deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

ESPRESSIF ESP32-DEVKITC

Software aspects and ADC demonstration

Part 2

TP1 - Grupo 7
Lúcia Sousa 93086
Raquel Pinto 92948

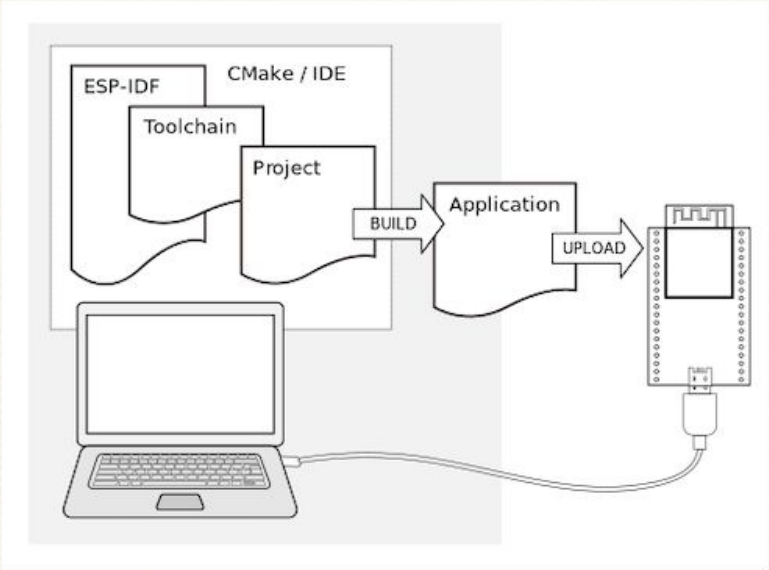
29/03/2022 ASE - Arquitetura de Sistemas Embutidos

Languages and development tools

- The lowest level option is **ESP-IDF** (Espressif IoT Development Framework), a software development kit.
- The mid-level option is **Arduino IDE** adapted for the ESP32 hardware, the language used is C++.
- The high-level option is **MicroPython** for ESP32.
- Languages: Lua, JavaScript

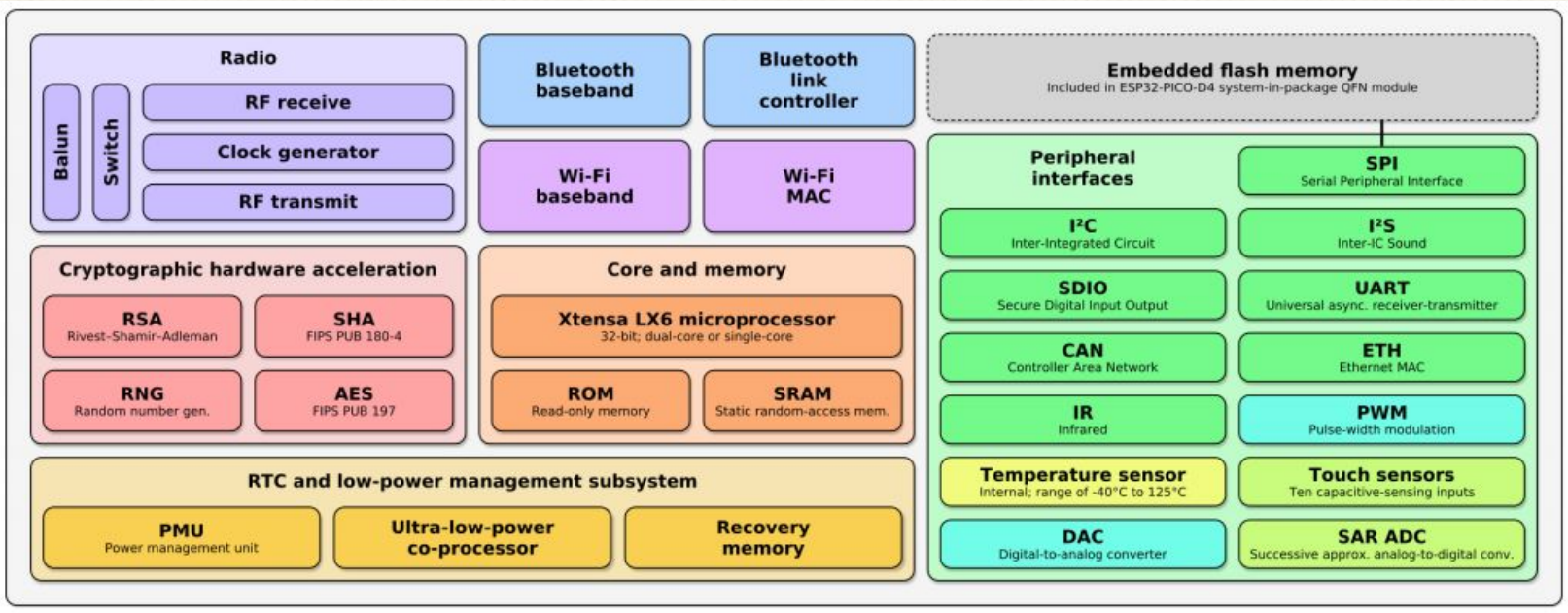
ESP-IDF

- Required: ESP32 board, USB cable and a computer running Windows, Linux, or macOS.
- To use ESP-IDF on ESP32 it is required:
  - **toolchain** to compile code for ESP32;
  - **build tools** (CMake and Ninja);
  - **ESP-IDF** that essentially contains API (software libraries and source code) for ESP32 and scripts to operate the toolchain.



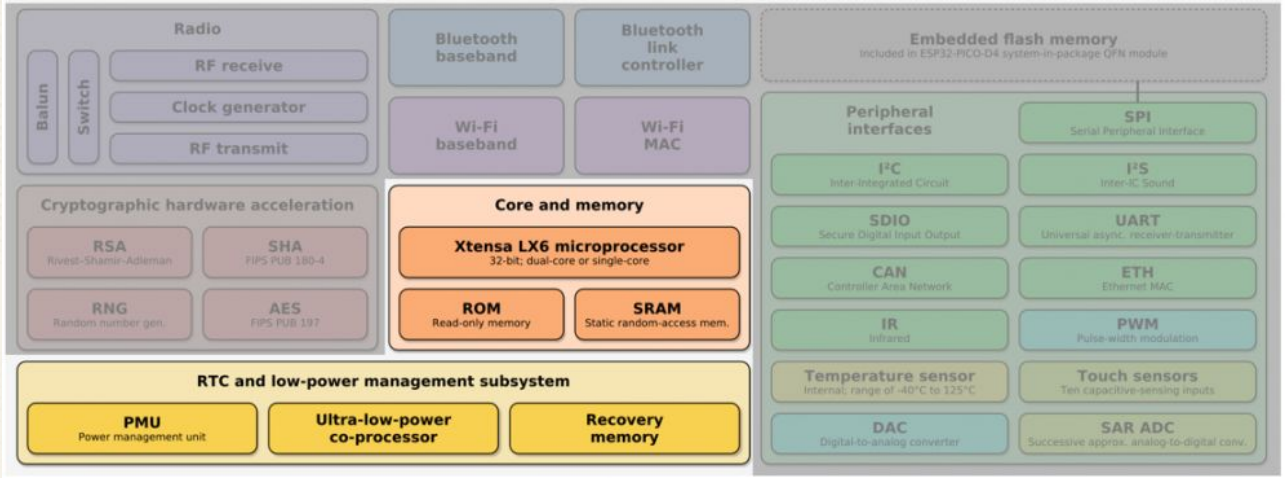
Power Modes

- Active Mode - All the features of the chip are active.
  - 160 - 260 mA or 790mA (Wi-Fi and Bluetooth together at the same time)



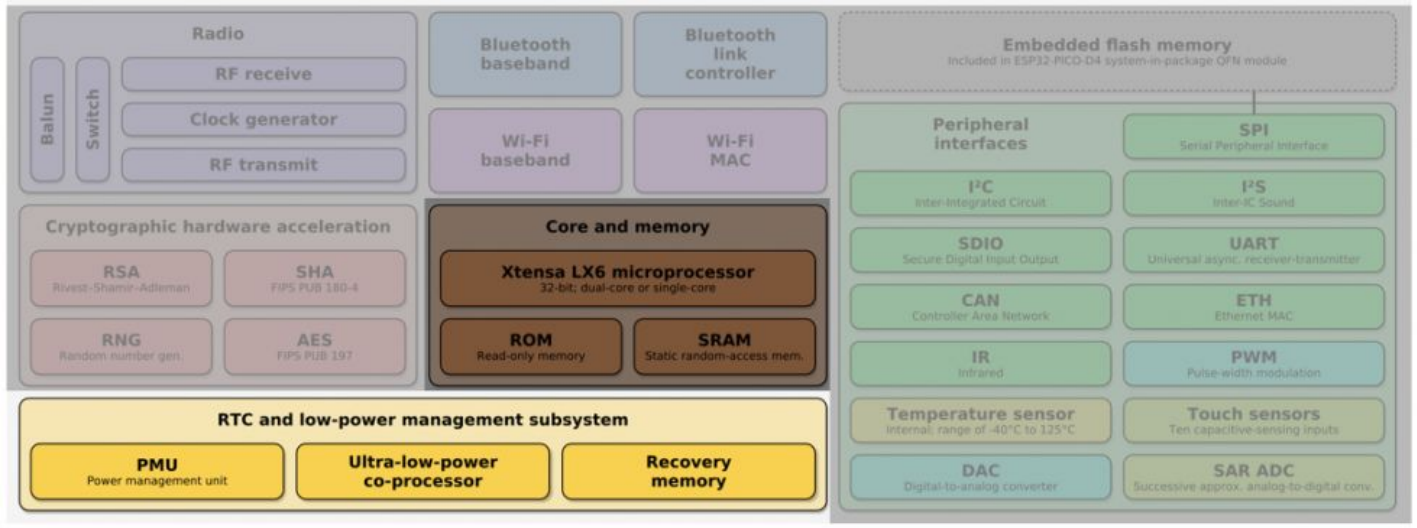
Power Modes

- Modem Sleep Mode - WiFi, Bluetooth and Radio are disabled, but are activated at predefined intervals to keep the connections.
  - Association sleep pattern - power mode switches between active mode and Modem sleep mode.
  - ESP32 remains connected to the router via the DTIM signaling mechanism (Beacon).
  - CPU is operational with a configurable clock.
  - 3 - 20 mA.



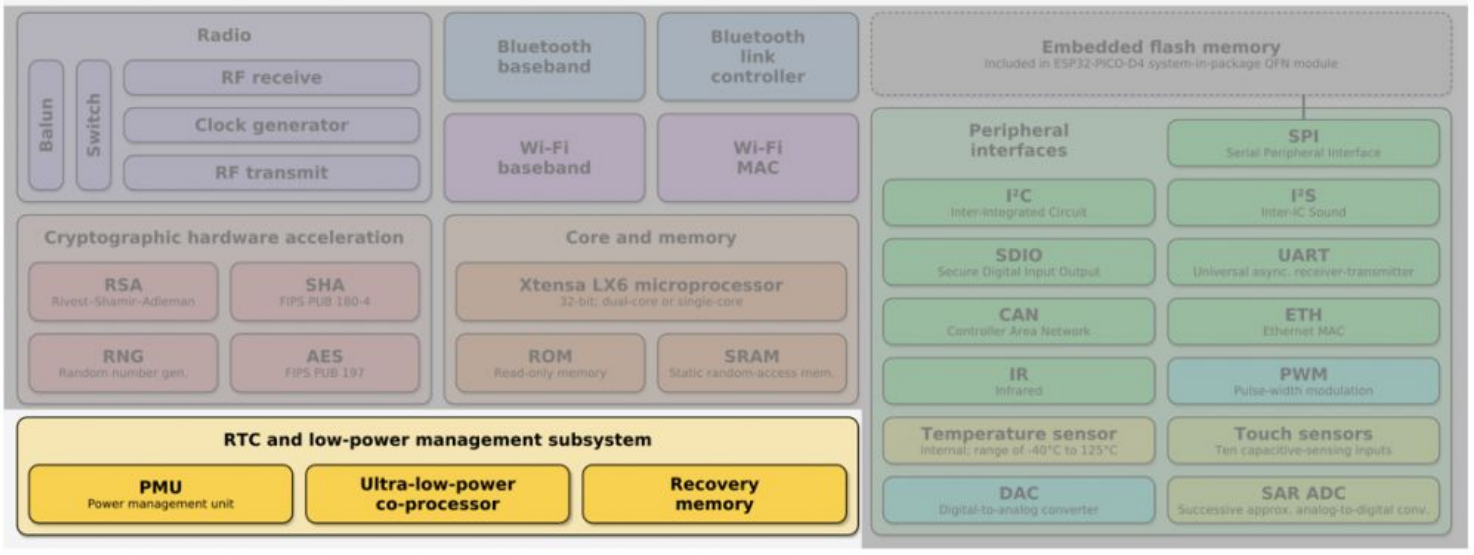
Power Modes

- Light Sleep Mode - CPU is paused by powering off its clock pulses, while RTC and ULP-coprocessor are kept active.
  - Before entering this mode, the ESP32 saves its internal state and resumes the saved state when exiting this mode.
  - 0.8 mA.



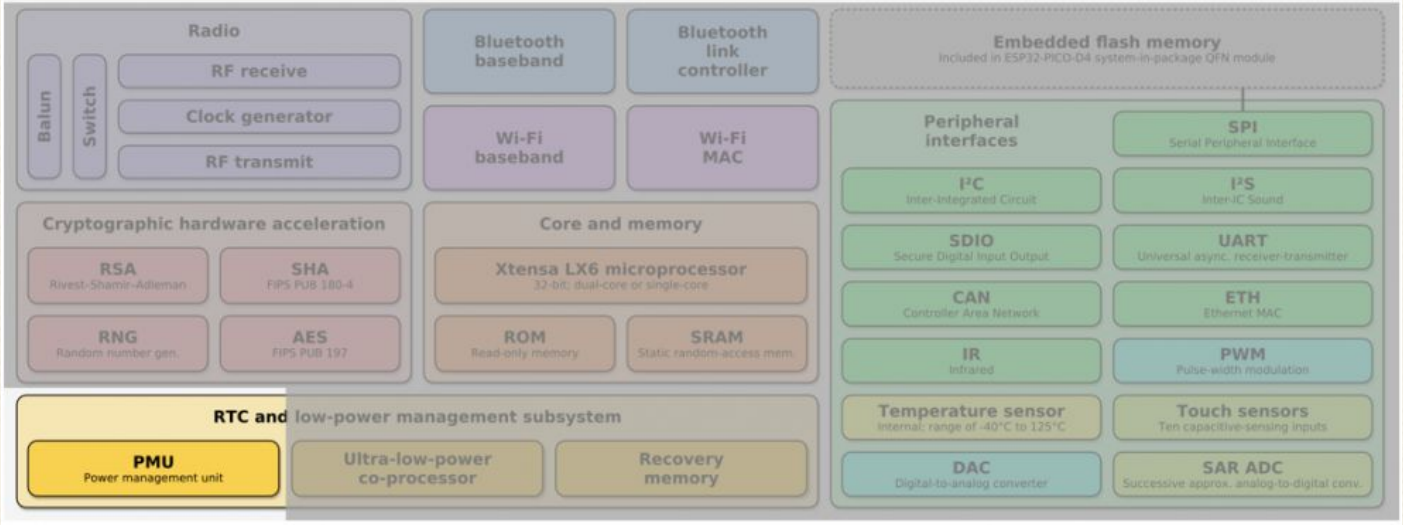
Power Modes

- Deep Sleep Mode - Only the RTC controller, the RTC peripherals, the ULP co-processor and RTC memories (slow and fast) remain on.
  - 0.15 mA or 0.01 mA (ULP co-processor not used)



Power Modes

- Hibernation Mode - Only one RTC timer on the slow clock and some RTC GPIOs are active (responsible for awakening the chip).
  - It is not possible to save data.
  - 0.0025 mA.



Power management features

Power management algorithm included in ESP-IDF can adjust the advanced peripheral bus (APB), CPU frequency and put the chip into light sleep mode. Application components can express creating and acquiring blocks to manage energy.

Examples:

- Driver for a peripheral clocked from APB can request the APB frequency to be set to 80 MHz while the peripheral is used.
- RTOS (Real Time Operating System) can request the CPU to run at the highest configured frequency while there are tasks ready to run.
- A peripheral driver may need interrupts to be enabled, which means it will have to request disabling light sleep.

Use of higher frequencies by APB or CPU and disabling light sleep causes higher power consumption.

Configuration

- CONFIG_PM_ENABLE - Activates Power Management at compile time.
  - increased latency → 0.2 us to 40 us (depends CPU frequency and single/dual core mode)
- Function esp_pm_configure() - Activates the Dynamic Frequency Scale (DFS) and automatic light sleep mode. With arguments:
  - max_freq_mhz : Maximum CPU frequency in MHz
  - min_freq_mhz : Minimum CPU frequency in MHz
  - light_sleep_enable : Signal indicating whether or not the system should go into light sleep automatically.
- CONFIG_PM_DFS_INIT_AUTO - The maximum CPU frequency will be determined by the CONFIG_ESP32_DEFAULT_CPU_FREQ_MHZ setting. The CPU frequency will be locked to the XTAL frequency (40MHz by default).

Power Management Locks

Applications have the ability to acquire/release locks in order to control the power management algorithm.

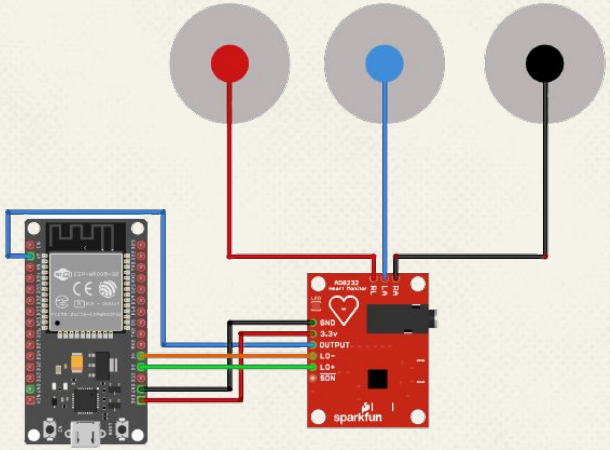
When an application acquires a lock, the power management algorithm operation is restricted.

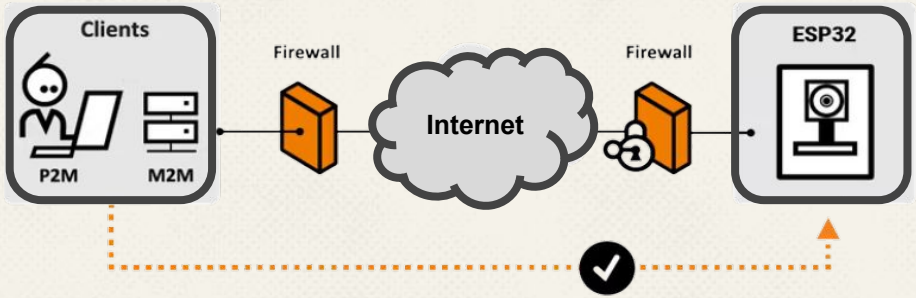
When the lock is released, such restrictions are removed.

Lock	Description
`ESP_PM_CPU_FREQ_MAX`	Requests CPU frequency to be at the maximum value set with `esp_pm_configure()`. For ESP32, this value can be set to 80 MHz, 160 MHz, or 240 MHz.
`ESP_PM_APB_FREQ_MAX`	Requests the APB frequency to be at the maximum supported value. For ESP32, this is 80 MHz.
`ESP_PM_NO_LIGHT_SLEEP`	Disables automatic switching to light sleep.

Applications

- Networking
- Data Processing
- P2P Connectivity
- Web Server
- Smart industrial devices
- Smart medical devices
- Smart energy devices
- Smart security devices





Remote Access and Upgrades

Remote Access

- Use ESP32 as a server
- Upload code to ESP32 wirelessly
- ESP32 Over-the-air (OTA) Programming

OTA - Over The Air

OTA programming is useful to update code to ESP32 boards that are not easily accessible.

A single central node can send an update to multiple ESPs on the same network.

Basic OTA: the program is updated into ESP32 over the air using Arduino IDE.

OTA web updater: the program is updated over the air using a web browser.

OTA Updates

OTA Process

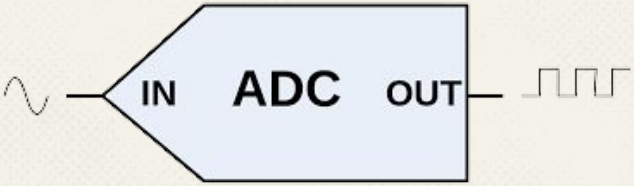
- OTA update mechanism allows a device to update itself based on data received while the normal firmware is running.
- OTA data partition should contain no data. ESP-IDF software bootloader will boot the factory app if it is present in the partition table.
- If no factory app is included in the partition table, the first available OTA slot (*ota_0*) is booted.
- After the first OTA update, the OTA data partition is updated to specify which OTA app slot partition should be booted next.

App Rollback

- Keep the device working after the update.
- Roll back to the previous working application in case a new application has critical errors.

Partition Table	
OTA App Slot	*ota_0*
OTA App Slot	*ota_1*
OTA Data Partition	**type** *data*, **subtype** *ota*

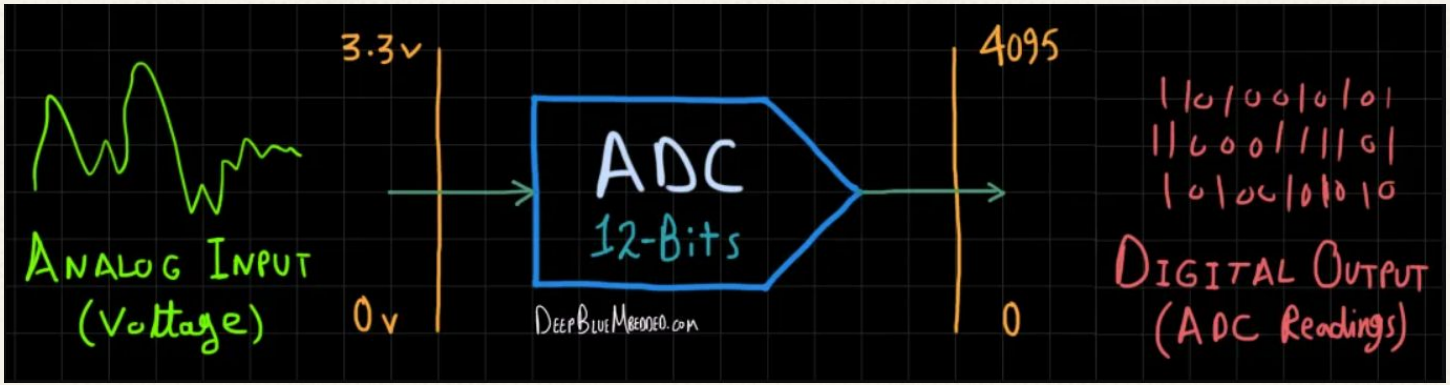
ADC demonstration



ESP32 ADC – Read Analog Values with Arduino IDE

Description

- Reading an analog value with the ESP32: measure voltage between 0 V and 3.3 V. The voltage measured is assigned to a value between 0 and 4095.



ADC Functions

There are nine function exposed by the ADC driver. They are:

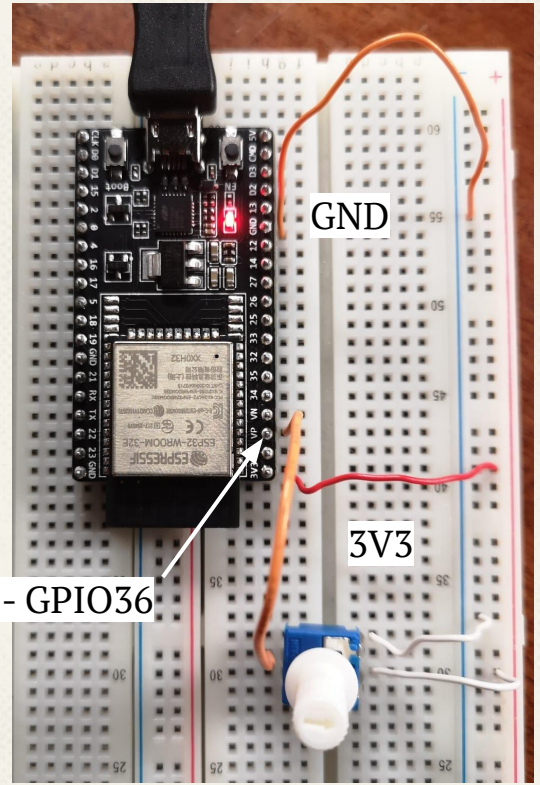
- analogRead(pin): Get the ADC Value for the specified pin.
- analogReadResolution(bits): Set the resolution of output of analogRead. Default is 12-bit but possible values are 9 to 12.
- analogSetWidth(bits): Sets the sample bits and read resolution.
- analogSetClockDiv(clockDiv): Set the divider for the ADC clock.
- analogSetAttenuation(attenuation): Set the attenuation for all channels. Default is 11db but possible values are 0db, 2.5db, 6db, and 11db.
- analogSetPinAttenuation(pin, attenuation): Set the attenuation for a particular pin.
- adcAttachPin(pin): Attach pin to ADC (done automatically in analogRead).
- analogSetVRefPin(pin): Set pin to use for ADC calibration if ESP32 is not already calibrated. Possible pins are 25, 26 or 27.
- analogReadMilliVolts(pin): Get millivolts value for pin.

ADC demonstration

Output

```
Raw: 1707       Voltage: 472mV
Raw: 1699       Voltage: 470mV
Raw: 1702       Voltage: 470mV
Raw: 1702       Voltage: 470mV
Raw: 3714       Voltage: 938mV
Raw: 4095       Voltage: 1026mV
Raw: 4095       Voltage: 1026mV
```

Circuit



Bibliography

- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/power_management.html
- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html
- https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/ota.html
- https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/
- https://www.makers.pt/2019/01/18/modos-de-sleep-do-esp32-e-seu-consumo-de-energia/
- https://deepbluembedded.com/esp32-adc-tutorial-read-analog-voltage-arduino/
- https://newscrewdriver.com/2021/02/22/evaluating-my-options-for-esp32-development/