Labwork 1 – GPIO and Timer

H. Cassé <hugues.casse@irit.fr>

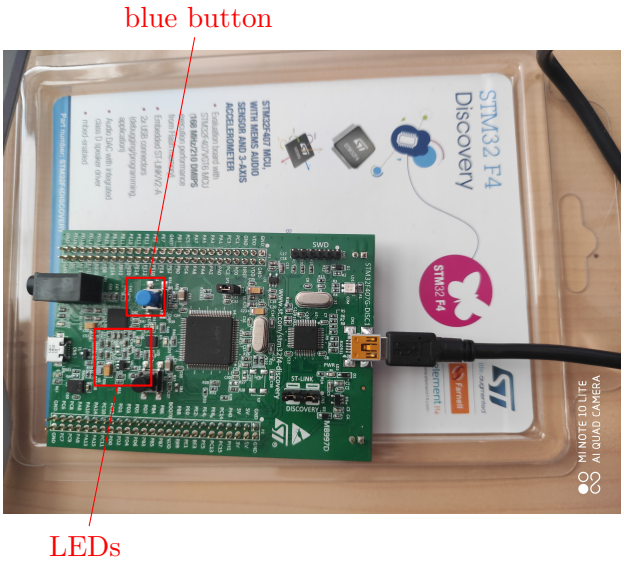
This labwork aims to demonstrate the programming of STM32F4 boards. This session topic is about programming the GPIO by polling.

1 Protocol

1.1 The board

An STM32 F4 Discovery board will be distributed to each student/pair of student with a mini-USB cable. In order to perform the labwork, the board has to be connected to the desktop using this cable.

Warning: to avoid short circuits with the pins of the board (located below) that could damage or break the board, you have to put it on the box blister as shown in the picture below:



The board is equipped with two buttons: as the black one is used to reset the board, we will juste use the blue one. There are also four LEDs: green, orange, red and blue.

The below sum up the wiring:

Device	GPIO	Pin
Blue button	A	0
Green LED	D	12
Orange LED	D	13
Red LED	D	14
Blue LED	D	15

In this session, we will only use these LEDs and this button but in next labworks, we will more devices to the board.

1.2 Programming the board

In order to do the labwork, you have to download and to extract the archive `labwork1.tgz` from the Moodle page. You get a directory named `labwork1` with the following structure:

- `include` Contain header files and particularly #defines useful to program the microntroller.
- `scripts` Contain scripts for linking the executable and configuring OpenOCD.
- `src` Contains the sources used for the exercises.

Each exercise (proposed in the next section) is associated with a source file in `src`.
To compile, you have just to type (in this directory):

```
> make
```

To test your code, you have to maintain 3 consoles (you can use tabs in your Linux environment) named thereafter *Console 1*, *Console 2* and *Console 3*.

In *Console 1*, you have to run `openocd` that is the monitor that connect to the STM32F4 board. After being launched, the console is blocd all the time it runs. Sometimes, messages may be displayed informing about some failure or error in communication:

Console 1

```
> make openocd
```

To stop it, just type ctrl+c.

While it is connected with the USB cable, the STM32F4 is able to perform outputs on the console of the developing desktop. With our configuration of OpenOCD, this is performed in the log file `stm32f4.log`. To get a continuous display of the new messages written in this file, you can type the following command:

Console 2

```
> make log
```

Now, we are able to upload our program on the board and to run and debug it. To do this, we have to launch the debugger GDB:

Console 3

```
> make debug_exN
```

GDB will be started, connect to OpenOCD, upload the program and restart the board.

After that, you can let the program or take the control with ctrl+c. The console is now available to type commands of GDB. Most commands encompasses:

break N (`b`) To put a breakpoint at line N.

continue (`c`) To continue execution (until a breakpoint is found).

next (`n`) To execute the next instruction.

backtrace (`bt`) To display the call stack.

print E Evaluates the C expression E and display its result (useful to display an hardwar register).

A more complete cheat sheet of GDB is profided on Moodle. Notice also that most commands support shortcuts.

To exit from GDB, type ctrl+d.

1.3 Printing to the console

To output to the console (**Terminal 2**), one can use a light version of `printf`:

```
printf("Hello, world!\n");
```

In **Terminal 2**, the output will be:

```
Hello, World!
```

Most common format escapes are available.

2 Exercises

In the following exercises, you have just to complete the given source file, to compile it and to apply the run & debug procedure presented in the previous section.

2.1 Exercise 1

Source: `src/ex1.c`

Write an embedded application that switches on then off the LEDs circularly.

First LED green is lighted on, then LED orange, then LED red and then LED blue and the cycle restarts.

Timing is performed by an empty loop iterating $30,000,000$ times.

Note: Think to initialize the GPIO in the right way.

2.2 Exercise 2

Source: 🗀`src/ex2.c`

Write an embedded application that switches on the green LED when the blue button is pushed and swicthes off the green LED when the blue button is released.

2.3 Exercise 3

Source: 🗀`src/ex3.c`

Write an embedded application that changes the status of the green LED each time there is a click on the blue button. We recall the click is a push on the button followed by a release. On first click, the LED is switched on. On the next click, the LED is switched off and so on.

Remark: you will possibly experiment multiple clicks induced by button spring bounces. How can you fix this?

2.4 Exercise 4

Source: 🗀`src/ex4.c`

Write an embedded application making the green LED blinking. First, select a blink time of 1 second. The waiting time must be implemented using the timer 4.

Then, clicking on the blue button may change the blinking time cyclically between the times $[1s, 500ms, 250ms]$.

2.5 Exercise 5 (optional)

Source: 🗀`src/ex5.c`

In this last exercice, you will implement a watchdog close to the one used in the course:

- First, it waits during 5s maintaing the green LED switched on. This wait time is reset each time the blue button is pushed.
- After 5s, it wait again for 5s. In this case, the green LED is switchd off and the red LED blinks with a period of 500ms (on phase of 250ms, off phase of 250ms). A click on the blue button makes the system to come back in the first phase (green LED on).
- After these 5s, the system freezes with the red light switched on.