Lab 4: Introduction to Debugging

COEN-4720 Embedded Systems
Cris Ababei
Dept. of Electrical and Computer Engineering, Marquette University

1. Objective

The objective of this lab is to start working with the most basic debugging functionalities offered by the STM32CubeIDE. The powerful set of debugging tools offered by STM32CubeIDE will allow you to easily investigate bugs and unwanted behaviors in your future projects.

2. Description

Example 1

Prerequisites for this example:

- Read Chapter 5 of the textbook [1].
- Watch the following youtube video: https://www.youtube.com/watch?v=BVC7KaUNCS8.

In this example – included as **lab4_ex1** in this lab files - we use the B1 User Button of the Nucleo board to control turning on/off the green LD2 (Green LED). When we press the button, the LED should turn off. When the button is not pressed, the LED should be on. For debugging, we use the debugger with **breakpoints**.

Note that the user button "B1 USER" is connected to the I/O PC13 (pin 2) of the STM32 microcontroller. Also, "User LD2", the green LED, is a user LED connected to ARDUINO signal D13 corresponding to STM32 I/O PA5 (pin 21). Please see the user manual and schematic diagram of the board to confirm that now! These pins are highlighted in Figure 1 below.

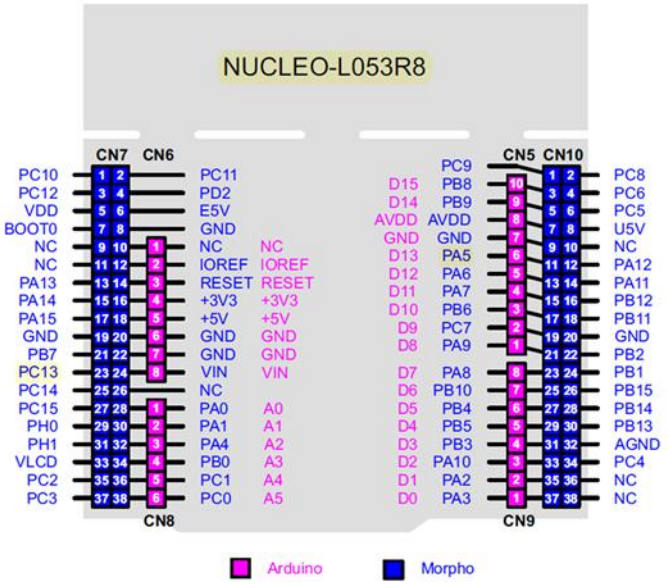

Figure 1: Pins of Nucleo-L035R8 board.

The STM32CubeIDE project for this example, **lab4_ex1**, is provided with the files for this lab. Open it and identify inside main.c how the LED is turned on and off with this code:

```
    if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13)) {
      // this branch is entered when button is NOT pressed!
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    } else {
      // this branch is entered when button IS pressed (because
      // of how button is connected to MCU pin - see schematic
      // diagram of NUcleo board);
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    }
    HAL_Delay(100);
```

And, how the pin PC13 is configured as INPUT with this code:

```
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT; // GPIO_MODE_IT_FALLING;
```

Build the project and then program the board (which you should have already connected to your computer with the USB cable). Observe operation and confirm this example works correctly.

Now, let us debug this simple program using one breakpoint, which should be set inside the main.c on the line with this code:

```
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
```

After setting the breakpoint, click on icon Debug->Debug Configurations…
In the window that pops-up, in the Main panel, browse to select lab4_ex1 as the project.
In the same panel, select lab4_ex1\Debug\lab4_ex1.elf as the C/C++ Application.

Then, select the Debugger panel, then click on the Debug button (bottom-right) to launch the debugger. Make sure you disconnect the STMCubeProgrammer first. Use the “step over” icon to step through several debug steps as discussed in the above prerequisite video. Observe operation and comment.

What you should observe is that the execution stops or not at the breakpoint – based on whether we press or not the Blue User Button on the board. To arrive to observe that, once the debug session is started, keep pressing the “Step Over” icon in the IDE until we get first time to the **if** statement, AFTER WHICH we continue by pressing “Resume” icon. At this point, the execution should be observed in two situations:

1. Without the button pressed: the execution keeps going doing the **while(1)** loop indefinitely without interruption (Notice that “Resume” icon is not clickable – as a confirmation of the above).
2. With the button pressed: the execution is stopped at the breakpoint – case when you could investigate the status of the program, registers, etc., as part of your debugging process . To

continue the execution, you need to press again "Resume" icon and thus get to go through ha new iteration of the **while(1).** (Notice that "Resume" icon is clickable in this case).

Example 2

Prerequisites for this example:

- Read Chapter 8, Sections 1-3 of the textbook [1].

This example – included as **lab4_ex2** in this lab files – is a modified version of example 1. The modification consists of the use of USART2 peripheral (or device) inside the microcontroller as **UART in Polling Mode** to send messages to a serial terminal on the PC. Such messages can be helpful for debugging too.

For now, we will just use the UART device as given in the provided code – to learn for the first time about it; later we will discuss more details about UART in general. Looking at the code, you will see that we use the function HAL_UART_Transmit() to send messages to a serial terminal. Here is one example of how the function is used in the code:

```
HAL_UART_Transmit(&huart2, (uint8_t*)BUTTON_PRESSED_MSG,
strlen(BUTTON_PRESSED_MSG), HAL_MAX_DELAY);
```

At this time look at the code to get a clear understanding of what it does (it will be discussed in class too). Before building, install a Serial terminal into the STM32CubeIDE as presented in Section 8.3.1 of the textbook.

Build the project and download to the Nucleo board using the STMCubeProgrammer.
Open the serial terminal and observe operation. You should see the serial terminal printing messages as shown in Figure 2 below.

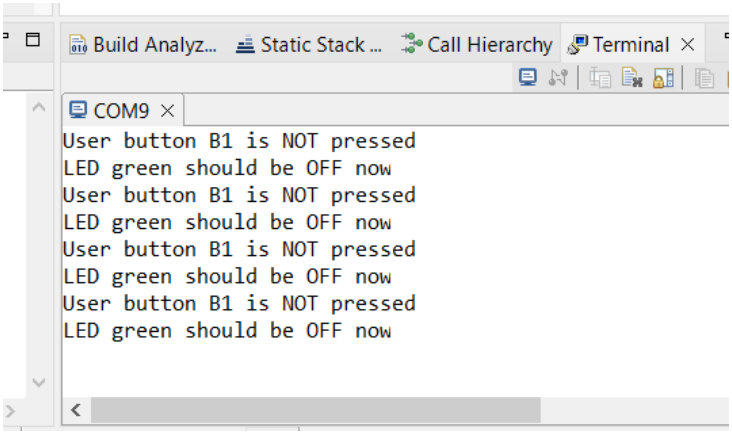

Figure 2: View of the Serial terminal with messages sent from the MCU via UART.

Keep in mind in your future work that you can use such "printing" technique to develop debugging strategies.

Example 3

Prerequisites:

- Download the source code for Chapter 5 from the GitHub repository of the textbook, let us say for board Nucleo-L073RZ (the closest to our own Nucleo board): https://github.com/cnoviello/mastering-stm32-2nd

Create a new project called lab4_ex3; of course, select Nucleo-L053R8 as the target board during the project creation process.

Open main.c of your newly created project inside the STM32CubeIDE.
Separately, in a text editor such as NotePad++, open the main-ex1.c of the project Nucleo-L073RZ\CH5.

Walk through both files side by side, main.c and main-ex1.c, and copy from main-ex1.c to your main.c the relevant code that your file is missing. For example, main.c has:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

While main-ex1.c has:

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <retarget.h>
#include <stdio.h>
```

So, copy the last two includes into your main.c, and so on.

Then, copy from Nucleo-L073RZ\CH5 the following two files:
Nucleo-L073RZ\CH5\Core\Inc\retarget.h
Nucleo-L073RZ\CH5\Core\Src\retarget.c
into your project; you can do that by simply using Windows Explorer.

Then, edit your syscalls,c to comment out several function descriptions:

```
int _close(int file)
int _fstat(int file, struct stat *st)
int _isatty(int file)
int _lseek(int file, int ptr, int dir)
```

as shown in the corresponding syscalls.c from Nucleo-L073RZ\CH5.

Now, build the project Project->Build Project.
The project should compile ok without any errors.
Program the Nucleo board and observe operation. Observe the Serial terminal, where you should be prompted to type in the number of times you wanted to have the message printed!

Now, start a new debug session, by creating a debug configuration: click on the Debug icon at the top-right of the IDE.
Insert a breakpoint inside your main.c file next to the line:

```
HAL_Delay(500);
```

Then, on the Eclipse toolbar, click on Debug->Debug As->STM C/C++ Application
Step Over through the main program.
Make sure that when you get to the line:

```
scanf("%hhu", &uTimes);
```
you switch panels to the Serial terminal to input value for uTimes variable (number of times to print message).
Switch back to Debugger Console panel and continue to step over until reaching the first time the breakpoint. Switch back to Serial terminal panel and observe that the message was printed the first time.
Continue to above steps and observe correct operation. Pretty cool!

Keep in mind in your future work that you can also use such technique - interaction with the terminal via retargeting - to develop your own debugging schemes for the application at hand.

3. Lab Assignment

Do all the provided examples and demo them to the TA.

Create a new project called lab4_assign1. Port into it the relevant code from the example Nucleo-L073RZ\CH8\Core\Src\main-ex1.c (which you downloaded from github already); that is: copy and paste into your main.c of your new project the missing code to implement the example main-ex1.c, which is described in Section 8.3 in the textbook. Then, change the functionality of this example such that when the user inputs the option "1", the LED Green is blinked twice in a row for 500ms and with 500ms delay between blinking – instead of being toggled.

4. References and Credits

[1] Textbook:
Carmine Noviello, Mastering STM32 - Second Edition, 2022, Available to purchase online:
https://leanpub.com/mastering-stm32-2nd <--- Buy from
https://github.com/cnoviello/mastering-stm32-2nd <--- Code examples