



Australian  
National  
University

## **Lab Seven**

# **Timer and Serial Communication of STM32 microcontroller**

**ENGN4213/6213**

**Digital Systems and Microprocessors**

Semester 1, 2023

Copyright © 2023, The Australian National University

# Table of Contents

1	Introduction .....	3
2	Pre-lab Task .....	3
3	What You Need .....	3
4	Timers on STM32F411RE.....	3
4.1.	Advanced-control timers (TIM1) .....	4
4.2.	General-purpose timers (TIMx) .....	4
5	USART on STM32F411RE .....	5
6	Activity-1: Escape Game with Timer [20 marks] .....	6
6.1	Timer Configuration .....	7
6.2	Timer as Internal Interrupt .....	8
7	Activity-2: Escape Game with UART Communication [10 marks] .....	10
7.1	UART configuration .....	11
7.2	UART DMA programming.....	11
	Appendix A: Hercules SETUP utility .....	12
	Appendix B: CubeMX & TrueStudio .....	13

# 1 Introduction

During this lab you learn the implementation of Timers and UART (Universal Asynchronous Receiver/Transmitter) communication using STM32 Nucleo board.

## 2 Pre-lab Task

- Complete Activity-2 of Lab-6
- Understand the concepts of Timers and UART communication on microcontrollers
  - Review **Week 7 Lecture B** and **Week 8 Lecture A**.
  - Refer Chapters 7 & 8 from the book 'Programming with STM32: Getting Started with the Nucleo Board and C/C++'. ANU students can get free access to this book at <https://learning.oreilly.com/library/view/programming-with-stm32/9781260031324/?ar>.
- Install serial port terminal
  - Windows: Hercules SETUP utility (Download from <https://www.hw-group.com/software/hercules-setup-utility>)
  - Mac: Coolterm (Download from - <https://freeware.the-meiers.org>), goSerial

## 3 What You Need

### Hardware:

- A STM32F411RE Nucleo board
- A USB type A to mini-B cable

### Software:

- STM32CubeIDE Development Platform
- Serial port terminal

## 4 Timers on STM32F411RE

A timer is simply a free-running counter that counts pulses from a clock source. Since the pulse train from a clock source has a known period or interval between pulses, the elapsed time is directly related to the number of pulses counted. The common hardware functions you'll see with timers are:

- **Output compare (OC):** toggle a pin when a timer reaches a certain value
- **Input capture (IC):** measure the number of counts of a timer between events on a pin
- **Pulse width modulation (PWM):** toggle a pin when a timer reaches a certain value and on rollover.

The STM32 device embeds multiple timers providing timing resources for software (providing time bases, timeout event generation and time-triggers) or hardware (generate waveforms, measure incoming signal parameters and react to external event) tasks. The STM32F411RE device have one advanced-control timer, seven general-purpose timers and two watchdog timers as listed in Figure 1.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/compare channels	Complementary output	Max. interface clock (MHz)	Max. timer clock (MHz)
Advanced-control	TIM1	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	100	100
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	50	100
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	50	100
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	100	100
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	100	100

Figure 1: Features of Timers in STM32F411RE

#### 4.1. Advanced-control timers (TIM1)

The advanced-control timer (TIM1) can be seen as three-phase Pulse Width Modulation (PWM) generators multiplexed on 4 independent channels. It has complementary PWM outputs with programmable inserted dead times. It can also be considered as a complete general-purpose timer. Its 4 independent channels can be used for 'Input capture', 'Output compares', 'PWM generation (edge- or center-aligned modes)', and 'One-pulse' mode output.

If configured as standard 16-bit timers, it has the same features as the general-purpose TIMx timers. If configured as a 16-bit PWM generator, it has full modulation capability (0-100%). The advanced-control timer can work together with the TIMx timers via the Timer Link feature for synchronization or event chaining. TIM1 supports independent DMA request generation.

#### 4.2. General-purpose timers (TIMx)

##### TIM2, TIM3, TIM4, TIM5

The STM32F411xC/xE devices have 4 full-featured general-purpose timers: TIM2, TIM5, TIM3, and TIM4. The TIM2 and TIM5 timers are based on a 32-bit auto-reload up/down counter and a 16-bit prescaler. The TIM3 and TIM4 timers are based on a 16-bit auto-reload up/down counter and a 16-bit prescaler. They all feature four independent channels for input capture/output compare, PWM or one-pulse mode output. This gives up to 15 input capture/output compare/PWMs.

The TIM2, TIM3, TIM4, TIM5 general-purpose timers can work together, or with the other general-purpose timers and the advanced-control timer TIM1 via the Timer Link feature for synchronization or event chaining. Any of these four general-purpose timers can be used to generate PWM outputs. They all have independent DMA request generation. They are capable of handling quadrature (incremental) encoder signals and the digital outputs from 1 to 4 hall-effect sensors.

### **TIM9, TIM10 and TIM11**

These timers are based on a 16-bit auto-reload upcounter and a 16-bit prescaler. TIM10 and TIM11 feature one independent channel, whereas TIM9 has two independent channels for input capture/output compare, PWM or one-pulse mode output. They can be synchronized with the TIM2, TIM3, TIM4, TIM5 full-featured general-purpose timers. They can also be used as simple time bases.

## **5 USART on STM32F411RE**

The Universal Synchronous/Asynchronous Receiver/Transmitter (USART) and the Universal Asynchronous Receiver Transmitter (UART) are the primary peripheral devices used for bidirectional data transfer between the STM32 microcontrollers and external devices. The UART communication is completely self-synchronizing while USART use synchronizing clock pulse train between the nodes.

The STM32F411RE device embed three universal synchronous/asynchronous receiver transmitters (USART1, USART2 and USART6). These three interfaces provide asynchronous communication, IrDA SIR ENDEC support, multiprocessor communication mode, single-wire half-duplex communication mode and have LIN Master/Slave capability. The USART1 and USART6 interfaces can communicate at speeds of up to 12.5 Mbit/s. The USART2 interface communicates at up to 6.25 bit/s. USART1 and USART2 also provide hardware management of the CTS and RTS signals, Smart Card mode (ISO 7816 compliant) and SPI-like communication capability. All interfaces can be served by the DMA controller.

USART name	Standard features	Modem (RTS/CTS)	LIN	SPI master	IrDA	Smartcard (ISO 7816)	Max. baud rate in Mbit/s (oversampling by 16)	Max. baud rate in Mbit/s (oversampling by 8)	APB mapping
USART1	X	X	X	X	X	X	6.25	12.5	APB2 (max. 100 MHz)
USART2	X	X	X	X	X	X	3.12	6.25	APB1 (max. 50 MHz)
USART6	X	N.A	X	X	X	X	6.25	12.5	APB2 (max. 100 MHz)

**Figure 2:** USART Feature Comparison

## ESCAPE GAME Cont...

### 6 Activity-1: Escape Game with Timer [20 marks]

-> Congratulations.

-> You successfully blinked the LED in Morse code and survived from the bird.

-> ...

-> You have been pressing the push button multiple times and feeling tired.

-> But the bird is still hovering around the chip.

-> You know, saving energy is the priority.

-> ...

-> !

-> You noticed that the bird is also coming to you periodically.

-> It is approximately ...

-> ...

-> Every 21 seconds!

-> Instead of a manual interrupt, why not for every 21 seconds, pause the blinking for 5 seconds, activate the blinking for 16 seconds?

-> Try to use a timer to defend automatically!

-> There still could be some urgent issues. So, keep the blue push button for emergency (the highest priority) in case!

## Instructions:

This is continuing work from Lab 6 section 7.2. You can make a copy of your previous code and make changes in it.

### 6.1 Timer Configuration

1. In your “Pinout & Configuration” window, click on TIMx (x could be any number from 2 to 11, check the timer features in Figure 1 before choosing).
2. Under “TIMx Mode and Configuration”, change **Clock Source** to **Internal Clock**. In this lab, we are going to use the timer in ‘PWM generation’ mode. Therefore, change **Channel 1** to **PWM Generation CH1**.

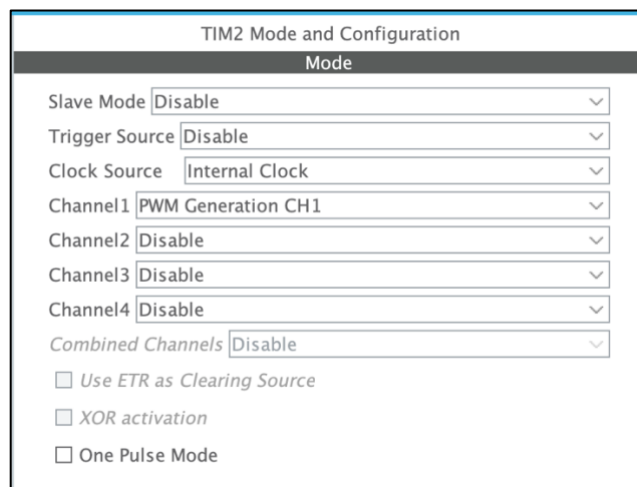


Figure 3: Timer Mode and Configuration

3. Read through **stm32f411re datasheet** (included in MC Lab resources folder) and understand how the TIMx timer you chose is scaled / divided from the internal clock. Identify the Advanced Peripheral Bus (APB) to which your chosen timer connected. (Hint: check Figure 3 of stm32f411re datasheet).
4. In “Clock Configuration” tab, **HCLK** (MHz) is your internal clock speed. Identify your **timer TIMx clock frequency** based on the relation to APB1 / APB2 using the knowledge from the datasheet. You can change your timer clock frequency by adjusting the APB1 / APB2 Prescaler and clock multiplier if required.
5. Based on the **timer TIMx clock frequency**, make changes in **Prescaler (PSC)**, **Counter Period (AutoReload Register)**, and **Pulse** values under ‘Parameter Settings’ of “TIMx Mode and Configuration” window (Figure 4) to generate a PWM signal that allows you to **display the message for 16 seconds** and **pause it for 5 seconds**, and repeat this cycle. Make note of your calculations to show it to your tutor during output demonstration.

Hints (refer **Week 7 Lecture B** for more information):

- $f_{COUNTER} = \frac{f_{CLK\_PSC}}{PSC+1}$
- $ARR = T_{PWM} * f_{COUNTER} - 1$
- $Pulse = T_{PWM\_ON} * f_{COUNTER} - 1$

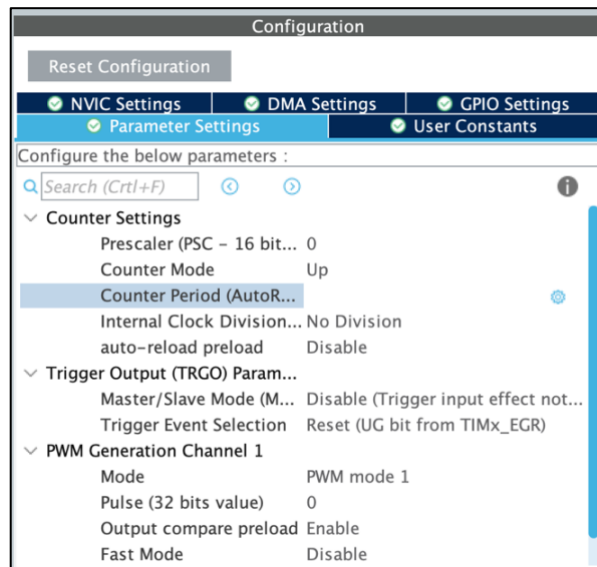


Figure 4: Timer Parameter Settings

## 6.2 Timer as Internal Interrupt

1. Similar to what you did in Lab 6 Section 7.2, enable your TIMx as an **internal interrupt** under NVIC settings of TIMx Mode and Configuration window.

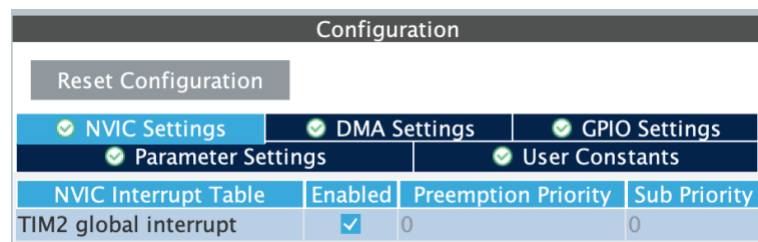


Figure 5: Enable Timer Interrupt

2. Since you have two interrupts – *external interrupt (blue push button)* and *timer interrupt*, the **priority** needs to be defined. As shown in Figure 6, in 'NVIC Mode and Configuration,' change **Priority Group** according to the total number of interrupts. Change **Preemption Priority** of your activated interrupts (Hint: the smaller the value, the higher the priority.)

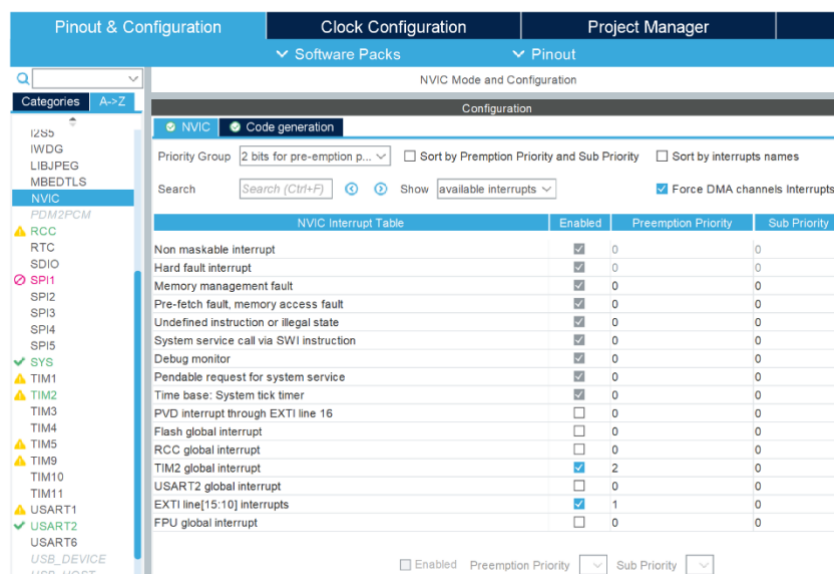


Figure 6: Interrupt Priority Configuration



3. Save your configuration and generate code.
4. From the HAL document ('stm32f4-hal' in MC Lab resources folder), find functions to **start your timer interrupt**. Put them in between `/* USER CODE BEGIN 2 */` and `/* USER CODE END 2 */`.

Hints:

- We need two timer interrupts:
    - To identify the end of PWM ON period to pause Morse code blinking.
    - To identify timer roll over (end of full period) to reactivate the blinking.
  - Start the PWM signal generation in interrupt mode to detect the end of PWM pulse.
  - Start the timer base generation in interrupt mode to detect the timer roll over.
  - To identify your TIM PWM handle, check under `/* Private variables-----*/` in `main.c`. Also check the C Primer document to see how pointer parameters are passed in function calls.
5. Similar to what you did in Lab 6 Section 7.2, use the **callback function(s) of your timer TIMx** to achieve the periodic interrupt from blinking LED. Modify other code if necessary.

Hints:

- Find the callback functions related to the two timer interrupts.
  - Use global variables to store the interrupts' states to control the message blinking similar to the implementation of blue pushbutton interrupt.
6. Demonstrate your work on Nucleo board as following
    - When the board is powered up, blink your SOS message
    - After every 16 seconds, the bird comes and stays for 5 second. So you need to
      - Stop blinking after every 16 seconds and the duration of unlit LED is 5 seconds.
      - Then start blinking from the beginning of the message.
      - If your message is a bit long, shorten the duration of dots, dashes and gap delays.
      - This activity is periodically performed automatically.
      - Whenever you press the blue push button, the blinking stops immediately and pause for 3 seconds.

## 7 Activity-2: Escape Game with UART Communication [10 marks]

-> The bird finally got bored and left.

-> But. . .

-> Unfortunately, the message you tried to convey was too fragmented, and no one came to help you. . .

-> !

-> A person noticed the message and became curious.

-> The person was trying to communicate with you, but you could not hear the voice.

-> . . .

-> You saw the person holding a keyboard.

-> You came up with an idea – send your message through UART serial communication (board to PC). Then the person will see your message and give a reply through keyboard. Receive the words the person typed through UART receiver (PC to board).


-> You do NOT need the push button interrupt, timer interrupt, and Morse code blinking in this section.

## Instructions

### 7.1 UART configuration

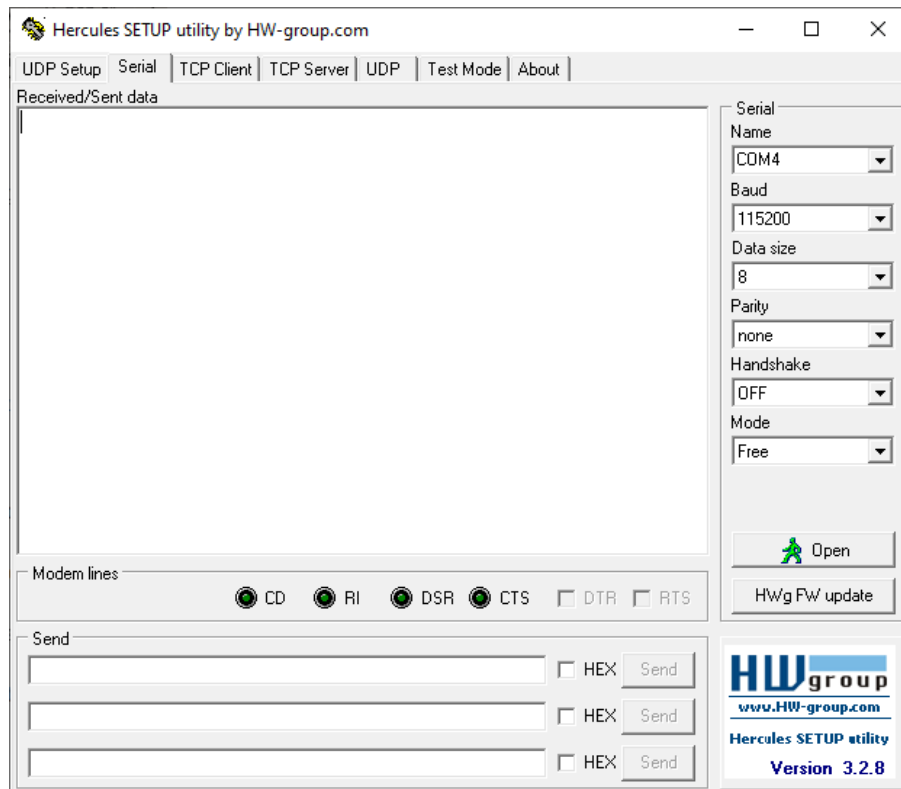
1. In the left selection bar of “Pinout & Configuration”, click on **USART2** and set the mode to **asynchronous**. Enable the **USART2 global interrupt** under **NVIC settings**.
2. In this lab, we going to receive data on UART using Direct Memory Access (DMA) mode (Refer **Week 8 Lecture A** for more information). Under **DMA Settings**, add **USART2\_RX** and set the mode to **Circular**, data width to **Byte** and **tick** the box for **Memory**.

### 7.2 UART DMA programming

1. Read the functions related to UART in the HAL document, find and understand the ones you could use to transmit data in polling mode (board to PC) and receive data in DMA mode (PC to board).
2. Define a global variable **buffer\_Tx** at an appropriate length to store the message you will transmit to your computer. Let the message be “**Please help <uni ID>**”.
3. Use the related UART functions from HAL library to **transmit** your words from **buffer\_Tx** to your computer **repeatedly**. Verify this through the serial port terminal (Refer Appendix A for setting the serial port console). You can add a delay to observe the message at an optimum speed.
4. Define a global variable **buffer\_Rx** at an appropriate length to store the words you will receive from your computer.
5. Use the related UART function from HAL library to **receive** the message from your computer to the Nucleo board in **buffer\_Rx** using DMA feature. Put it between `/* USER CODE BEGIN 2 */` and `/* USER CODE END 2 */`.
6. Use UART receiver **callback** function from HAL library and **set a flag** in this function to indicate that the data receiving has been completed. Set a **break point** inside this function to monitor the callback trigger.
7. Build and debug your code.
8. Add **buffer\_Rx** and **flag** as ‘Live Expression’ (Window menu -> Show View -> Live Expressions).
9. Click on Resume button  and demonstrate your UART communication in the open Serial port terminal.
  - The serial port terminal repeatedly displays the *help* message from you.
  - The tutor replies to this message from the serial port terminal
  - Show the tutor’s reply saved in the **buffer\_Rx**.
  - Show the trigger of callback function and change in flag value. Explain when this change happens.

## Appendix A: Hercules SETUP utility

Hercules SETUP utility is useful serial port terminal, UDP/IP terminal and TCP/IP Client Server terminal. After you set up the UART transmission between your board and PC, check out which **COM port** your board is connected (via **Device Manager**).



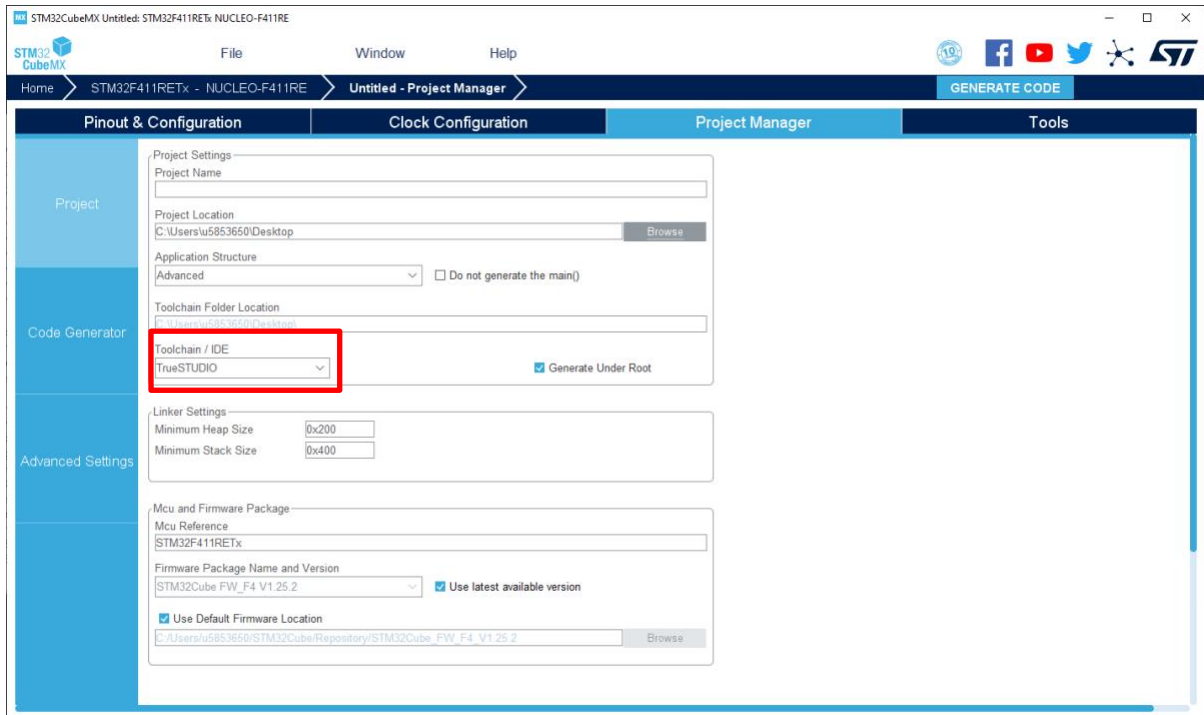
In the Hercules window,

- Set the **serial** port to the exact **COM port** your board is connected.
- Set the **Baud rate** as the same number as your USART configuration.
- Click **Open** to open the COM port and communication between the board and PC.

If you are transmitting data from the board to PC, you should see message in the big blank area. If you are receiving data (PC to board), simply type some message in the input bar and click **Send**. Any input bar among the three is fine.

## Appendix B: CubeMX & TrueStudio

If you are not using STM32CubeIDE, but using a combination of CubeMX and some other compilers (for example TrueSTUDIO), you can simply change the **Toolchain / IDE** in **Project Manager** tab (in your CubeMX).



Then, click on **GENERATE CODE** on the right top corner, you will have a project folder.

Open TrueStudio, go to File -> Open Projects from File System, import the folder from the location you just saved. Then you can do the same thing as using STM32CubeIDE.