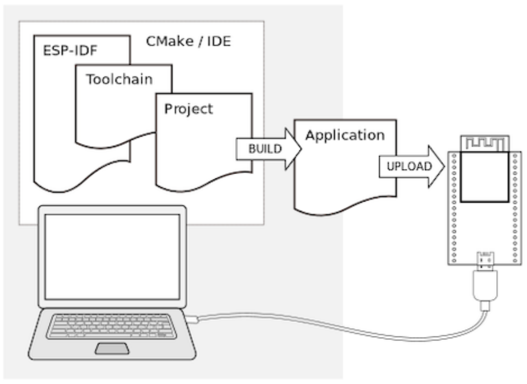
João Gameiro, 93097
Pedro Abreu, 93240

Arquiteturas para Sistemas Embutidos

Aula4

Development Framework

ESP-IDF (**Espressif IoT Development Framework)** is Espressif's official IoT Development Framework for the ESP32, ESP32-S and ESP32-C series of SoCs. It provides a self-sufficient SDK for any generic application development on these platforms, using programming languages such as C and C++. It powers millions of devices in the field, and enables building a variety of network-connected products, ranging from simple light bulbs and toys to big appliances and industrial devices.



ESP-IDF needs a set of cross-compiler tool-chains, tools to build the project (CMake and Ninja) and ESP-IDF itself that is responsible for operating with the toolchain. The installation can be made one of two ways: through an IDE or manually.

Manual Installation

Can be made in Linux, MacOS or in Windows. For Linux and MacOS it is necessary to install a set of prerequisites and then download the code from the Expressif Github repository and proceed to its installation. In Windows one also needs to install some prerequisites and then download an installer that will take care of the installation of ESP-IDF.

IDE Installations

It is possible to install ESP-IDF through an Eclipse (Java development IDE) plug-in that provides developers an environment to develop ESP32 based applications. It offers advanced editing, compiling, flashing and debugging features with the addition of Installing the tools, SDK configuration and CMake editors. In order to install the plug-in it is necessary to have the following prerequisites:

- Java >= 11
- Python >= 3.6
- Eclipse IDE for C/C++ and Git

If VSCode is the prefered option is also possible to install ESP-IDF through a VSCode extensions that supports:

- Building and flashing projects to ESP32
- Monitoring, debugging and tracing
- System trace viewer and core dumps, and other features.

VSCode extension already takes care of all the tools that need to be installed to run ESP-IDF.

It is also possible to use PlatformIO, which is a user-friendly and extensible IDE, with a set of professional development instruments, providing modern and powerful features to speed up yet simplify the creation and delivery of embedded products that can be integrated with VSCode or CLion. This cross-platform embedded development environment provides support for ESP-IDF.

Programming Languages

ESP32 has support for several programming languages in several platforms which include: C/C++, Python (MicroPython, CircuitPython), Lua (LuaRTOS, NodeMCU) and Javascript.

Power Management

ESP32 offers efficient and flexible power-management technology to achieve the best balance between power consumption, wakeup latency and available wakeup sources. Users can select out of five predefined power modes of the main processors to suit specific needs of the application. To save power in power-sensitive applications, control may be executed by the Ultra-Low-Power coprocessor, while the main processors are in Deep-sleep mode.

Active Mode

All the features of the chip are active. Active mode keeps everything ON at all times, so the chip requires more than 240mA current to operate. Most inefficient mode and will drain the most current. 160~260 mA.

Modem Sleep

Everything is active while only WiFi, Bluetooth and radio are disabled. The CPU is also operational and the clock is configurable. To keep WiFi/Bluetooth connections alive, the Wi-Fi, Bluetooth, and radio are woken up at predefined intervals. The Wi-Fi/Bluetooth baseband is clock-gated or powered down. Immediate wake-up. 3~20mA.

Light Sleep

Digital peripherals, most of the RAM and CPU are clock-gated. The CPU is paused by powering off its clock pulses, while RTC and ULP-coprocessor are kept active. 0.8mA. Before entering light sleep mode, ESP32 preserves internal state and resumes operation upon exit from the state. Less than 1 ms to wake up. 0.8mA.

Deep Sleep

The CPU is powered down, while the ULP co-processor does sensor measurements and wakes up the main system. The main memory of the chip is also disabled.RTC memory is kept powered on. Power is shut off to the entire chip except the RTC module. So, any data that is not in the RTC recovery memory is lost, and the chip will thus restart with a reset. Less than 1 ms to wake up. 10µA.

Hibernation Mode

The chip disables CPU and ULP-coprocessor. The RTC recovery memory is also powered down, meaning there's no way we can preserve any data during hibernation mode. Everything is shut off except only one RTC timer on the slow clock and some RTC GPIOs are active. They are responsible for waking up the chip from hibernation mode. Less than 1 ms to wake up. 1µA.

The System API also contains methods that can adjust advanced peripheral bus (APB) frequency or CPU frequency as well as managing the enter and leaving of the sleep modes. Application components can specify requirements by creating power management locks.

Remote Access

ESP Local Control (**esp_local_ctrl**) component in ESP-IDF provides capability to control an ESP device over Wi-Fi + HTTPS or BLE. There is the possibility of creating lightweight HTTP(S) servers that run on the board and can be configured to be accessible from the internet and add the necessary methods to control the kit. It is necessary the correct configuration of the server which not only includes the necessary methods for dealing with requests as well as the configuration of a domain to make the server accessible from the Internet.

esp_http_server and **esp_https_server** allow for the creating and configuration of the server.

Firmware Upgrades

esp_https_ota provides simplified APIs to perform firmware upgrades over HTTPS. It's an abstraction layer over existing **OTA** APIs.

esp_https_ota function allocates HTTPS OTA Firmware upgrade context, establishes HTTPS connection, reads image data from HTTP stream and writes it to **OTA** partition and finishes HTTPS **OTA** Firmware upgrade operation.

This API is capable of dealing with the entire OTA upgrade process however if more control information regarding the operation is necessary there available other APIs to be called in succession to accomplish the upgrade

It is also possible to perform OTA upgrades with pre-encrypted firmware by enabling an option to decrypt the data received before being processed.

SPI - Master Driver

SPI Master Driver is a program that controls ESP32s SPI peripherals while they function as masters. The SPI master driver governs communications of Hosts with Devices. An SPI Transactions include

- **Command** -> In this phase, a command (0-16 bit) is written to the bus by the Host.
- **Address** -> an address (0-64 bit) is transmitted over the bus by the Host.
- **Write** -> Host sends data to a Device.
- **Dummy** -> Configurable phase used for timing and synchronization purposes
- **Read** -> Device sends data to Host

The attributes of a transaction are determined by the bus configuration structure, device configuration structure and transaction configuration structure.

An SPI Host can send full-duplex transactions, during which the read and write phases occur simultaneously. In half-duplex transactions, the read and write phases are not simultaneous (one direction at a time).

The data that needs to be transferred to or from a Device will be read from or written to a chunk of memory indicated by the members **rx_buffer** and **tx_buffer** of the structure **spi_transaction_t**. In terms of interruptions, there are two types in SPI:

- **Interrupt Transactions**
  - Interrupt transactions will block the transaction routine until the transaction completes, thus allowing the CPU to run other tasks.
- **Polling Transactions**
  - The routine keeps polling the SPI Host's status bit until the transaction is finished. Smaller transaction duration, but CPU is busy pooling the Host.

In terms of data structures that allow the configuration of the transaction and of the intervinientes, the following are defined:

- **spi_transaction_t -** describes one SPI transaction
- **spi_device_interface_config_t -** configuration for a SPI slave that is connected to one of the buses
- **spi_transaction_ext_t -** for SPI transactions which may change their address and command length.
- **spi_bus_config_t -** configuration structure for a SPI bus

SPI - Slave Driver

SPI Slave driver is a program that controls ESP32's SPI peripherals while they function as slaves. ESP32 integrates two general purpose SPI controllers which can be used as slave nodes (SPI2 and SPI3). Initialization of an SPI peripheral as slave is done by making a call to **spi_slave_initialize** with it's specific configurations.

In terms of data structures that allow the configuration of the transaction and of the slave the following are defined:

- **spi_slave_interface_config_t -** Configuration for a SPI host acting as a slave device.
- **spi_slave_transaction_t -** describes one SPI transaction

Applications

Some examples of applications where ESP32 is used include:

- Generic Low-power IoT Sensor Hub and IoT Data Loggers
- Home Automation
  - Light Control, Smart Plugs, Smart Door Locks
- Industrial Automation
  - Industrial wireless control and industrial robotics
- Smart agriculture
  - Smart greenhouses, smart irrigation, agriculture robotics
- Health Care applications
  - Health monitoring, baby monitors
- Audio Applications
  - Internet music players, Live streaming devices, Audio headsets
- Wearable Electronics
  - Smart watches, Smart bracelets

Bibliography

- ESP-IDF Programming Guide
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html
- Insight Into ESP32 Sleep Modes & Their Power Consumption - Last Minute Engineers
  - https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/
- Programming Languages for ESP32 and ESP8266
  - https://www.electronicdiys.com/2018/11/programming-languages-for-esp32-esp8266.html
- ESP32 - Technical Reference Manual
  - https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
- ESP32 Series - Datasheet
  - https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
- ESP32WROOM32E WROOM 32 EU - Datasheet
  - https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf
- Espressif on Github
  - https://github.com/espressif