#cec32xNotes, cec32x_devtool_20_organization_of_proj_files

Organization of project files

There are many different approaches to organize the projects files. Here we prescribe a certain approach to simplify the delivery of the program files to students to the class.

After we install the development tools, the files for each tool as well as the precompiled libraries (in `.lib` or `.a` format) are installed for us automatically. We don't have to worry about these files. We can compile some of our library files into `.lib` or `.a` format as well, but we will not spend our time to discuss this.

What we will discuss is the organization of the files that are hosted on our file system which are under our direct control.

Before moving on for the details of the file structure, we take a look at the groups of files for the embedded C and assembly projects.

Three groups of files for embedded C and assembly projects

There are three groups of files for embedded C and assembly projects:

- Source files and project management files. Source files are user programmed C and assembly language files which prescribe the logic of our project, and the management files are those files help us manage the source files and various library files so that we can build the project using the dev tools.
- Library files in source format. Library files can be provided in two formats, the the `.lib` or `.a` format and the source format. The advantages for the former is that we don't need to spend to recompile the library files; they are linked directly after the compilation of the source files in the project. However, the latter has a unique advantage—we can browse the source code to know the details of the implementation of the library. Proprietary library usually comes in the first format, and the open-source library usually comes in the second format. We will organize the files in the second format into our file organization.
- Build files. These are the files created by the dev tools. There are two types of build files:
    - Object files for each source file, including the source level library file.
    - Linked files, including the final file for simulation or programming the HW kit.

High level file structure

Since Keil MDK-ARM only works on Windows, we will put all these files under the C drive to facilitate the delivery of files between the members of the class. Specifically, we put all our files under `C:\pjct_arm`.

Within this folder, we have the following subfolders:

- `_lib`: All source level libraries
    - `cec32xlib`: Library files developed by members of the class
    - `STM32Cube_FW_F4_V1.25.2`: Firmware libraries for various STM32F412 boards
    - `STM32Cube_FW_L4_V1.16.0`: Firmware libraries for various STM32L476 boards
    - `unity` - Unity **unit testing** framework
- `build`: Common build directory
    - `F412Build`: Build directory for the STM32F412 projects
    - `L476Build`: Build directory for the STM32L476 projects
    - `sim`: Build directory for the simulator-based projects
- `projects`: Top level directory for CEC32x projects (e.g. labs, workshops, etc.)
    - `examples` - Project templates, in-class examples, etc.
    - `labs` - Directory for all lab projects (including final)
    - `workshops` - Directory for all workshop projects

We need to follow this high level file structure in a flexible way. **This means we can put the `projects` folder somewhere else. If you use Git to do version control, it may be a good idea to put it in `C` as indicated above. If not, you are suggested to have the entire `projects` folder moved to your could drive, such as OneDrive or GoogleDrive, so that you can recover your source code when there is anything wrong with your computer.**

As for the other files, we need to use the above high level file structure. If you wanted to know the reason, here it comes.

The main reason we put the `_lib` folder directly under `C:\pjct_arm` (can be another name) is that CubeMX, which we will use to generate source files for HW based projects, can only use absolute directory for linking the library files. We have stripped down the firmware library files so the they can be copied to this folder.

The `build` folder can be put any where; it is usually in the folder of each project. As we need to put the project source files in the cloud drive or GitHub, it is awkward to have the `build` folder, which is very big for each project, to be mingled with the source files. We use three subfolders in this `build` folder to reduce the interference between the library object files. Note that when you switch to a new project, it is always a good idea to **rebuild** all the files for that project.

Project file structure

We will have slightly different project file structure for the simulator-based and HW-based projects.

Project file structure for simulator-based projects

For the simulator-based projects, we will just use the following simple structure, under an example project folder `expl_010_template_for_simulator_prjc_with_c`:

- `expl_010_template_for_simulator_prjc_with_c` (folder): The folder for the entire project. All source and project management files are included here.
    - `source` (folder): All source files. There may be subfolders under this folder for bigger projects. For small projects, we have all the source files in this folder. For example:
        - `main.c` (file)
    - `expl_c_prjt_sim.uvoptx` (file): A project management file
    - `expl_c_prjt_sim.uvprjx` (file): The **main project management file**

To open the project, we just need to double-click the main project management file. After that, you may see other files and folders created under the project folder. No need to worry about these files and folders for the moment.

Project file structure for HW-based projects

For all the HW-based projects, we have the same project functionality for both HW kits. As such, we program the project functionalities in common source files. We will then use HW specific files for each HW kit. As such, we use the following architecture, under an example project folder `expl_014_template_for_HW_prjt`:

- `expl_014_template_for_HW_prjt` (folder): The folder for the entire project. All source and project management files are included here.
    - `usr_src` (folder): All source files. There may be subfolders under this folder for bigger projects. For small projects, we have all the source files in this folder. For example:
        - `usr_tasks.c` (file): This is the source file used to program the project functionality. The logic here should be common and use as less HW specific handling as possible.
        - `usr_tasks.h` (file): This is the header file. Again, it should use as less HW specific handling as possible.
    - `F412Discovery` (folder): This folder contains all the HW specific files. This folder includes the following:
        - `Drivers` (folder): This is the folder saving all the HW driver library files. We use **Links** to the common library files in `_lib`, so this folder is empty, but it is here as a place holder.
        - `Inc` (folder): This folder include all the HW specific header files that are specifically configured for this project.
        - `MDK-ARM` (folder): This is the folder containing the various project related files, such as the project management files, for the project. To open the

    project, we need to double-click the main project management file in this folder. If we use another other development tool, we will have a corresponding folder for that tool.
    - `Src` (folder): The source files generated by CubeMX. Note that the `main.c` file is in this folder. We need to hack this file a bit so that the control will go to the `usr_tasks.c`. This hack will be done only once; even if we regenerate the code, the changed code is still saved in this `main.c` file.
    - `F412Discovery.ioc` (file): This is the CubeMX project file which contains the setup of the hardware and other project info, such as the hardware peripheral used in the project.
- `L476Discovery` (folder): A counterpart of `F412Discovery`, used for the L476 Discovery kit.