Documented Tasks

// SHOULD I EXPLAIN WHAT SYNCHRONOUS AND ASSYNCHRONOUS COMMUNICATION IS?

// SHOULD I EXPLAIN WHAT A MASTER AND SLAVE CONFIG IS? FEEL LIKE ITS UNECESSARY

**Development Environment for ESP32 and read manual**

I’m currently reading the manual as I go and complete tasks on my project.

**Test ESP32 W/ some simple code**

* Imported hello\_world from Expressif examples and learnt how to flash micro;
* Imported blink project from Expressif examples – found where LED is in the pins;
* Created a project that measures the internal temperature of the micro – code from YouTube;
* Created a project that takes measures from Hall effect sensors of the micro – code from YouTube;
* Created a project that simulates a traffic light with 3 leds: green, yellow and red.
* Created a standard project and made 2 tasks turn on and off the LED22;

**Search how to make libraries in ESP-IDF and search about Project Layering**

Given that the build manager toolchain is Cmake, the way the layering is done in Expressif IDE is using components you create/call from their API/import from their IDF Component Repository. Components, as far as I’ve understood, are meant to be thought as libraries but with additional features – pretty much libs with additional Cmake wrapper commands. As such, after investigating multiple components from their repo and comparing the file and folder structure of each one (they were varied), decided on layering my project with a look I’m more familiar with – something that came from SE subject project structuring.

**How to set up an IDE - for annex**

Even for a setup, it is easy to start with the IDE. I did it myself while looking at this [link](https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32/get-started/windows-setup.html#get-started-windows-first-steps) (ESP-IDF Programming Guide) and at the IDF Documentation (after installing the IDE it is presented as an option to open this documentation. Initially, I copied and edited code that the IDE comes with (examples folder in the IDE installation directory) though their IDE, using their ESP-IDF CMD (tool that lets you execute build/config/monitor-related commands through a terminal interface) tool to set the target (esp32), build the project files, flash the micro and monitor its serial console. These are simple one-line commands that are on the programming guide link above.

After that, I explored the basic parts of the IDE interface until I could run the code on the micro through the IDE: setting the target through their interface, building and cleaning projects, and running code. **Debugging** is difficult here, so I researched the 3 debugging methods available to me as a user of the IDE: GDBStub Debugging, Core Dump Debugging (both are post-mortem) and OpenOCD Debugging. Since the LOLIN32 Lite also does not support JTAG debugging over USB, OpenOCD Debugging is out of the question as [I would need an external adaptor to be connected to certain pins in the board](https://www.espboards.dev/esp32/lolin32-lite/) and I don’t have one. As a result, I decided to stick with the basic GDBStub Debugging, that is pretty much the monitor command I’ve mentioned above.

**Check if I can change CPU frequency**

I found how to change the CPU frequency in a project and that is to access via ESP-IDF CMD terminal to its menuconfig (idf.py menuconfig) and change a setting in the sdkconfig file (Component config -> ESP System Settings - > CPU Frequency); or to access the file itself via project explorer on the project I’m on, and then find the same choices In the menu the other method had to change the settings of the file. I can change the CPU frequency to one of three values: 80, 160 or 240 MHz, the default being 160MHz.

**Think about architecture (queue with raw values - filtering - server task with filtered values)**

Following what the Embedded Systems has been teaching in class and using some references online of Expressif slideshow presentation on YouTube, I slowly came up with something that might be, in my opinion, viable as the architecture of the project.

The project consists of a main task that creates the other tasks and starts the scheduler.

**Study SPI**

SPI (Serial Peripheral Interface) is a protocol widely used between microcontrollers and peripherals such as sensors, ADCs e DACs. SPI communication is a synchronous type of communication protocol. Using a synchronous type of communication protocol means that all the devices that are communicating with each other have the same clock signal.

Usually, people use this communication protocol as it is very fast and supports very high communication speeds and high CPU clock frequencies as well. SPI communication is a full duplex communication protocol, having a master and slave configuration, something that I’m already familiar with since I learned about the I2C communication protocol in the Embedded Systems subject. In this configuration, two devices (using a slave and a master as an example, although there can be multiple slaves connected to one master) can transmit and receive data at the same time: SPI communication doesn’t rely only on one data bus.

At the master side, there’s the MOSI pin (Master Out Slave In), the MISO pin ()

**Study UART**

**NMEA frames check**

**Map communication protocol in micro: make document about protocols and read about them in ESP**

**Compare 3 different modules to explain the protocols**

**Tabela comparativa com os parâmetros que me interessam para decidir no melhor**

**Make communication protocol code (to do)**

**Make display font**