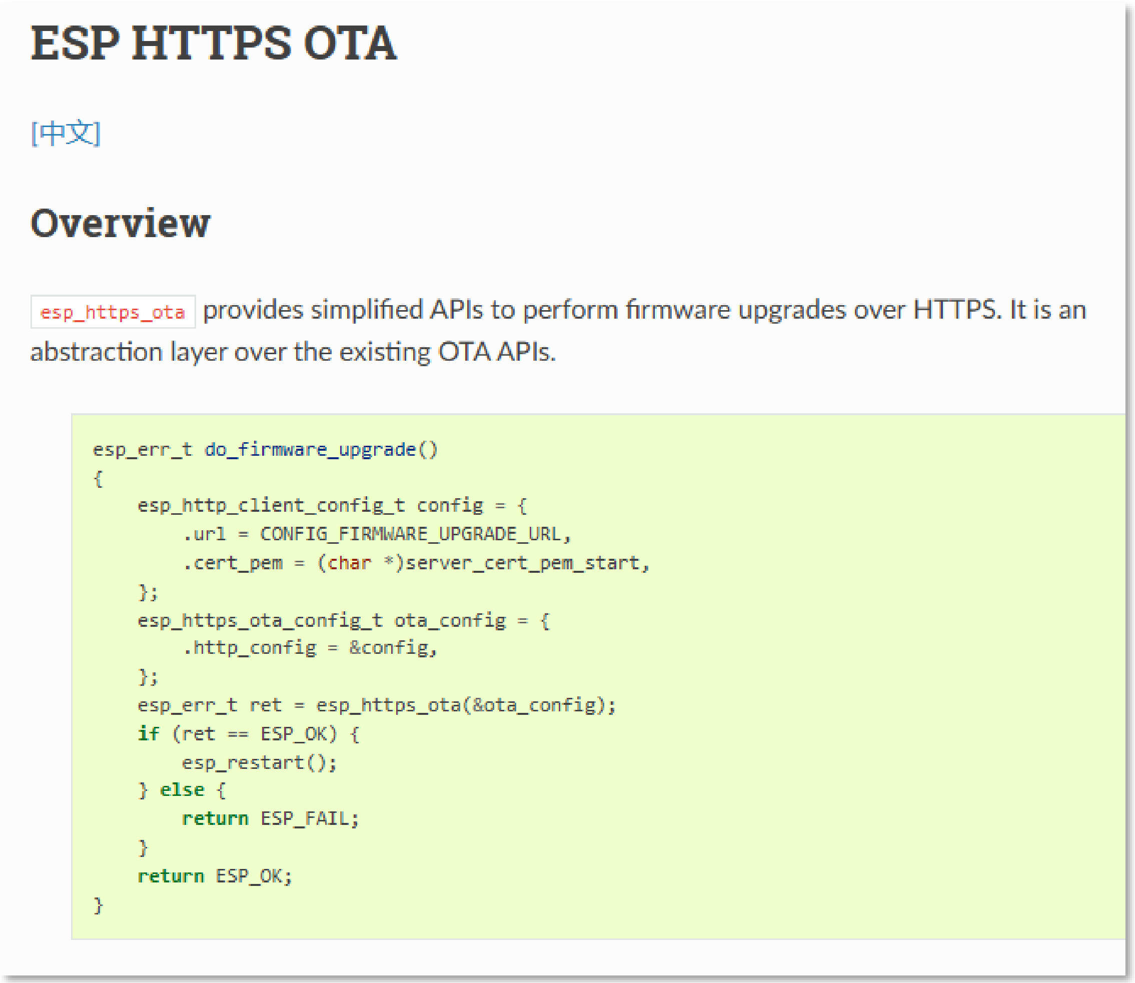
OTA updates

- It's *possible* that your FW will not be perfect
  - It might have bugs
  - It might need new features
  - It might have a testing variant and a release variant
- ESP32 can do over-the-air (OTA) FW updates
  - Decreases allowable FW size by ~2x



OTA updates

- It’s *possible* that your FW will not be perfect
  - It might have bugs
  - It might need new features
  - It might have a testing variant and a release variant
- ESP32 can do over-the-air (OTA) FW updates
- Built into ESP-IDF



ESP HTTPS OTA

[中文]

Overview

`esp_https_ota` provides simplified APIs to perform firmware upgrades over HTTPS. It is an abstraction layer over the existing OTA APIs.

```
esp_err_t do_firmware_upgrade()
{
    esp_http_client_config_t config = {
        .url = CONFIG_FIRMWARE_UPGRADE_URL,
        .cert_pem = (char *)server_cert_pem_start,
    };
    esp_https_ota_config_t ota_config = {
        .http_config = &config,
    };
    esp_err_t ret = esp_https_ota(&ota_config);
    if (ret == ESP_OK) {
        esp_restart();
    } else {
        return ESP_FAIL;
    }
    return ESP_OK;
}
```

OTA updates

- It’s *possible* that your FW will not be perfect
  - It might have bugs
  - It might need new features
  - It might have a testing variant and a release variant
- ESP32 can do over-the-air (OTA) FW updates
- Built into ESP-IDF

- Store program bin on server
  - With associated JSON telling FW version
- On ESP32
  - Go to server and get JSON
  - Check version against current
  - If update needed, pull new bin
  - Place into new OTA partition
  - Reboot and run new partition FW

OTA updates

- It should be possible to do OTA with cellular
  - Several teams have claimed in the past to do this
- Challenge with LoRa due to low data-rate vs. ~MB code size