Zephyr development

When developing embedded applications using the Zephyr ecosystem, some setup is needed. Zephyr is less beginner friendly because of its many dependencies between files and its lack of follow-along tutorials for setup. Since Zephyr is relatively young and open-source, there have been a lot of revisions in its systems over the years and examples from a previous major SDK version are often not good to reference anymore.

Setup

Setting up an environment for Zephyr consists of installing the Zephyr SDK and setting up a workspace to develop your project. When you want to contribute to the Zephyr Project, you will be doing in-tree development. To have your contributions added to the Zephyr Project, your code needs to be licensed and documented, as described in the Contribution Guidelines.

When you don't want to contribute to the Zephyr Project, it's recommended to use an out-of-tree workspace for development. This option is best for when you want a compact and modular project repository. For this project, the project repository is accompanied by a README with a setup tutorial and a short explanation to get started with an out-of-tree workspace.

Describe hardware

The hardware descriptions in Zephyr are in device tree format and these are used together with device tree bindings in yaml format to create a uniform Hardware Abstraction Layer.

Write applications

Writing applications in Zephyr is comparable to other C/C++ projects. the code is found under */src* with main.c as the main application and libraries and different code modules that get included in the code and get compiled in by CMake following the CMakeLists.txt

Modules

External modules like Zephyr subsystems, libraries or device drivers can be implemented, enabled and configured using KConfig. These configurations are made during compilation by specifying different configuration variables. KConfig Search is helpful for existing modules to check specific variables and their parameters. When you want to add your own dependencies, you can make Kconfig files describing the configuration variables. Depending on how your application is written and configured, it's possible to compile applications for many different boards. The Button example can compile for your board if you have buttons defined with matching aliases in your device tree and there are optional LEDs that will work if they are specified as well.

Implement drivers

Drivers in Zephyr are split into two parts: The hardware description and the driver software implementation. The LED controller driver is a good example driver: a generic api for using RG-leds with a specific implementation for the SC620 ledcontroller that uses red and green LEDs to form RG-leds.

Hardware description

A driver uses device tree bindings to specify its dependencies. When a driver is used in an application and the required dependencies are not present in the device tree as specified, the application will not compile.

Driver software

Driver software should be a generic api with a specific implementation of that api for each hardware target.