Practice 2: Introduction to *Embedded* systems (STM32F767ZI)

Curse 2022/2023

Made by: Mario Pereira Martin

Date: October 16, 2022

Campus Sevilla
Avda. de las Universidades s/n.
41704 - Dos Hermanas, Sevilla, España.
Tel. +34 955 641 600

Campus Córdoba
Escritor Castilla Aguayo, 4.
14004 - Córdoba, España.
Tel. +34 957 222 100

1. Objectives

This practice is an introduction to embedded systems. These systems will be used in the project related to the curse: '***Real Time and embedded systems / Concurrent and real time systems***. The topics introduced during this session are the following:

- STM32F767ZI Connected to the Laptop and STM32CubeIDE software introduction.
- Ozone software introduction for debugging.
- Introduction to C/C++ Code for GPIO management.

2. Materials

The materials needed for this practice are the following:

- STM32F767ZI board with power supply from laptop (provided by teacher).
- Protoboard, LED (diode) and resistance (provided by teacher).
- Laptop with Windows operating system (from students or the labs).
- USB cable (provided by teacher).

3. STM32F767ZI board

Figure 1 shows a top view of the stm32 Nucleo board that can be found in practice laboratories. This is the **STM32F767ZI** model board.

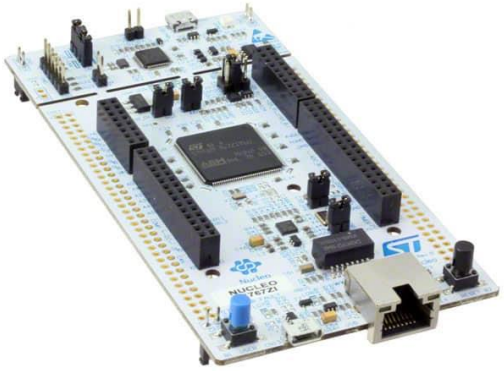

Figure 1. STM32 Nucleo board (STM32F767ZI model) – top view

[1] More information can be found at https://www.st.com/en/evaluation-tools/nucleo-f767zi.html#st_description_sec-nav-tab

[2] More information can be found at https://www.freertos.org/

The main characteristics of this board are the following:

- Common features
  - STM32 microcontroller in the LQFP144 package
  - Three user LEDs
  - Two users and reset push-buttons
  - 32.768 kHz crystal oscillator
  - Board connectors: SWDST Zio expansion connector including ARDUINO® Uno V3ST morpho expansion connector
  - Flexible power-supply options: ST-LINK, USB VBUS, or external sources
  - On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port, and debug port
  - Comprehensive free software libraries and examples available with the STM32Cube MCU Package
  - Support of a wide choice of Integrated Development Environments (IDEs), including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE
- Board-specific features
  - External or internal SMPS to generate Vcore logic supply
  - Ethernet compliant with IEEE-802.3-2002
  - USB OTG full-speed or device only
  - Board connectors: USB with Micro-AB or USB Type-C® Ethernet RJ45
  - Arm® Mbed Enabled™ compliant

More details in project documents and ST web page[1].

4. Operating System

Usually, there is no operating system running in this stm32 Nucleo board, and it uses a super loop-based structure. However, in this practice, we will work with a relevant RTOS. The RT operating system running in this stm32 Nucleo board will be FreeRTOS[2]. It's a market-leading real-time operating system that works with microcontrollers and small microprocessors. FreeRTOS includes a kernel and a growing set of IoT libraries for use across all industry sectors.

FreeRTOS operating system is just in the provided project as a middleware. The development environment inserts this operating system.

5. STM32F767ZI connected to the laptop and STM32CubeIDE software introduction.

STM32 board is prepared to be connected to different development environments (using the USB port, via ST-Link protocol or J-Link protocol). However, trying to reduce the number of steps that student needs and simplify the corresponding practice, it's proposed to use a preset project. The objective is to create an application with the operating systems and all the drivers to work with several board peripherals. This code is needed to manage any proposed system, avoiding building the project from the beginning.

✓ **Step 1**. Download (from Moodle) and install the STM32CubeIDE software (see Chapters 5-6 from the book *RTOS with Microcontrollers* or chapter 2 from *Practice: Working with STM32F767ZI*). This program must be installed on your laptop following the screen instructions. It's an eclipse variation used such as cross-compiler but taking into account that st-link is not used and the debugging program will be another software.

✓ **Step 2**. Download (from Moodle) and decompress in a located folder the resource 'Projects from RTOS on microcontrollers'. Then import only chapters 5-6.

✓ Describing the process in some micro-step:

a. In the C/C++ view, go to *File → Import → General → Projects from a folder or archive → click on next.* In the windows that appear, choose *Archive* and search the file. You can put the path directly on the Input source text box. Click on the chapter5_6 project only. Then click on Finish.

✓ **Step 3**. Try to build the new project on your laptop by following these micro-steps:

a. In the C/C++ view, go to project explorer and right-click on *<project_name> → Build project.* In the console windows must appear the same as shown in figure 2.

```
arm-none-eabi-size   Chapter5_main.elf
   text    data     bss     dec     hex filename
  32656      48   23004   55708    d99c Chapter5_main.elf
Finished building: default.size.stdout

arm-none-eabi-objcopy -O ihex "Chapter5_main.elf" "Chapter5_main.hex"


13:22:20 Build Finished. 0 errors, 0 warnings. (took 3s.885ms)
```

Figure 2 Console Windows output when you build the project

In the 'Project Explorer' window, you can watch all the folders related to the project. This project contains information about all the board peripherals (drivers and others), the RT operating systems and our customized source code that must run on the board. Now we are going to explain details about the main folder:

- Binaries: Include information about the final file (*.elf) that must be inserted in the board. This file contains all the source code (operating system, filesystem, drivers, application and other necessary codes).
- Includes: shows the .h libraries used for project compilation. You can introduce any library you need, but taking care of it is recommended.
- BSP: it includes code and libraries related to peripherals. Usually, timers, GPIO, USB, UART, ADC and others. In future, it's recommended to use the code generator to manage these drivers to avoid problems developing our application.
- Debug: It's a reduced version of the complete application to create a debug version of our application.
- Drivers: It contains specific libraries for peripherals.
- Inc: Setting files for drivers and other middleware.
- Middleware: It contains Third-party software to be used in our project. In particular, we will introduce the FreeRTOS, USB and SEGGER debugging tools.
- Src: This is the main folder. Here we create our application introducing code in the main.c file.
- Startup: it contains files to manage the start-up of our application. Try not to change it.
- Loose files: The main files here are the *.jdebug file used to manage the debugging software (see details in documentation) and the *.ioc file used to manage the code generator. Due to this project being developed on another version of the STM32CubeIDE, this file is not working. The teacher will provide the instructions to solve this inconvenience in class.

In the new project, some of these folders are included in a new folder called 'Core' but essentially is the same structure.

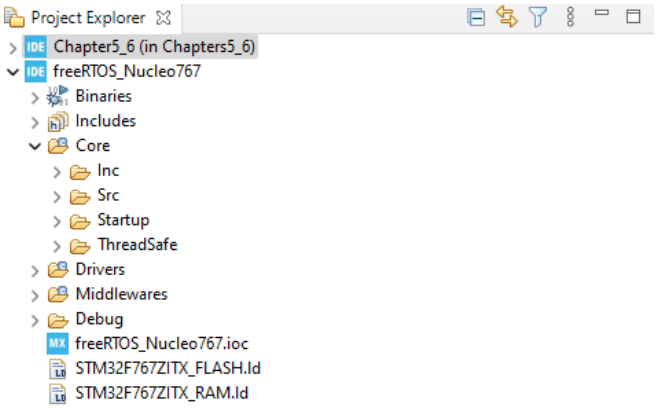

Figure 3 Project Explorer with the previous and new project

Afterwards, The main file (in the 'src' folder) is a significant source code with many labels that must be respected for working with the code generator. Students must only add code in the corresponding labels. If you include code in other places, it could be deleted. Thus read all tags and focus your attention on writing code on titles related to threads, semaphores, queues and mutexes (see figure 4).

```
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
MX_ETH_Init();
MX_USB_OTG_FS_PCD_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
/* USER CODE END RTOS_QUEUES */

/* Create the thread(s) */
/* creation of defaultTask */
defaultTaskHandle = osThreadNew(StartDefaultTask, NULL, &defaultTask_attributes);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
/* USER CODE END RTOS_THREADS */

/* USER CODE BEGIN RTOS_EVENTS */
/* add events, ... */
/* USER CODE END RTOS_EVENTS */
```

Figure 4 Tags in main.c

6. Ozone software introduction for debugging

At this time, STM32CubeIDE is ready to work, and we have a *.elf file to be inserted into the development board. To insert our code and debug its behaviour, the student must download and install the Ozone software available in Moodle.

Then, start Ozone J-Link Debugger v3.24f and close the first windows that appear (see figure 5).

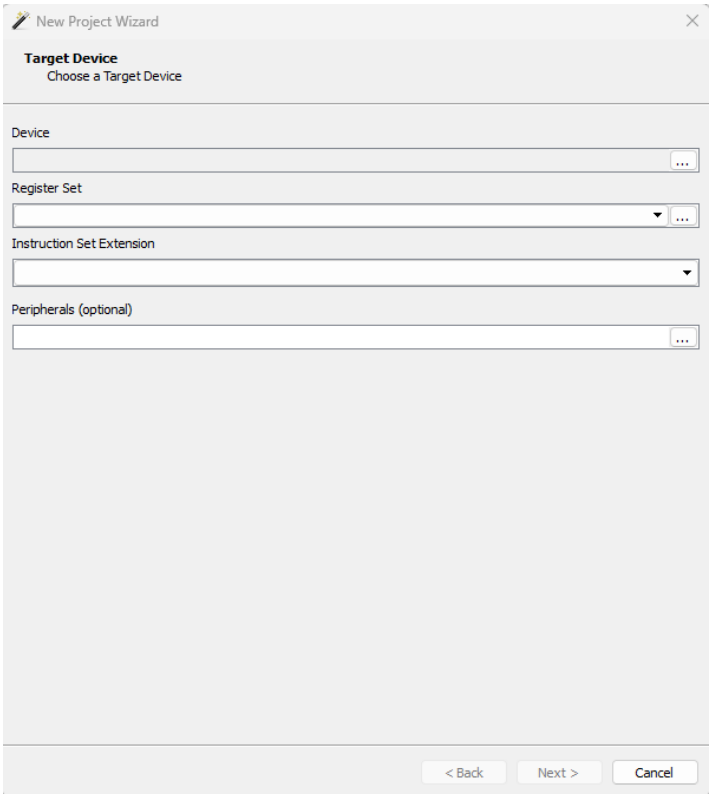

Figure 5 First Windows on J link debugger

Click on *File* → *Open* and search the path of the *.jdebug file in the chapter 5_6 project. You can find the path file doing right-click on Chapters5_6.jdebug file → *show in* → *System Explorer*.

If something is wrong, edit the project and place the right path in the settings. Once all is right, you obtain a screen similar to figure 6.

The steps to insert a code into the board are the following:

- ✓ Step 1. Connect the board to your computer using the USB cable.
- ✓ Step 2. Click on the down icon near the power icon (green colour) and choose download and reset program. Click on yes or accept in any appearing Windows. Don't take into account the licence windows. Only accept it.

In main.c you can watch all your code and introduce any breakpoint to check it.

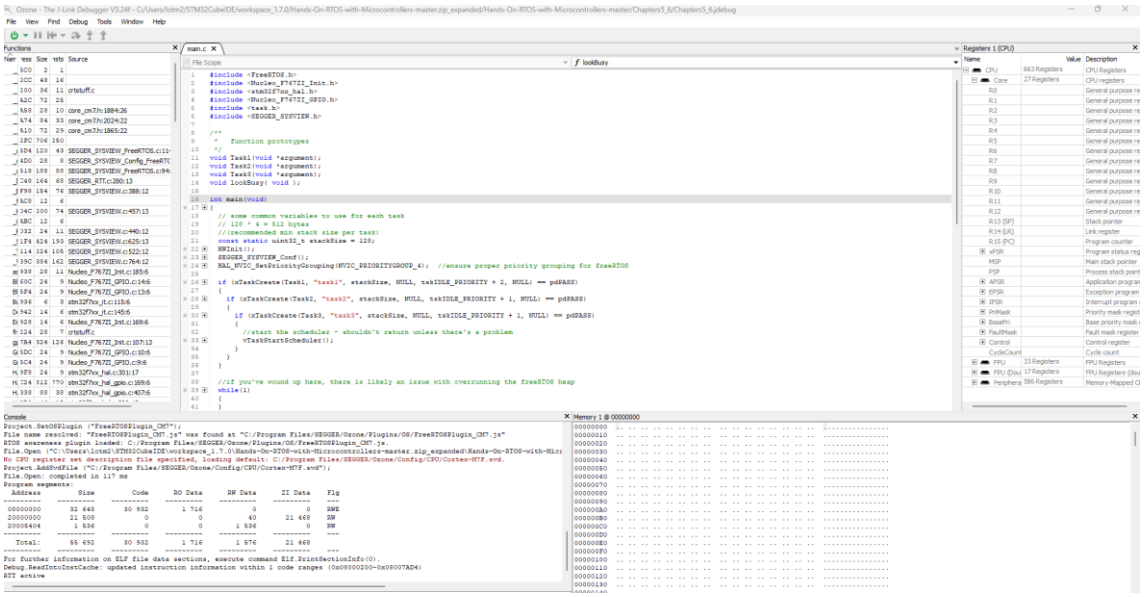

Figure 6 Ozone J link debugger

7. GPIO module in the STM32F7 board using STM32CubeIDE.

One of the main objectives of this practice is to learn to work with the General Purpose Input Output port (GPIO) of the main microcontroller in STM32F767. In this case, the student should learn how to work with it using the Graphic environment of STM32CubeIDE (Code Generator). This environment is included in the development environment and makes it easier to work with peripherals.

The exercises in this practice are easy and focus on setting up some pins as input or output. The student must activate pins in the graphical environment and wait for the code generator to introduce all the changes in the code. Another important topic is that the student must know all the details about the board's pinout (see details in the other documents or the web page) and that students must take care of the labels in the code.

Pinout web page: https://os.mbed.com/platforms/ST-Nucleo-F767ZI/

Exercises

1. Modify the status of a pin to switch on a LED following these steps:
    a. Configure a pin in the Graphical environment as output and switch on it, changing its value to one.
    b. Connect the previous output pin to a LED ***in series with a resistor*** (100 ohms).
2. Change the task on your code to toggle the previous LED between logic one and zero in a periodic way. To do this, students must watch the previous code done in chapter 5_6.

3. Try to read a GPIO input in the Stm32. To check if it is all right, the student must use the 3.3V/0V pin to switch on/off the GPIO input.
4. Create a new task to read periodically the GPIO input configured in exercise 3.

Homework

Search information about the following topics related to the practice:

1. Graphics environment in STM32CubeIDE. Functioning (What is it and how it works).
2. PWM signals. Functioning (What is it and how it works).
3. How to configure PWM pins on STM32CubeIDE**.**

Evaluation

The evaluation of this practice is done taking into account the performance observed during the practice sessions in the laboratory (it means attention, participation, care of laboratory material, involvement in the right development of the actual session and so on) and a ***practice report*** about the following topics:

- ✓ Results of exercises 1 to 4.
- ✓ Results of the homework: all the information obtained at home for exercise 5 (see homework).

This ***report*** should be delivered by Moodle.