Robot - TP3 - Embarqué - ADC

Setup

To be able to flash a ESP32, you should not use a VM or WSL.

- For Windows only:
  - from https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads download CP210x_Universal_Windows_Driver.zip
  - Unzip this file, right clic on `silabser.inf` then `Install`
  - Do not forget to follow `epitaGitlabSshKey.pdf` and execute from a git console:
  ```
  git clone git@gitlab.cri.epita.fr:jeremie.graulle/esp-idf-cxx.git
  ```
- For Personal Linux (not Epita computer):
  - VS code installation: (add PPA and install from https://doc.ubuntu-fr.org/visual_studio_code#via_ppa_de_microsoft or manual download from https://code.visualstudio.com/download and install)
  - Before connect the ESP32 to the computer, run:
  ```
  echo -e "# CP210X USB UART\nATTRS{idVendor}==\"10c4\", ATTRS{idProduct}==\"ea60\", " \
    "MODE:=\"0666\", ENV{ID_MM_DEVICE_IGNORE}=\"1\", ENV{ID_MM_PORT_IGNORE}=\"1\"" \
    | sudo tee /etc/udev/rules.d/97-cp210x.rules
  ```
  - Connect the ESP32 to the computer, then run `lsusb`, you should see a line contains: `ID 10c4:ea60 Silicon Labs CP210x UART Bridge`
  - Then run: `ls -l /dev/ttyUSB*` you should have `crw-rw-rw-` at the begin of the line, if not reboot your computer.
- Run Visual Studio Code
- In menu `View` option `Extensions`, search and install the extension `ESP-IDF` from `Espressif Systems`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: Configure ESP-IDF extension`.
- Choose `Express`
- Leave `Github` in `Select download server`
- Select `v5.4.1 (release version)` in `Select ESP-IDF version:`
- Leave `/home/<user>/esp` in `Enter ESP-IDF container directory`
- Leave `/home/<user>/.espressif` in `Enter ESP-IDF Tools directory (IDF_TOOLS_PATH)`
- Leave `/usr/bin/python3` in `Select Python version:`
- Clic on `install`
- You should get `ESP-IDF has been configured` popup
- Search and install the extension `C/C++ Extension Pack`

If the installation failed from VS code you can try to setup it in console mode using the document from https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/linux-macos-setup.html

If you prefered you can execute all command from console simply run `. $HOME/esp/v5.4.1/esp-idf/export.sh` (do not forget the dot) then you can run:

- `idf.py menuconfig`
- `idf.py build`
- `idf.py flash`
- `idf.py monitor` (To exit IDF monitor use the shortcut `Ctrl+]`)

1. First C Project example (optionnal)

Purpose: To be able to import an ESP-IDF example, build, flash and see debug message.

Step 1.1

- In menu `View` option `Command Palette`, type and select `ESP-IDF: Show Examples Projects`.
- Select `Use ESP-IDF /home/<user>/esp/v5.4.1/esp-idf ESP-IDF v5.4.1`
- Select `esp-idf/get-started/blink`
- Clic on `Select location for creating blink project` then select an empty folder on your disk

- VS code should change the working directory (the top display directory) to `blink`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: SDK Configuration editor (Menuconfig)` (same as the bottom shortcut `SDK Configuration Editor (menuconfig)`)
- In `Serial flasher config/Flash size` select `4MB`
- In `Example Configuration/Blink GPIO number` type `12`
- Clic `Save`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: Build your project` (same as the bottom shortcut `Buid Project`)
- In menu `View` option `Command Palette`, type and select `ESP-IDF: Select port to use` select `/dev/ttyUSB0` (same as the bottom shortcut `Select Port to Use (COM, tty, usbserial)`)
- In menu `View` option `Command Palette`, type and select `ESP-IDF: Flash your project` then `UART` (same as the bottom shortcut `Flash Device`)
- In menu `View` option `Command Palette`, type and select `ESP-IDF: Monitor your device` (same as the bottom shortcut `Monitor Device`)
- Use Ctrl+T then x to exit from monitor (usefull when reboot in loop)

Step 1.2

- Update the period by menuconfig
- Open the main.c and updated period directly in C source

2. First C++ Project example (optionnal)

Purpose: To be able to import an ESP-IDF module and use C++ instead of C. Definition: a ESP-IDF module is an external lib can be easilly add to a ESP-IDF project.

Step 2.1

- From previous VS code project
- In menu `View` option `Command Palette`, type and select `ESP-IDF: open ESP-IDF Terminal`, then execute `idf.py add-dependency espressif/esp-idf-cxx^1.0.0-beta`
- In `Explorer`
  - open `main/idf_component.yml`, then remove `espressif/led_strip: ^2.4.1`
  - rename `main/blink_example_main.c` into `main/main.cpp` and replace contains by `https://github.com/espressif/esp-idf-cxx/blob/main/examples/blink_cxx/main/main.cpp`
  - check in `main\CMakeLists.txt`, `blink_example_main.c` have been replace by `main.cpp`
- In `Explorer` open `main/main.cpp` and update `GPIONum(4)` with `GPIONum(12)`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: SDK Configuration editor`
- In `Compiler options/Enable C++ exceptions` check `Enable C++ exceptions` and `Enable C++ run-time type info (RTTI)`
- Clic `Save`
- Do Build, Flash and Monitor like step 1
- To have the code navigation and autocompletion, in menu `View` option `Command Palette`, type and select `ESP-IDF: Add vscode configuration folder`
- Do Flash and Monitor like previous project

Step 2.2

- Open the main.cpp and updated period directly in C++ source

3. First new C++ Project

Purpose: To be able to create a new project and import a custom ESP-IDF module dedicated for this robot.

Step 3.1

- In menu `View` option `Command Palette`, type and select `ESP-IDF: new Project`
- In `Project Name` type `robot-esp-idf`
- In `Enter Project directory` select your workspace (this is a new project create next to `blink` project, you should select the parent folder of `blink` folder, do not select `blink` folder)
- In `Choose ESP-IDF Board` select `ESP32 chip (via ESP-PROG)`
- In `Choose serial port` select `/dev/ttyUSB0`
- Click on `Choose Template`
- Select `template-app`
- Click on `Create project using template template-app`
- On popup `Project robot-esp-idf has been created. Open project in a new window?` Clic `Yes`
- Now in Explorer you should see a top level `robot-esp-idf`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: open ESP-IDF Terminal`, then execute:

```
idf.py add-dependency \
  --git git@gitlab.cri.epita.fr:jeremie.graulle/esp-idf-cxx.git \
  jgraulle/esp-idf-cxx
```

- Rename `main/main.c` into `main/main.cpp`
- From `https://github.com/espressif/esp-idf-cxx/blob/main/examples/blink_cxx/main/main.cpp` copy contains into file `main/main.cpp`
- Check in `main\CMakeLists.txt`, `main.c` have been replace by `main.cpp`
- In menu `View` option `Command Palette`, type and select `ESP-IDF: SDK Configuration editor (Menuconfig)` (same as the bottom shortcut `SDK Configuration Editor (menuconfig)`)
- In `Serial flasher config/Flash size` select `4MB`
- In `Compiler options/Enable C++ exceptions` check `Enable C++ exceptions` and `Enable C++ run-time type info (RTTI)`
- Clic `Save`
- Do Build, Flash and Monitor like step 1
- To have the code navigation and autocompletion, in menu `View` option `Command Palette`, type and select `ESP-IDF: Add vscode configuration folder`

Step 3.2

- Open the main.cpp and updated period directly in C++ source
- Create new personnal private project `ssie-robot-embedded` on the Epita gitlab.
- Add your two-person team and `jeremie.graulle` as `maintainer`
- Clone or add remote this project in local
- Do not commit generated files. Create a .gitignore file and add:
  - build/
  - sdkconfig
  - sdkconfig.old
  - .*
  - managed_components
  - dependencies.lock
- Commit all sources files of this project in main branch.

4. Add new hardware component for ADC

Step 4.1

- Download robot-tp3-embedded-adc-header.tar.gz and extract into the project created in previous step `First new C++ Project` next to `main.cpp`.
- Create a new C++ body C++ file adc_cxx.cpp and add stub (empty body function) for each header function to make it compile again (do not forget to add body in CMakeLists.txt)

Step 4.2

- Create a `tp3` branche to commit this step
- From official SDK help: https://docs.espressif.com/projects/esp-idf
- Select the stable version (v5.4)
- Goto `API Reference` and look for ADC
- Update file adc_cxx.cpp and adc_cxx.hpp to have reel implementation. You can have a look in other hardware C++ class in `jgraulle/esp-idf-cxx:` (include/gptimer_cxx.hpp, include/pulse_counter_cxx.hpp) to have the same behavior (use CHECK_THROW to convert return code to exception).
- Update main to add a new thread, in this thread you will read battery voltage every seconds and print it into serial console (use ADC on GPIO 36).

Step 4.3

In addition to logging the voltage, a brief beep (100ms) should be made using the buzzer if the voltage is below 7V (you have an example of a beep in the folder `managed_components/esp-idf-cxx/examples/ledc_cxx/main/main.cpp`).

Use a relatively low-pitched sound to limit disturbances during the lab. If the board is powered via USB-C, you should not activate the buzzer. To validate this step, you will need to call me so that I can come and test your algorithm by simulating a discharged battery.

I will ask you to always include this code in all your projects to protect batteries from deep discharges.

Step 4.4

- Move the battery management to a dedicated thread by adding a `while(true)` in the main loop.
- Create a merge request from `tp3` branch to main.
- Add jeremie.graulle as reviewer of this MR