



Title: Getting Started with STM32 using STM32CubeIDE and HAL

Prepared by:
PM Ir Dr Tee Kian Sek
Dr Chew Chang Choon
Prof. Ir. Dr. Soon Chin Fhong
FKEE, UTHM

1st Draft: April 2024
Rev 1: June 2025

*A short course to promote competency skills
in electronics and embedded systems*

Shared Documents
https://github.com/rtlab1417/STM32_intro_shared

1 Table of Contents

2	Disclaimer	1
3	Reminder.....	1
3.1	GitHub – Online Shared Documents	1
3.2	Preparation Before Short Course	1
3.3	The Training Kit.....	2
4	About the instructors.....	5
5	A Brief on ARM.....	6
5.1	ARM architecture family	6
5.2	Design and Licensing.....	6
5.3	Why popular?.....	6
5.4	Market Share in Processor	7
6	Family of STM32 MCU.....	8
6.1	STM32 MCU Naming Convention.....	9
6.2	ST Reference.....	10
6.3	Competitors.....	10
7	Introduction To The Short Course.....	11
7.1	Key Elements (To Highlight).....	11
7.2	List of Market Parts	12
8	Knowing The Development Board	12
8.1	Schematic.....	13
8.2	Boot Configuration	15
8.3	The MCU – STM32F103C8T6.....	15
8.4	Pin Definitions	15
8.5	Device Features And Peripherals	16
8.6	ST Reference.....	17
9	Programming Software	18
9.1	STM32 – Hardware Abstract Layer (HAL)	18
9.2	C Language	18
10	Installing STM32CubeIDE	20
10.1	First Glance.....	20
10.2	ST reference	21
10.3	Important skills	21
10.4	STM32CubeIDE – Shortcut Keys	22
11	First Project - Create A New Project.....	23
11.1	Create A New Project.....	23

11.2	File Structure.....	25
11.3	Build – main.c.....	25
11.4	Pinout & Configuration.....	26
11.5	Run – Upload To MCU	27
11.6	Re-configure Pinout	28
12	Practices	30
12.1	GPIO- LED	31
12.1.1	Diagram.....	31
12.1.2	Practices	32
12.2	GPIO- LEDs Blink.....	34
12.2.1	Diagram.....	34
12.2.2	Practices	35
12.3	GPIO- LED-Buttons	37
12.3.1	Diagram.....	37
12.3.2	Practices	38
12.4	GPIO- EXTI.....	41
12.4.1	Diagram.....	41
12.4.2	Practices	41
12.5	UART in Polling Mode.....	44
12.5.1	Diagram.....	44
12.5.2	Practices	46
12.6	UART With Interrupt	49
12.6.1	Diagram.....	49
12.6.2	Practices	53
12.7	UART With DMA.....	56
12.7.1	Diagram.....	56
12.7.2	Practices	59
12.7.3	References on UART	62
12.7.4	Discussion on UART – Polling Mode, Interrupt, DMA	62
12.8	UART With BLE_DMA	63
12.8.1	Diagram.....	63
12.8.2	Practices	65
12.9	IIC_AHT20.....	68
12.9.1	Diagram.....	68
12.9.2	Practices	69
12.10	IIC_OLED.....	71
12.10.1	Diagram.....	71

12.10.2	Practices	71
13	Tools.....	74
13.1	Serial App – CoolTerm	74
13.1.1	Self-loop Test With UART-to-USB	75
13.1.2	Self-test With Two Units of UART-to-USB.....	75
13.2	Flashing Using UART Port.....	76
13.2.1	Procedures to Write.....	77
13.2.2	Procedures to Read/Read ChipInfo.....	79
13.3	ST Tools	79
14	STM32 ST-LINK Utility.....	80
14.1	Write HEX / Bin File to MCU	80
14.2	Read HEX / BIN File and Erase Full Chip	81
14.3	Why not STMCubeProgrammer?	81
15	STM32CubeIDE – Build binary, hex files.....	82
15.1	Option Setting at the IDE.....	82
15.2	How to use the binary file and the hex file?	84
16	Version Control	84

2 Disclaimer

The instructors do not sell market parts. Each participant should make online purchases, as prices may vary.

3 Reminder

1. Before attending the short course, please install STM32CubeIDE on your personal computer. Refer to the installation guide for instructions. See **Installing STM32CubeIDE**.
2. The training kit, including MCU, ST-Link V2, parts, and accessories, is **NOT INCLUDED** in the training fee. You will need to purchase the market parts yourself as listed in [\[3.3 The Training Kit\]](#).

3.1 GitHub – Online Shared Documents

https://github.com/rtlab1417/STM32_intro_shared

Note: The content will be updated from time to time.

3.2 Preparation Before Short Course

Know-How:





(Having this fundamental knowledge or experience is beneficial. You can still proceed with the short course even without it..)






1. Experience in programming microcontrollers using Arduino
2. Basic knowledge of C programming
3. Understanding of the STM32 microcontroller family


Software:

1. STM32CubeIDE
 - a. Download from the ST website
 - b. For the downloading and installation guide, refer to the steps explained in [\[10 Installing STM32CubeIDE\]](#)
2. Serial Communication app
 - a. Any free download apps available online
 - b. Some tools are recommended by the instructors. See [\[13 Tools\]](#)
 - c. Share tools by the instructors. See the shared folder. See [\[3.1 GitHub – Online Shared Documents\]](#)

3.3 The Training Kit

No	Item	Qty	Unit Price (RM)	Amount (RM)	Images	Comment
1	MCU Training Board STM32F103C8T6, Micro USB a.k.a: Bluepill	1	6.00	6.00		If your budget allows, consider purchasing a few sets.
2	ST-LINK V2	1	8.00	8.00		If your budget allows, consider purchasing two sets
3	micro USB to USB cable Programming cable (normal Android data cable would work)	1	2.00	2.00		Use a handphone charger cable
4	LED (5mm) Mix colour (Red, yellow, Blue, Green) (10 pcs per colour - purchase online)	6	1.20	7.20		You don't need to buy if you already have it.
5	Resistor 0.25 W, any value from 220~470 Ohm (in pack of 10 pcs)	1	1.00	1.00		You don't need to buy if you already have it.

6	Momentary pushbutton, 4 pins	2	1.00	2.00		You don't need to buy if you already have it.
7	breadboard, MB102	1	6.00	6.00		You don't need to buy if you already have it.
8	Male to Male 20mm length, (normally in bundle of 40pcs or less) Dupont Jumper Wire	1	4.00	4.00		You don't need to buy if you already have it.
9	Male to Female 20mm length, (normally in bundle of 40pcs or less) Dupont Jumper Wire	1	4.00	4.00		You don't need to buy if you already have it.
10	USB to Uart Converter CH340 or CP2102 (either one would work)	1	6.00	6.00		If your budget allows, consider purchasing a few sets. Many breakout boards are available online, such as CP2102, PL2303, CH341, CH9102, FTDI, etc. They work the same. Some may need a driver. All drivers are available online.
12	OLED 0.96", I2C 128x64	1	6.00	6.00		If your budget allows, consider purchasing a few sets. You might need to solder the header pins.
13	A laptop OS - Window 10 or 11	1				Your own personal computer. Sufficient storage

14	Bluetooth Module, HC05	1	16.00	16.00		If your budget allows, consider purchasing at least one pair
----	------------------------	---	-------	-------	---	--

Note:

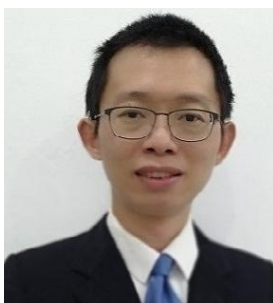
The market parts will be updated and added as the tutorial progresses.

4 About the instructors



PM Ir Dr Tee Kian Sek

<https://community.uthm.edu.my/staff/people/tee>



Dr Chew Chang Choon

<https://community.uthm.edu.my/chewcc>



Prof. Ir. Dr. Soon Chin Phong

<https://community.uthm.edu.my/soon>

5 A Brief on ARM

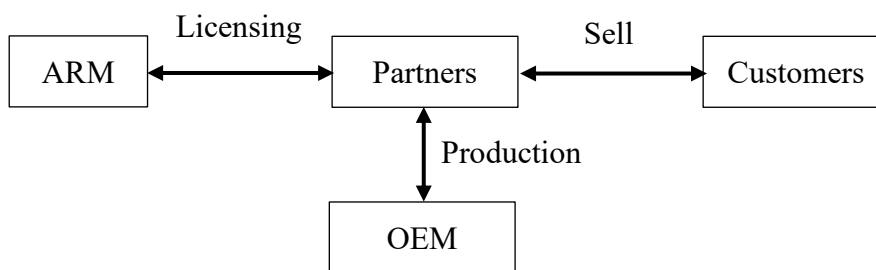
You can skip the background information and go straight to the practices at [\[11 First Project - Create A New Project\]](#) and [\[12 Practices\]](#)!

5.1 ARM architecture family

https://en.wikipedia.org/wiki/ARM_architecture_family

ARMv1, ARMv2, ARMv3... ARMv9

5.2 Design and Licensing



Partners:

1. Broadcom
2. Apple
3. ST
4. ARM licenses IP to over 1,000 global partners (including Samsung, Apple, Microsoft).

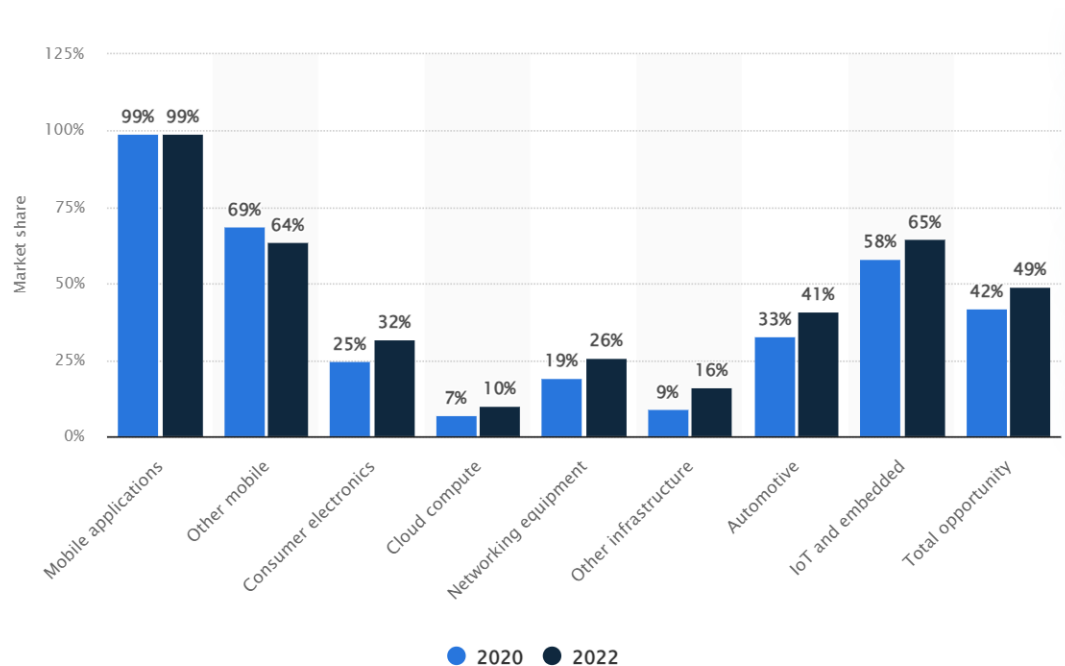
<https://www.arm.com/partners/catalog/results#sort=date%20descending&numberOfResults=12>

5.3 Why popular?

1. Low energy consumption, low cost, high performance
2. Support 16/32 instruction sets
3. Many partners
4. Rich ecosystem
5. Many more reasons...

https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers.html

5.4 Market Share in Processor



Source: <https://www.statista.com/statistics/1132112/arm-market-share-targets/>

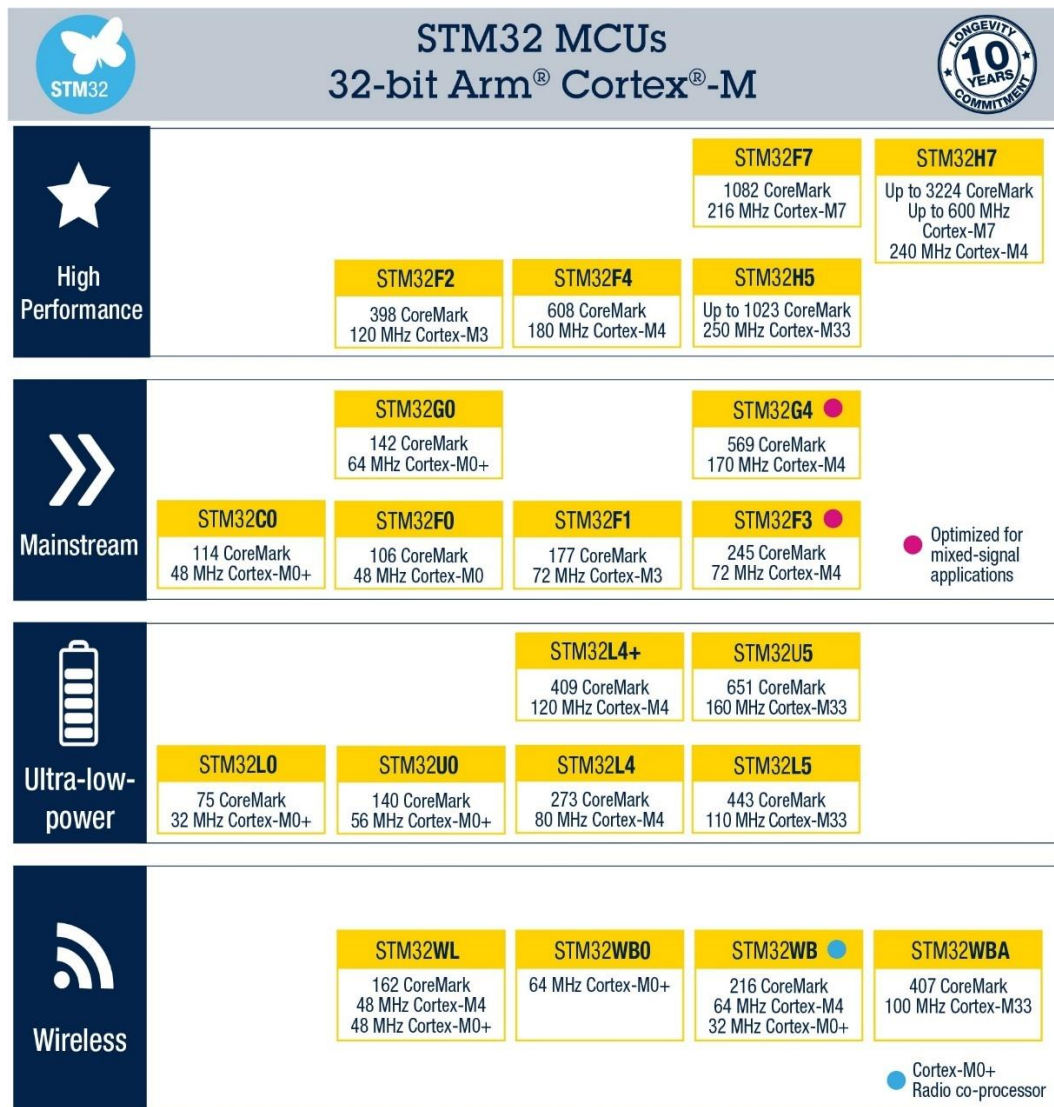
As of 2022, over 230 billion ARM chips have been produced, making ARM the most widely used family of instruction set architectures.

Please update the latest info online!

6 Family of STM32 MCU

You may skip this section and directly jump to the practices at [\[12 Practices\]](#).

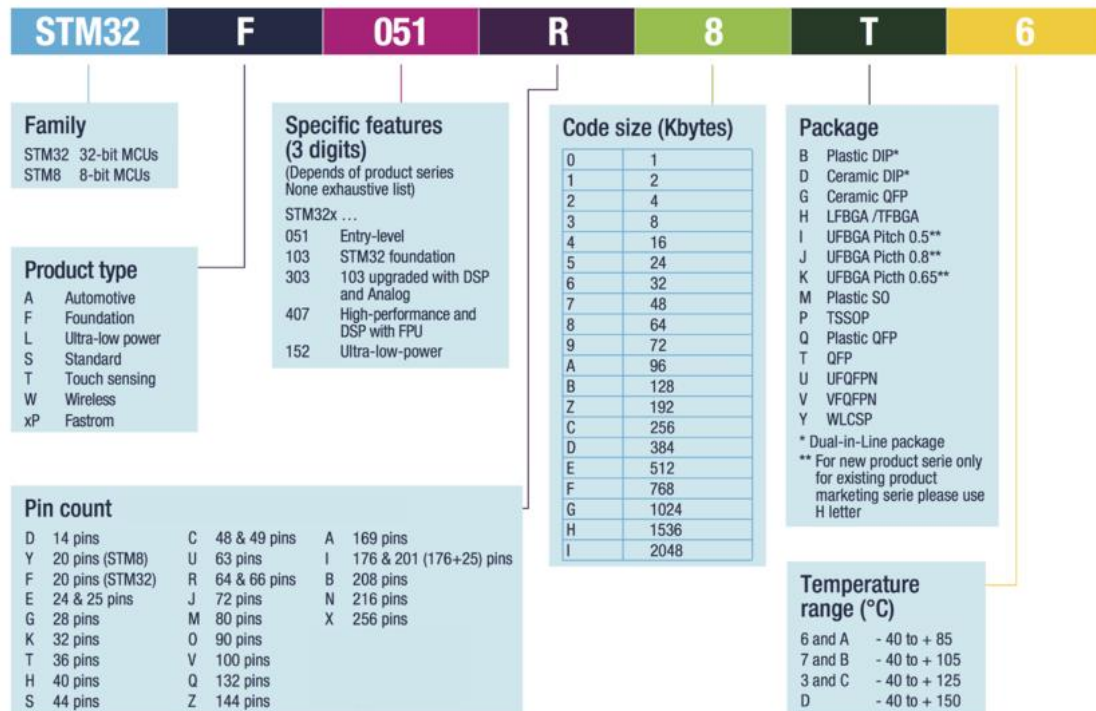
ST offers a variety of products for various industrial applications, including microcontrollers. Please visit the ST website for more information.



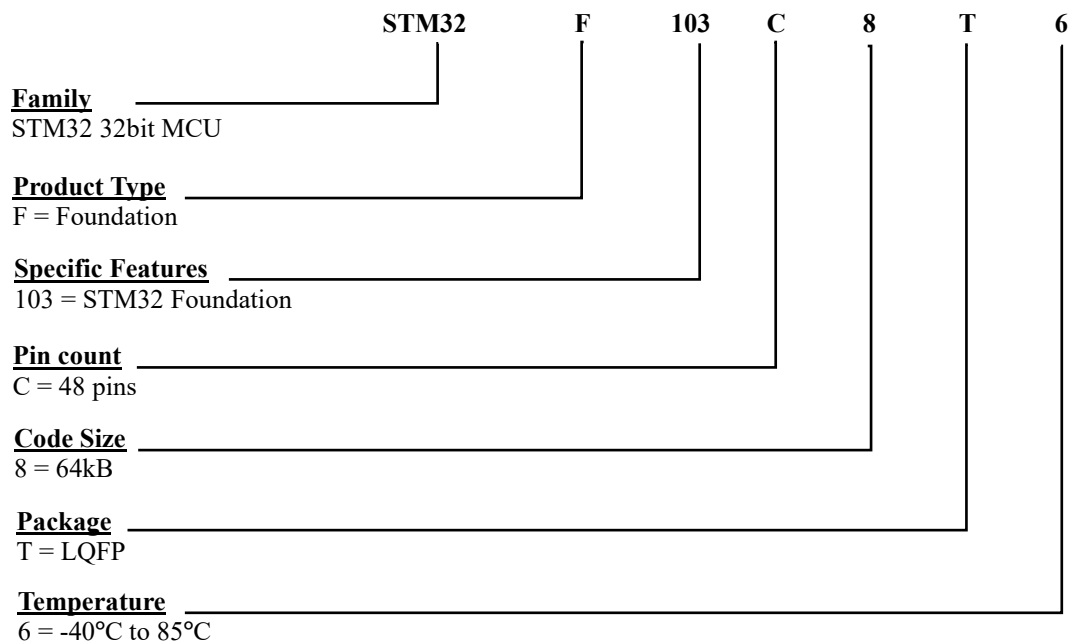
Source: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>

- STM32 is a 32-bit microcontroller developed by ST Company based on the ARM Cortex-M core (M0, M0+, M3, M4, M7, M33)
- STM32 is often used in embedded fields, such as smart cars, unmanned machines, robots, wireless communications, internet of things, industrial control, entertainment electronics, etc.
- STM32 is a powerful, excellent-performing, rich in resources, low-power-consumption, classic embedded microcontroller.

6.1 STM32 MCU Naming Convention



For example, STM32F103C8T6 (it is the MCU deployed in this training!)



6.2 ST Reference

Learners are advised to consult official documentation such as datasheets, instruction manuals, and notes provided by ST in PDF format. You may click the link and search for PDF:

<https://www.st.com/en/microcontrollers-microprocessors/stm32f103/documentation.html>

The MCU is STM32F103C8T6 (We recommended a datasheet and a reference manual.)

1. Datasheet
 - a. Medium-density performance line Arm
 - b. STM32F103x8, STM32F103xB
2. Reference Manual - RM0008, for STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs
3. Others.

More readings:

Comprehensive explanation –

<https://stm32-base.org/guides/getting-started.html>

https://stm32world.com/wiki/Blue_Pill

<https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>

Highlighted information:

1. Based on ARM architecture.
2. Feature
3. Provided STM - https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/boards-and-hardware-tools.html
4. Provided in market
5. Programming Language – C language

6.3 Competitors

Key players in the semiconductor industry.

1. Microchips (Atmel was acquired in 2016)
2. NXP Semiconductors
3. Infineon
4. Renesas Electronics
5. Texas Instruments
6. Many more...

7 Introduction To The Short Course

You may skip this section and directly jump to the practices at [12 Practices].

To start, we need to make sure we have everything ready. This includes:

1. **Hardware:** This includes the Microcontroller Unit (MCU), a programmer, and peripheral devices as the minimum system requirements.
2. **Software:** An Integrated Development Environment (IDE) is necessary for development.
3. **DC Power Source:** You can use 5VDC or a 3.3V DC source.
 - a. For most projects, we use the 3.3V from the MCU.
 - b. If a 5VDC source is required, connect to an external adapter providing 5VDC 2A.
4. **Breadboard and Jumper Wires:** These are essential for making connections.
5. **Personal Computer:** Essential for programming and development tasks. Our demonstration is based on Windows 11. If you are using macOS, it should be similar.
6. **Sensors and Actuators:** Refer to the relevant practices for specific components.

7.1 Key Elements (To Highlight)

1. STM32 MCU and development board
 - a. Recommended here – STM32F103C8T6
 - b. Do not confuse with STMF03C6T6
2. Programmer
 - a. Recommended here – ST-Link V2
3. Software – Recommended here - STM32CubeIDE
 - a. Download for free
 - b. https://www.st.com/content/st_com/en/stm32cubeide.html
 - c. Install – See next.



7.2 List of Market Parts

Refer to [\[3.3 The Training Kit\]](#)

Note:

1. All market components that you purchase belong to you as participants.
2. These components are currently available for purchase in the market.
3. Prices may vary based on individual purchases.
4. If your budget allows, consider buying spare units such as the MCU, USB to UART converter, and OLED display.
5. While it's not required, it would be very helpful to equip your toolbox with a digital multimeter, wire cutters, screwdrivers, tweezers, and other tools, all within your personal budget..

8 Knowing The Development Board

You may skip this section and directly jump to the practices at [\[12 Practices\]](#).

The development board is a minimal system featuring the STM32F103C8T6 microcontroller, commonly referred to as "BluePill" in the market.

STM32F103 microcontrollers are built on the Cortex-M3 core and can operate at a maximum CPU speed of 72 MHz. The range of these microcontrollers includes variants with Flash memory from 16 Kbytes to 1 Mbyte, and they also offer motor control peripherals, a USB full-speed interface, and CAN support.

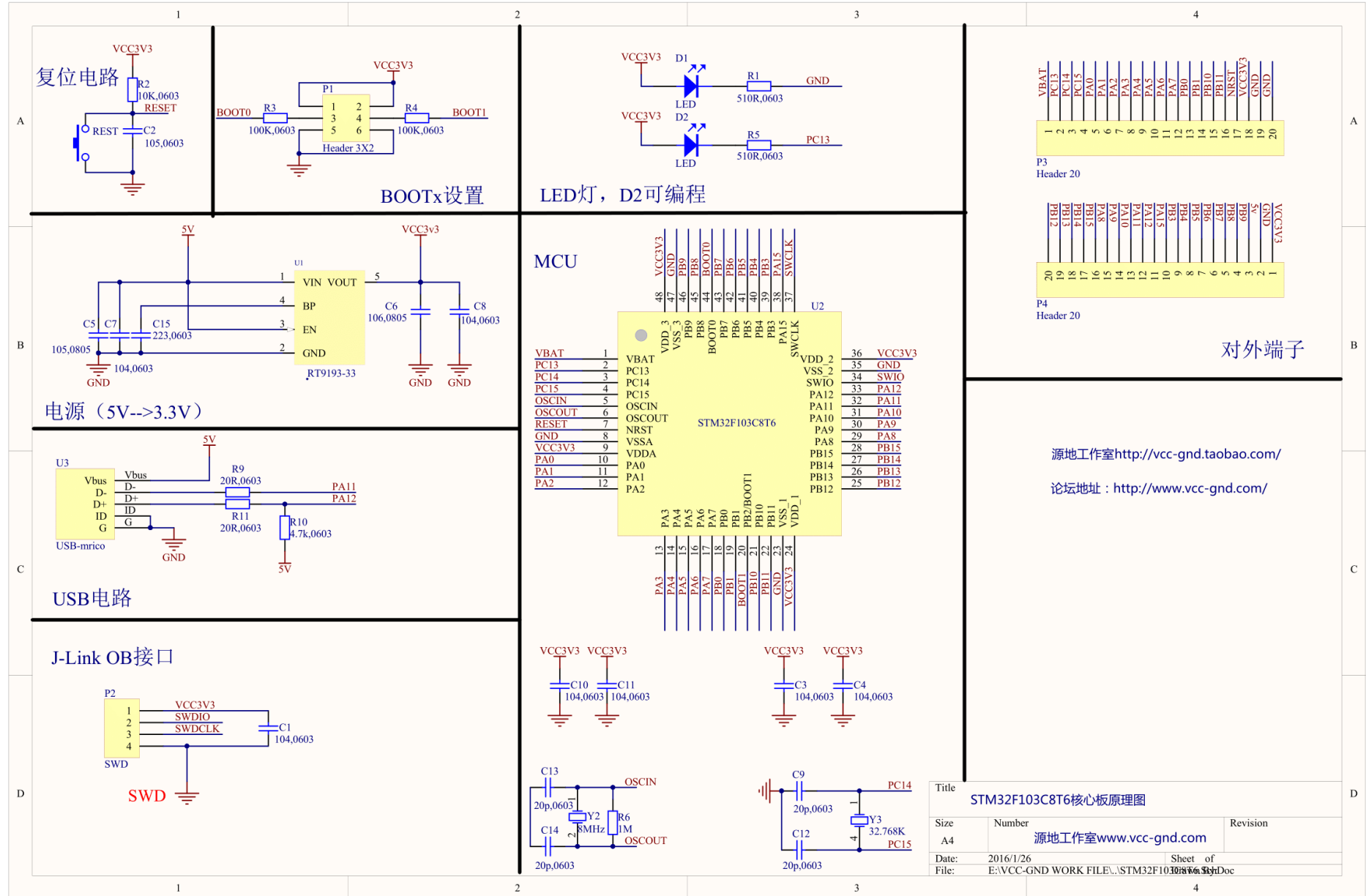


Source: <https://www.st.com/en/microcontrollers-microprocessors/stm32f103.html>

8.1 Schematic

See attachment. *Note: Attached in GitHub*

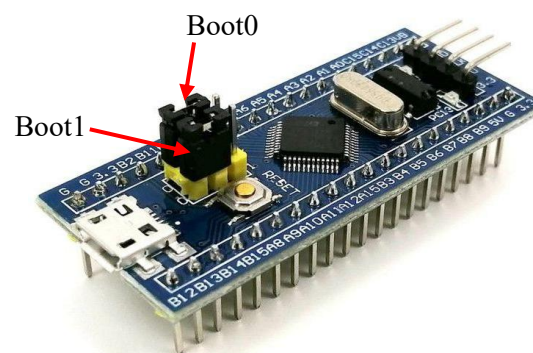
[original-schematic-STM32F103C8T6-Blue_Pill.pdf](#)



8.2 Boot Configuration

Table 9. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space



8.3 The MCU – STM32F103C8T6

1. Family: STM32F1
2. Core: Arm® 32-bit Cortex®-M3 CPU core
3. Frequency: Max. 72MHz
4. Memory: 20K (SRAM) , 64K (Flash)
5. Supply: 2.0~3.6V (Standard 3.3V)
6. Package: LQFP48

Refer to the official datasheet (see attachment)

STM32F103x8 / STM32F103xB - Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. Interfaces

8.4 Pin Definitions

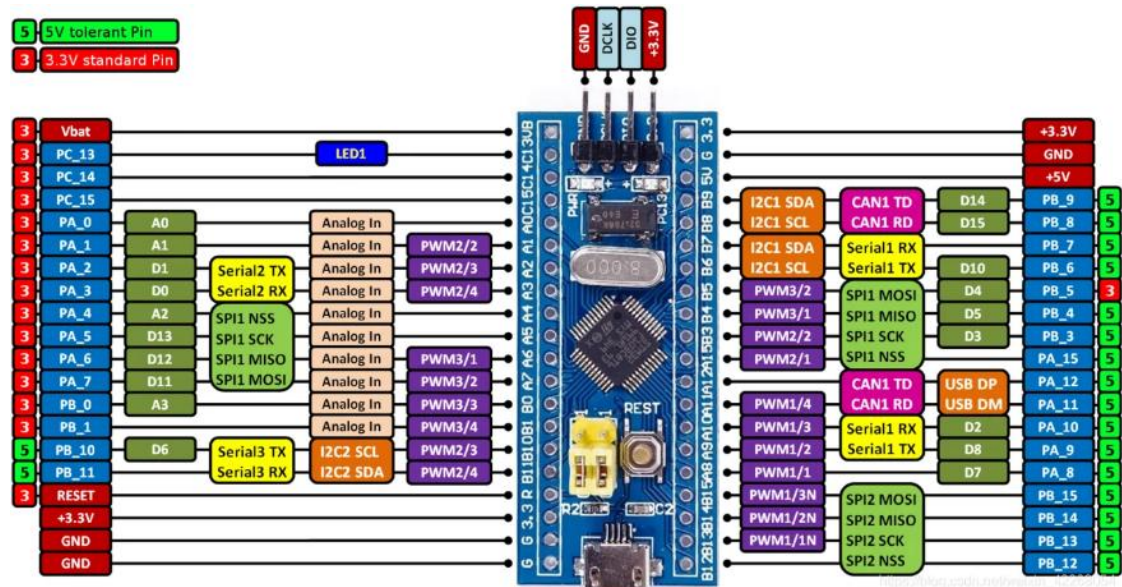
Alternatively, re-arrange in an Excel spreadsheet.

See attachment. *Note: Attached in GitHub*

[*STM32F103C8T6 -pin definition.xlsx*](#)

We shall examine this document often.

Alternatively, we could refer to this pin assignment below for quick review and assignment.

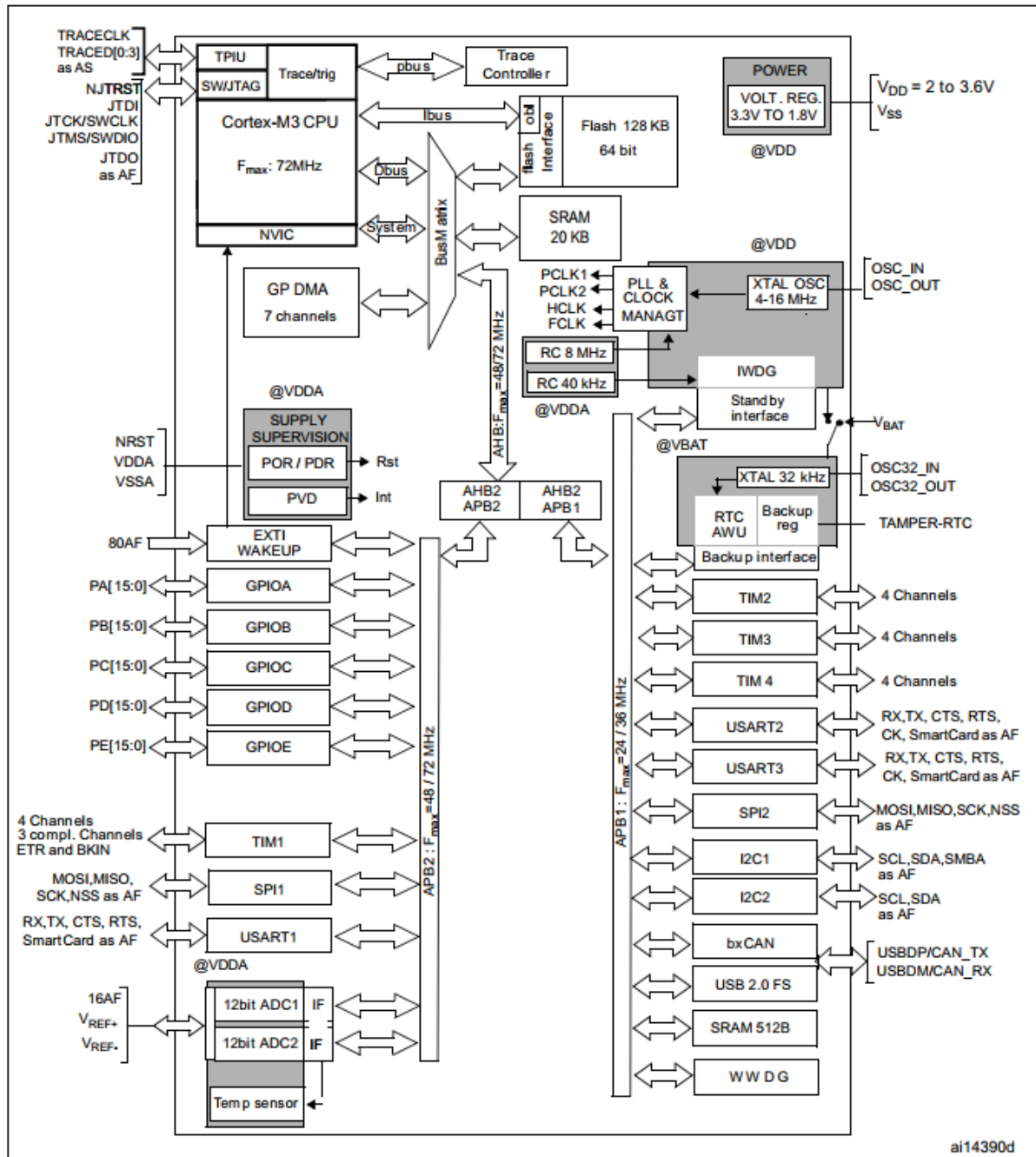


Source: <https://reversepcb.com/stm32f103c8t6/>

8.5 Device Features And Peripherals

Term	Description	Term	Description
NVIC	Nested Vectored Interrupt Controller	CAN	CAN Comm.
SysTick	the Cortex® System Timer	USB	USB Comm.
RCC	Reset and Clock Control	RTC	Real-time clock
GPIO	General-purpose I/O	CRC	Cyclic Redundancy Check
AFIO	Alternate-function I/O	PWR	Power Control
EXTI	External interrupt/event controller	BKP	Backup registers
TIM	Timer	IWDG	Independent watchdog
ADC	Analog-to-Digital Converter	WWDG	Window watchdog
DMA	Direct memory access	DAC	Digital-to-analog converter
USART	USART Comm.	SDIO	SD Interface
I2C	I2C Comm.	FSMC	Flexible static memory controller
SPI	SPI Comm.	USB OTG	USB

Refer to the manual for a detailed description. Don't worry if these terms seem intimidating.



(Source: Datasheet)

8.6 ST Reference

At ST's official website, search for: STM32F103

<https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html#documentation>

1. Datasheet
2. Reference Manuals

9 Programming Software

You may skip this section and directly jump to the practices at [\[12 Practices\]](#).

A few Integrated Development Environments (IDEs) are available in the market to code STM32 and its family. Some of them are paid services, yet some are open-source. Among them:

1. **STM32CubeIDE – official IDE by ST - free - deployed in this short course.**
2. Keil MDK - paid service
3. IAR Embedded Workbench – paid service
4. Arduino IDE - free
5. PlatformIO - free
6. Matlab - Hardware support needed.
7. Etc.

9.1 STM32 – Hardware Abstract Layer (HAL)

To code an MCU, there are a few options:

1. Bare metal programming
 - a. Configure the registers directly and manually
 - b. Good for understanding the MCU. It might be intimidating for a beginner.
2. Standard peripheral library
 - a. Provided by ST
 - b. Many API could be shared across the MCU family.
3. **Hardware Abstract Layer (HAL)**
 - a. **ST provides HAL for its MCU family.**
 - b. **This is implemented in this short course.**

The STM32 Hardware Abstraction Layer (HAL) provides APIs to connect with user applications, libraries, and stacks, making it easier to code embedded systems. HAL will make development faster because it's easy to use. This is especially true for first prototypes or examples, where you do not need extensive tests, but you need something to show quickly. HAL usually makes code easily portable within the same brand/manufacturer.

Further readings:

1. <https://www.linkedin.com/pulse/bare-metal-vs-hal-unleashing-power-embedded-systems-daniel-oluwole/>
2. <https://embeddedthere.com/understanding-stm32-hal-library-fundamentals/>

9.2 C Language

STM32CubeIDE deploys C language for HAL and coding. A basic understanding of the C programming language can be very beneficial.

Some C elements:

Variables, Data type, Operators, Loops, Struct, Pointer, Function, type cast, etc.

Note: C is not equal to C++; however, C could be implemented in C++.
Note: This short course should not and will not deliver C programming.

10 Installing STM32CubeIDE

You may skip this section if you have downloaded and installed the IDE, and directly jump to the practices at [12 Practices].

No worries, it should be a typical installation process.

1. It is free and can be downloaded from (or Google it!)
https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/software-development-tools.html
2. Choose the Windows version (I use Windows)
3. Follow the instructions. Register your name and email. Once registered, check your email. Click the link to download the software.
4. Unzip and then install STM32CubeIDE.
5. Official guide by STM32
 - a. Getting Start video – by STM32 website
 - b. https://www.st.com/content/st_com/en/stm32cubeide.html

Friendly suggestion:

1. You could click ‘ok’ and accept all options along the installation.
2. Alternatively, when being prompted to enter the local folder for workspace, Suggestion: create a specific folder in your computer as the workspace for all projects.

Example:

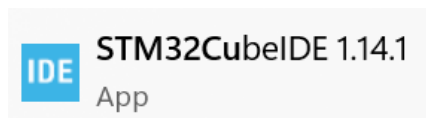
D:\STM32CubeIDE\workspace\

Because C-drive is for apps and os. D drive is my working drive.

10.1 First Glance

We will explore this IDE in this training.

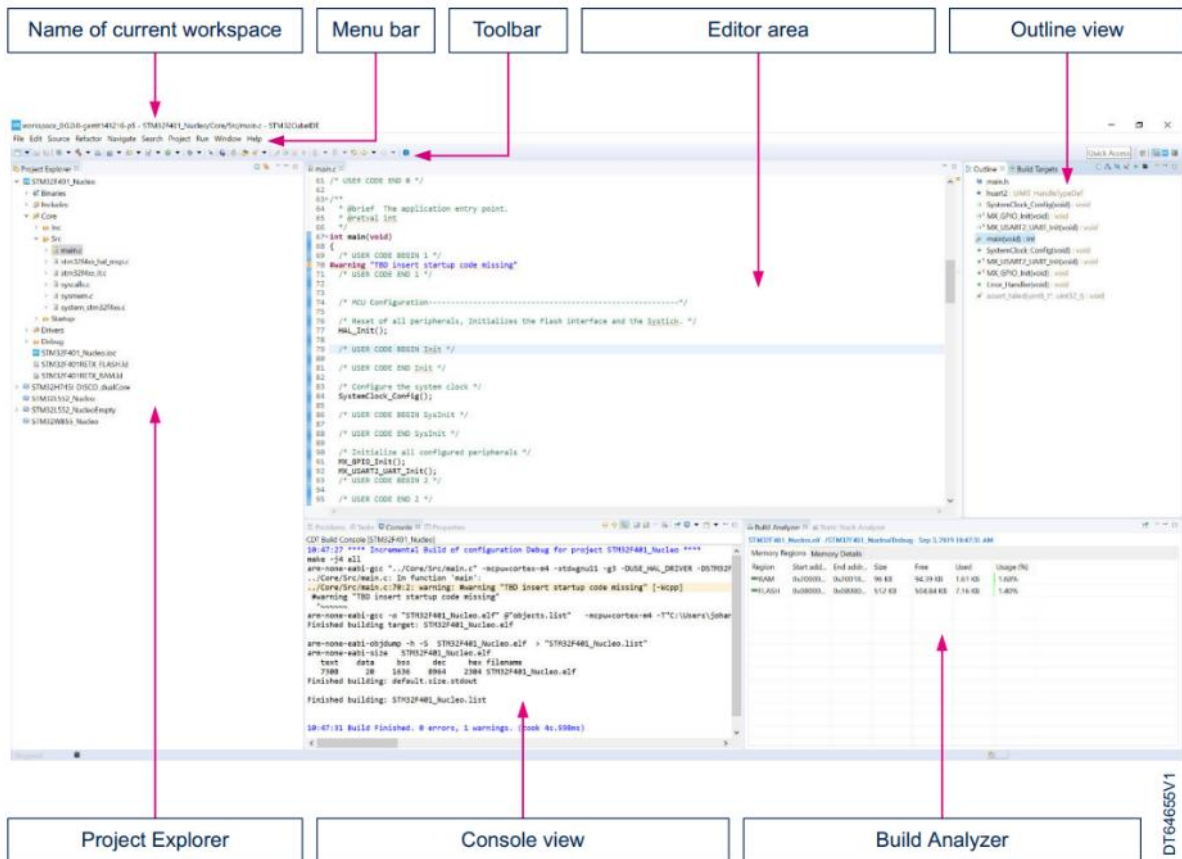
Once installed, search for “STM32CubeIDE 1.14.1”. (Your version might be different!). Click and run.



Note:

When you install the latest version, you might notice that the version number is a bit different from this tutorial. No worries, that's all part of the update!

The IDE should run similarly to the image below. We shall go through the IDE as we practice.



Source: UM2609 – User Manual - STM32CubeIDE user guide

10.2 ST reference

You may skip this section and directly jump to the practices at [\[12 Practices\]](#).

<https://wiki.st.com/stm32mcu/wiki/Category:STM32CubeIDE>

No worry about it. We would know the operations as soon as we create a new project.

10.3 Important skills

You may ignore this section and directly jump to the practices at [\[12 Practices\]](#). We could jump into this section when we need these skills.

Every often, a program will do these:

1. Clone a project – either as the progression of coding or importing from a shared project.

See the link:

<https://community.st.com/t5/stm32-mcus/how-to-clone-a-dual-core-project-in-stm32cubeide/ta-p/619747>

10.4 STM32CubeIDE – Shortcut Keys

You may skip this section and directly jump to the practices at [\[12 Practices\]](#).

Coding becomes much more efficient if some frequent commands are repeated. A few of them are:

CTRL+SHIFT+L	List keyboard shortcuts List all defined keyboard shortcuts
CTRL+Left Mouse or F3	Open Declaration
CTRL+/ CTRL+SPACE	comment or uncomment // Code Completion Code completion/parameter hints depending on context Parameter hints
F11	Debug project

Search in STM32CubeIDE for a complete list of shortcut keys. Or simply google it.

https://www.st.com/resource/en/application_note/atollic_editing_keyboard_shortcuts-atollic-editing-keyboard-shortcuts-stmicroelectronics.pdf

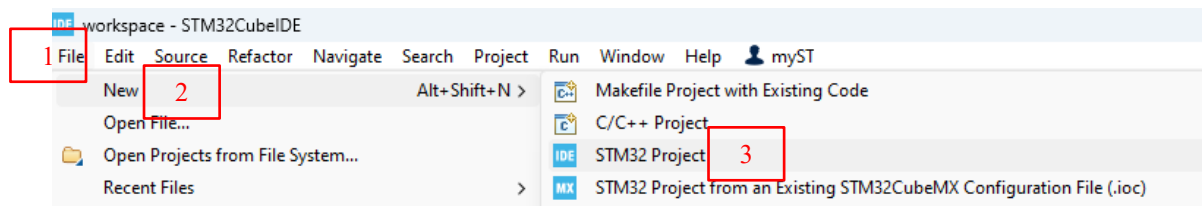
11 First Project - Create A New Project

Following the instructor's demonstration would make it much easier! This process will be repeated for each project, helping you become accustomed to it.

These are the general steps for creating a new project. We would repeatedly create a new project with the following practices.

11.1 Create A New Project

Step 1. Create a new project. File | New | STM32 Project



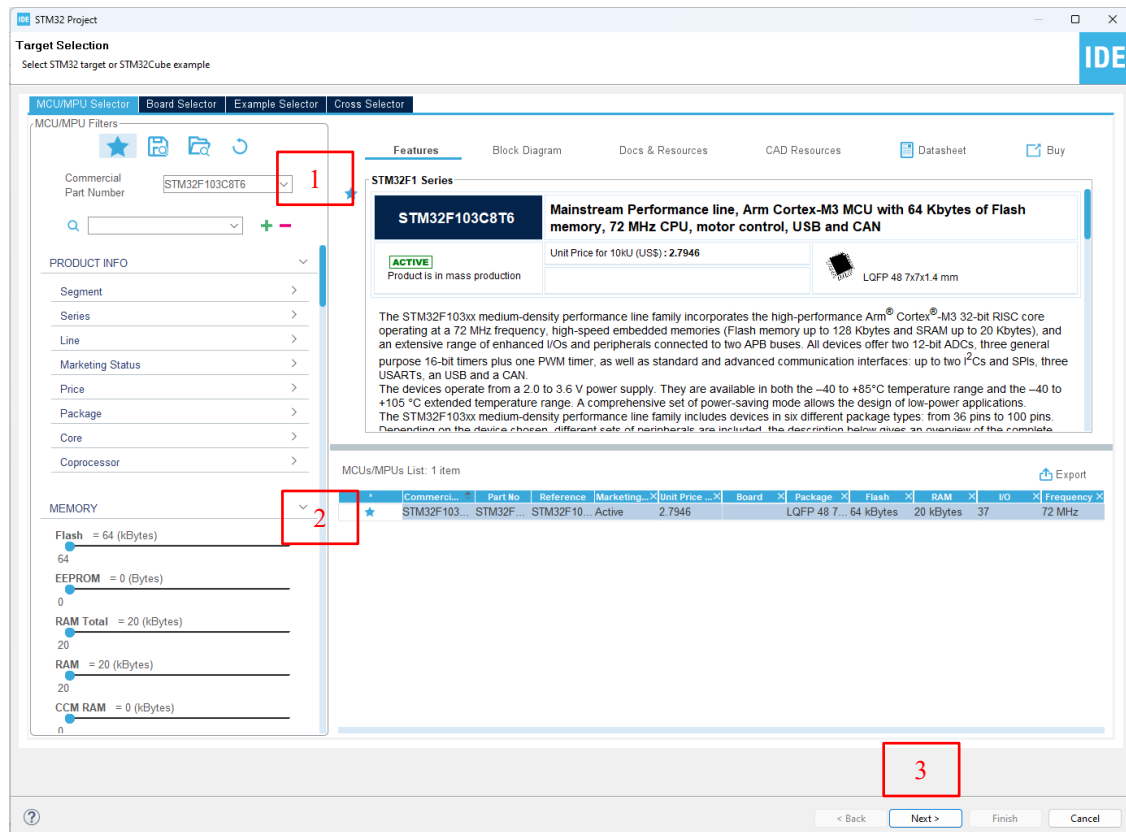
Step 2. Target Selection

[1] Type in Commercial Part number – STM32F103C8T6

[2] Select the part as listed. Click the star ☆. Save it as favorite. The star turn solid blue ★. The IDE would remember the favorite choice. If the same MCU is chosen for the following practice, just click the favorite.

[3] Click 'Next'

Note: For first-timer, spend sometime on the information shared.

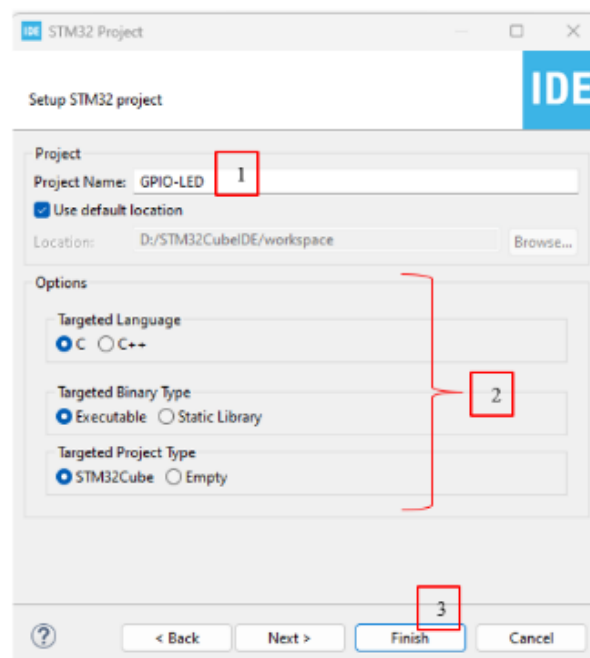


Step 3. Save to default location (workspace location)

[1] Type in Project Name – “GPIO-LED”

[2] Options. Do not change.

[3] Click Finish.



11.2 File Structure


The IDE would create a template for the selected MCU. Notice the listing in Project Explorer. A project is created.

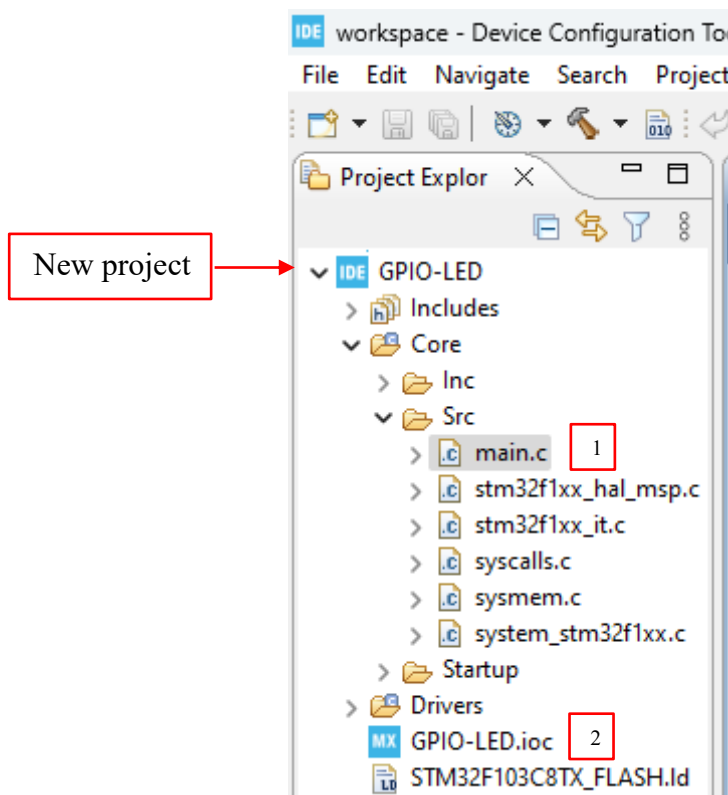
1. A folder is created and saved in the default location (You specified it during installation, but you could still choose another location).
2. Open File Explorer and the save location. Notice the folder structure. No worry! We will explore and use the standard C and HAL libraries as we explore further.
3. At the moment, we are concerned about two files

[1] `./Core/Src/main.c`

[2] `./GPIO-LED.ioc`

11.3 Build – main.c

1. Check the project and compiler
 - a. Do nothing on the new project.
 - b. Double click on `main.c`. It should pop up in the editor.
 - c. Build the code.
 - d. Click  or choose from the menu Project | Build All
 - e. The compiled results should be printed on the console window.
Example: Build Finished. 0 errors, 0 warnings. (took 209ms)
 - f. The template is working. Ready for further work!



```

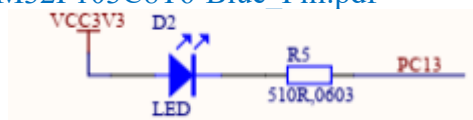
CDT Build Console [GPIO-LED]
15:29:39 **** Incremental Build of configuration Debug for project GPIO-LED ****
make -j16 all
arm-none-eabi-size  GPIO-LED.elf
   text    data    bss     dec     hex filename
   3492     20   1572    5084    13dc GPIO-LED.elf
Finished building: default.size.stdout

15:29:39 Build Finished. 0 errors, 0 warnings. (took 209ms)

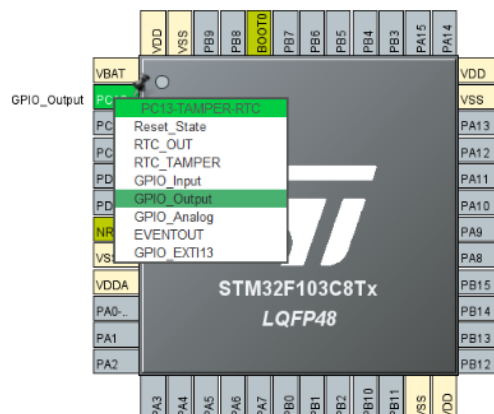
```


11.4 Pinout & Configuration

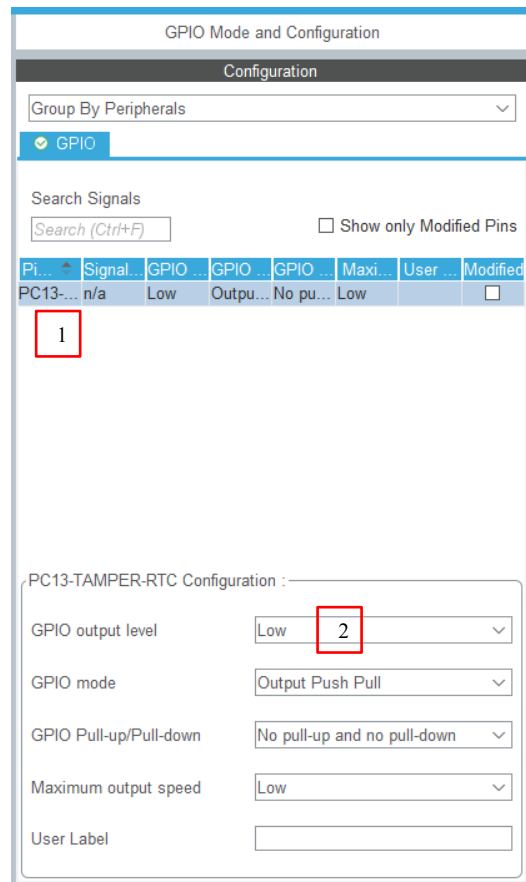
The development board has a built-in LED, at PC13, active LOW.
See: [original-schematic-STM32F103C8T6-Blue_Pill.pdf](#)




1. At the editor or Project Explorer, click Pinout & Configuration -- [GPIO-LED.ioc](#)
2. At the figure, click PC13 and set it as [GPIO_Ouput](#)

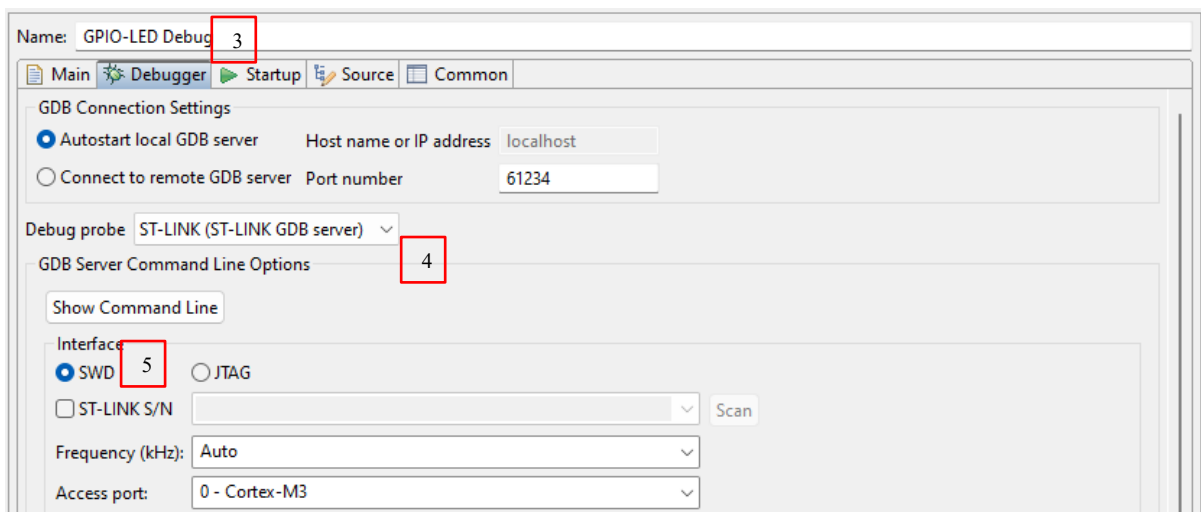
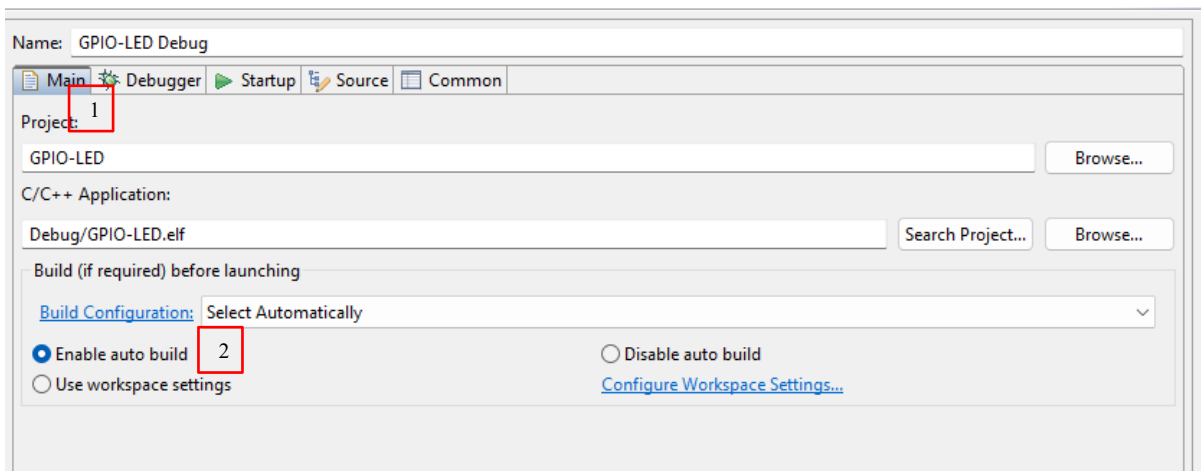


3. GPIO Mode and Configuration
 - [1] Click and select GPIO pin to configure
 - [2] GPIO output level – select **LOW**. (We expect LED is lit ON.)
Keep the rest unchanged.
4. Code Generation
 - a. Click the icon  or from the menu, Project | Generate Code
 - b. The IDE would convert the selected configuration into code (updated in main.c)
5. Click main.c, build the project.



11.5 Run – Upload To MCU

1. Click the Run icon,  or click Run | Run Configurations... for the first time.
2. Run Configuration (only one time configuration)
 - [1] Main Tab
 - [2] Select **Enable auto build**. This option would build the project before running the project.
 - [3] Debugger Tab
 - [4] At the dropdown menu, select **ST-Link**
 - [5] Select the interface as **SWD**. (Namely Serial Wire Debug)

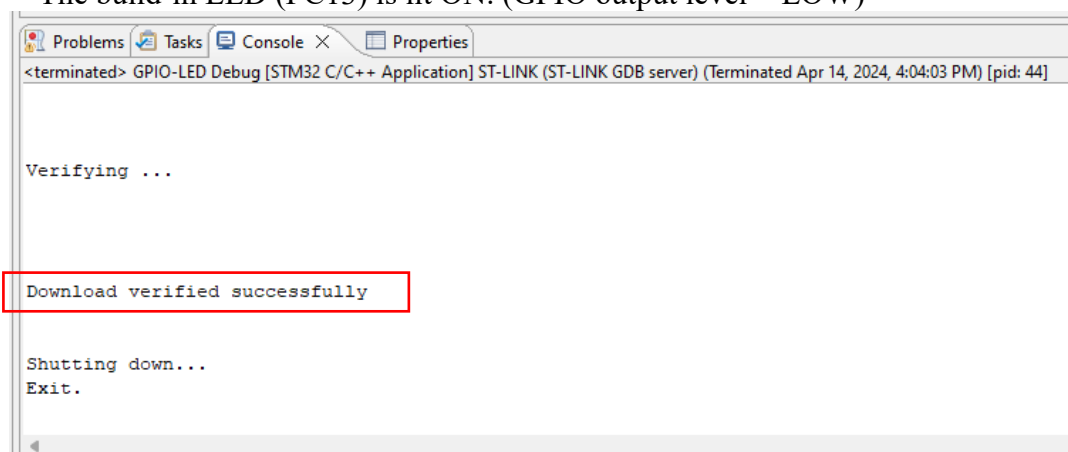


3. Make sure ST-Link V2 is plugged in and connected to the MCU.

4. Click **Run**. Download the project to the MCU.

Download verified successfully.

The build-in LED (PC13) is lit ON. (GPIO output level = LOW)



11.6 Re-configure Pinout

1. Repeat Pinout & Configuration, but this time, change the GPIO output level to **HIGH**

2. Compile and run.
3. The built-in LED (PC13) should be lit OFF. (GPIO output level = HIGH)

12 Practices

List of practices:

- 12.1 GPIO- LED**
- 12.2 GPIO- LEDs Blink**
- 12.3 GPIO- LED-Buttons**
- 12.4 GPIO- EXTI**
- 12.5 UART in Polling Mode**
- 12.6 UART With Interrupt**
- 12.7 UART With DMA**
- 12.8 UART With BLE_DMA**
- 12.9 IIC_AHT20**

And more projects would be included from time to time. Be patient and enjoy the practices.

12.1 GPIO- LED

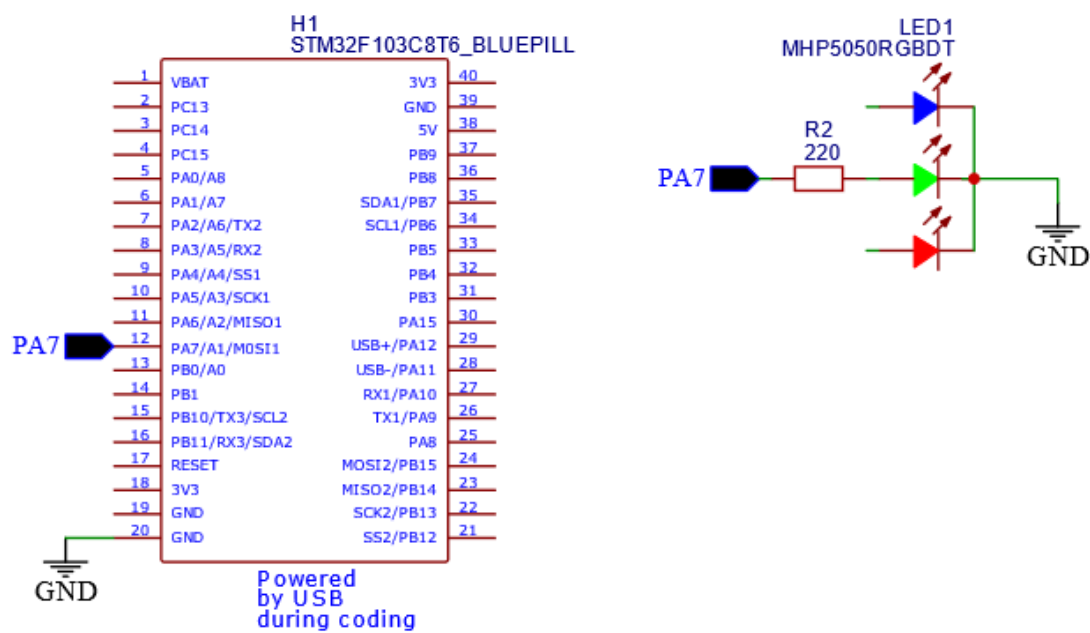
Requirement:

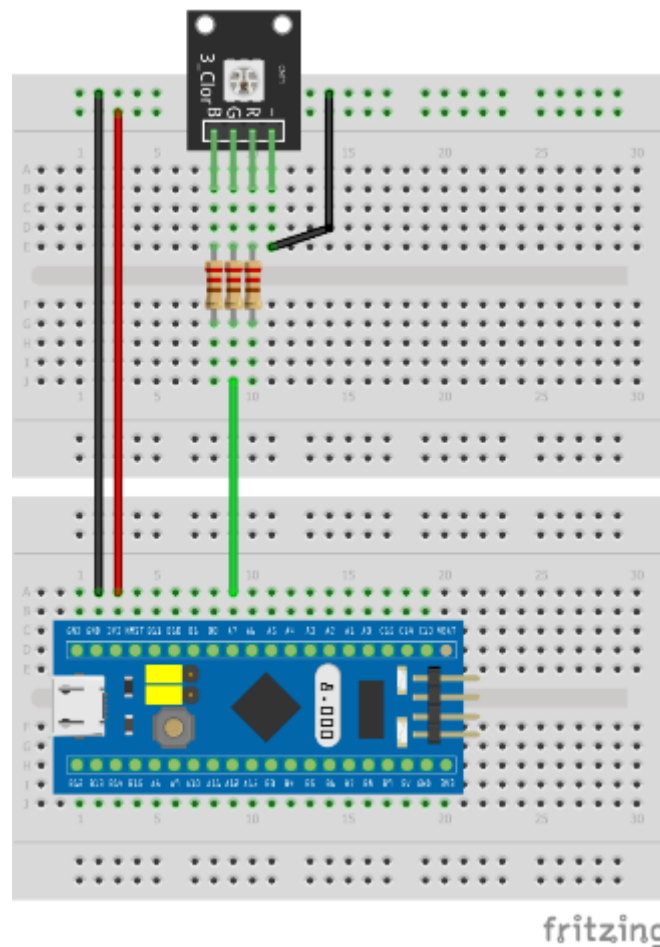
1. LED_GREEN blinks on specified time intervals.

12.1.1 Diagram

Pin	Mode	User Label
PA7	GPIO_Output, Low , Output Push Pull , No pull-up and no pull-down	LED_GREEN

- Pay attention of the direction of current flow. This determines GPIO output level, either active *LOW* or active *HIGH*.





Current limiting
resistors
220 ~ 470 Ω

12.1.2 Practices

1. Refer to the sample code.
2. Explore GPIO – try different configurations.
 - a. The initial GPIO is configured as [GPIO_Output, **Low**, **Output Push Pull**, No pull-up and no pull-down]. [Build and run](#)
 - b. Update GPIO to [GPIO_Output, **High**, **Output Push Pull**, No pull-up and no pull-down]. [Build and run](#)
 - c. Update GPIO to [GPIO_Output, **Low**, **Output Open Drain**, No pull-up and no pull-down]. [Build and run](#).
 - d. Update GPIO to [GPIO_Output, **High**, **Output Open Drain**, No pull-up and no pull-down]. [Build and run](#).
3. Exploring main.c (update GPIO as [GPIO_Output, **Low**, **Output Push Pull**, No pull-up and no pull-down])
 - a. Scroll by mouse or click the outline in Outline View
 - b. For the first time, observe the template (main.c)
 - i. header
 - ii. Function declarations in C
 - iii. Comments – you could try to comment/uncomment. Shortcut: CTRL+/
iv. `int main(void)`
 1. Initialization – default and user
 2. `while (1)`
 - c. `void MX_GPIO_Init(void)` - Generated template by STM32CubeIDE

1. struct and HAL

4. Exploring STM32CubeIDE. A quick search of the function used.
 - a. Within the same file. Practice: At main.c, hover the mouse cursor at `HAL_Init()`, right-click the mouse, and a window will pop up. Click [Open Declaration \(F3\)](#).
 - b. Source files and header files of STM32 HAL. Practice: At main.c, hover the mouse cursor at `HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, GPIO_PIN_SET);`, right-click the mouse; a window will pop up. Click [Open Declaration \(F3\)](#).
 - c. Spend some moments glaring at the source files. Observe HAL and its declaration.
5. Exploring STM32CubeIDE. Add user codes (Important! If you want to update configuration repetitively.)

```
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
LED_GREEN_Pin, GPIO_PIN_SET);
HAL_Delay(1000);
HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
LED_GREEN_Pin, GPIO_PIN_RESET);
HAL_Delay(1000);
```

Identify the coding area as commented in the template.

Example:

```
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */
```

- a. Add code within comment / before or after comment respectively in two attempts.
- b. Do update Pinout & Configuration and observe the difference.

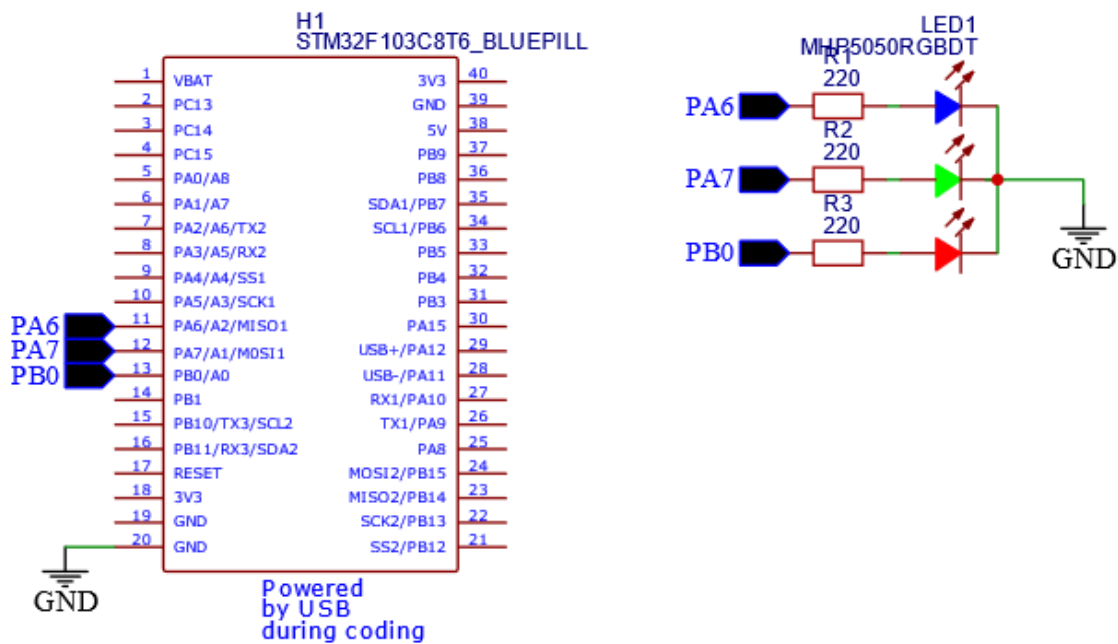
12.2 GPIO- LEDs Blink

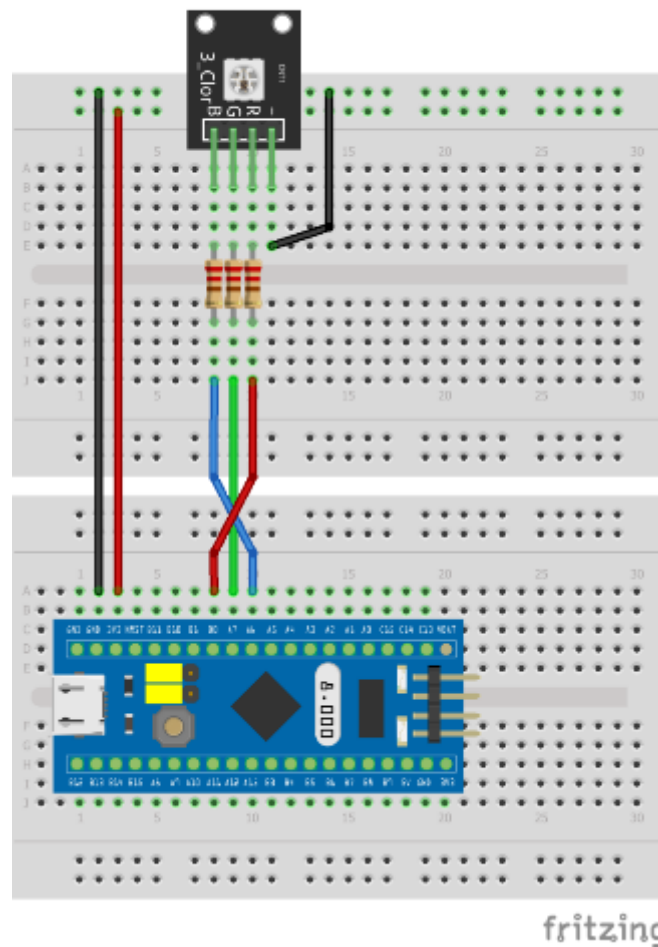
Requirement:

1. LED_BLUE, LED_GREEN, and LED_RED would be lit ON/OFF according to a pre-defined sequence.

12.2.1 Diagram

Pin	Mode	User Label
PA6	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_BLUE
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED





Current limiting
resistors
220 ~ 470 Ω

12.2.2 Practices

Let's say we want to mix the color repetitively using these sequences to update RGB led.

	State0	State1	State2	State3	State4	State5
LED_BLUE	0	0	1	1	1	1
LED_GREEN	0	1	1	1	0	0
LED_RED	1	1	1	1	0	1

- 1 – mean light ON
- 0 – mean light OFF

For each state count, it would take 1 second (Anyway, you could change it).

1. Update main.c, add in between the comment (after **while** (1))

```

/* USER CODE BEGIN 2 */
int state = 0;
/* USER CODE END 2 */
/* USER CODE BEGIN WHILE */
while (1)
{
    // red state
    if(state <=3 || state == 5){
        HAL_GPIO_WritePin(LED_RED_GPIO_Port,
        LED_RED_Pin,GPIO_PIN_SET);
    }else{
        HAL_GPIO_WritePin(LED_RED_GPIO_Port,
        LED_RED_Pin,GPIO_PIN_RESET);
    }
    // green state
    if(state >=1 && state <= 3){
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
        LED_GREEN_Pin,GPIO_PIN_SET);

        }else{
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
        LED_GREEN_Pin,GPIO_PIN_RESET);
        }

    // blue state
    if(state >=2&& state <= 5){
        HAL_GPIO_WritePin(LED_BLUE_GPIO_Port,
        LED_BLUE_Pin,GPIO_PIN_SET);
    }else{
        HAL_GPIO_WritePin(LED_BLUE_GPIO_Port,
        LED_BLUE_Pin,GPIO_PIN_RESET);
    }

    HAL_Delay(1000);

    state++;
    if(state>5){
        state = 0;
    }
}

```

2. Change `HAL_Delay(1000);` to shorter time. Say: 0.1 sec
3. Design your own sequence. Then update the code.
4. **Refer to the sample code.**

12.3 GPIO- LED-Buttons

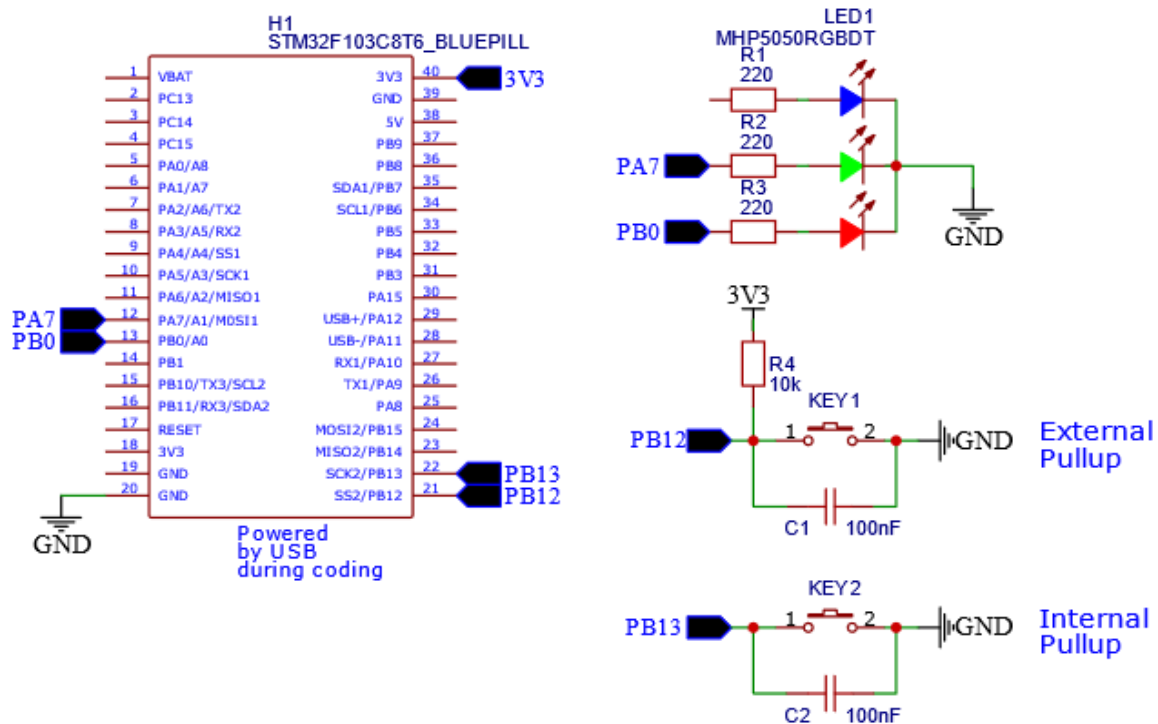
Requirement

1. We shall explore GPIO input and mechanical button bouncing issues.

12.3.1 Diagram

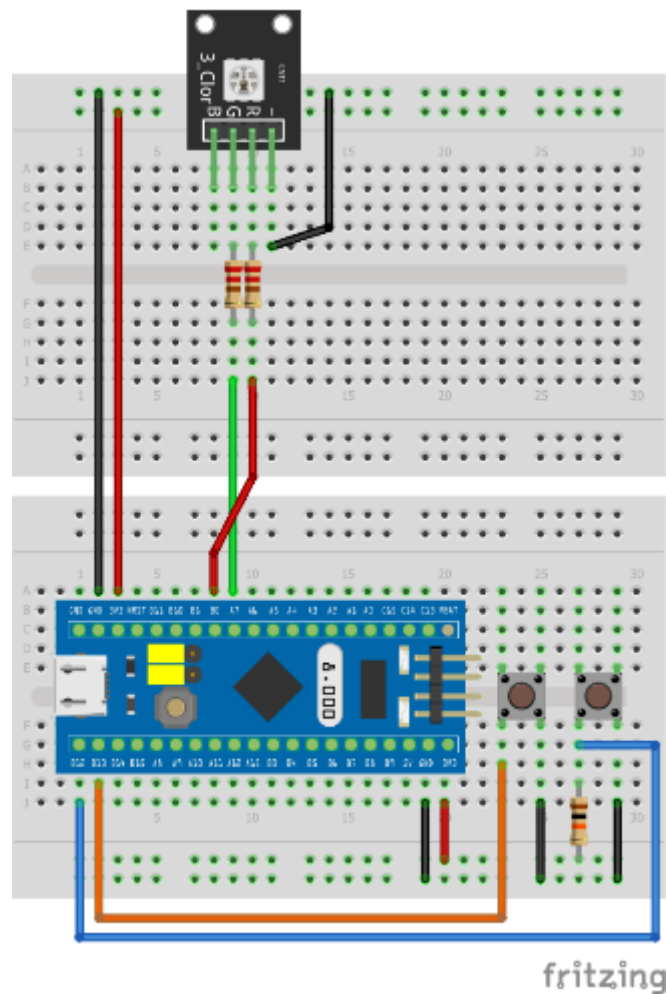
Pin	Mode	User Label
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PB12	GPIO Input, No pull-up and no pull-down	KEY1
PB13	GPIO Input, Pull-up	KEY2

- KEY1 and KEY2 are mechanical momentary type.



*Adding capacitor 100nF to some extent could resolve the bouncing issue.

** Ignore the capacitor connection if not available.



12.3.2 Practices

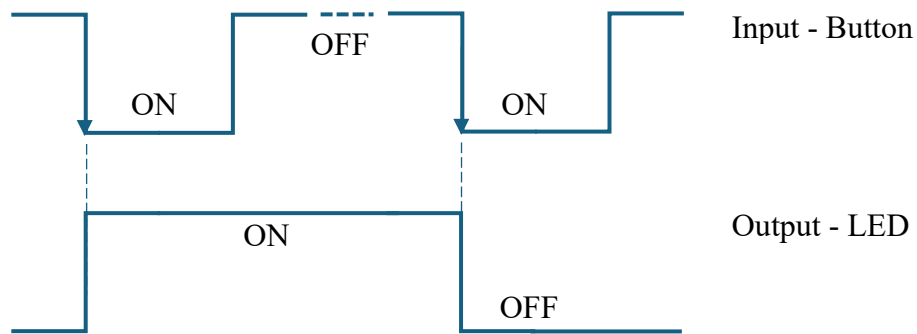
1. Refer to the sample code.
2. KEY1 – not self holding.
 - a. Add in these codes (after **while** (1)).

```

if(HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin)==GPIO_PIN_RESET ){
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
        LED_GREEN_Pin, GPIO_PIN_SET);
} else{
    HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,
        LED_GREEN_Pin, GPIO_PIN_RESET);
}

```

- b. Build and run
 - c. Press KEY1. Observe **GREEN** led.
3. KEY2 – self holding. We expect the button to operate this way, i.e., toggle the LED alternatively.



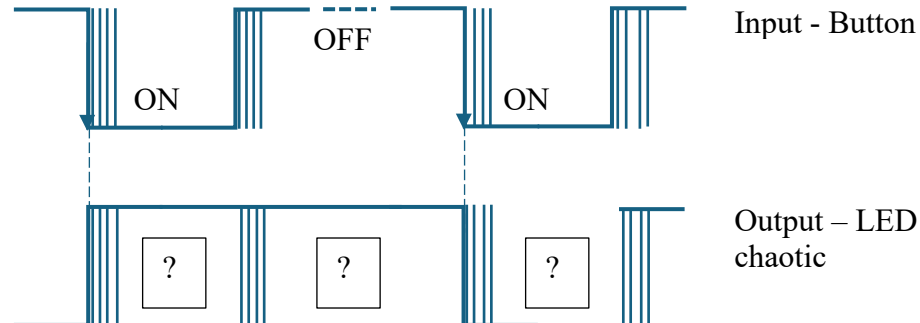
- a. Add these codes within the while loop.

```
if(HAL_GPIO_ReadPin(KEY2_GPIO_Port, KEY2_Pin)==GPIO_PIN_RESET ){

    if(HAL_GPIO_ReadPin(KEY2_GPIO_Port,KEY2_Pin)==GPIO_PIN_RESET ){
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
    }
}
```

- b. Build and run
c. Press KEY2. Observe **RED** led.

4. A momentary mechanical pushbutton displays a phenomenon called a bouncing effect. Practically, the button would exhibit a short moment of state changes as its switch contact displays the spring effect. This uncertainty could disturb the expected outcome.



- a. Add software de-bouncing – hopefully to eliminate the bouncing effect.

```
if(HAL_GPIO_ReadPin(KEY2_GPIO_Port, KEY2_Pin)==GPIO_PIN_RESET ){
    HAL_Delay(10);
    if(HAL_GPIO_ReadPin(KEY2_GPIO_Port, KEY2_Pin)==GPIO_PIN_RESET ){
        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        while(HAL_GPIO_ReadPin(KEY2_GPIO_Port,
                                KEY2_Pin)==GPIO_PIN_RESET){};
    }
}
```

- b. Build and run
c. Press KEY2. Observe **RED** led.

Think about this:

1. Bouncing effects are found during:

- a. High \rightarrow Low
 - b. Low \rightarrow High
- 2. Is software debouncing the ultimate solution?

12.4 GPIO- EXTI

EXTI (EXtErnal Interrupt/Event).

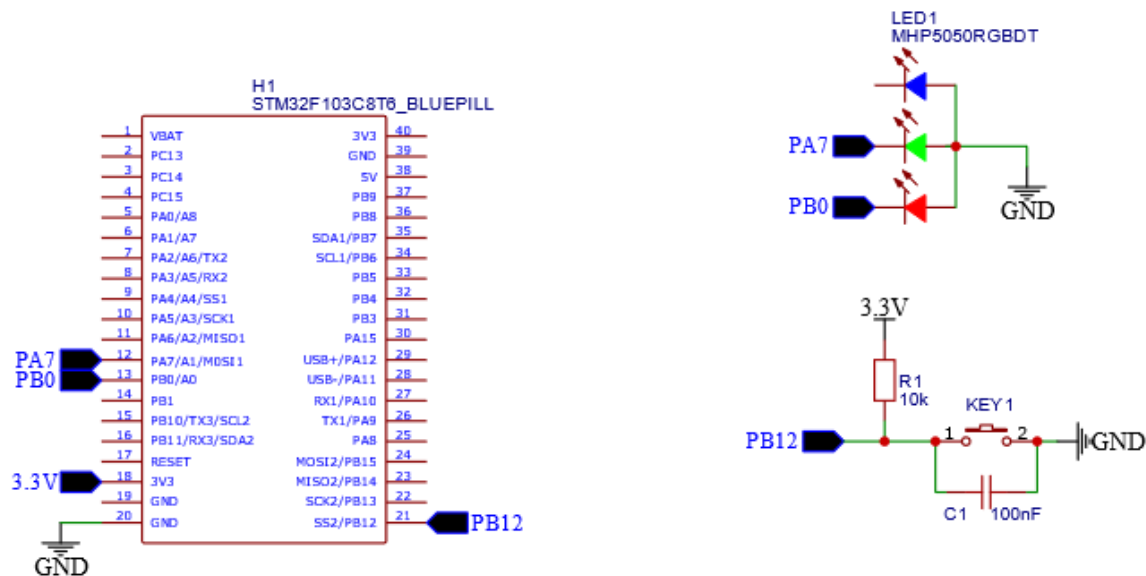
Requirement:

1. LED_RED is blinking at intervals of 2 seconds repetitively.
2. Pressing KEY1 would toggle LED_GREEN at any time.

12.4.1 Diagram

Pin	Mode	User Label
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PB12	GPIO_Input, No pull-up, and no pull-down	KEY1

- KEY1 is a mechanical momentary type.



12.4.2 Practices

1. Refer to the sample code.
2. KEY1 – as GPIO_Input – code within main.c - **while** (1) .
 - a. Add in these codes.

```

HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET);
HAL_Delay(1000);
HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET);
HAL_Delay(1000);

// without interrupt
if (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == GPIO_PIN_RESET) {
    HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
    while (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == GPIO_PIN_RESET);
}

```

- b. Build and run.
- c. Press KEY1. Observe RGB LED.
3. KEY1 – as GPIO interrupt
 - a. Re-configure PB12 as [GPIO_EXTI12, No pull-up and no pull-down]. Name as KEY1. Generate the codes.
 - b. In main.c – remove `// without interrupt` and if-statement
 - c. In `stm32f1xx_it.c`, scroll to the end and find:

```

void EXTI15_10_IRQHandler(void)
{
    /* USER CODE BEGIN EXTI15_10_IRQn 0 */
    // de-bouncing by delay
    HAL_Delay(10);
    if (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == GPIO_PIN_RESET) {
        HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
    }

    /* USER CODE END EXTI15_10_IRQn 0 */
    HAL_GPIO_EXTI_IRQHandler(KEY1_Pin);
    /* USER CODE BEGIN EXTI15_10_IRQn 1 */

    /* USER CODE END EXTI15_10_IRQn 1 */
}

```

**Note: Adding `HAL_Delay(10)` is not a good practice whenever deploying EXTI*

- d. Build and run. (Well! It won't work as expected!)
- e. Open [Pinout & Configuration](#) – update **Preemption Priority**
 - i. Time base: System tick timer > EXTI line[15:10] interrupts

Configuration		
NVIC		Code generation
NVIC interrupt table		Enabled
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	14
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
EXTI line[15:10] interrupts	<input checked="" type="checkbox"/>	15

- f. Build and run. (Well! This time it should work as expected!)

g. Press KEY1. Observe RGB LED.

Explanation:

1. The higher the priority, the lower the assigned number.
2. `HAL_Delay(10)` is dependant on `Systick()`.

12.5 UART in Polling Mode

UART STM32

- Simple UART communication in polling mode
- UART with Interrupt
- UART with DMA

This example features UART in polling mode.

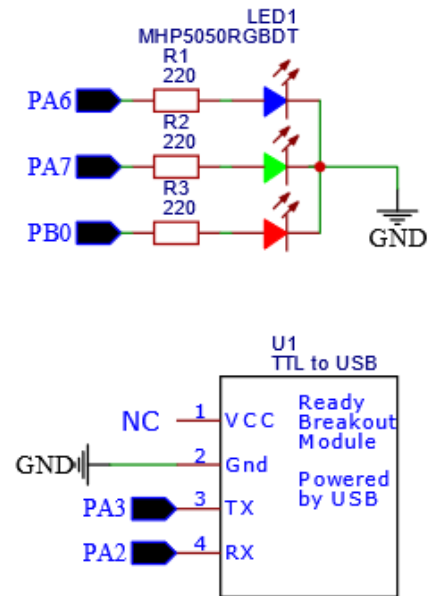
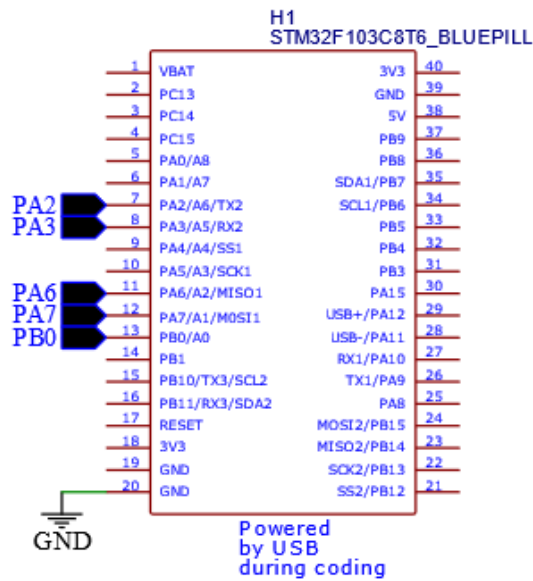
Requirement:

1. Each LED is lit up on the text command via the serial terminal app.
Press enter to send. Observe the change.

Command	Color	ON/OFF
R0	LED_RED	OFF
R1	LED_RED	ON
G0	LED_GREEN	OFF
G1	LED_GREEN	ON
B0	LED_BLUE	OFF
B1	LED_BLUE	ON

12.5.1 Diagram

Pin	Mode	User Label
PA6	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_BLUE
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PA2	USART2_TX GPIO settings completed by Pin & Configuration	
PA3	USART2_RX GPIO settings completed by Pin & Configuration	



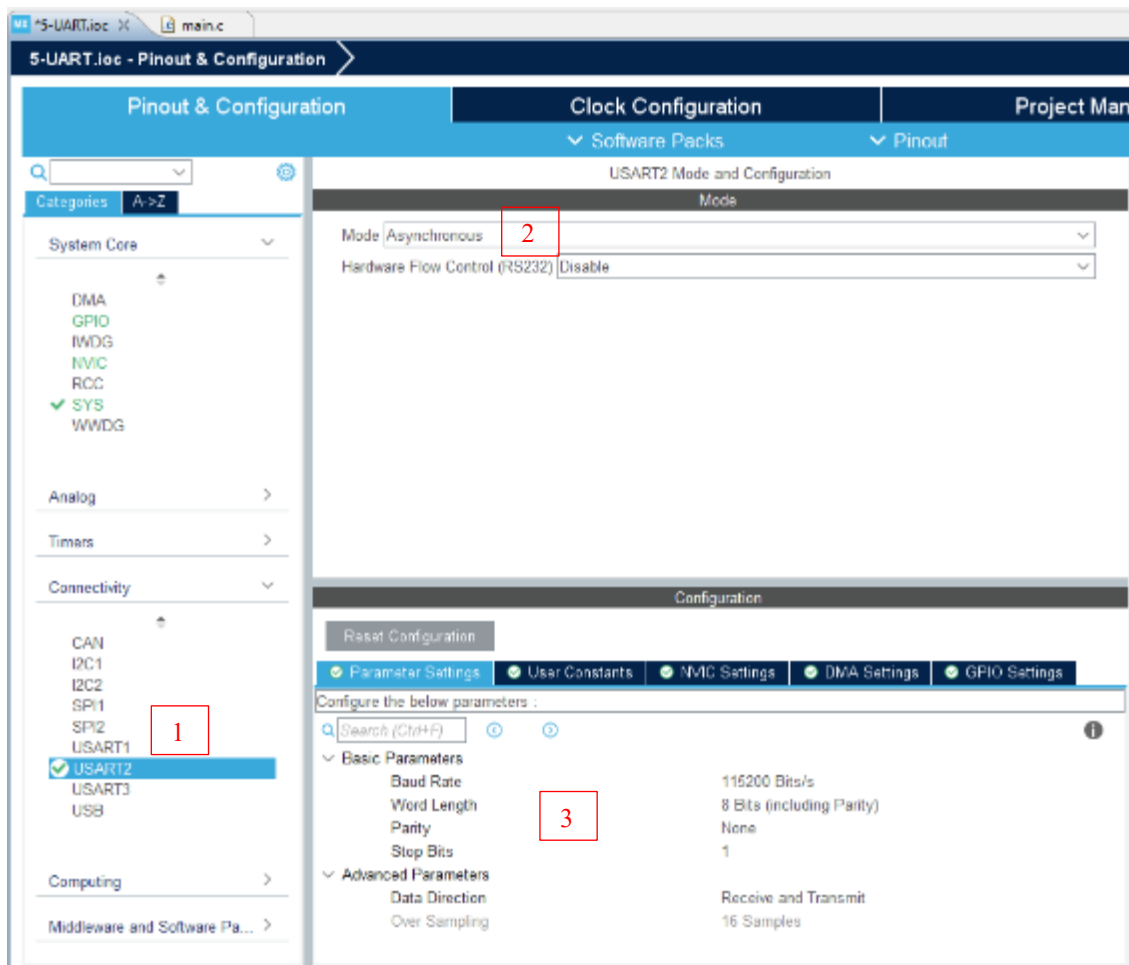
Note:

The connection of TX-RX between an MCU and a UART-to-USB converter must be criss-crossed.

USART2 – Configuration (refer to the figure below.)

- 1 Under Connectivity, select USART2
- 2 Mode: Asynchronous
- 3 Basic parameters: 115200 Bits/s, 8N1

Note: The same configuration must be set up for the serial terminal app.




12.5.2 Practices

1. Refer to the sample code.
2. At main.c, observe `static void MX_USART2_UART_Init(void)`. Check if USART2 is configured as planned.
3. Update the code.
 - a. Add these codes to main.c

```

uint8_t receiveData[2]; // buffer

while (1) 
{
    HAL_UART_Receive(&huart2, receiveData, 2, HAL_MAX_DELAY);
    HAL_UART_Transmit(&huart2, receiveData, 2, 100);
    GPIO_PinState state = GPIO_PIN_SET;

    if (receiveData[1]=='0'){
        state = GPIO_PIN_RESET;
    }else if (receiveData[1]=='1') {
        state = GPIO_PIN_SET;
    }

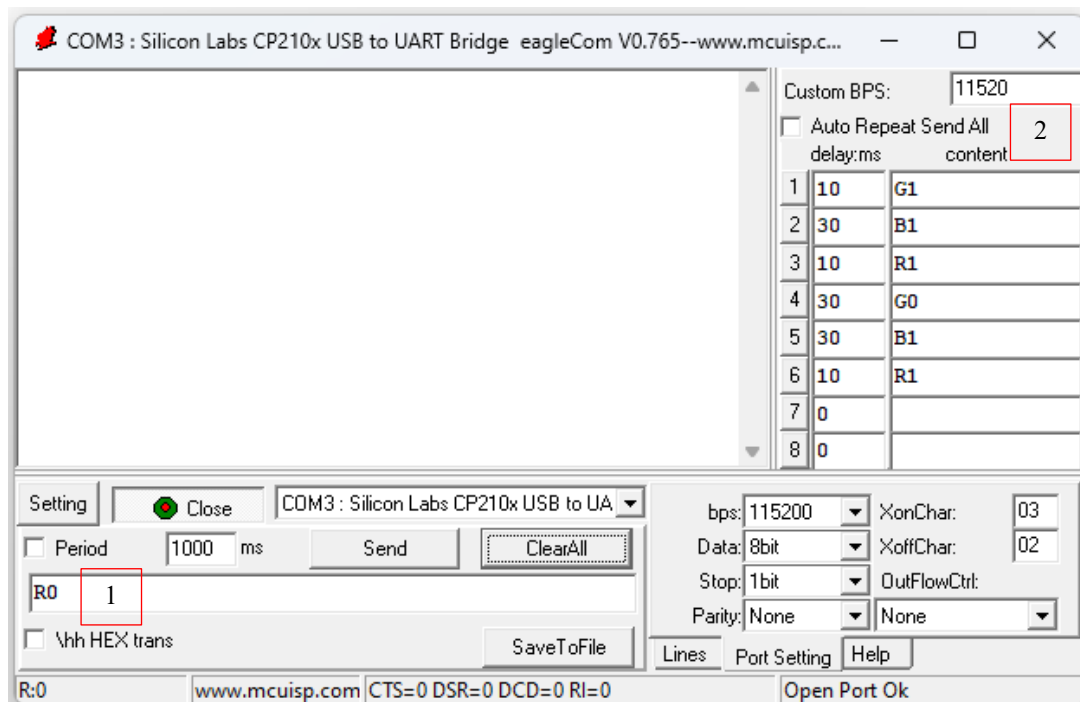
    if (receiveData[0]=='R'){
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, state);
    }else if (receiveData[0]=='G') {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port,LED_GREEN_Pin, state);
    }else if (receiveData[0]=='B') {
        HAL_GPIO_WritePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin, state);
    }
}

```

- b. Build and run
- c. Open serial terminal app. Configure port setting (115200, 8N1). Click Open to establish serial communication with the MCU. At serial terminal app. This example deploys – eagleCom downloaded from www.mcuisp.com
- d. At serial terminal app. Refer to the figure below and enter the text command:

At 1, try to enter these commands. The commands are listed in **UART in Polling Mode**.

Refer to the figure below, enter text command: At 2, try to enter a series of commands and time intervals. Click “Auto Repeat Send All”. Observe the change.



4. What's wrong?

e. Received buffer

`HAL_UART_Receive(&huart2, receiveData, 2, HAL_MAX_DELAY);`
 configured to read 2 bytes of data

- i. At transmit side, enter more than two characters. Ex: R00
- ii. The 'expected' sequence is not in a row! Only 2 characters is read, and the rest is left to the following read.

f. Missing UART-Rx

- i. Notice that in main.c, Uart receive is coded within **while** (1) loop. What if there are codes that take longer processing time? Missing data?
- ii. Try add in `HAL_Delay(2000);` within **while** (1) loop. Build and run. Examine the LEDs.

12.6 UART With Interrupt

This is another mode of UART.
UART with Interrupt.

Requirement:

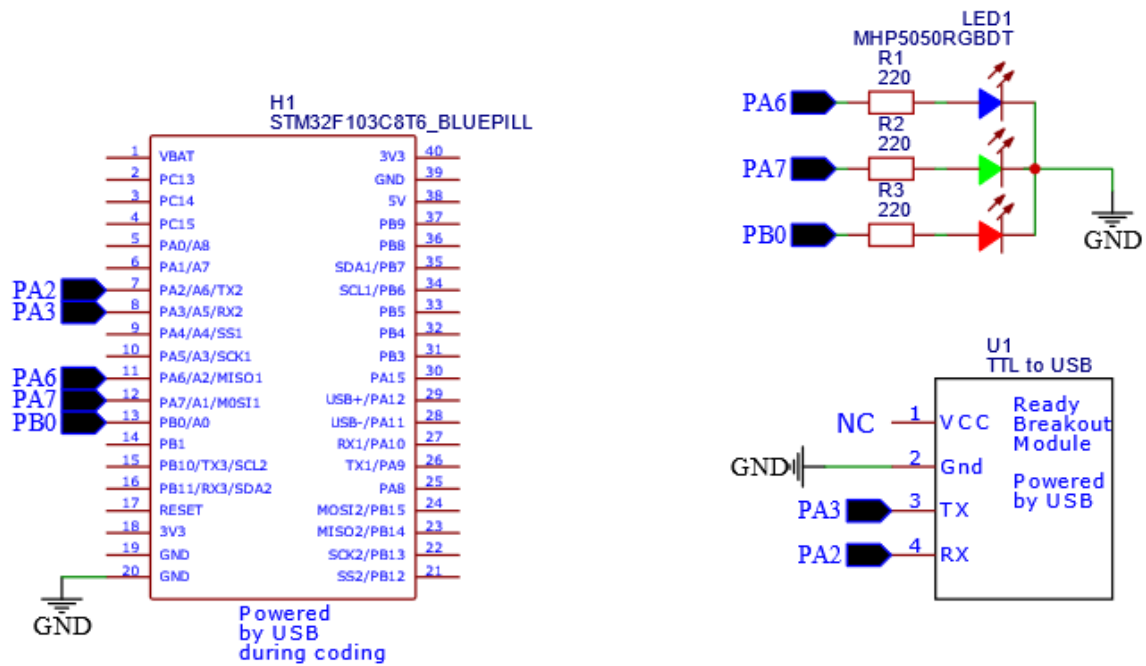
1. Each LED is lit up on text command via a serial terminal app in a callback function.
Press enter to send. Observe the change.

Command	Colour	ON/OFF
R0	LED_RED	OFF
R1	LED_RED	ON
G0	LED_GREEN	OFF
G1	LED_GREEN	ON
B0	LED_BLUE	OFF
B1	LED_BLUE	ON

12.6.1 Diagram

Hardware wiring and Pin & Configuration are the same as before; add in the configuration of NVIC Settings.

Pin	Mode	User Label
PA6	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_BLUE
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PA2	USART2_TX GPIO settings completed by Pin & Configuration	
PA3	USART2_RX GPIO settings completed by Pin & Configuration	



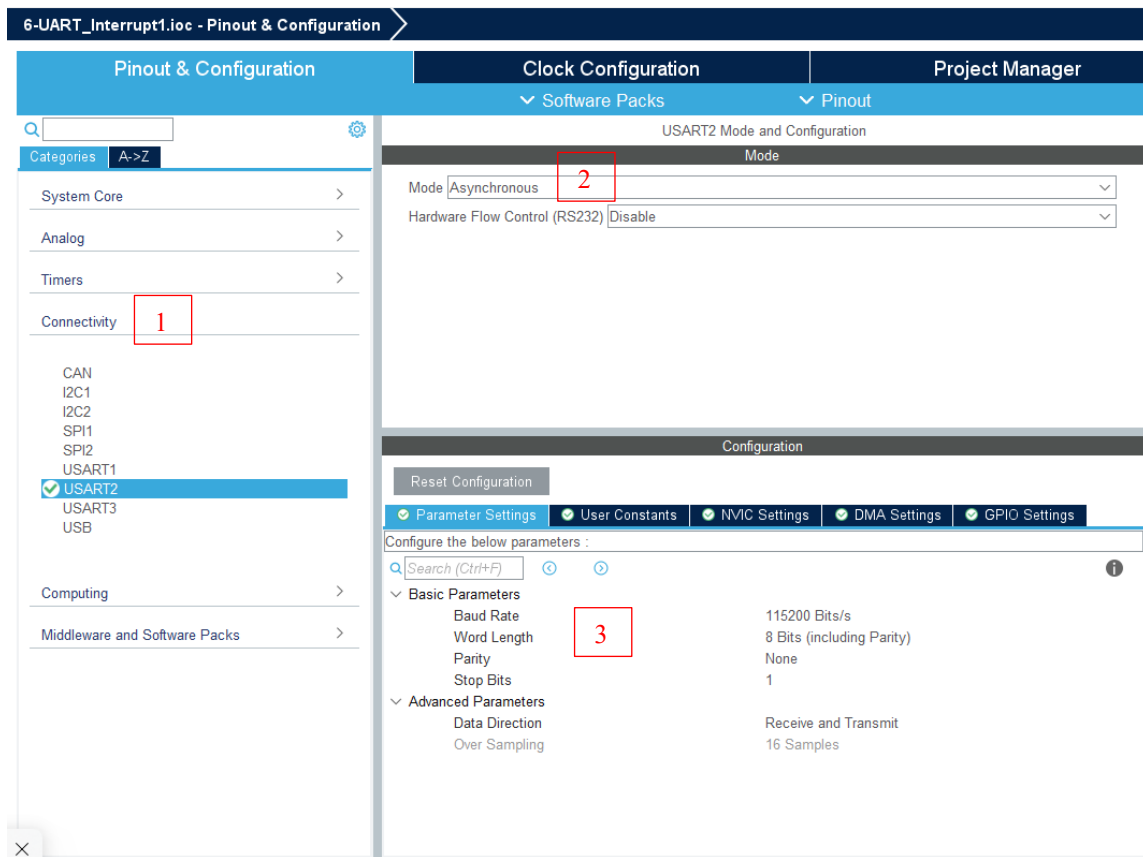
Note:

The connection of TX-RX between an MCU and a UART-to-USB converter must be criss-crossed.

USART2 – Configuration (refer to the figure below.) Under Parameter Settings

- 1 Under Connectivity, select USART2.
- 2 Mode: Asynchronous.
- 3 Basic parameters: 115200 Bits/s, 8N1.

Note: The same configuration must be set up for the serial terminal app.



USART2 – Configuration (refer to the figure below.) Under NVIC Settings

- 4 Select tab | NVIC Settings
- 5 Click Enabled – USART2 global interrupt

6-UART_Interrupt1.ioc - Pinout & Configuration

Pinout & Configuration

Categories A->Z

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- ☒ SYS
- WWDG

Analog >

Timers >

Connectivity >

CAN
I2C1
I2C2
SPI1
SPI2
USART1
☒ **USART2**
USART3
USB

Computing >

Middleware and Software Packs >

Clock Configuration

Pinout

USART2 Mode and Configuration

Mode

Asynchronous

Hardware Flow Control (RS232)

Disable

Configuration

Reset Configuration 4

NVIC Settings
DMA Settings
GPIO Settings

Parameter Settings
User Constants

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART2 global interrupt 5	<input checked="" type="checkbox"/>	0	0

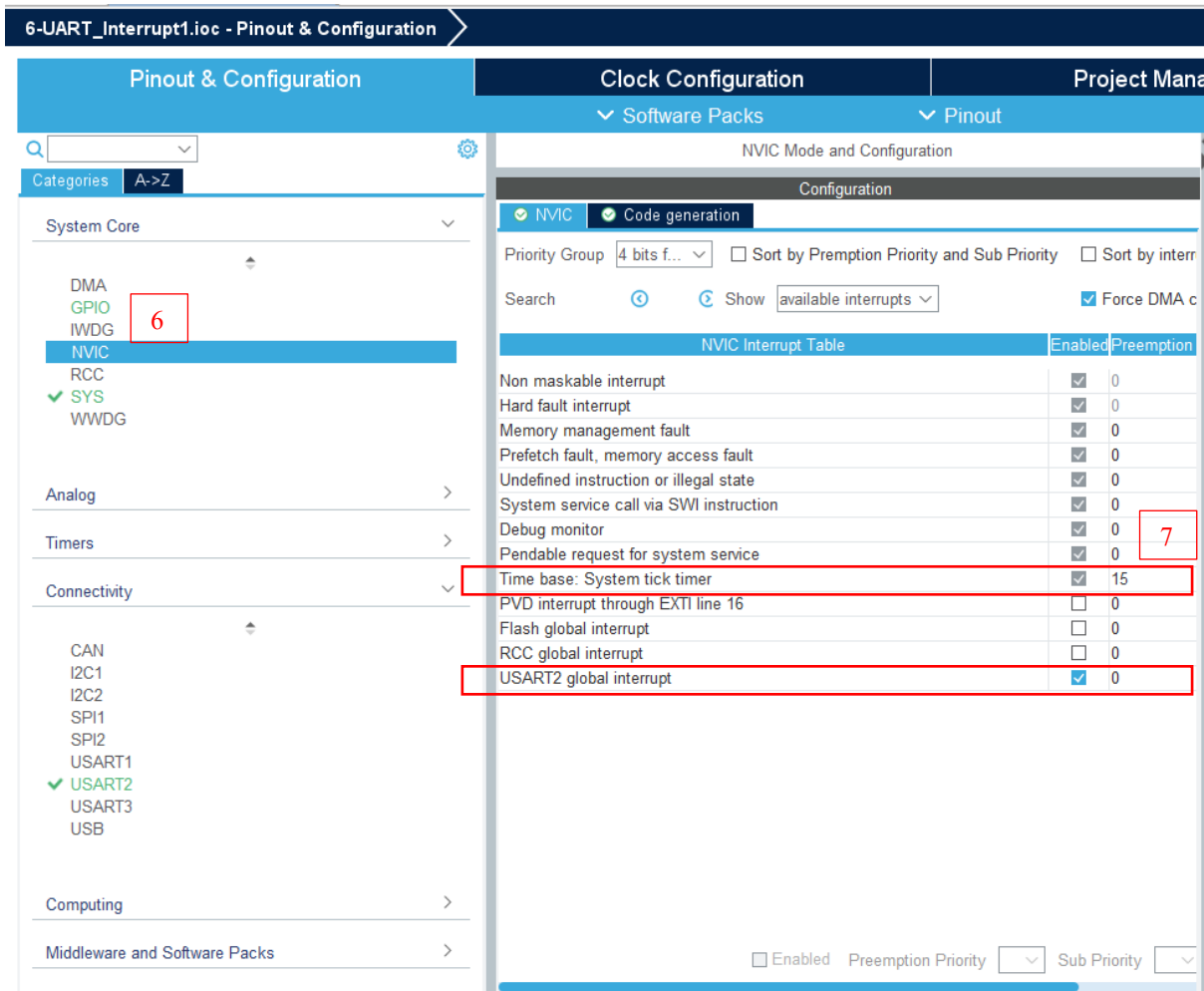
NVIC – Preemption Priority

6

Under NVIC, select USART2

7

Update preemption priority as shown. *[note: Lower number has higher priority.]*



12.6.2 Practices

1. Refer to the sample code.
2. Check the self-generated, `void USART2_IRQHandler(void)` at source file - `stm32f1xx_it.c`. Do nothing.
3. At `main.c`, observe `static void MX_USART2_UART_Init(void)`. Check if USART2 is configured as planned. Do nothing.
4. At `main.c`, update the code.
 - a. Add in a callback function at `main.c`
 - b. Build and run.
 - c. Open serial terminal app. Configure port setting (115200, 8N1). Click Open to establish serial communication with the MCU. At serial terminal app. This example deploys – eagleCom downloaded from www.mcuisp.com
 - d. Examine the LEDs. See the command text in the beginning of **UART With Interrupt**.

Refer to 12.5 UART in Polling Mode.

Compared to the previous example on UART-Polling Mode, communication commands are removed in `while` (1)

```

/* Private user code -----
-----*/
/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    HAL_UART_Transmit_IT(&huart2, receiveData, 2);
    GPIO_PinState state = GPIO_PIN_SET;

    if (receiveData[1]=='0'){
        state = GPIO_PIN_RESET;
    }else if (receiveData[1]=='1') {
        state = GPIO_PIN_SET;
    }

    if (receiveData[0]=='R'){
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, state);

    }else if (receiveData[0]=='G') {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, state);
    }else if (receiveData[0]=='B') {
        HAL_GPIO_WritePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin, state);
    }
    HAL_UART_Receive_IT(&huart2, receiveData, 2);
}
/* USER CODE END 0 */

```

```

int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the
    SysTick. */
    HAL_Init();

    /* USER CODE BEGIN Init */


    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();


    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */

    HAL_UART_Receive_IT(&huart2, receiveData, 2); 
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {

        /* USER CODE END WHILE */ 

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

```

5. At main.c, let's pretend to do some work in **while** (1)
 - a. Add a delay - `HAL_Delay(2000);`
 - b. Build and run.
 - c. Examine the LEDs.
 - d. Is the LEDS being delayed if `HAL_Delay(2000)` is prolonged? Try.

12.7 UART With DMA

This is another mode of UART, deploying Direct memory access (DMA).
UART2 is configured as an event.

Requirement:

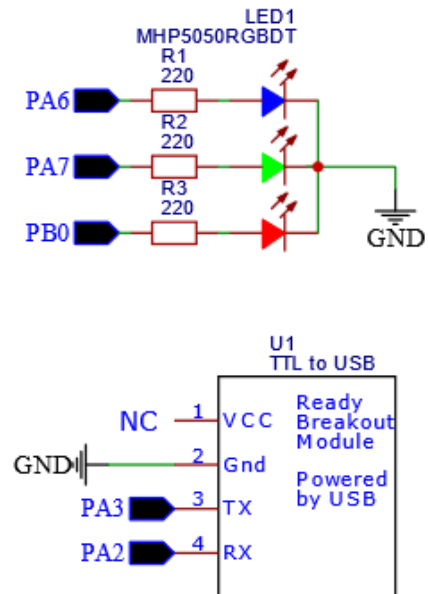
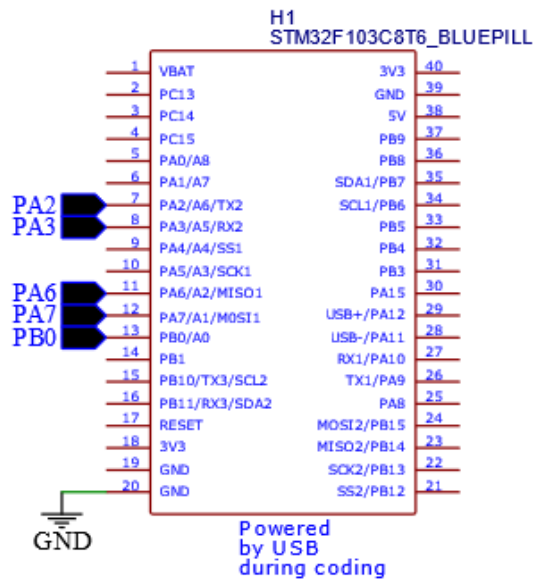
1. Each LED is lit up on text command via serial terminal app in a callback function.
Press enter to send. Observe the change.

Command	Colour	ON/OFF
R0	LED_RED	OFF
R1	LED_RED	ON
G0	LED_GREEN	OFF
G1	LED_GREEN	ON
B0	LED_BLUE	OFF
B1	LED_BLUE	ON

12.7.1 Diagram

Hardware wiring and Pin & Configuration are the same as before; add in the configuration of NVIC Settings.

Pin	Mode	User Label
PA6	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_BLUE
PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PA2	USART2_TX GPIO settings completed by Pin & Configuration	
PA3	USART2_RX GPIO settings completed by Pin & Configuration	



Note:

The connection of TX-RX between an MCU and a UART-to-USB converter must be criss-crossed.

Note:

Repeat the same configuration for UART With Interrupt; add in the selection DMA Settings.

Note:

Alternatively, you could clone a project.

NVIC – Preemption Priority

- 1 Under NVIC, select USART2
- 2 Automatic update preemption priority as shown. *[note: Lower number has higher priority.]*

7-USART_Interrupt_DMA.ioc - Pinout & Configuration

Pinout & Configuration | Clock Configuration | Project M

Software Packs | Pinout

NVIC Mode and Configuration

Configuration

✓ NVIC | ✓ Code generation

Priority Group: 4 bits for pre-... | ☐ Sort by Preemption Priority and Sub Priority | ☐ Sort by interrupts names

Search: | Show: available interrupts | ☒ Force DMA channels Interrupts

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0	0
Memory management fault	<input checked="" type="checkbox"/>	0	0
Prefetch fault, memory access fault	<input checked="" type="checkbox"/>	0	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0	0
Debug monitor	<input checked="" type="checkbox"/>	0	0
Pendable request for system service	<input checked="" type="checkbox"/>	0	0
Time base: System tick timer	<input checked="" type="checkbox"/>	15	0
PVD interrupt through EXTI line 16	<input type="checkbox"/>	0	0
Flash global interrupt	<input type="checkbox"/>	0	0
RCC global interrupt	<input type="checkbox"/>	0	0
DMA1 channel6 global interrupt	<input checked="" type="checkbox"/>	0	0
DMA1 channel7 global interrupt	<input checked="" type="checkbox"/>	0	0
USART2 global interrupt	<input checked="" type="checkbox"/>	0	0

Categories: A->Z

System Core

- DMA
- GPIO
- IWDG
- NVIC
- RCC
- ✓ SYS
- WWDG

Analog >

Timers >

Connectivity

- CAN
- I2C1
- I2C2
- SPI1
- SPI2
- USART1
- ✓ USART2
- USART3
- USB

Computing >

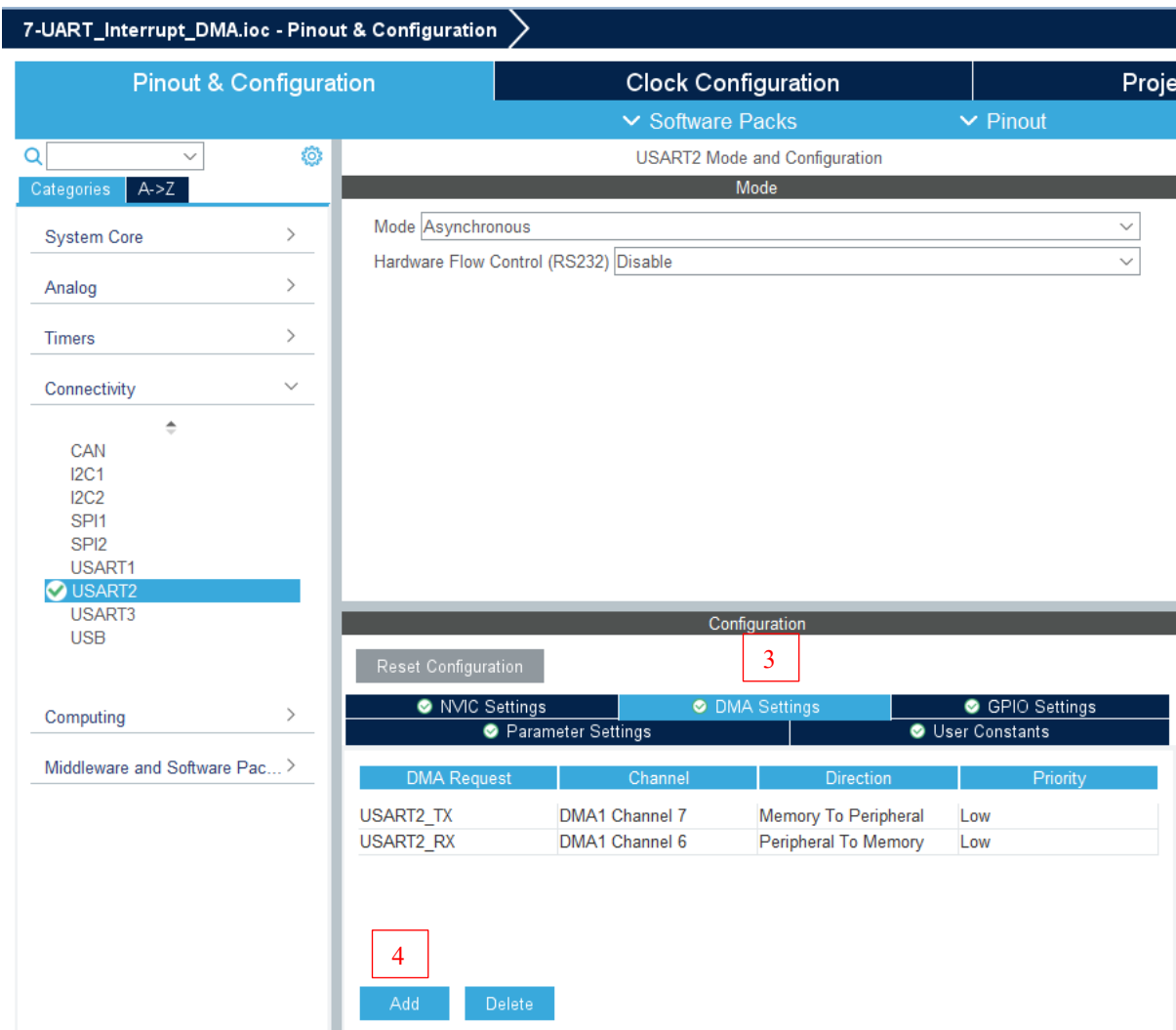
Middleware and Soft... >

☐ Enabled | Preemption Priority: | Sub Priority:

USART2 – DMA Settings

- 3 Under USART2, select DAM Settings
- 4 Add TX and RX. Leave the setting alone.

Note: We configure a channel between UART2 and DMA. The CPU would not be called for each interrupt.



12.7.2 Practices

1. Refer to the sample code.
2. Check the self-generated, `void USART2_IRQHandler(void)` at source file - `stm32f1xx_it.c`. Do nothing.
3. At `main.c`, observe `static void MX_USART2_UART_Init(void)`. Check if USART2 is configured as planned. Do nothing.
4. At `main.c`, observe `static void MX_DMA_Init(void)`. Check if DMA is configured accordingly. Do nothing.
5. At `main.c`, update the code.
 - a. Add in a callback function at `main.c`
 - b. Build and run.
 - c. Open serial terminal app. Configure port setting (115200, 8N1). Click Open to establish serial communication with the MCU. At serial terminal app. This example deploys – eagleCom downloaded from www.mcuisp.com
 - d. Examine the LEDs.
6. When DMA is used, this is taken over `void USART2_IRQHandler(void)`. Check this `void HAL_UARTEx_RxEventCallback()`.

```

/* Private user code -----
-----*/
/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
    HAL_UART_Transmit_DMA(&huart2, receiveData, 2);
    GPIO_PinState state = GPIO_PIN_SET;

    if (receiveData[1]=='0'){
        state = GPIO_PIN_RESET;
    }else if (receiveData[1]=='1') {
        state = GPIO_PIN_SET;
    }

    if (receiveData[0]=='R'){
        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, state);

    }else if (receiveData[0]=='G') {
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, state);
    }else if (receiveData[0]=='B') {
        HAL_GPIO_WritePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin, state);
    }
    HAL_UART_Receive_DMA(&huart2, receiveData, 2);
}

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t
Size){
    if(huart==&huart2){
        HAL_UART_Transmit_DMA(huart, receiveData, Size);

        HAL_UARTEx_ReceiveToIdle_DMA(&huart2, receiveData,
sizeof(receiveData));
        __HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);
    }
}

/* USER CODE END 0 */

```

Bypassed when DMA is selected.


```

int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the
    SysTick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_USART2_UART_Init();
    /* USER CODE BEGIN 2 */


    // HAL_UART_Receive_DMA(&huart2, receiveData, 2);
    HAL_UARTEx_ReceiveToIdle_DMA(&huart2, receiveData,
    sizeof(receiveData));
    __HAL_DMA_DISABLE_IT(&hdma_usart2_rx, DMA_IT_HT);
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

```



7. At main.c, let's pretend to do some work in **while** (1)
 - e. Add a delay - `HAL_Delay(2000);`
 - f. Build and run.
 - g. Examine the LEDs.
 - h. Is the LEDs being delayed if `HAL_Delay(2000)` is prolonged? Try.

8. Study the difference of UART in Polling Mode, UART With Interrupt, UART With DMA.

12.7.3 References on UART

https://wiki.st.com/stm32mcu/wiki/Getting_started_with_UART

12.7.4 Discussion on UART – Polling Mode, Interrupt, DMA

The MCU has three key resources for GPIO: USART, I2C, analog IN/OUT, and other peripherals. The resources are the CPU, peripherals, and memory. A CPU will bridge between a peripheral and the memories.

UART works on the agreed-upon baud rate between the two devices. In the first practice, which involves using [UART in Polling Mode], the CPU is continuously engaged, waiting to receive data from a UART to memories and ready to transmit data from memories to a UART, respectively. Any coded functions or peripheral usages (if coded) can jeopardize the UART timing issue, potentially resulting in lost data. To mitigate this issue, the UART can operate with interrupts, allowing the CPU to attend to tasks in priority order and resume to normal. This practice is demonstrated in [UART With Interrupt]. The third option is to deploy [UART With DMA], rechanneling a peripheral directly to the memory area, and the CPU is freed. *Note: NVIC for UART is enabled.*

12.8 UART With BLE_DMA

Refer to the previous project [[UART With DMA](#)], but now we have added a **Bluetooth module, HC06, connecting to USART3**. The port settings are identical. Here, we will try a customized protocol to turn ON/OFF the LEDs.

The **protocol** is defined as (in HEX):

- Format: **Header Length Data Checksum**
- header: 0xAA
- Data definition: 0x01 = red, 0x02 = green, 0x03=blue, 0x00 = OFF, 0xFF = ON
- checksum = sum (byte) of all previous bytes

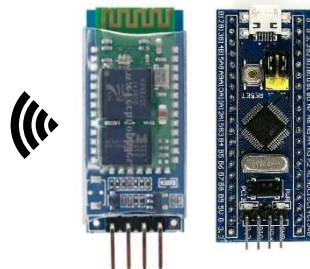
Example: (data in HEX)

- AA 05 01 00 B0
- AA 07 01 00 03 FF B4
- Red OFF == AA 05 01 00 B0
- Red ON == AA 05 01 FF AF
- Green OFF == AA 05 02 00 B1
- Green ON == AA 05 02 FF B0
- Blue OFF == AA 05 03 00 B2
- Blue ON == AA 05 03 FF B1

A laptop or a smartphone is always configured as a BLE-Master, while HC06 (or HC05) is configured as a BLE-Slave (**default setting: Slave, 9600, 8N1**).



BLE - Master



BLE - Slave

Some info about HC05: (You may skip this part!)

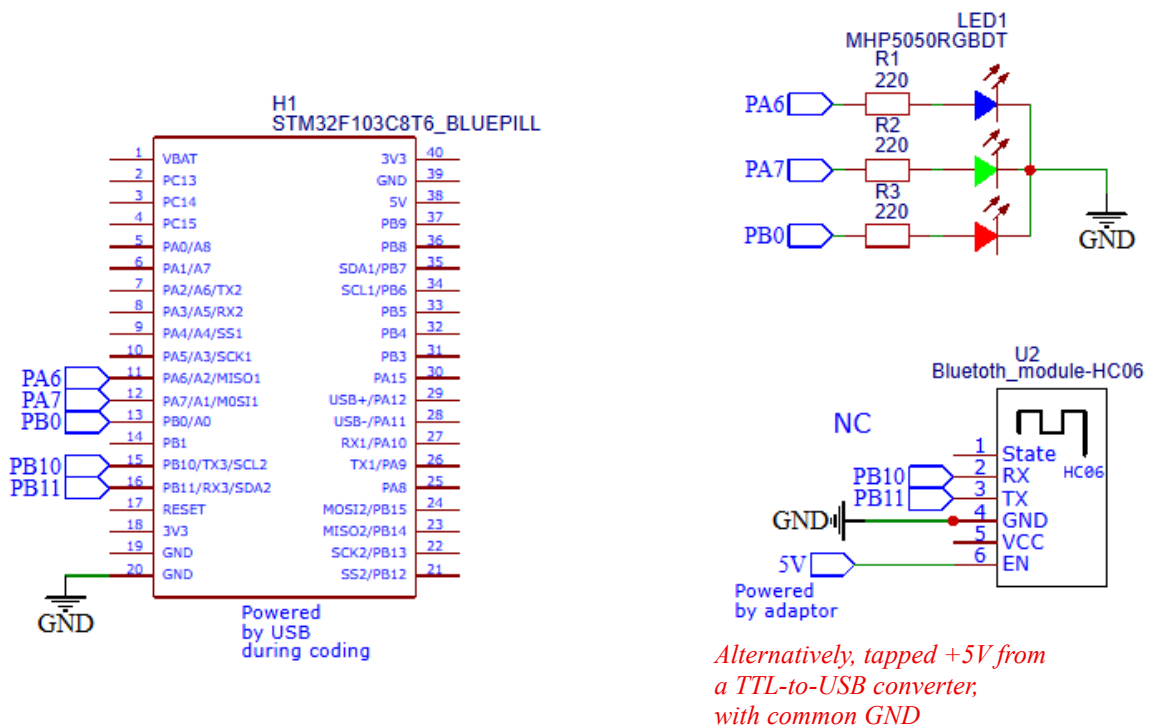
<https://www.electronicwings.com/sensors-modules/bluetooth-module-hc-05->

12.8.1 Diagram

Hardware wiring and Pin & Configuration are the same as before; add in the configuration of NVIC Settings.

Pin	Mode	User Label
PA6	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_BLUE

PA7	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_GREEN
PB0	GPIO_Output, Low, Output Push Pull, No pull-up and no pull-down	LED_RED
PB10	USART3_TX GPIO settings completed by Pin & Configuration	BLE
PB11	USART3_RX GPIO settings completed by Pin & Configuration	BLE

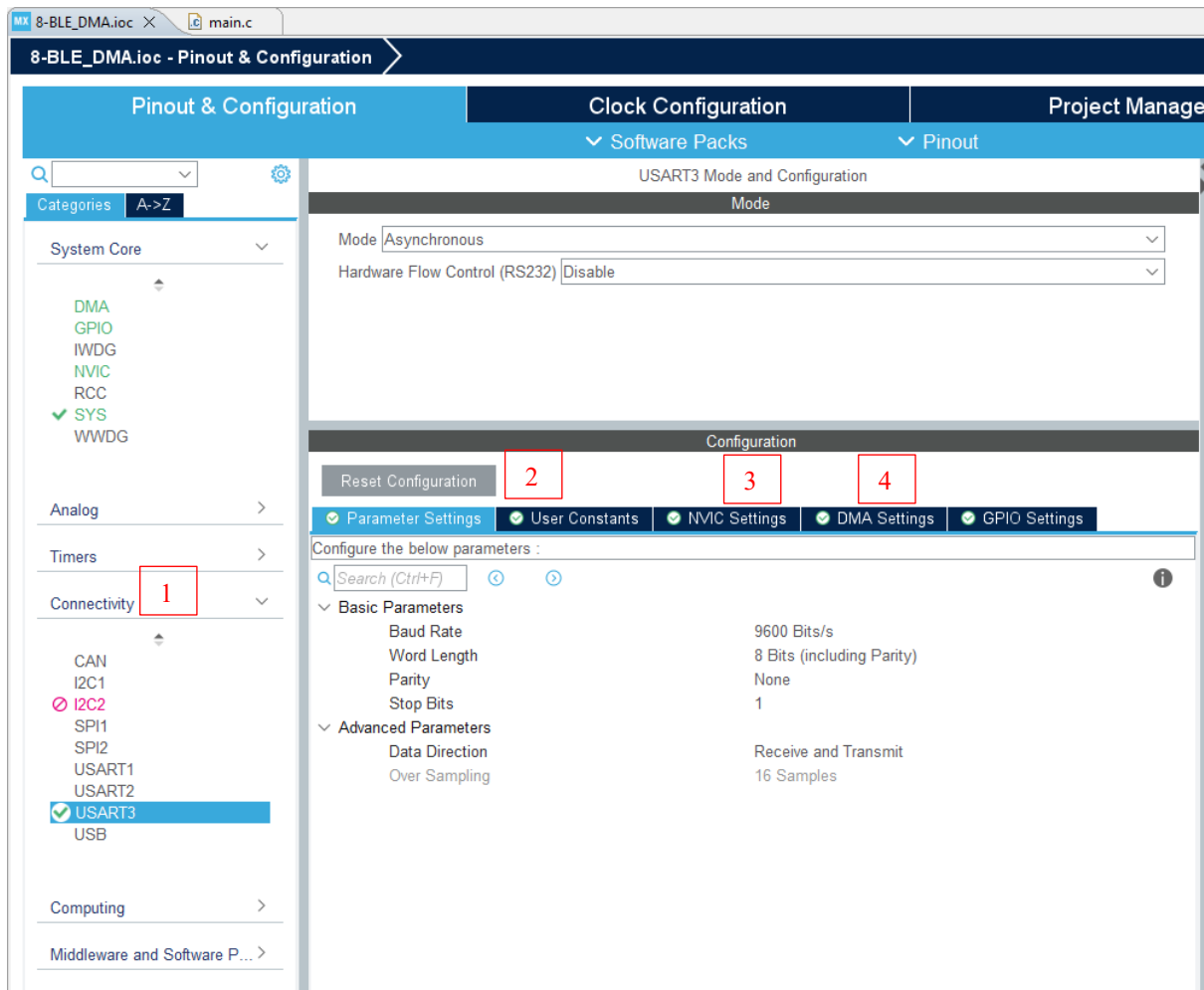


Note:

HC06/HC05 is powered individually by 5VDC.

USART3 – Configuration – Parameter Settings, NVIC, DMA

- 1 Under Connectivity, select USART3
- 2 Select Parameter Settings. 9600, 8N1 (Default setting to Bluetooth module, HC06)
- 3 Select NVIC. Check USART3 global interrupt.
- 4 Select DMA Settings. Add in DMA channel – USART3-TX and USART3-RX. Use default settings.



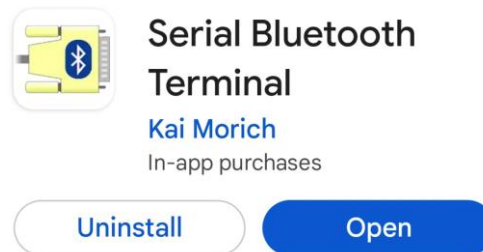
12.8.2 Practices

1. Refer to the sample code.
2. At main.c, update the code.
 - a. Add a callback function to main.c


```
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart,
                                uint16_t Size)
```
 - b. Build and run.
3. On the laptop,
 - a. Enable Bluetooth, HC06. (scan for BLE devices)
 - b. Open a serial terminal app. Configure port setting (9600, 8N1). Click open the port. At the laptop, Bluetooth & device > Devices, HC05 should be shown as "Connected"
 - c. Examine the LEDs. Use the protocols mentioned above.

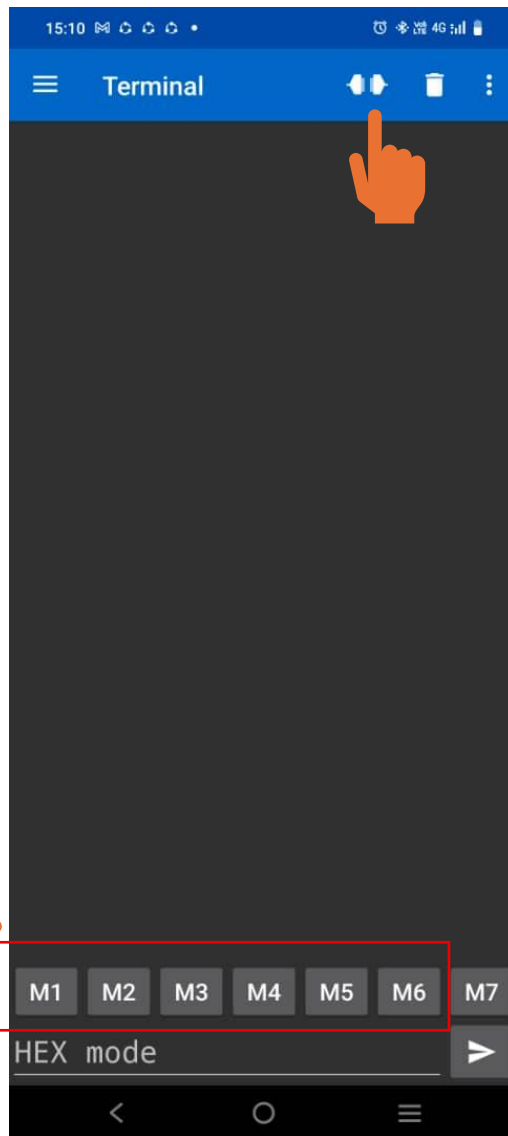


Alternatively, using a smart handphone (Android), search in the Play Store | Serial Bluetooth Terminal. Install it.



On a smartphone, enable Bluetooth and search for HC05. Pair it. Click connect. Configure the button as macro (M1, M2, M3, M4, M5, M6 to hex codes above). It is rather intuitive, or else the instructor will demonstrate it.

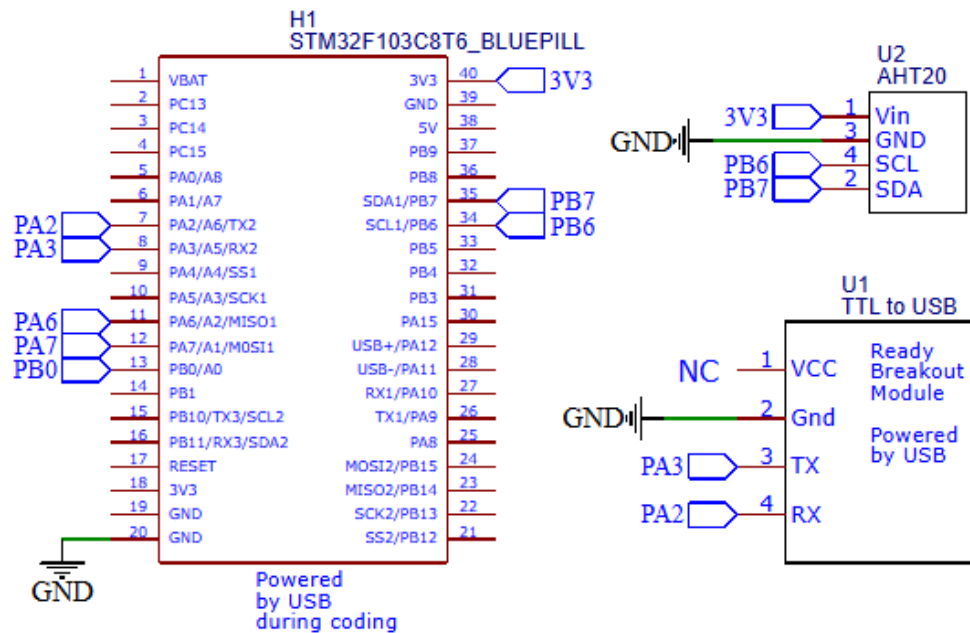
Configure the macro,
M1 to M6



12.9 IIC_AHT20

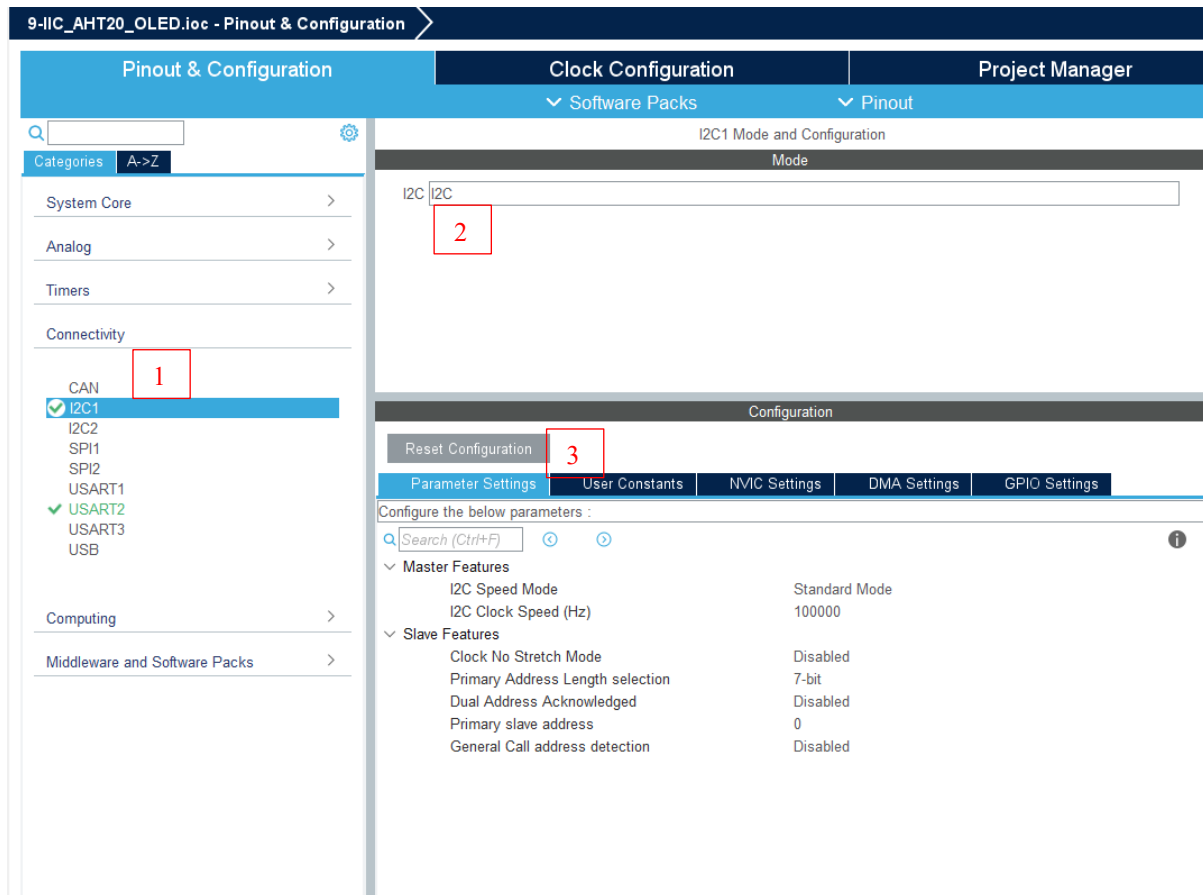
12.9.1 Diagram

Pin	Mode	User Label
PA2	USART2_TX Polling Mode - default settings (115200, 8N1)	USART2
PA3	USART2_RX	USART2
PB6	I2C1_SCL Use default settings – standard mode, 100k	I2C1
PB7	I2C1_SCA	I2C1



I2C1 – Configuration – Parameter Settings

- 1 Under Connectivity, select I2C1
- 2 Mode. Choose I2C.
- 3 Select Parameter Settings. Standard mode, 100000 (Default setting)



12.9.2 Practices

1. Refer to the sample code.
2. At *.ioc, click Project Manager|Code Generator, and check – **Generate peripheral initialization as a pair of .c/.h' files per peripheral**
3. Now, check in the Project Explorer, .c and .h files for each peripheral are created. (Check under Core|Inc and Core|Src)
4. Create .c and .h for the sensor, AHT20. (parked under Core|Inc and Core|Src)
5. Refer to the sample code.
6. Temperature and Humidity will be posted to USART2.
 - a. Open a serial app and read the data.
 - b. Blow your breath on the sensor and observe the data change.

Note:

I2C1 is in polling mode. No interrupts or DMA are configured.

UART Serial Monitor

Serial Configuration

COM Port: COM8 Refresh

Baud Rate: 115200 About

Data Bits: 8

Parity: None

Stop Bits: 1

Flow Control: None Disconnect

Communication Display

☐ Display HEX ☐ Show Timestamp ☒ Autoscroll ☐ DTR ☐ RTS Clear Display Save to File

[RX] Temperature: 27.0 C, Humidity: 81.2 %

[RX] Temperature: 27.0 C, Humidity: 81.2 %

[RX] Temperature: 27.0 C, Humidity: 81.2 %

[RX] Temperature: 27.0 C, Humidity: 81.2 %

[RX] Temperature: 27.0 C, Humidity: 81.3 %

Transmit Data

☐ Send as HEX ☐ Auto Transmit (ms): 1000 ^ v

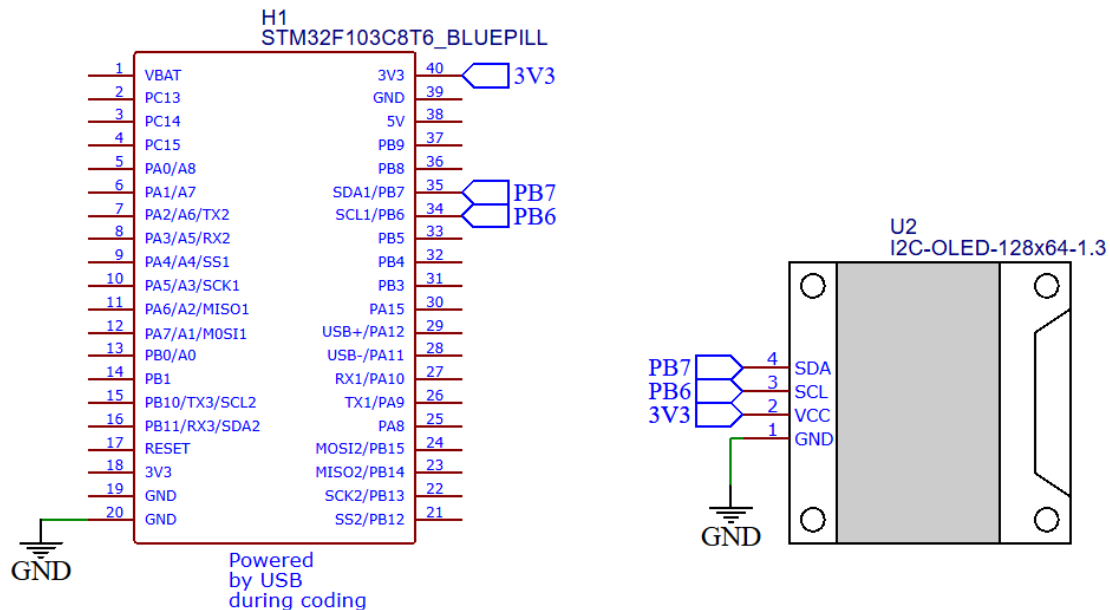
Transmit Clear Input

12.10 IIC_OLED

In this practice, we do not want to rewrite the code from scratch. Instead, we look for a pre-existing library created by someone on GitHub. This exercise illustrates the procedure for utilizing a third-party library.

12.10.1 Diagram

You may continue from Practice 9 -IIC-AHT20, without removing the wiring.



Pin	Mode	User Label
PB6	I2C1_SCL Use default settings – standard mode, 100k	I2C1
PB7	I2C1_SCA	I2C1

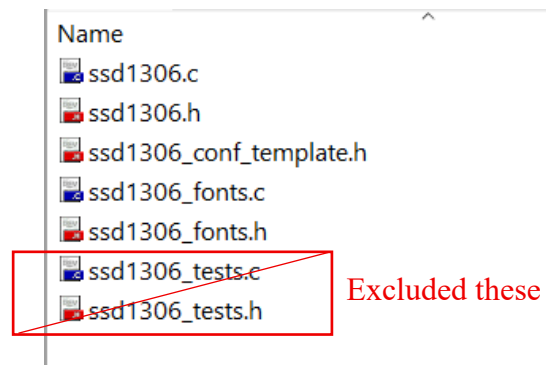
12.10.2 Practices

GitHub

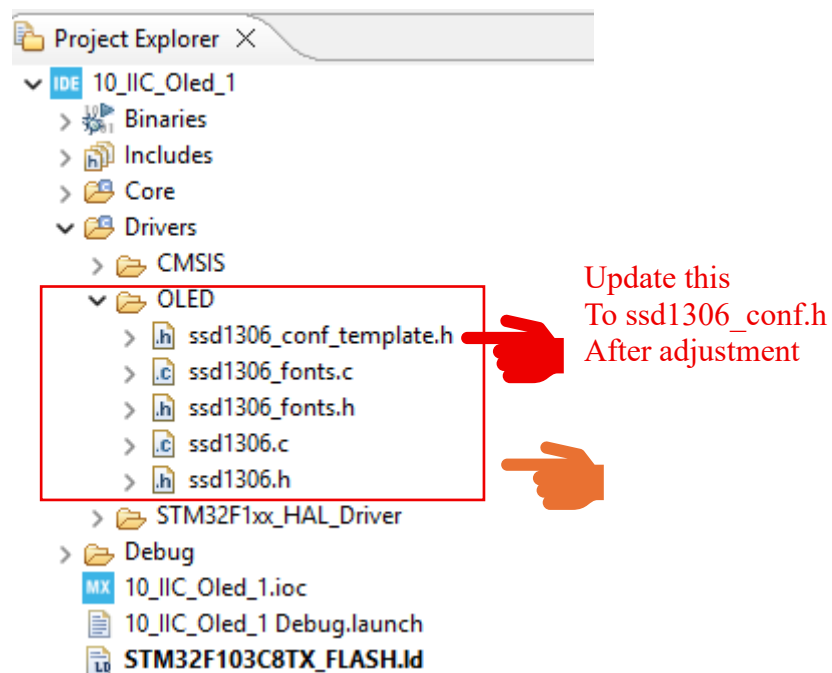
1. GitHub – download the ZIP files
<https://github.com/afiskon/stm32-ssd1306/tree/master>
2. Unzipped the driver and open the folder.
3. Copy all files under the folder./ssd1306/” except *ssd1306_tests.c* and *ssd1306_tests.h*

STM32CubeIDE

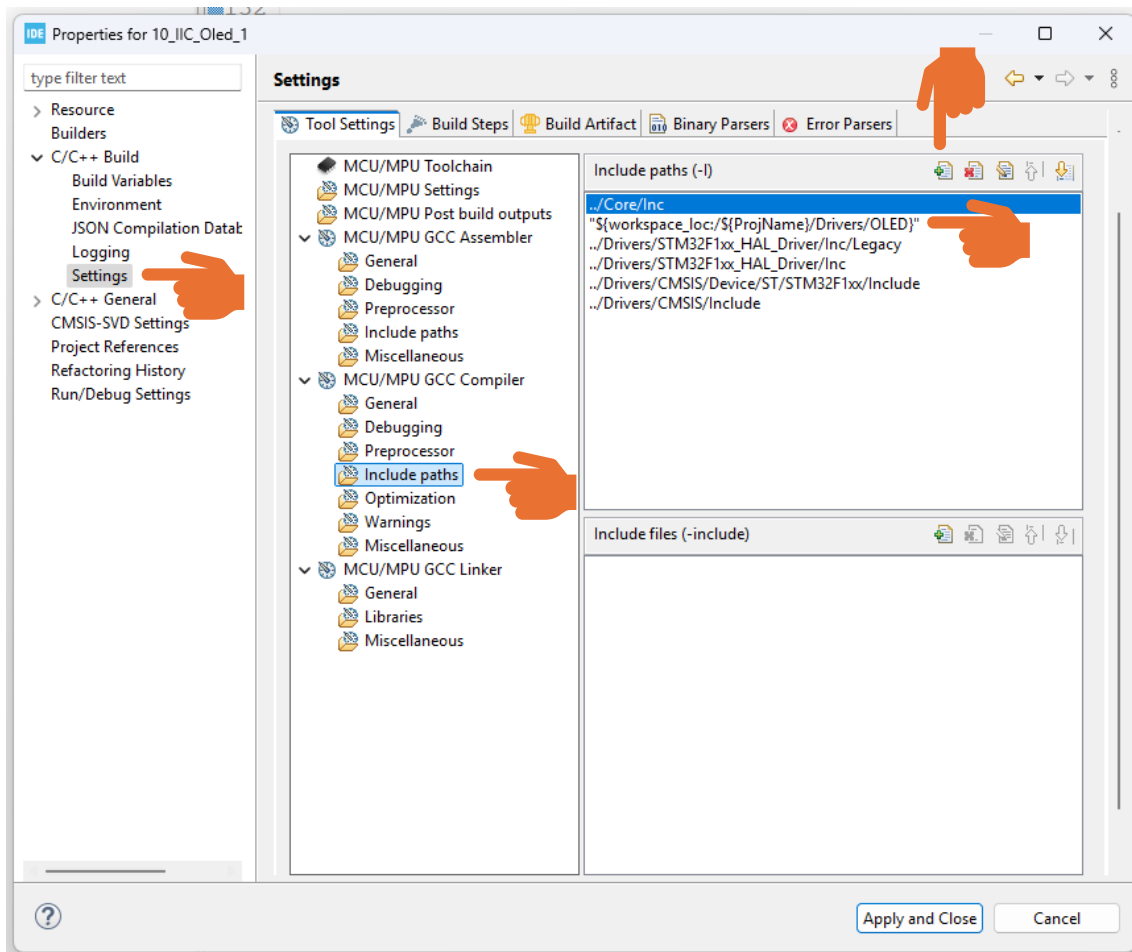
1. Create a new project named “IIC_OLED”. Configure for I2C1.
2. Try to build. No error, no warning.
3. At the project folder (.\workspace\10_IIC_Oled_1\Drivers\) create a new folder named “OLED”
4. Paste all files downloaded as figure below:



And at Project Explorer, refresh, and the files should be added as below:



5. Add path to the project. Right-click on the project and go to "Properties". Do the following: Add the path to the project.



6. Rebuilt. Error. No worry. Configure
7. Adjust "ssd1306_conf_template.h". The instructor will show.
8. Code at main.c. The instructors will demonstrate it.

The steps are demonstrated in this video:

<https://youtu.be/z1Px6emHIeg?si=2rF47ub8JtQxVgpg>

13 Tools

You may skip this section and directly jump to the practices at [12 Practices]. No worries, you will use these tools as you progress.

A few useful tools are introduced.

1. Serial terminal
Preferably GUI and idiot proof.
A number of free apps are available online. **CoolTerm** and **eaglecom** are introduced; however, you could learn and use any serial terminal app for the application in reading/writing serial data. Used together to read/write serial data via USB port at the computer. For example: UART-to-USB, RS485-to-USB, RS232-to-USB, etc.
Note: We will explore other serial terminal apps throughout the practices. Some may appear to be convenient and intuitive than others.
2. In System Programming (ISP), include In Circuit Programming and In Application Programming.
3. ST-Link Utility
STM32 ST-LINK Utility (STSW-LINK004) is a full-featured software interface for programming STM32 microcontrollers.

13.1 Serial App – CoolTerm

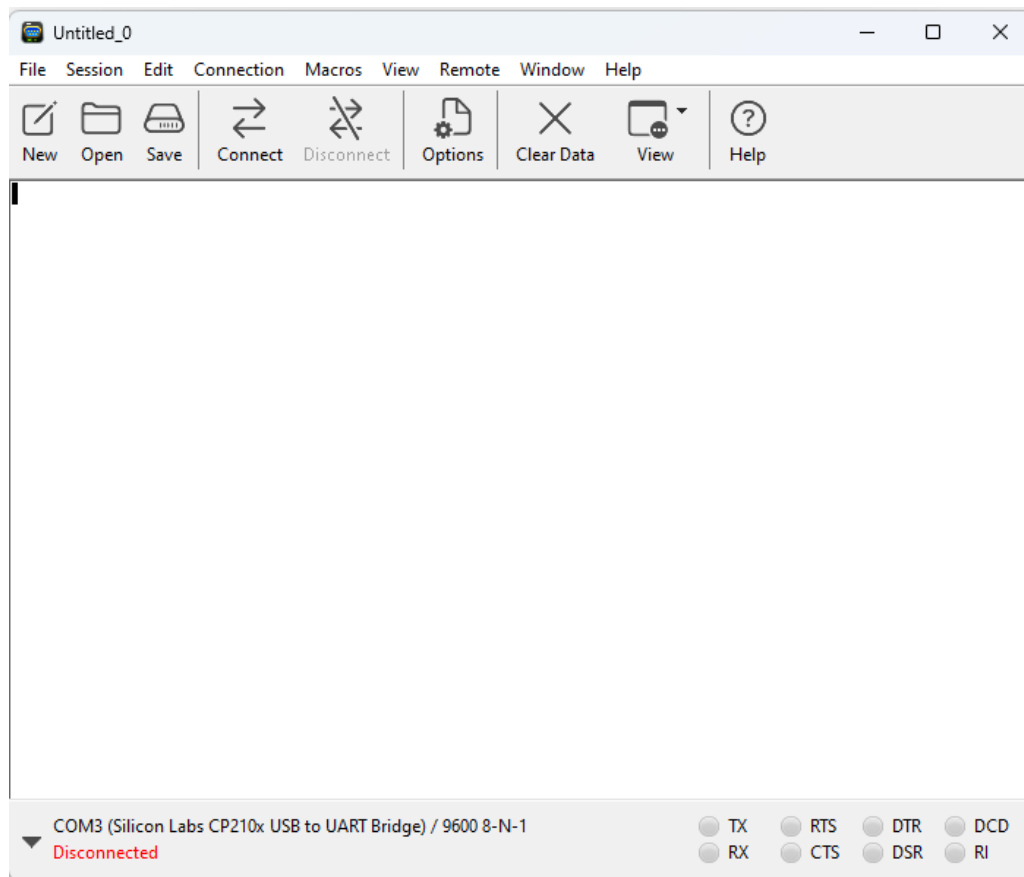
No worries. It is a GUI and rather easy to use. Just play around, and you could get in handy fast.

Serial terminal app. Free to download.

<https://coolterm.en.lo4d.com/windows>

Tutorial on CoolTerm

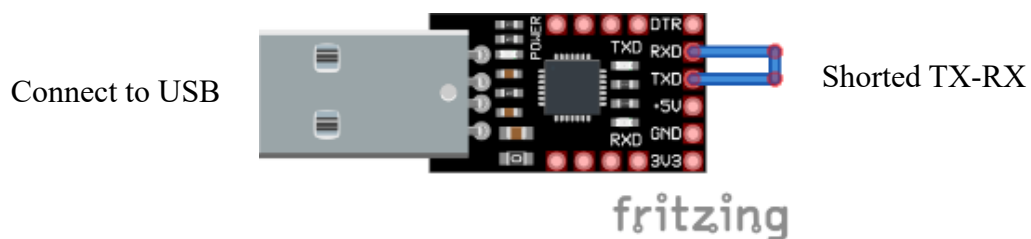
1. <https://learn.adafruit.com/getting-started-with-binho-nova/quickstart-with-coolterm>
2. Brief Steps:
 - Step 1. Plug in the UART-to-USB converter in the computer. Open **Device Manager** check COM port availability.
 - Step 2. Open CoolTerm – click **Options** – set **Serial Ports** - COM port, baud rate, data bits, parity, stop bits
 - Step 3. Click **Connect**. You are ready to connect
 - Step 4. Click **Connection** | Send String. Select Plain or Hex.



13.1.1 Self-loop Test With UART-to-USB

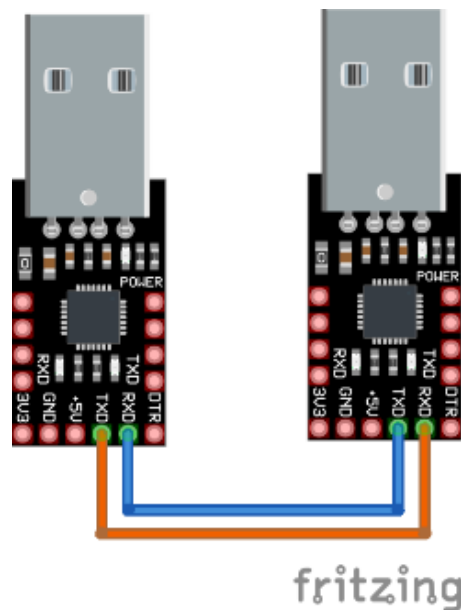
Do sel-loop, by connecting TX to RX. Plug in the converter to the computer, then open a serial terminal app. Do send anything, if the converter is working and the connection is correct, you should receive the return data.

- Note: The converter must be detected and listed in Device Manager. Use the same configuration settings as listed in Device Manager. Example, 9600, 8N1.



13.1.2 Self-test With Two Units of UART-to-USB

Imagine that two serial devices are communicating. Connect RX to TX and TX to RX. Open two serial terminal apps and select a COM port for each converter. Simulate Read/Write data.



13.2 Flashing Using UART Port

There are a number of options to flash an MCU. One way is to flash via UART1 port.

Note: This differs from using ST-Link V2 using the SWD (Serial Wire Debug) interface or the full JTAG interface.

Download “Free STM32 ISP software” from

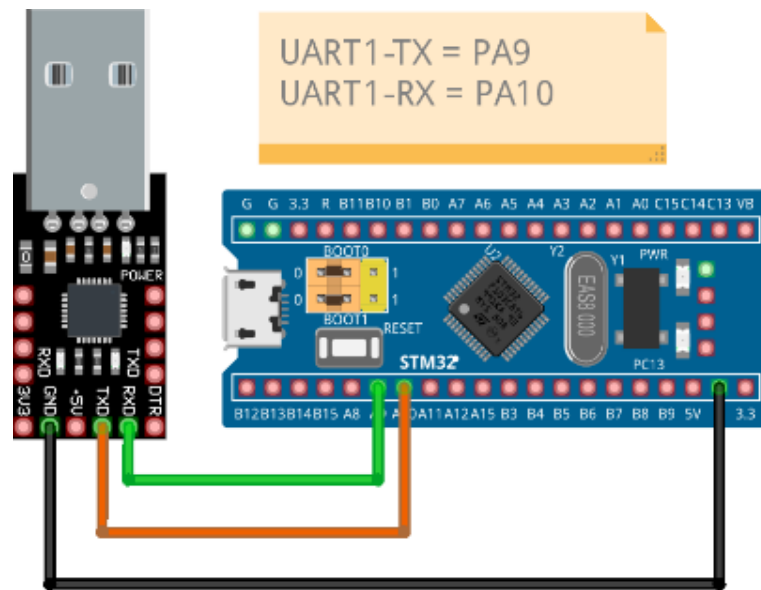
<http://www.mcuisp.com/English%20mcuisp%20web/ruanjianxiazai-english.htm>

This method deploys the bootloader located the system memory. Refer to the BOOT modes.

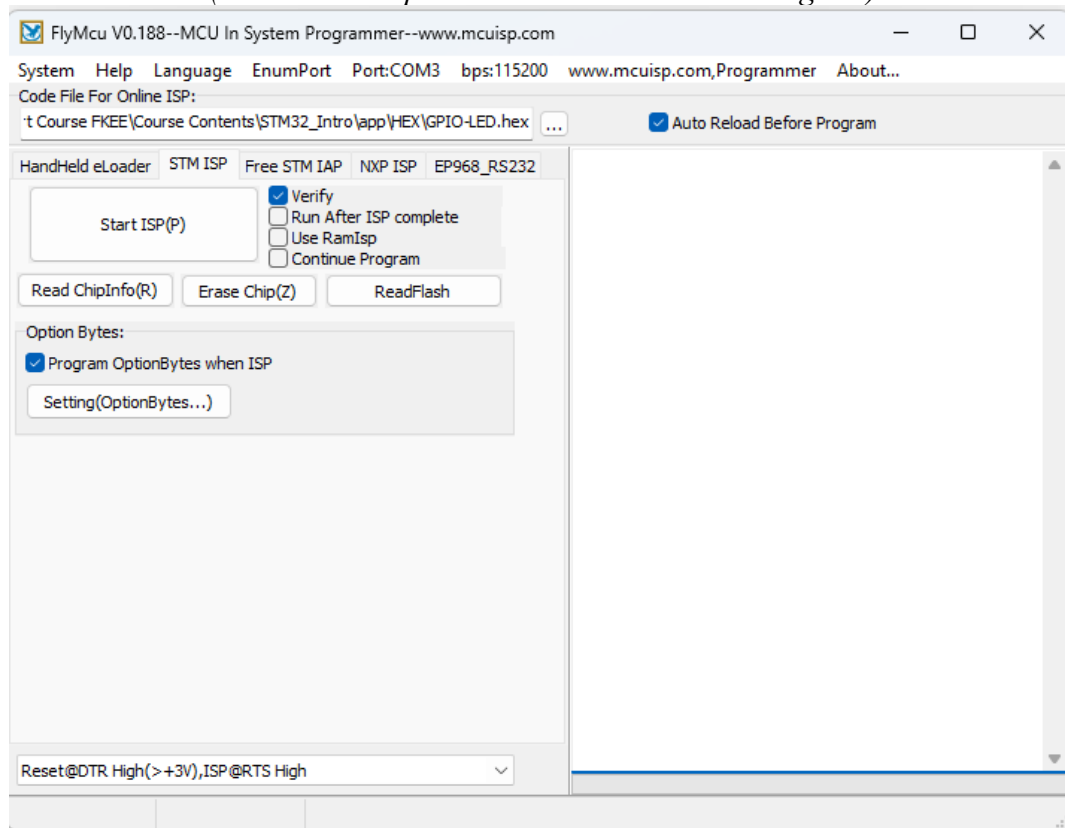
Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as boot space
0	1	System memory	System memory is selected as boot space
1	1	Embedded SRAM	Embedded SRAM is selected as boot space

(Source: RM0008 Reference manual)

Criss-crossed connection of [TX, RX] between UART-to-USB and UART1 as shown in the figure below.



fritzing

(MCU must be powered. Not Shown in the diagram)

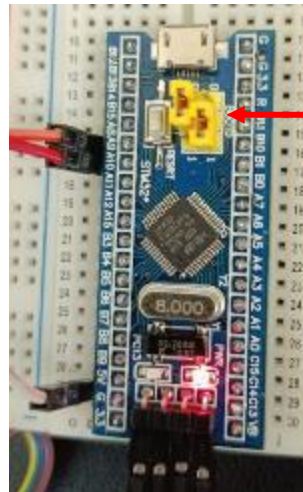
13.2.1 Procedures to Write

- Step 1. Identify connection and COM port name – check Device Manager
- Step 2. At MCUISP, choose the COM port. Leave the settings as it is.
- Step 3. At Codes Files For Online ISP – choose the hex file.
- Step 4. Hardware. Switch jumper **BOOT0** → 1. Press reset button. MCU would switch to System memory - Bootloader.

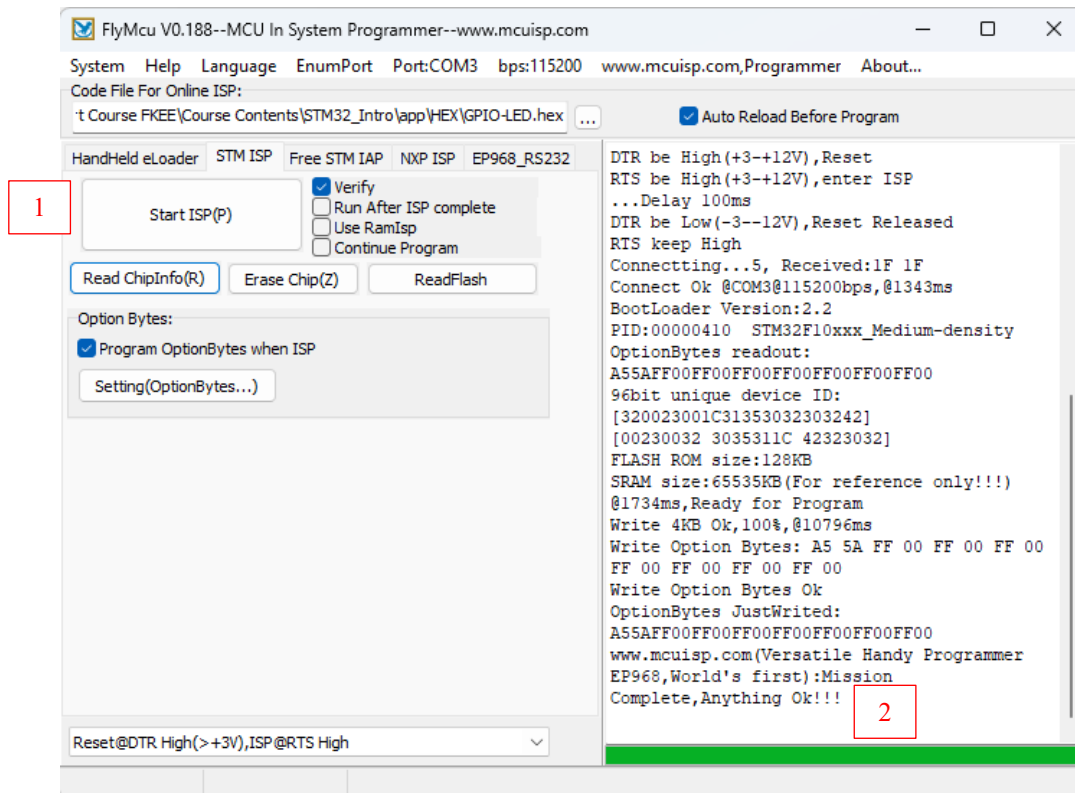
- Step 5. At MCUISP, click the button 1 – Start ISP(P). Wait till the operation is completed 2.
- Step 6. Hardware. Switch jumper BOOT0 to 0. Press reset button. MCU would switch to Main Flash memory.

Info:

Switch BOOT0 from 0→1→0 is done manually for this training board (Blue Pill – STM32F103C8T6) because no flow control circuitry is designed. Other development with flow control (DTR, RTS) circuitry, switching between Main Flash memory and System memory is completed electronically.



BOOT0 → 1
Press Reset



13.2.2 Procedures to Read/Read ChipInfo

The same procedures as Write.

For Read, save Bin file in location.

13.3 ST Tools

Some tools provided by ST.

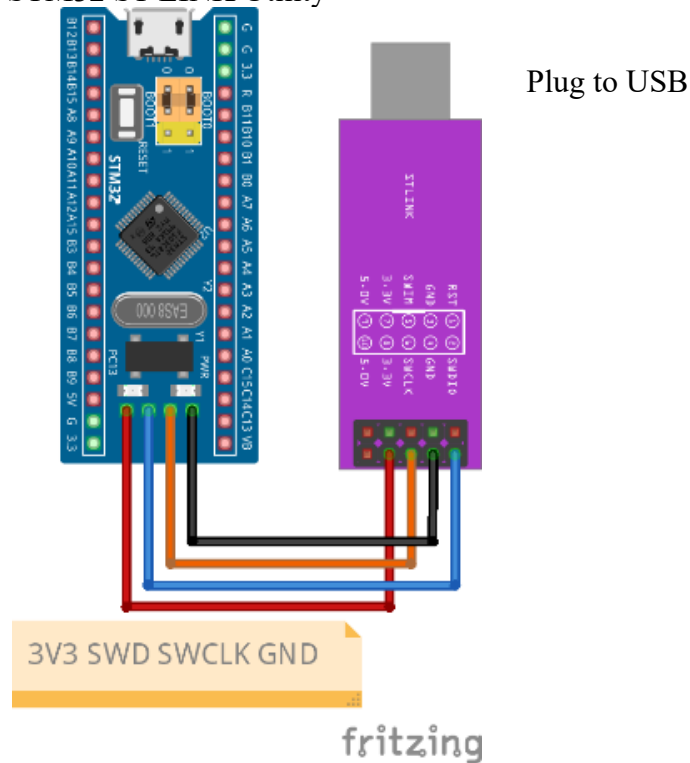
1. STM32 Flash loader demonstrator
<https://www.st.com/en/development-tools/flasher-stm32.html#get-software>
2. STM32CubeProgrammer software for all STM32
<https://www.st.com/en/development-tools/stm32cubeprog.html#get-software>

14 STM32 ST-LINK Utility

Alternatively, we could use **STM32 ST-LINK Utility** (STSW-LINK004) via the SWD interface. It is a full-featured software interface for programming STM32 microcontrollers. Download the software and install it on the computer (Windows).

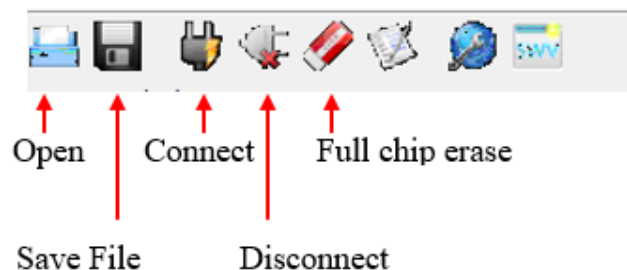
<https://www.st.com/en/development-tools/stsw-link004.html>

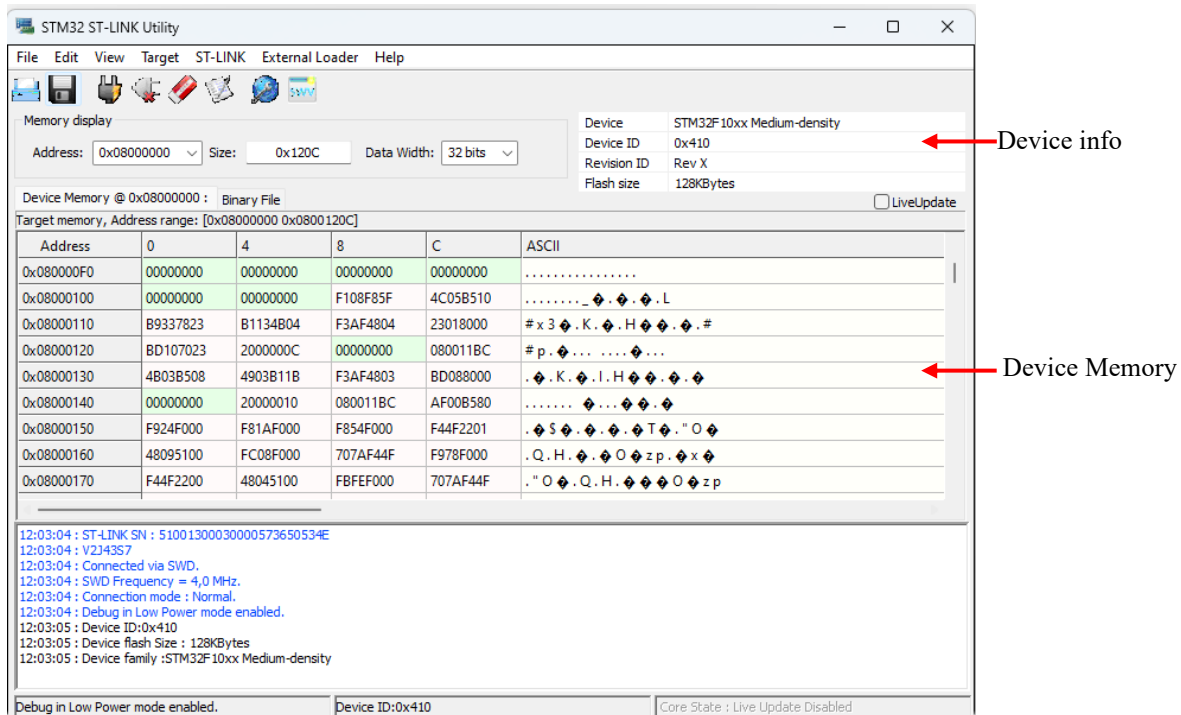
1. Connect ST-Link V2 as shown. [3V3, SWD, SWCLK, GND]
2. Plug in ST-Link V2 to the USB port. At Device Manager, under Universal Serial Bus devices | STM32 STLink should be detected. The MCU is powered up.
3. Open the software – STM32 ST-LINK Utility



14.1 Write HEX / Bin File to MCU

Step 1. At the app, click connect.





- Step 2. Open and choose a Hex file.
- Step 3. At menu, choose Target | Program & Verify. At pop-up, choose Start.
- Step 4. Observe the MCU.

14.2 Read HEX / BIN File and Erase Full Chip

The same procedures.

Save read into HEX or BIN file in location.

For full chip erase, the device memory would be set to 0xFF.

14.3 Why not STMCubeProgrammer?

The ST-Link used in practices is a third-party clone version; the latest STM32CubeProgrammer does not detect the serial number of this debugger/programmer.



Source: [stlink-debugging-and-programming-tools-overview.pdf](#)

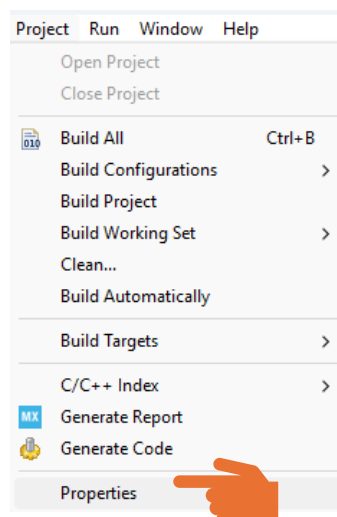
15 STM32CubeIDE – Build binary, hex files

For the developer, a tested project is likely to be saved in a binary file or a hex file. These files (either one) are used to share the project with others without exposing the source files. For production, these files are used to reflash a new MCU.

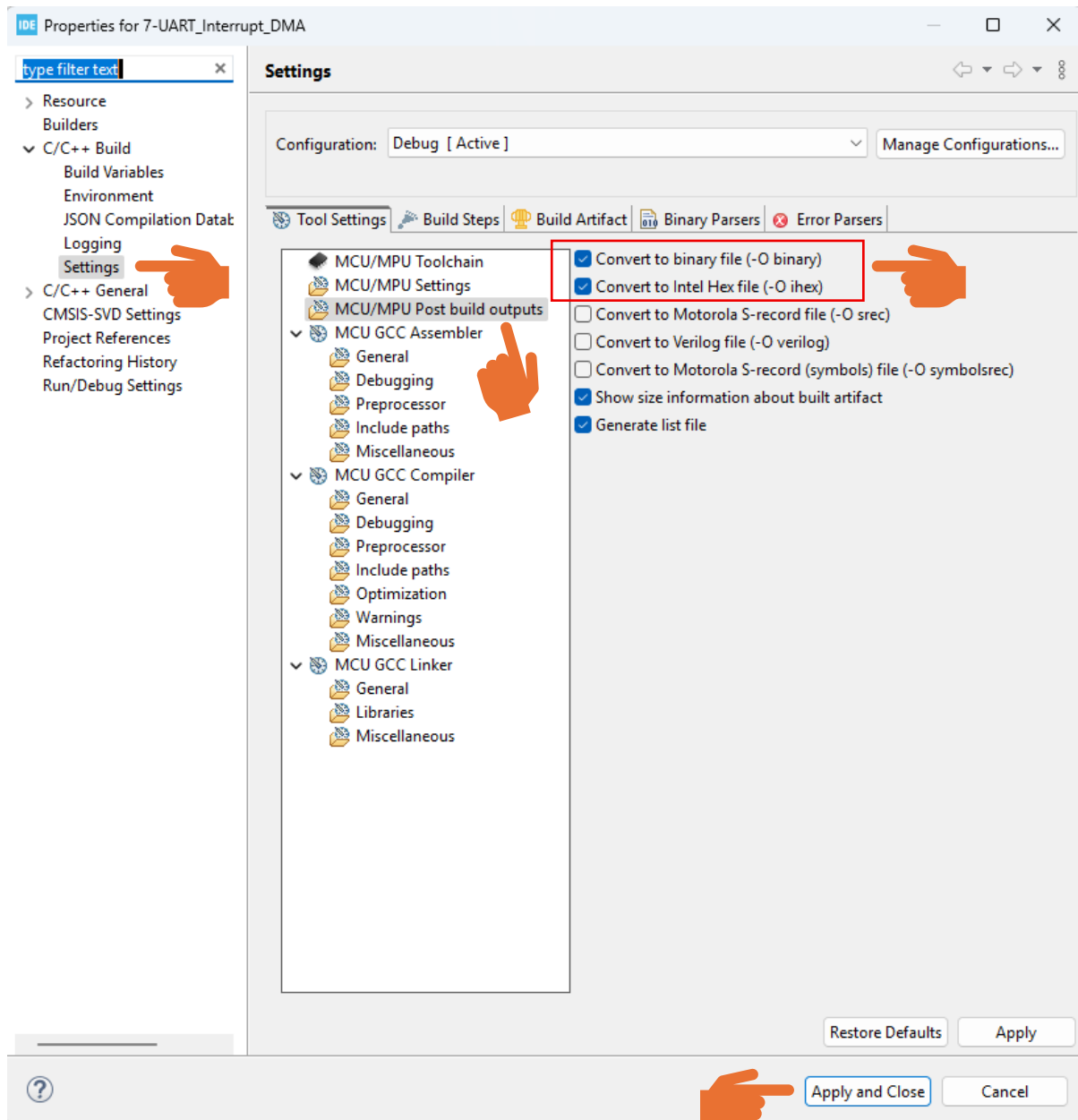
15.1 Option Setting at the IDE

To create a binary file and a hex files, check out and tick the options.
At any project, do:

Step 1: Click Project | Properties

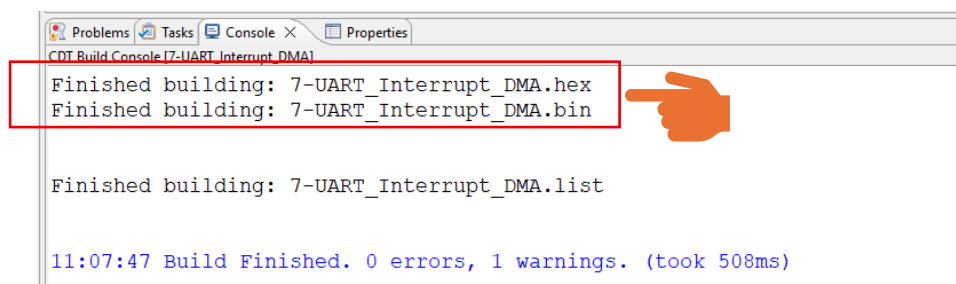


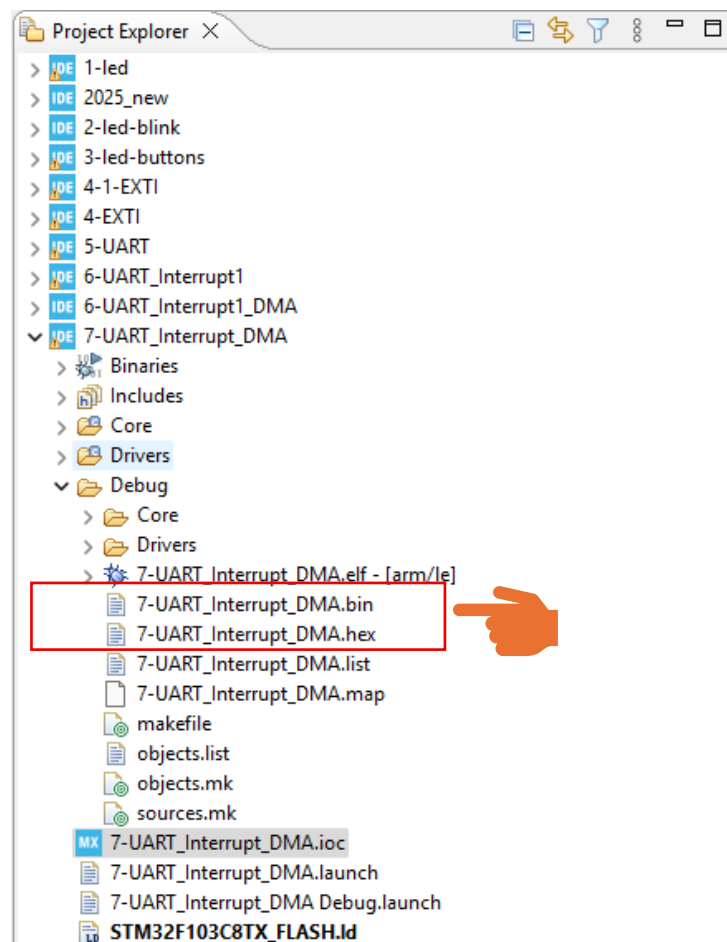
Step 2: Choose the options and save the settings



Step 3: Re-build the project

At the build console, the binary file and the hex file are created. They could be seen under the project explorer (on the left of the IDE) and allocated under the folder location (./Debug/)!





15.2 How to use the binary file and the hex file?

Jump to [\[13.2 Flashing Using UART Port\]](#).

16 Version Control

1. 1st Draft – April 2024
2. Rev 1 -June 2025

----- The text concludes at this point.-----