PROGRAMMERING AV INBYGGDA SYSTEM

The ESP32

Dario Salvi, 2025

Materials

- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/
- ESP32 Technical Reference Manual
- Making embedded systems: chapter 1 “Introduction”
- Making embedded systems: chapter 3 “Getting the Code Working”
- Making embedded systems: chapter 4 “Outputs, Inputs, and Timers”

ESP32

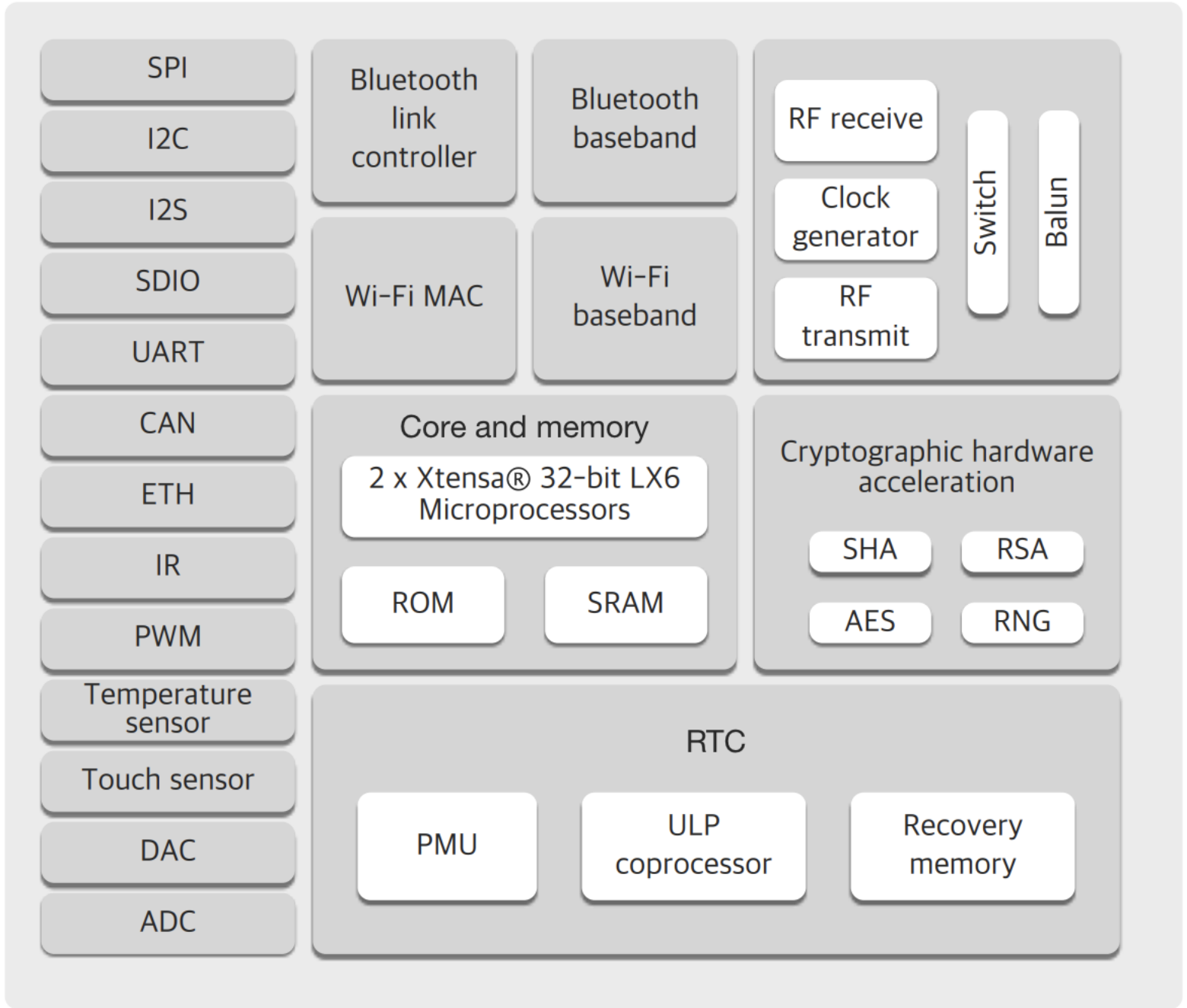
- Processors:
  - CPU: Xtensa dual-core 32-bit LX6 (160 or 240 MHz)
  - Ultra low power (ULP) co-processor
- Memory: 520 KiB SRAM
- Wireless connectivity:
  - Wi-Fi: 802.11 b/g/n
  - Bluetooth: v4.2 BR/EDR and BLE
- Security:
  - IEEE 802.11 WFA, WPA/WPA2 and WAPI
  - Secure boot
  - Flash encryption
  - 1024-bit OTP, up to 768-bit for customers
  - Cryptographic hardware acceleration
- Power management:
  - Internal low-dropout regulator
  - Individual power domain for RTC
  - 5µA deep sleep current
  - Wake up from GPIO interrupt, timer, ADC measurements, capacitive touch sensor interrupt

- Peripheral interfaces:
  - 12-bit SAR ADC up to 18 channels
  - 2 × 8-bit DACs
  - 10 × touch sensors (capacitive sensing GPIOs)
  - 4 × SPI
  - 2 × I^2S interfaces
  - 2 × I^2C interfaces
  - 3 × UART
  - SD/SDIO/CE-ATA/MMC/eMMC host controller
  - SDIO/SPI slave controller
  - Ethernet MAC interface with dedicated DMA
  - CAN bus 2.0
  - Infrared remote controller
  - Motor PWM
  - LED PWM (up to 16 channels)
  - Hall effect sensor
  - Ultra low power analog pre-amplifier

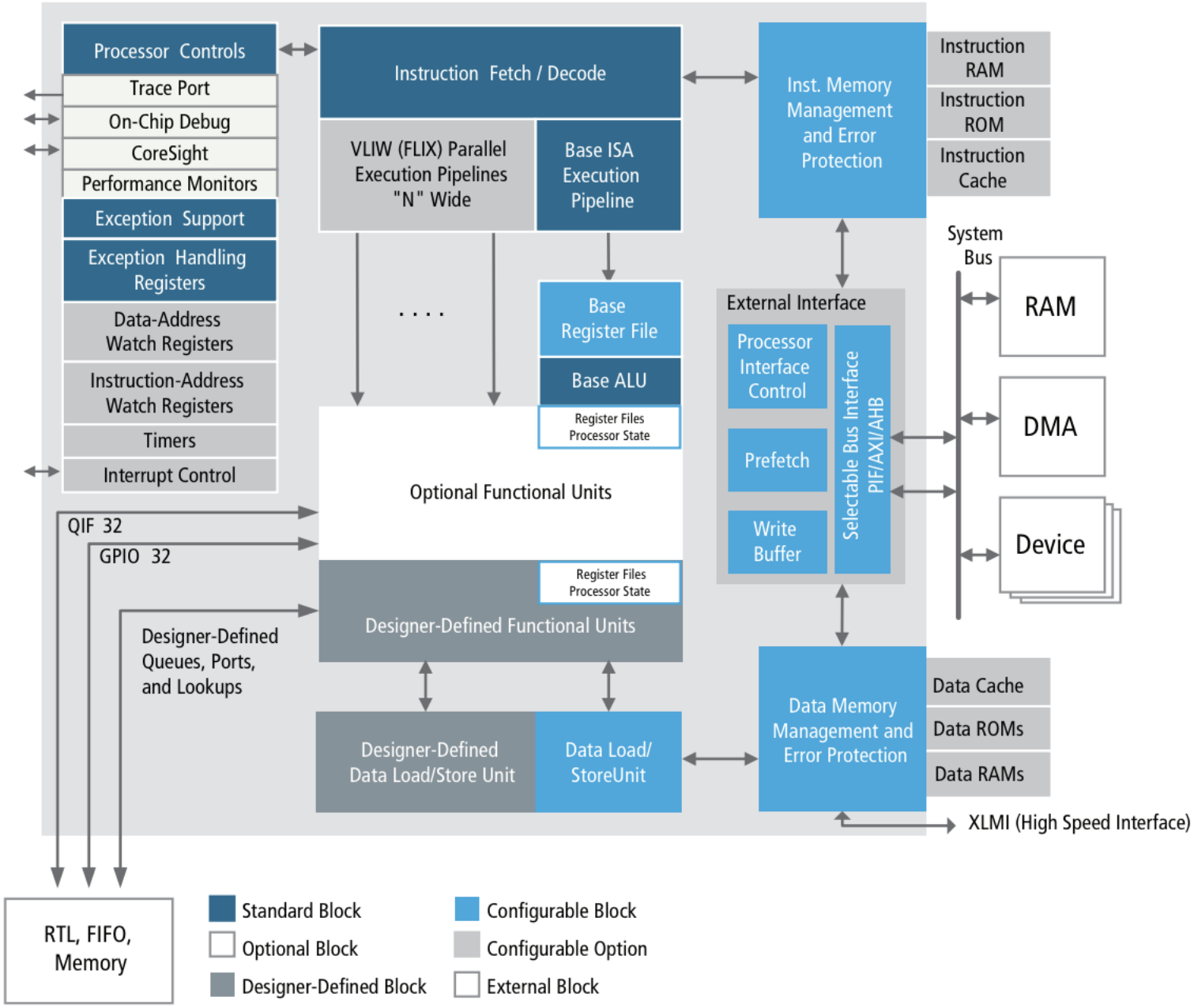
Comparison of ESP32 Series Microcontrollers

	ESP32	ESP32-S2	ESP32-S3	ESP32-C2	ESP32-C3	ESP32-C5	ESP32-C6	ESP32-H2	ESP32-P4
Release Date	2016	2020	2020	2022	2020	2022	2021	2021	2023 ⭐
Processor	Xtensa dual-core 32-bit LX6	Xtensa single-core 32-bit LX7	Xtensa dual-core 32-bit LX7	32-bit single-core RISC-V	32-bit single-core RISC-V	32-bit single-core RISC-V	32-bit single-core RISC-V	32-bit single-core RISC-V	Dual-core 32-bit RISC-V ⭐
Frequency	160/240 MHz	Up to 240 MHz	Up to 240 MHz	Up to 120 MHz	Up to 160 MHz	Up to 240 MHz	Up to 160 MHz	Up to 96 MHz	Up to 400 MHz (main), 40 MHz (low-power core) ⭐
SRAM	520 KB	320 KB	512 KB	272 KB	400 KB	400 KB	512 KB	256 KB	768 KB ⭐
ROM	448 KB	128 KB	384 KB	576 KB ⭐	384 KB	384 KB	320 KB	128 KB	8 KB TCM
Flash	Up to 4 MB	Up to 4 MB	Up to 8 MB ⭐	Up to 4 MB	Up to 4 MB	Up to 4 MB	Up to 4 MB	External	External
Wi-Fi	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 b/g/n, 2.4 GHz	802.11 ax, 2.4/5 GHz ⭐	802.11 ax, 2.4 GHz	❌	❌
Bluetooth	v4.2 BR/EDR and BLE	❌	v5.0 BLE	v5.0 BLE	v5.0 BLE	v5.2 BLE	v5.3 BLE ⭐	v5.0 BLE	❌
Zigbee	❌	❌	❌	❌	❌	802.15.4 (Thread, Zigbee)	802.15.4 (Thread, Zigbee)	❌	❌
GPIO	34	43	45 ⭐	14	22	20	30	19	50+ ⭐
ADC	Two 12-bit, 18 channels	Two 13-bit, 20 channels ⭐	Two 12-bit, 20 channels	One 12-bit, 5 channels	Two 12-bit, 6 channels	One 12-bit, x channels	One 12-bit, 7 channels	❌	❌
DAC	Two 8-bit channels ⭐	Two 8-bit channels ⭐	❌	❌	❌	❌	❌	❌	❌
SPI	4	4	4	3	3	2	2	2	❌
I2C	2	1	2	1	1	2	2	1	❌
I2S	2 ⭐	1	2 ⭐	1	1	1	1	❌	❌
RMT	8 channels ⭐	4 channels	8 channels ⭐	❌	4 channels	2 channels	2 channels	❌	❌
Touch Sensor	10	14 ⭐	14 ⭐	❌	❌	❌	❌	❌	❌
Hall Sensor	✔ ⭐	❌	❌	❌	❌	❌	❌	❌	❌
LCD Interface	✔	✔	✔	❌	❌	❌	❌	❌	Yes (via MIPI-DSI) ⭐
Camera Interface	✔	✔	✔	❌	❌	❌	❌	❌	Yes (via MIPI-CSI) ⭐
Deep Sleep	~10 µA	~20 µA	~10 µA	~5 µA ⭐	~5 µA ⭐	~5 µA ⭐	~5 µA ⭐	~5 µA ⭐	N/A
Size	5x5 mm or 6x6 mm	7x7 mm	7x7 mm	4x4 mm ⭐	5x5 mm	5x5 mm	5x5 mm	4x4 mm ⭐	Custom
Other Features	❌	USB OTG	USB OTG	❌	❌	❌	❌	❌	HMI, USB OTG, Ethernet, AI features ⭐

https://www.espboards.dev/blog/esp32-soc-options/


SPI
I2C
I2S
SDIO
UART
CAN
ETH
IR
PWM
Temperature sensor
Touch sensor
DAC
ADC
Bluetooth link controller
Bluetooth baseband
Wi-Fi MAC
Wi-Fi baseband
RF receive
Clock generator
RF transmit
Switch
Balun
Core and memory
2 x Xtensa® 32-bit LX6 Microprocessors
ROM
SRAM
Cryptographic hardware acceleration
SHA
RSA
AES
RNG
RTC
PMU
ULP coprocessor
Recovery memory

Xtensa LX6 microprocessor



Useful resources

- **Espressif ESP-IDF SDK**
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html
- ESP32 official forum
  - https://esp32.com/
- ESP32 projects
  - https://randomnerdtutorials.com/projects-esp32/
- Luca Dentella example projects
  - http://www.lucadentella.it/en/category/esp32/
  - https://github.com/lucadentella/esp32-tutorial

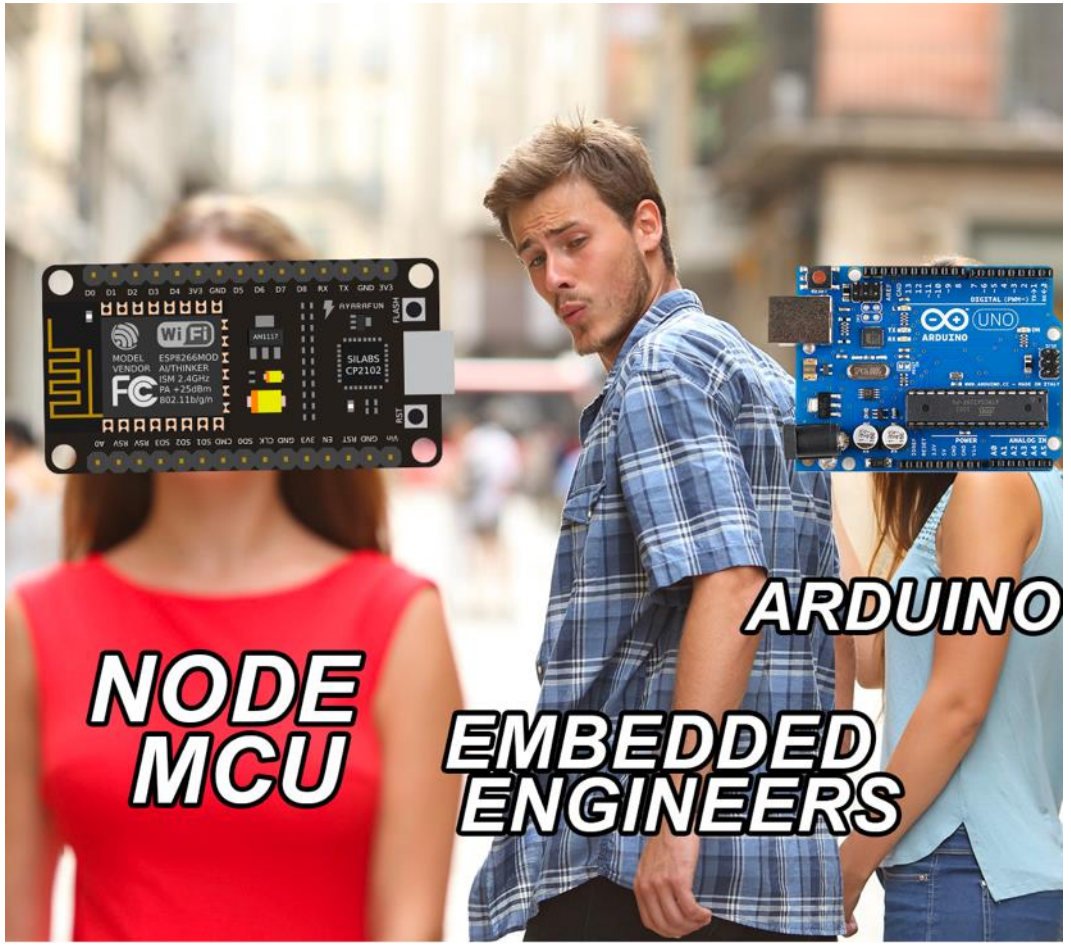
Cool projects and inspirations

- Cool projects for ESP32
  - https://github.com/agucova/awesome-esp
- Doom on ESP32
  - https://www.youtube.com/watch?v=y6PP_IBbOTY
- Gameboy emulator
  - https://github.com/lualiliu/esp32-gameboy
- Windows 3.0 on ESP32
  - https://hackaday.com/2021/07/28/emulating-the-ibm-pc-on-an-esp32/
- Andreas Spiess Youtube channel with loads of great projects
  - https://www.youtube.com/channel/UCu7_D0o48KbfhpEohoP7YSQ
- Bitluni's Youtube channel, some nice ESP32 projects
  - https://www.youtube.com/channel/UCp_5PO66faM4dBFbFFBdPSQ

On the board

- **ESP-WROOM-32:** This is the heart of the ESP32 module. It is a 32-bit microprocessor developed by Espressif systems.
- **Micro-USB jack:** The micro USB jack is used to connect the ESP32 to our computer through a USB cable. It is used to program the ESP module and for serial debugging as it supports serial communication
- **RST Button:** This is the reset button of the ESP module. Pressing this button will reset the code running on the ESP module
- **Boot Button:** This button is used to upload the code to the ESP module. It has to be pressed after clicking on the upload icon on the IDE. When the Boot button is pressed along with the RST button, ESP enters into firmware uploading mode. Do not play with this mode unless you know what you are doing.
- **Red LED:** The Red LED on the board is used to indicate communication.

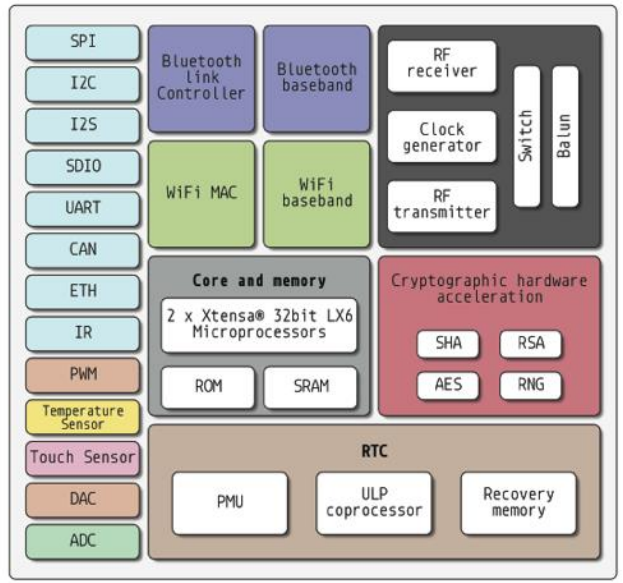
Why not Arduino?



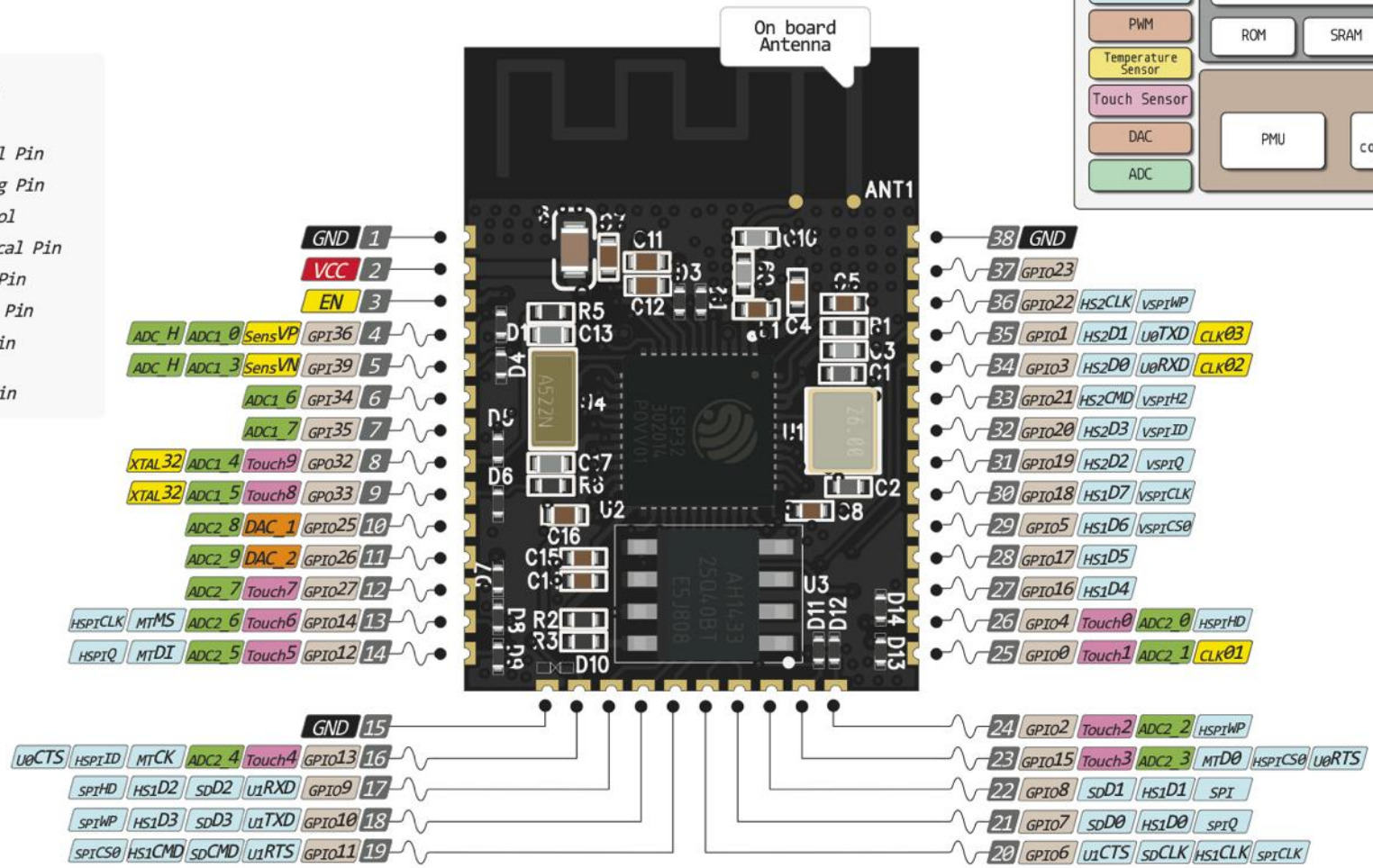
Pins

- The ESP32 chip comes with 48 pins with multiple functions. Not all pins are exposed in all ESP32 development boards, and there are some pins that cannot be used.
- Each specific GPIO works in the same way regardless of the development board you're using.
- The ESP32 comes with several peripherals including:
  - 18 Analog-to-Digital Converter (ADC) channels
  - 3 SPI interfaces
  - 3 UART interfaces
  - 2 I2C interfaces
  - 16 PWM output channels
  - 2 Digital-to-Analog Converters (DAC)
  - 2 I2S interfaces
  - 10 Capacitive sensing GPIOs
- The ADC (analog to digital converter) and DAC (digital to analog converter) features are assigned to specific static pins.
- You can decide which pins are UART, I2C, SPI, PWM, etc – you just need to assign them in the code. This is possible due to the ESP32 chip's multiplexing feature.

ESP32 PINOUT



Power
GND
Serial Pin
Analog Pin
Control
Physical Pin
Port Pin
Touch Pin
DAC Pin
PWM Pin



http://esp32.com/

bq

10 DEC 2015
ver 1 rev 3

3V3

EN Used by program logic. SW2 grounds via resistor.

GPIO34 ADC1_CH6 RTC_GPIO4

GPIO35 ADC1_CH7 RTC_GPIO5

GPIO32 32K_XP (32.768 kHz osc i/p) ADC1_CH4 TOUCH9 RTC_GPIO9

GPIO33 32K_XN (32.768 kHz osc o/p) ADC1_CH5 TOUCH8 RTC_GPIO8

GPIO25 DAC_1 ADC2_CH8 RTC_GPIO6 EMAC_RXD0

GPIO26 DAC_2 ADC2_CH9 RTC_GPIO7 EMAC_RXD1

GPIO27 ADC2_CH7 TOUCH7 RTC_GPIO17 EMAC_RX_DV

GPIO14 ADC2_CH6 TOUCH6 RTC_GPIO16 MTMS HSPI-CLK HS2_CLK SD_CLK EMAC_TXD2

GPIO12 ADC2_CH5 TOUCH5 RTC_GPIO15 MTDI HSPIQ HS2_DATA2 SD_DATA2 EMAC_TXD3

Gnd

GPIO13 ADC2_CH4 TOUCH4 RTC_GPIO14 MTCK HSPID HS2_DATA3 SD_DATA3 EMAC_RX_ER

GPIO9 SD_DATA2 SPIHD HS1_DATA2 U1RXD

GPIO10 SD_DATA3 SPIWP HS1_DATA3 U1TXD

! CMD

5V

Gnd

GPIO23 VSPID HS1_STROBE

GPIO22 VSPIWP U0RTS EMAC_TXD1

GPIO1 U0TXD CLK_OUT3 EMAC_RXD2

GPIO3 U0RXD CLK_OUT2

GPIO21 VSPIHD EMAC_TX_EN

Gnd

GPIO19 VSPIQ U0CTS EMAC_TXD0

GPIO18 VSPICLK HS1_DATA7

GPIO5 VSPICS0 HS1_DATA6 EMAC_RX_CLK

GPIO17 HS1_DATA5 U2TXD EMAC_CLK_OUT_180

GPIO16 HS1_DATA4 U2RXD EMAC_CLK_OUT

GPIO4 ADC2_CH0 TOUCH0 RTC_GPIO10 HSPIHD HS2_DATA1 SD_DATA1 EMAC_TX_ER

GPIO0 ADC2_CH1 TOUCH1 RTC_GPIO11 CLK_OUT1 EMAC_TX_CLK

GPIO2 ADC2_CH2 TOUCH2 RTC_GPIO12 HSPIWP HS2_DATA0 SD_DATA0 External pulldown

GPIO15 ADC2_CH3 TOUCH3 RTC_GPIO13 MTDO HSPICS0 HS2_CMD SD_CMD EMAC_RXD3

GPIO8 SD_DATA1 SPID HS1_DATA1 U2CTS

GPIO7 SD_DATA0 SPIQ HS1_DATA0 U2RTS

GPIO6 SD_CLK SPICLK HS1_CLK U1CTS

ADC: FSD = 4095 = 1.109V (Because 693mV gave 2559. Is the limit 1.0V?)

DAC: FSD = 255 = 3.19V (Vs = 3.3V). 127 gave 1.63V implying 3.3V FS.

Used for internal flash, not recommended for other use

Input only. No internal pullup or pulldown.

Used by USB/REPL

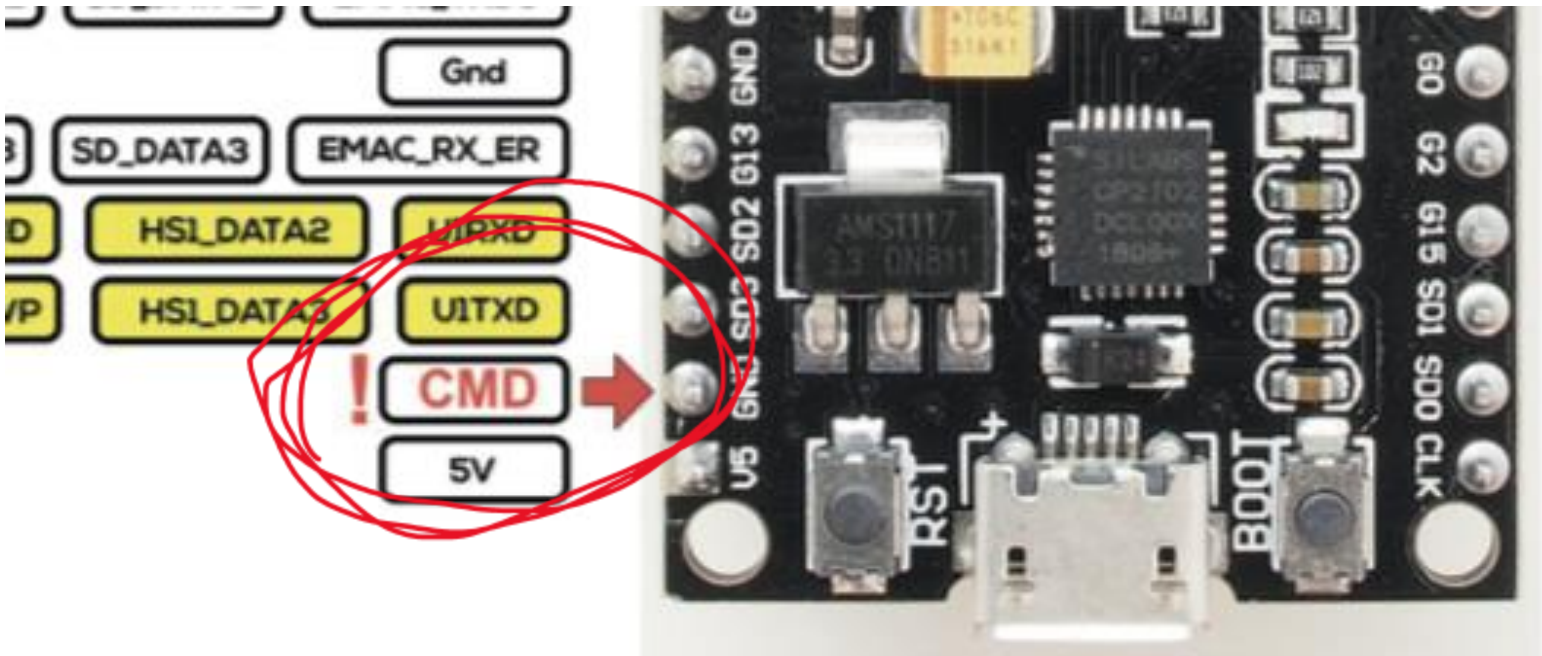
GPIO0 has a 5KΩ external pullup. SW0 grounds via 470Ω

Used on ESP32-WROVER-KIT etc to access external SPI RAM

Value	Expected	Actual	Error %
10	0.13	0.21	2.4
20	0.26	0.33	2.1
127	1.64	1.63	-0.3
200	2.58	2.53	-1.5
240	3.11	3.01	-3
255	3.3	3.19	-3.3

Remapping peripherals:
uart = machine.UART(1,baudrate=115200,tx=25,rx=26)

ESP32-D2WD is the chip with embedded 2MB flash and the internal flash is connected to different pins (GPIO16, GPIO17, SD_CMD, SD_CLK, SD_DATA_0 and SD_DATA_1)


Gnd
SD_DATA3
EMAC_RX_ER
HS1_DATA2
U1RXD
HS1_DATA3
U1TXD
!
CMD
5V
AMS1117
3.3
RST
BOOT
G0
G2
G15
SD0
CLK
G13
SD2
GND
5V

Safety Precaution:

All GPIO runs at 3.3V !!!

GPIO current drawn

The absolute maximum current drawn per GPIO is 40mA

What pins can I use?

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED
3	OK	RX pin	HIGH at boot
4	OK	OK	
5	OK	OK	outputs PWM signal at boot
6	x	x	connected to the integrated SPI flash
7	x	x	connected to the integrated SPI flash
8	x	x	connected to the integrated SPI flash
9	x	x	connected to the integrated SPI flash
10	x	x	connected to the integrated SPI flash
11	x	x	connected to the integrated SPI flash
12	OK	OK	boot fail if pulled high
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	
23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

https://randomnerdtutorials.com/esp32-pinout-reference-gpios/

Datasheet

- The main reference for the ESP32 is the Technical Reference Manual
- The chip manual is your bible. You will have to learn how to navigate it as you would learn how to navigate a new software library.
- You don’t need to learn everything by heart, but you need to know **where to find information**.
- Google is your friend, but the Datasheet is your best friend.
- Reading datasheets is an art, one that requires experience and patience. These technical documents are packed of information and are written by electrical engineers for electrical engineers.
- Datasheets (of other components) are also needed when you need to integrate with external chips (more on this later…)

Quick go through the ESP32 datasheet

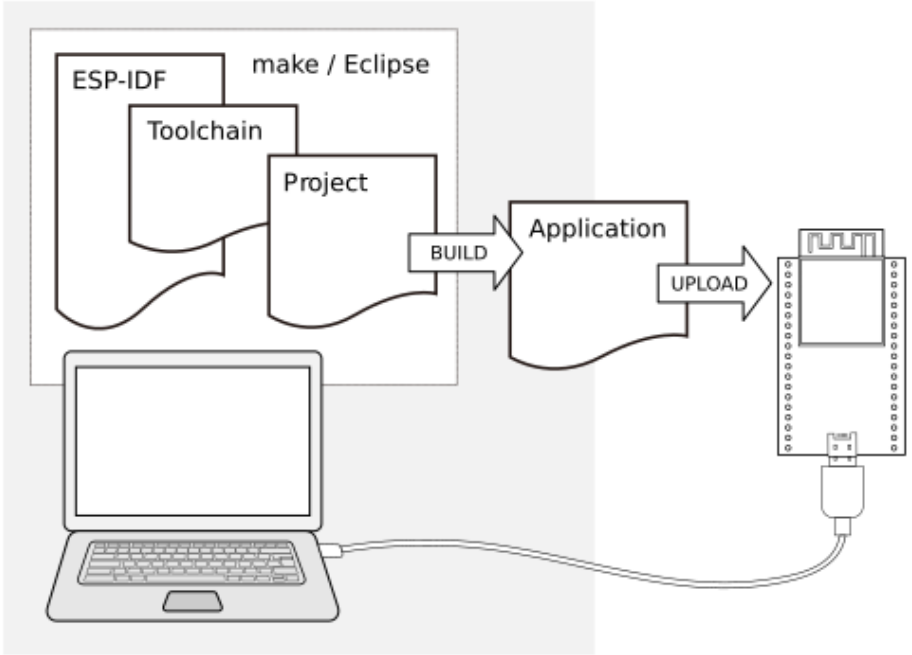
- Available on Canvas.
- Be aware of the difference between the “datasheet” and the “technical reference manual”!

Programming the ESP32

- The ESP-IDF (Espressif IoT Development Framework) and related toolchain is completely open source and can be used in different ways:
  - Command line
  - With Eclipse
  - With the Arduino IDE
  - With **VSCode** (<- our choice)

The tools

- **Toolchain** to compile code for ESP32
- **Build tools** - CMake and Ninja to build a full Application for ESP32
- **ESP-IDF** essentially contains an API (software libraries and source code) for the ESP32 and scripts to operate the **Toolchain**
- **Text editor** to write programs in C (Eclipse, Notepad, Vim)



Arduino

- EspressIF provides libraries compatible with Arduino.
- These abstract some common functions with a simplified API.
  - https://www.arduino.cc/reference/en/
- You can also use the Arduino IDE to program the ESP, but it's a bit limited.
- If interested, see: https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/

Platformio

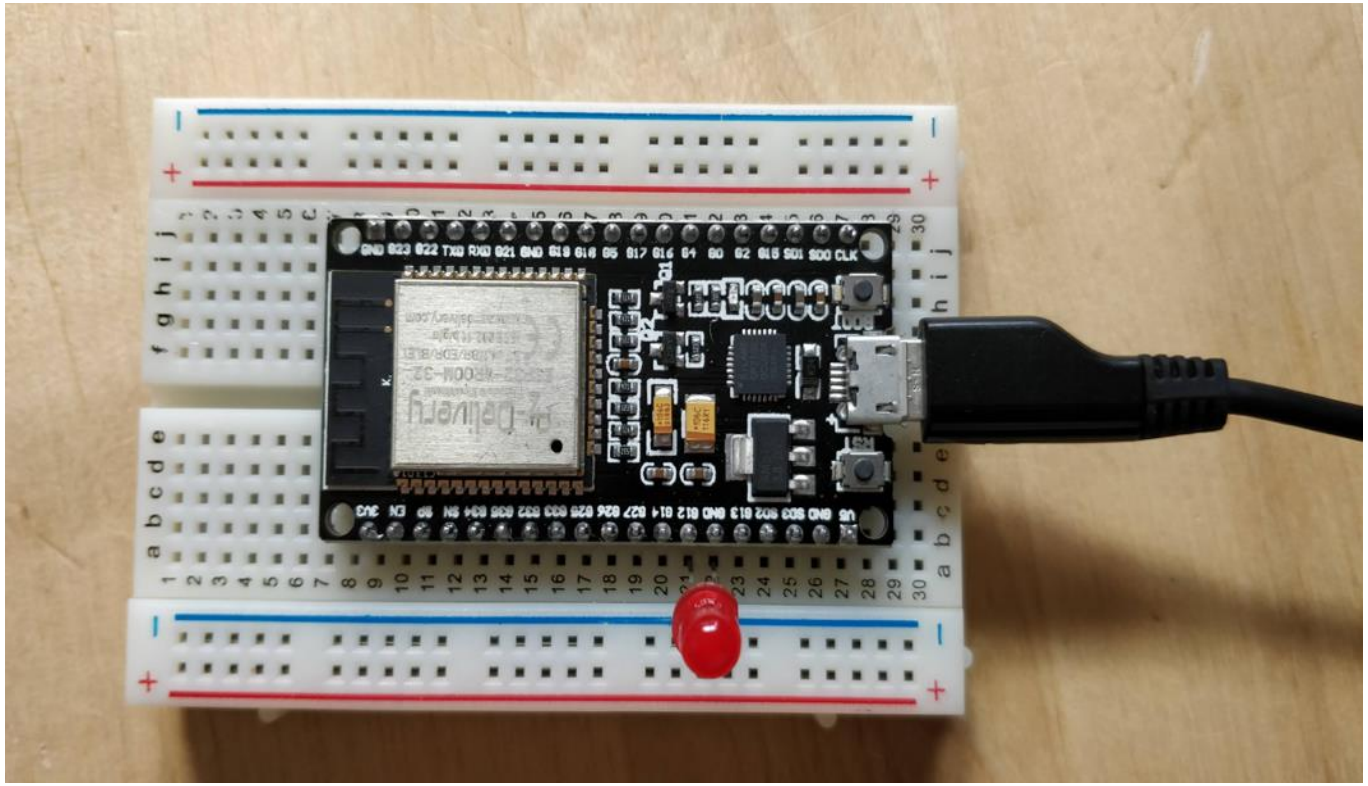
- We are going to use Visual Studio Code with the Platformio plugin.
- https://platformio.org/
- Compilers, toolchain and libraries all in one place.
- Easy to use and multiplatform.
- **Includes both the Arduino libraries and the ESP IDF**.

Arduino code structure

- Setup()
  - Is executed once at startup.
- Loop()
  - Is an infinite loop that repeats itself forever.

Build the hardware

- Connect an LED between GPIO 13 and GND



Live coding

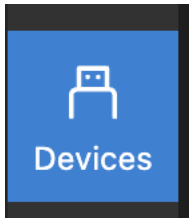
- Example of how to blink an LED with the Arduino libraries.
- See Canvas.

Configuring Platformio

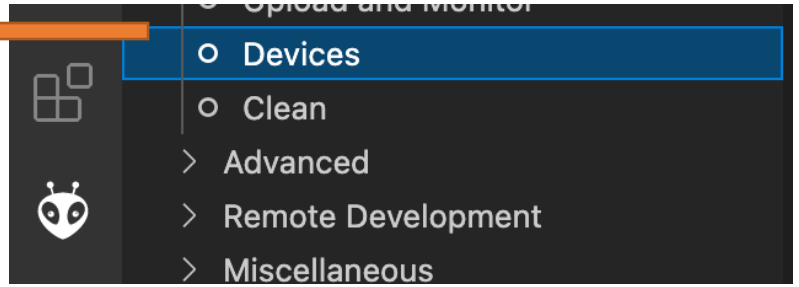
- Remember to setup platfromio.ini!

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s
framework = espidf
monitor_speed = 115200
upload_port = /dev/cu.SLAB_USBtoUART
```

1) From the Platformio home page



2) From the commands palette



Compile / run / clean / debug

Compile

Upload / Run

Clean

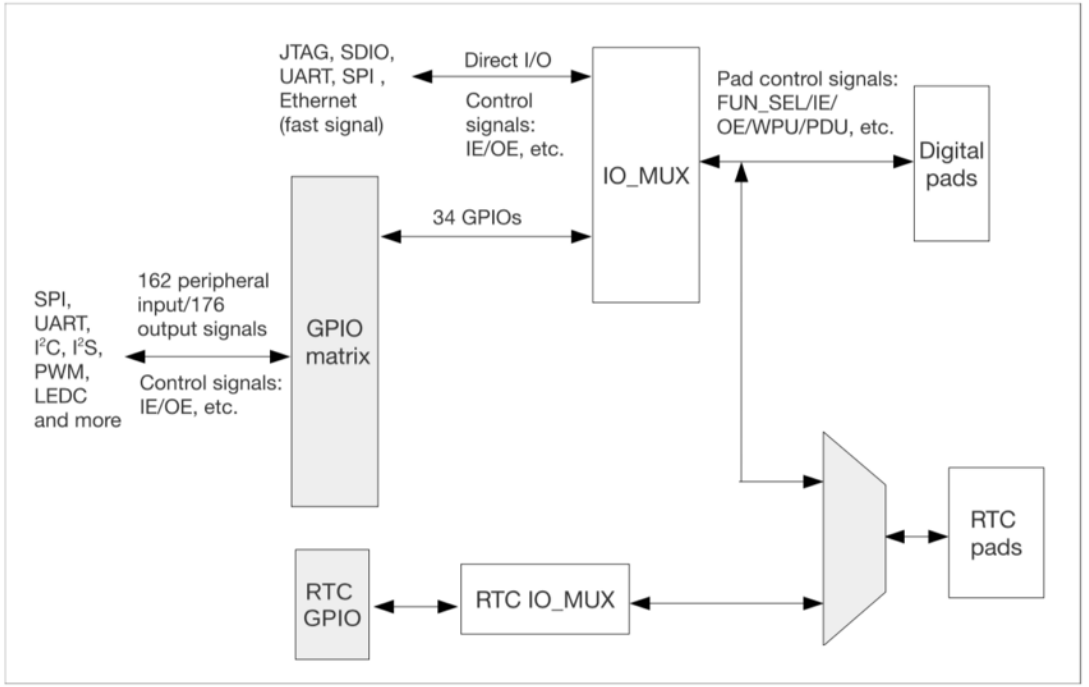
Open serial

Open terminal

Some of these are only visible when the project has been correctly identified as a Platformio project. If they are not there, re-create the project using Platformio.

GPIOs on the ESP32

- The ESP32 chip features 40 physical GPIO pads.
  - GPIO6-11 are usually used for SPI flash.
  - GPIO34-39 can only be set as input mode
- Each pad can be used as a general-purpose I/O, or be connected to an internal peripheral signal.
- The IO_MUX, RTC IO_MUX and the GPIO matrix are responsible for routing signals from the peripherals to GPIO pads.
- Separate “RTC GPIO” functions when GPIOs are routed to the “RTC” low-power subsystem. These pin functions can be used when in deep sleep.



GPIO Matrix

- The GPIO Matrix is a full-switching matrix between the peripheral input/output signals and the pads. To configure it, use registers:
  - GPIO_ENABLE_REG (pins 0 to 31) and GPIO_ENABLE1_REG (pins 31 to 39) enable the pin
  - GPIO_PINn_REG (one per pin n): configures the behaviour of the interrupts.
  - GPIO_FUNCn_IN_SEL_CFG_REG (one per pin n) and GPIO_FUNCn_OUT_SEL_CFG_REG select the input or output function for pin n.
  - GPIO_OUT_W1TS_REG: set the output of GPIOs 0-31 to HIGH (GPIO_OUT1_W1TS_REG for pins 31-39).
  - GPIO_OUT_W1TC_REG: set the output of GPIOs 0-31 to LOW (GPIO_OUT1_W1TC_REG for pints 31-39)
  - GPIO_IN_REG: reads the value of pins 0-31 (GPIO_IN1_REG for pins 31/39).
- For input to the chip: Each of the 162 internal peripheral inputs can select any GPIO pad as the input source.
- For output from the chip: The output signal of each of the 34 GPIO pads can be from one of the 176 peripheral output signals.

IO_MUX

- The IO_MUX contains one register per GPIO pad: IO_MUX_n_REG
- Each pad can be configured to perform a ”GPIO” function (when connected to the GPIO Matrix) or a direct function (bypassing the GPIO Matrix).
- Some high-speed digital functions (Ethernet, SDIO, SPI, JTAG, UART) can bypass the GPIO Matrix for better high-frequency performance (set in GPIO_FUNCy_IN_SEL_CFG_REG and GPIO_FUNCy_OUT_SEL_CFG_REG).
  - In this case, the IO_MUX is used to connect these pads directly to the peripheral.

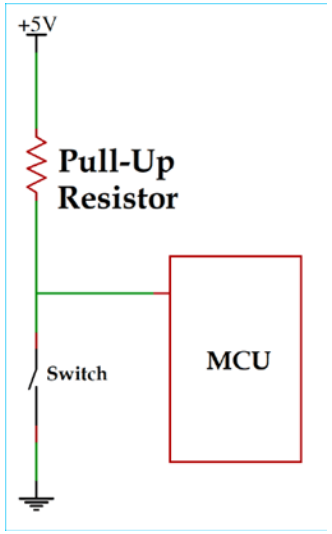
RTC IO_MUX

- RTC IO_MUX is used to connect GPIO pads to their low-power and analog functions. Only a subset of GPIO pads have these optional ”RTC” functions.

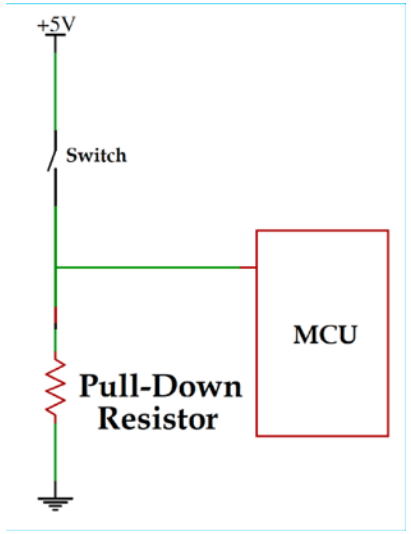
Live coding

- Programming the blink example using direct access to registers inside the ESP-IDF.
- See Canvas.

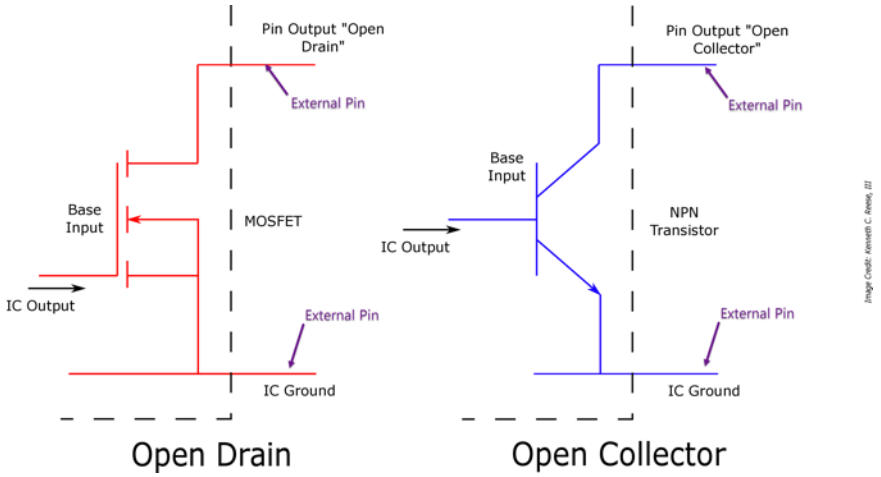
Pull-up / Pull-down / open drain



Pullup: the pin is internally connected to Vcc through a resistor.



Pulldown: the pin is internally connected to GND through a resistor.

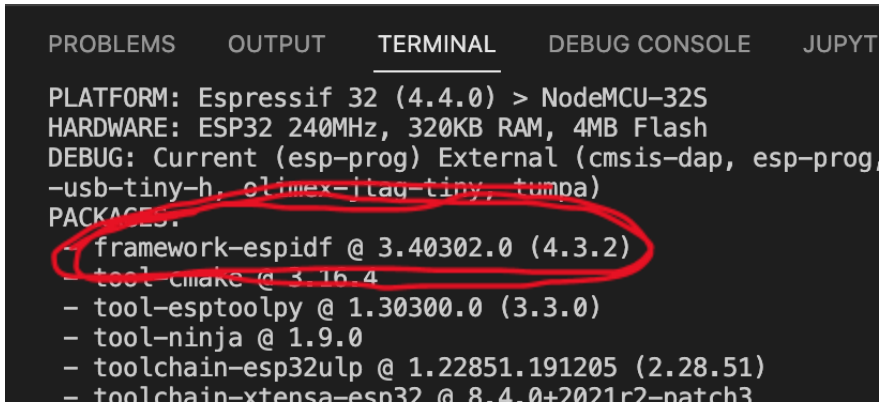


Open collector/drain: A pin configured in *open drain* mode can only sink current (if set to a logic level **0**) but cannot provide current to an external load; if the pin is set to logic level **1**, it is **disconnected** from the external circuitry

Using the ESP-IDF

- Which version?
  - When you install Platformio and create a ESP-IDF project, it will download a certain version.
  - The actual version depends… You can retrieve it, when compiling the project, by looking at the compilation messages, at the beginning:
  - **The API changes depending on the version!**
  - The documentation for each version is online:

https://docs.espressif.com/projects/esp-idf/en/v4.3.2/esp32/get-started/index.html



Using the ESP-IDF

- You can pin the version of the platform in platformio.ini

```
[env:nodemcu-32s]
platform = espressif32@3.5.0
board = nodemcu-32s
framework = espidf
monitor_speed = 115200
```

- Each version of the platform uses a selected version of the ESP-IDF framework, see https://github.com/platformio/platform-espressif32/releases
- Let's use:
  - platform v 5.1.1
  - ESP-IDF: 4.4.1

3.5.0

- Added new boards:
  - Franzininho WiFi
  - LionBit
- Updated ESP-IDF to `v4.3.2` (release notes)
- Updated toolchains for ESP-IDF to `v8.4.0r2-patch2`
- Removed redundant PSRAM fix for M5 Stack Core2 (#676)

Using the ESP-IDF

- You can use
  - esp_err_t **gpio_config**(const gpio_config_t *pGPIOConfig)
    - Needs to include "driver/gpio.h"
    - The gpio_config_t contains:
      - **pin_bit_mask**: 64 bits indicating which GPIO(s) you want to configure
      - **mode**: enum indicating the possible configurations (disabled, input, output, output with open drain, input and output with open drain, output and input)
      - **pull_up_en**: enum to specify if you want to use an internal pull-up resistor
      - **pull_down_en**: enum to enable an internal pull-down resistor
      - **intr_type**: interrupt type (more about this later…)
  - esp_err_t **gpio_set_level**(gpio_num_t gpio_num, uint32_t level)
    - Level must be 0 or 1
  - void **ets_delay_us**(uint32_t us)
    - Keeps the MCU busy doing nothing for us microseconds.

Live coding

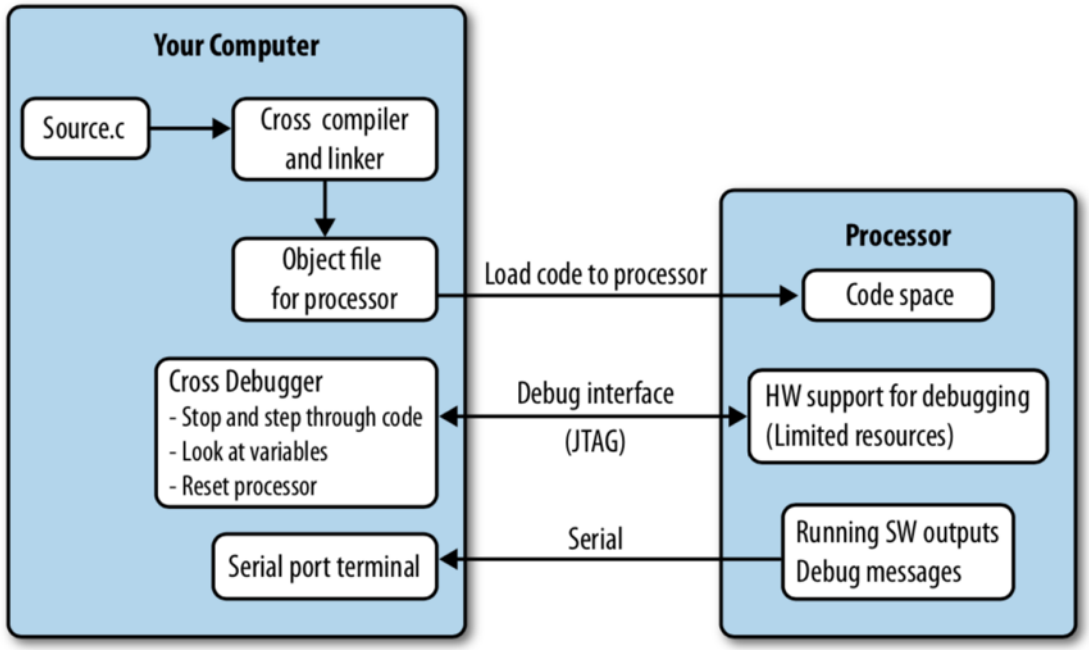
- Programming the blink example using the ESP-IDF.
- See Canvas.

Arduino vs Registers vs ESP/IDF

- Manipulating registers is usually harder, use libraries when provided.
  - The only exception is when you need absolute control on low level operations.
- Arduino is usually more “friendly” but more limited.
- The ESP/IDF is **very extensive**, but powerful!
- The Arduino framework can be included as a library inside the ESP-IDF if you want to mix the two.
- We are going to focus on the ESP/IDF from now on.

Debugging

- If you were to debug software running on a computer, you could compile and debug on that computer. The system would have enough resources to run the program and support debugging it at the same time.
- In embedded systems, in addition to a cross compiler, you'll need a **cross debugger**. The debugger sits on your computer and communicates with the target processor through a special processor interface often called JTAG.



Debugging

- The MCU must expend some of its resources to support the debug interface, allowing the debugger to halt it as it runs and providing debug information.
- To keep costs down, some processors support a **limited subset of features**.
- For example, adding a breakpoint causes the processor to modify the machine code to say “stop here.” If your code is programmed to read-only memory, instead of modifying the machine code, the processor has to set an internal register (hardware breakpoint) and compare it at each execution cycle to the address being run, stopping when they match.
- This can change the timing of the code**, leading to annoying bugs that occur only when you are debugging**.

Printf

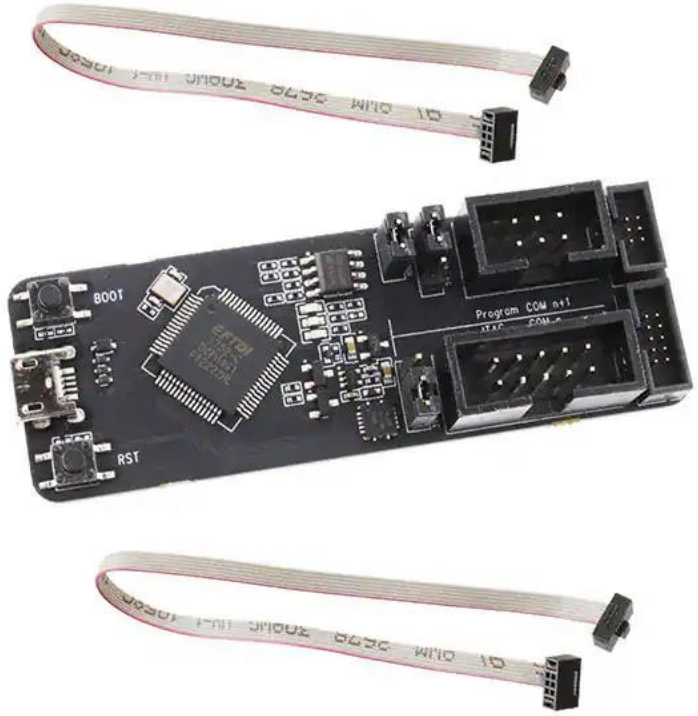
- To avoid buying a debugger you can still debug your code via printf or some sort of light-weight logging to an otherwise unused communication port.
- While incredibly useful, this can also change the timing of the system, possibly leaving some bugs to be revealed only after debugging output is turned off.

Live coding

- Adding printf() to both Arduino and ESP-IDF.
- See Canvas.

JTAG on ESP32

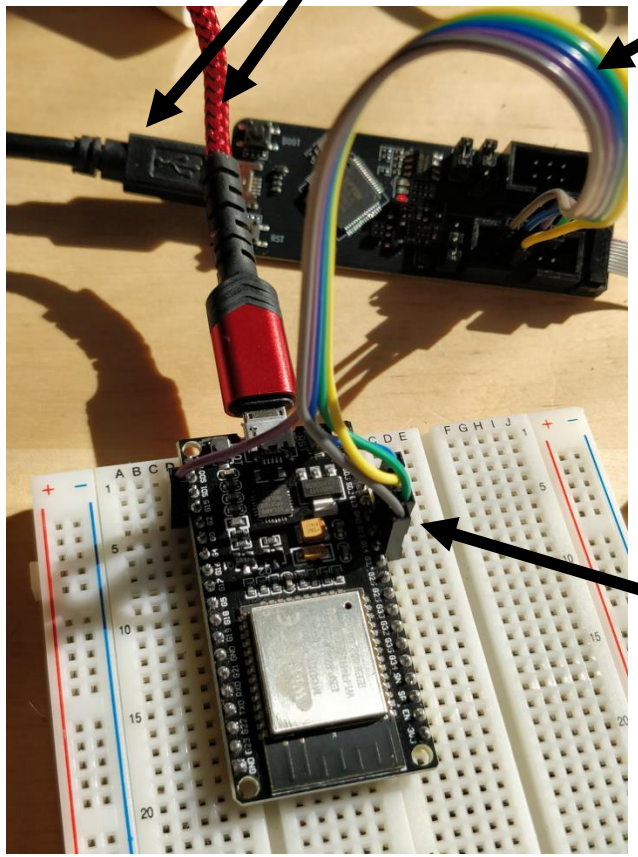
- You need an external hardware, a JTAG programmer.
  - Example: the ESP-PROG
- https://docs.platformio.org/en/latest/plus/debug-tools/esp-prog.html
- https://medium.com/@manuel.bl/low-cost-esp32-in-circuit-debugging-dbbee39e508b
- https://www.youtube.com/watch?v=psMqilqlrRQ
- https://www.youtube.com/watch?v=TivyIFF-dzw

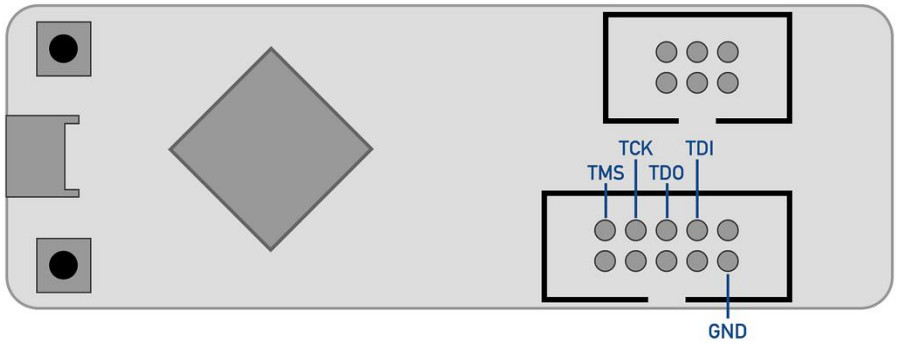


Setup

2 USB cables

Male-female connectors





- GPIO12 — TDI
- GPIO15 — TDO
- GPIO13 — TCK
- GPIO14 — TMS

Live coding

- Using the JTAG programmer for debugging.

Example exam questions

- Name 5 features included in the ESP32.
- What is a datasheet? Why do you need it?
- How can software running on a microcontroller be debugged?
- Describe what pull-down, pull-up and open drain configurations are.
- How can a pin configured with an internal pull-up used for?
- You are connecting a pushbutton to a pin of the MCU on one end and to Vcc on the other end. How should you configure your pin to detect when the button is pressed? You have the following 2 options to configure:
  - Direction: input/output
  - Connection: direct/pulldown/pullup.