# Zephyr RTOS

## Setup

### Prerequisites

#### Install

* Go to `C:\` to install zephyr in the directory `zephyrproject`
* Follow the [Getting Started Guide](https://docs.zephyrproject.org/latest/develop/getting_started/index.html) to install the Zephyr source code.
* Install `openocd` if needed or `[jlink](https://www.segger.com/downloads/jlink)` depending on the runner needed by the board:

> choco install openocd

⚠️ For Jlink don’t forget to install the USB driver, otherwise it will not work.

#### Update

`west update`, use to keep your Zephyr repositories up to date, if zephyr is already installed.

Creating a Zephyr Application

* Create a project folder with the following structure:

<app>

├── CMakeLists.txt

├── prj.conf

├── src

| └── main.c

└── boards

└── <board>.overlay or custom\_board.dts

* Set the ZEPHYR\_BASE environment variable to the Zephyr source code path:

set ZEPHYR\_BASE=C:\zephyrproject\zephyr

* Active Python virtual environment.

C:\zephyrproject\.venv\Scripts\activate.bat

**OR**

* Alternatively, use the `env.cmd` script to set up the environment automatically. Make sure to update the script with the correct Zephyr source path.

⚠️ Run the above command every time you open a new terminal.

## Build and Flash the Project:

* `west build` to compile the project, ex: `west build -p always -b samd21\_xpro app`
  + `-b <board>` ex: `-b samd21\_xpro`: Specifies the target board.
  + `-p` pristine: (optional)
    - `auto`
    - `always` => default option
    - `never`
  + Path to the <app> to build, if not specified use `.`
  + `-- -DDTC\_OVERLAY\_FILE=board.overlay` (optional): Specifies the path to an overlay file for the devicetree.
  + `-- -DBOARD\_ROOT=<path>` (optional): Specifies the path to the `boards` directory where the custom board is located. Need to be an absolute path.
* `west flash` to flash the compile project to the board
  + `--build-dir <path>` or `-d` (optional): Specifies the directory to build the project
  + `--runners jlink` or `-r` (optional): Specifies the runners if supported by the board

## Other West command

* `west build -t menuconfig` or `west build -t guiconfig` to see or change Kconfig variables.
* `west config build.board <board>` sets permanently the default board if `-b` not specified
* `west debug` => attach debugger to the board, open `gdb`
  + `--runner openocd` or `-r`

## Vscode setting and extensions

### Extensions

* `ms-vscode.cpptools-extension-pack` => C/C++
* `ms-vscode.cmake-tools` => CMake
* `ms-vscode.vscode-serial-monitor` => Serial monitor
* `marus25.cortex-debug` => Embedded Debugger
* `nordic-semiconductor.nrf-devicetree` => Devicetree

### Configuration

#### Settings.json

* `"C\_Cpp.default.compileCommands": <path>` after building the project add the path for the `compile\_commands.json` in the build directory.

#### Tasks.json

Create tasks to build and flash the code on the board, in the tasks specify the env variable `ZEPHYR\_BASE` to the zephyr code source, otherwise you will have a build error.

You can add input parameters if you want to dynamic parameter to the command run.

#### Launch.json

Create 2 launch configurations, one for launch and one for attach.

In each launch configuration add the following parameters:

* `name` => Name of the configuration
* `svdPath` => Path to svd file or ex: "[Microchip::SAMD21\_DFP@3.6.144](mailto:Microchip::SAMD21_DFP@3.6.144)"
* `deviceName` => Name of the board, need depending on the svd file, this setting is not recognise by Vscode but works.
* `device` => Name of the board
* `cwd`
* `executable` => Path to executable file .elf
* `request` => launch or attach
* `type` => cortex-debug
* `runToEntryPoint` => main function
* `servertype` => openocd, jlink ... or external to connect to an already existing session (with openocdm you want to connect to the debug session launch by `west flash`
* `gdbTarget` => Use only with serverType = external, define address to find the debugger server, ex: `:3333`
* `gdbPath` => path to gdb in zephyr sdk directory
* `preLaunchTask` => only for launch, command to run before debug, to build the project
* `configFiles` => for openocd, configuration file provides by zephyr

## Debug the Project

### Cortex Debug

Install the Vscode extension Cortex-Debug, then you need to create a `[launch.json](#_Launch.json)` file with 2 launch configuration. One to launch a debug session and one to attach a debug session. The launch session will build the project and flash it on the board then connects a debug session. The attach session will directly connect to the board executing code and open a debug session.

In the launch request you need to specify a `preLaunchTask` to build the project. So, you need to add a `[tasks.json](#_Tasks.json)` file with a task to build the project.

Go to the onglet `Run and debug` in Vscode, then chose the configuration you want to use, and start debugging.

To access the register, you need to specify a `svdPath` in the `[launch.json](#_Launch.json)`, and if needed a `deviceName`.

### West debug

Run the command `[west debug](#_Other_West_command)` in the terminal. It will open a `gdb` session.

Gdb Commands:

* `Ctlr + x`, then `2` => open the graphical interface
* `list` => with the graphical interface convert asm in C code
* `b <function>` => create a break point on a function
* `b <file:line>` => create a break point
* `b <address>` => create a break point
* `cont` or `c` => continue to run until the next break point or the end
* `next` or `n` => step over
* `step` or `s` => step in
* `backtrace` or `bt` => print callstack
* `print <var>` or `p <var>` => print variable, register…
* `info registers` or `i r` => print register value

## Devicetree

Use to define the hardware, all the peripheric.

### Overlay

File with the extension ` .overlay` are used to add peripheric or overwrite the devicetree of the original board.

Default location for devicetree when building the project:

1. socs/<SOC>\_<BOARD\_QUALIFIERS>.overlay
2. boards/<BOARD>.overlay
3. boards/<BOARD>\_<revision>.overlay
4. <BOARD>.overlay
5. app.overlay

Or use `-- -DDTC\_OVERLAY\_FILE=board.overlay` to explicitly define the path to the board in the `west build` command. It takes the priority over the default location define above.

You can add new node(device) to the devicetree or modify existing node.

To add node:

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

To modify existing node (label):

Une image contenant Police, texte, capture d’écran, Graphique

Le contenu généré par l’IA peut être incorrect.

Depending on the `compatible` key each node will need different parameters, that can be find on the doc. It can be specific to the board used.

Led => `compatible = "gpio-keys"`

Button => `compatible = "gpio-keys"`

After building the project the complete devicetree use will be in the file `<build directory>/zephyr/zephyr.dts`

### Custom Board

In your app directory `boards/<vendor>/<board\_name>/` that will contain all the file describing the board. If no vendor name it `other`.

## Kconfig

Use to enable or disable some features. The file `prj.conf` will contains the Kconfig at the app root.

* To enable Gpio: `CONFIG\_GPIO=y`
* To enable Pwm: `CONFIG\_PWM=y`
* …

## CMake

The `CMakeLists.txt` is located at the root of the app.

The default file contains:

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

Before pulling the project zephyr, you van define any variable you want.