

Virtual Lab

M. Briday

November 18, 2020

1 Hardware/Virtual Platform

1.1 Hardware interactions

The real environment when programming the Coro lab board is in Fig.1

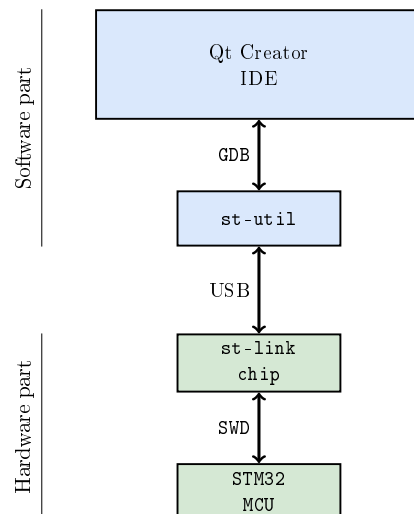


Figure 1: Connection between tools and hardware

- the st-link chip and the STM32 MCU are located on the nucleo32 board
- When starting a debug session, the `st-util` tool is launched by QtCreator

The GDB connection is done through a socket on port 4242.

1.2 Virtual Hardware interactions

The virtual lab is based on the QEMU tool, on which the backend has been modified to:

- model STM32F303 specific hardware (gpio, timers...)
- add a communication layer to display the state of the virtual hardware

The tools are organized as in Fig.2

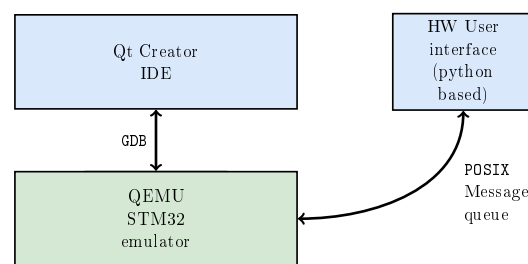


Figure 2: Connection between tools and hardware

The big picture is simpler... but the tools are not connected automatically. You can notice:

- Qt Creator is connected *exactly* as with the real hardware, and it will behave *in exactly the same way*.
- Qemu and the Graphical user interface communicate through POSIX message queue. This means:
 - it will require a POSIX compliant OS (Linux, Mac OS).
 - if one of the processes is not started, the other will freeze as soon as the message queue is full. When a process is frozen, you can just restart the other process to consume messages.

2 How to Start

A basic project have:

- A CMake project file: `CMakeLists.txt`
- a basic C file: `main.c`

2.1 Design with Qt Creator

With Qt Creator, open the project: **File** → **Open file or project** and choose the `CMakeLists.txt` file.

The first time, it will ask for the appropriate kit (Desktop, STM32,...): configure the project for the STM32 target. You should be able to edit your source file (and have a project on the left side), as in Fig.3.

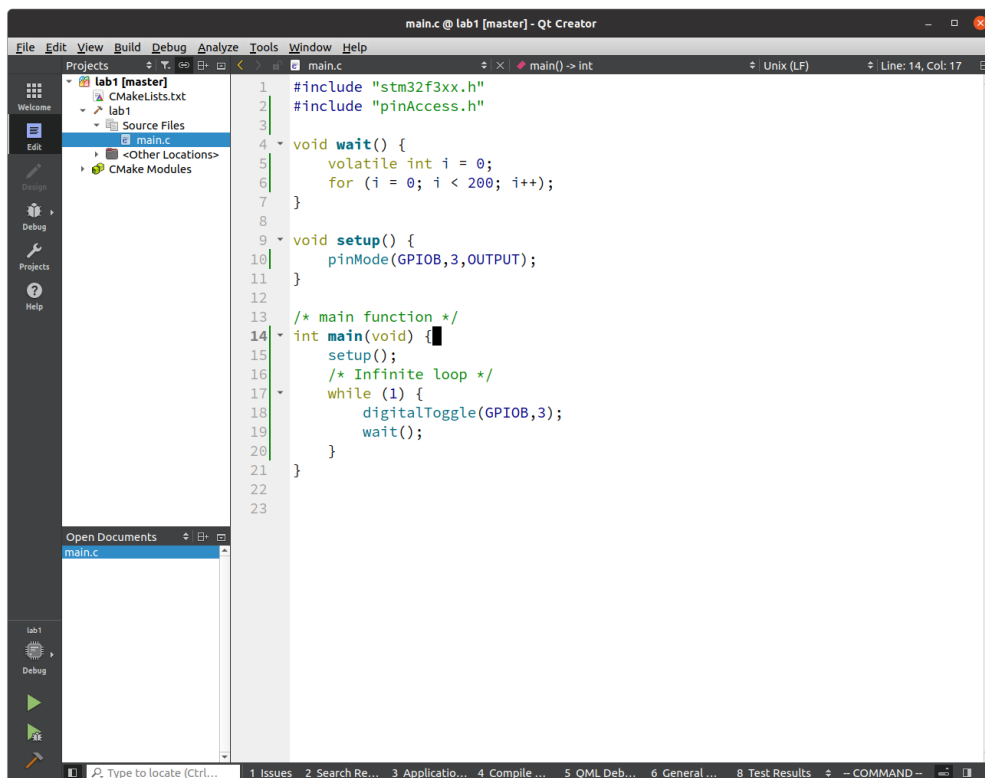


Figure 3: Qt Creator - main project

2.2 Run on Qemu

When the project is built (**Build** → **Build project "lab1"** or **Ctrl+B**), then a new directory is created: `../build-blink-STM32-Debug` (where blink is the project name,

STM32 the kit name and Debug the build mode).

you will have to start a terminal (from the project explorer, right click inside the directory and choose **open in terminal**).

From the terminal, you can start qemu in debugger mode (to be used with Qt Creator)

```
make qemu-gdb
[ 87%] Built target lab1
Scanning dependencies of target qemu-gdb
[100%] Generating lab1.bin
*****
```

Now, qemu is started and waiting for a GDB connexion.

We have to start the user interface that gives information about the environment (buttons, leds, ...). This is a python script that can be run as a program. You have to use another terminal (you can press **Ctrl+Shift+T** to get another tab:

```
-> % ../qemu_io.py
```

A graphical interface should appear, as in Fig.4.

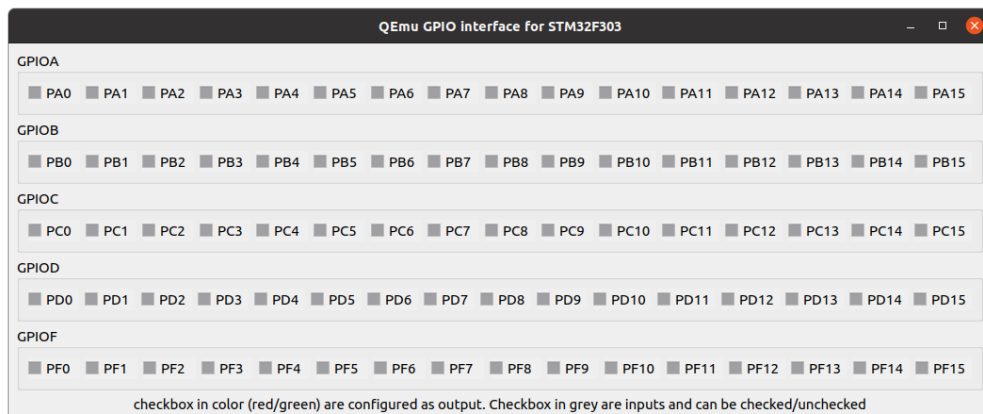


Figure 4: User interface

Important Note:

- Both Qemu *AND* the graphical interface should be started. If a process is not launched, the other will send messages until the message buffer is full, and then will be freezed!. In that case, just relaunch the missing process to unfreeze.
- to exit qemu, you should type **Ctrl+A x**. When a debug session ends (by Qt Creator), qemu should returns.
- the python user interface will be different from one lab to another. Just launch the correct one (and only one!)

2.3 Debug with Qt Creator

Qt Creator should now connect to the GDB server. Chose **Debug** → **Start Debugging** → **Attach to running debug server...** as in Fig.5

You will have to configure (the first time only):

- the local port: 4242
- the local executable: the binary file (same as project name in the build dir, without any extension)
- select **Break at "main"**

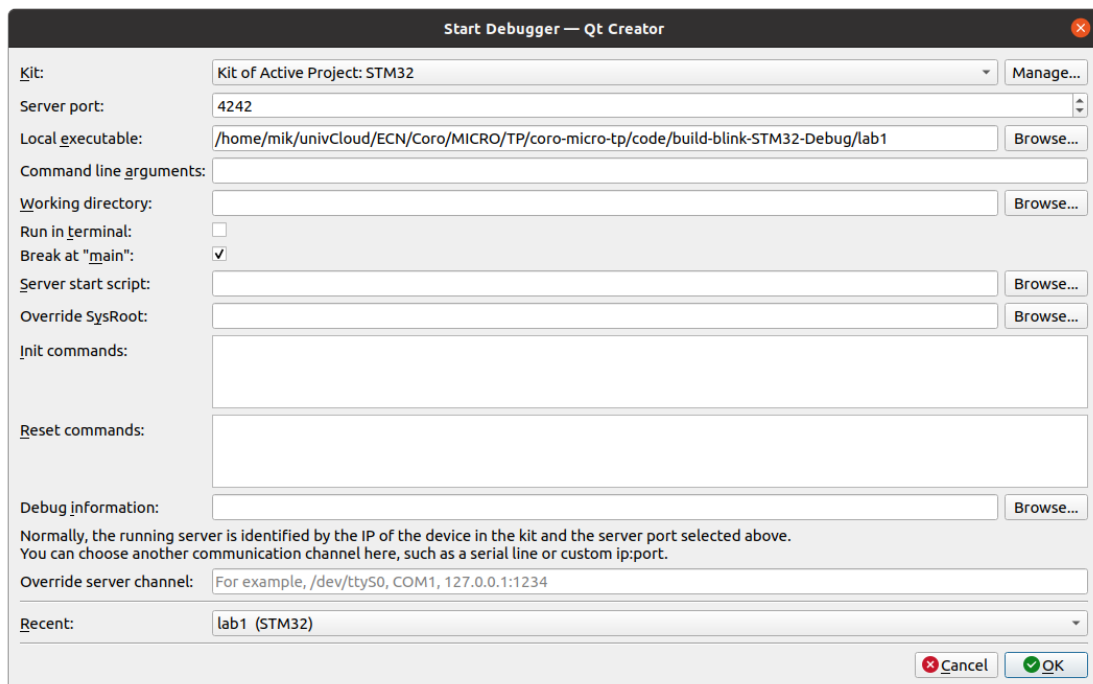


Figure 5: Attach to a running debug server

The debugger should connect and show the code of the `main()` function. The debugger will behave as on the real target, and you should see the led updated after each loop on the user interface as in Fig.6

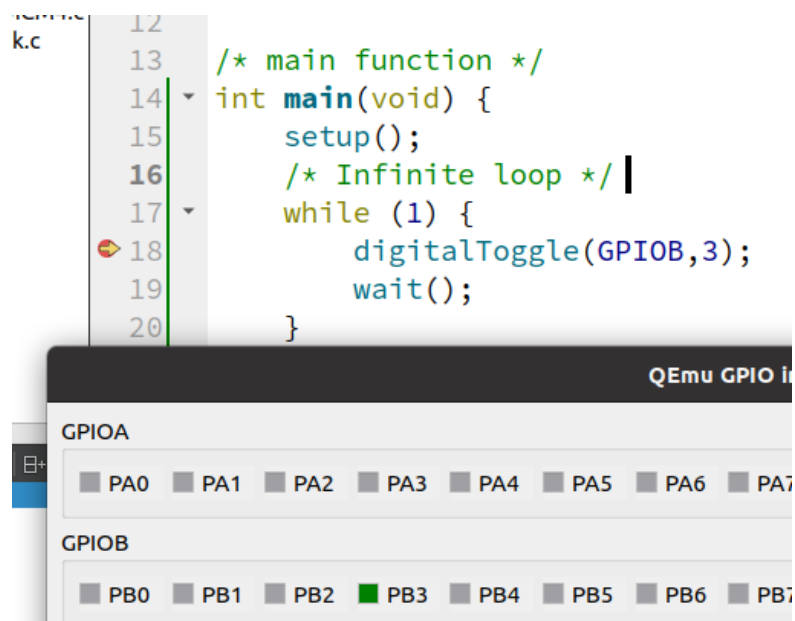


Figure 6: Debug session in progress...