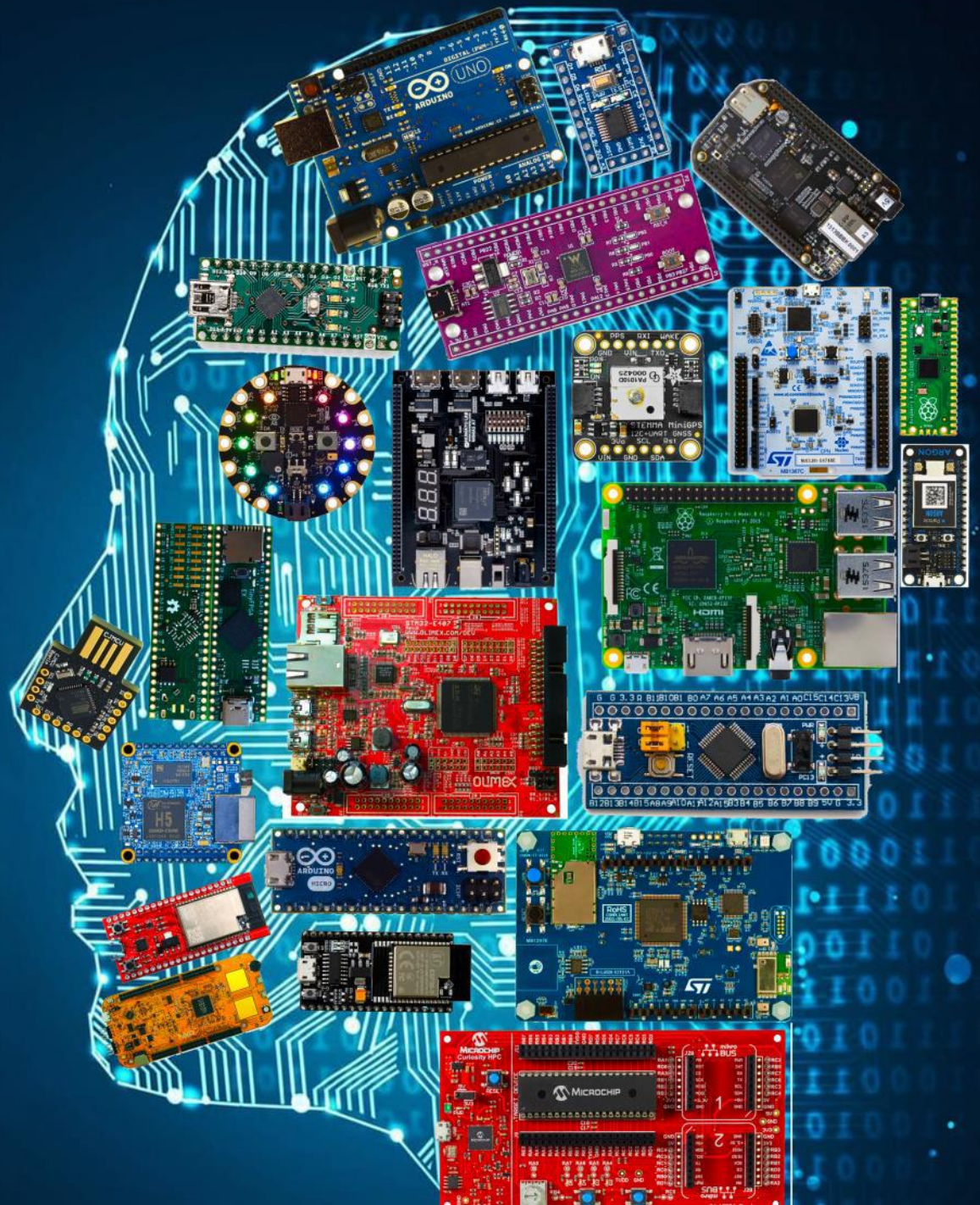
STM32F4xx Introduction



IE 5995: IoT and Edge AI Programming
Dr. Yanchao Liu
Industrial & Systems Engineering
Wayne State University

Agenda

- Learn the STM32F4xx NUCLEO board layout and features
- Install the development toolchain
- Create STM32 projects, configure, build, and download
- Understand the code organization
- Use GPIO for digital input and output
- Learn basic debugging functions
- Use Interrupts
- Use Timers and Timer-triggered Interrupts
- Use UART to transmit and receive data (poll, interrupt and DMA)
- Use ADC, PWM, DAC, DMA

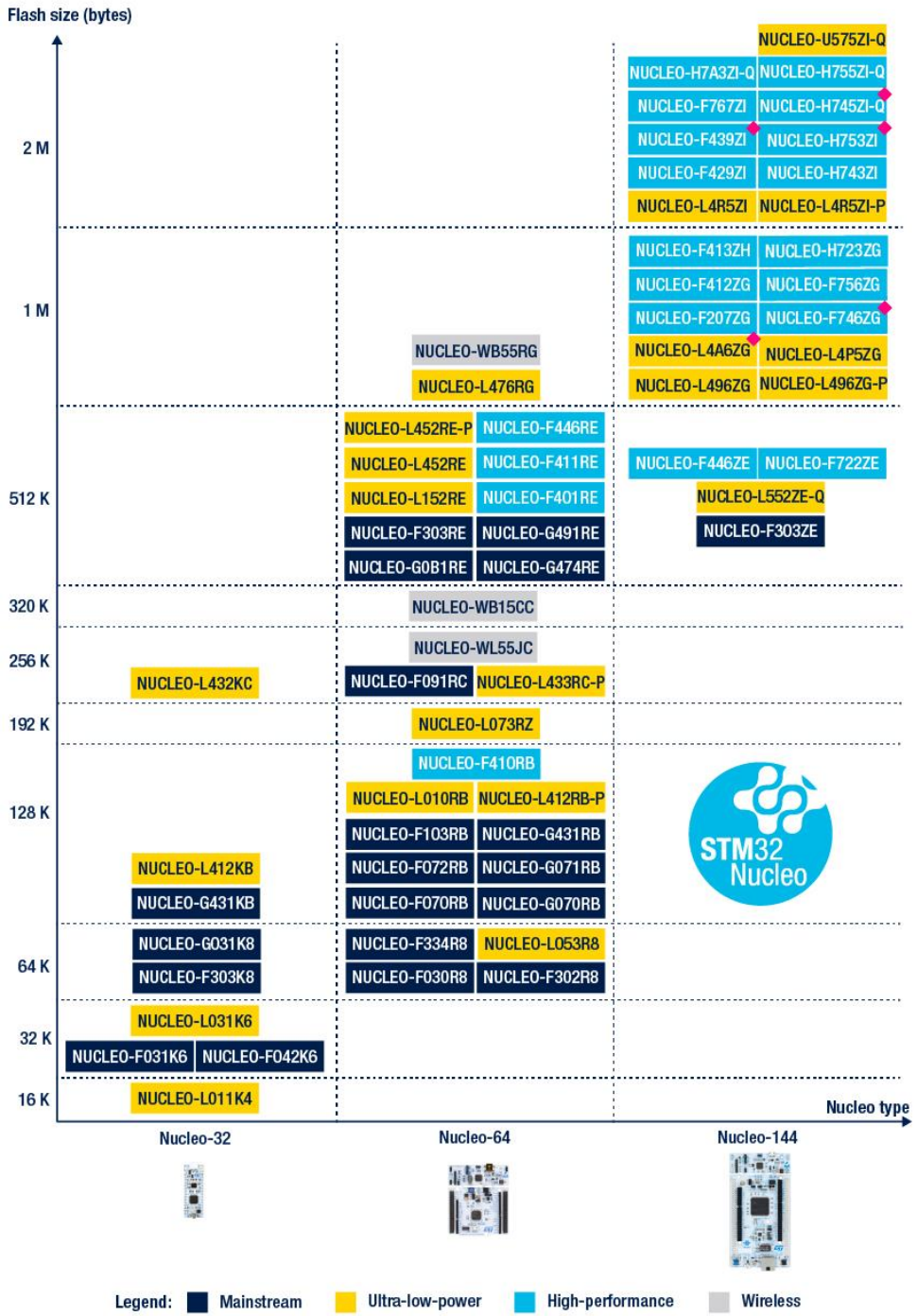
NUCLEO board codification

NUCLEO-XXYYZT NUCLEO-XXYYZT-P	Description	Example: NUCLEO-L452RE
XX	MCU series in STM32 Arm Cortex MCUs	STM32L4 Series
YY	MCU product line in the series	STM32L452
Z	STM32 package pin count • C for 48 pins • R for 64 pins	64 pins
T	STM32 Flash memory size: • 6 for 32 Kbytes • 8 for 64 Kbytes • B for 128 Kbytes • C for 256 Kbytes • E for 512 Kbytes • G for 1 Mbyte • Z for 192 Kbytes	512 Kbytes
-P	STM32 has external SMPS function	No SMPS

STM32 Nucleo-64 boards

Common features

- STM32 microcontroller in LQFP64 or LQFP48 package
- 1 user LED shared with ARDUINO®
- 1 user and 1 reset push-buttons
- 32.768 kHz crystal oscillator
- Board connectors:
  - ARDUINO® Uno V3 expansion connector
  - ST morpho extension pin headers for full access to all STM32 I/Os
- Flexible power-supply options: ST-LINK USB VBUS or external sources
- On-board ST-LINK debugger/programmer with USB re-enumeration
- Capability: mass storage, Virtual COM port, and debug port
- Comprehensive free software libraries and examples available with the
- STM32Cube MCU Package
- Support of a wide choice of Integrated Development Environments (IDEs)
  - Including IAR Embedded Workbench®, MDK-ARM, and STM32CubeIDE



STM32 Nucleo product line

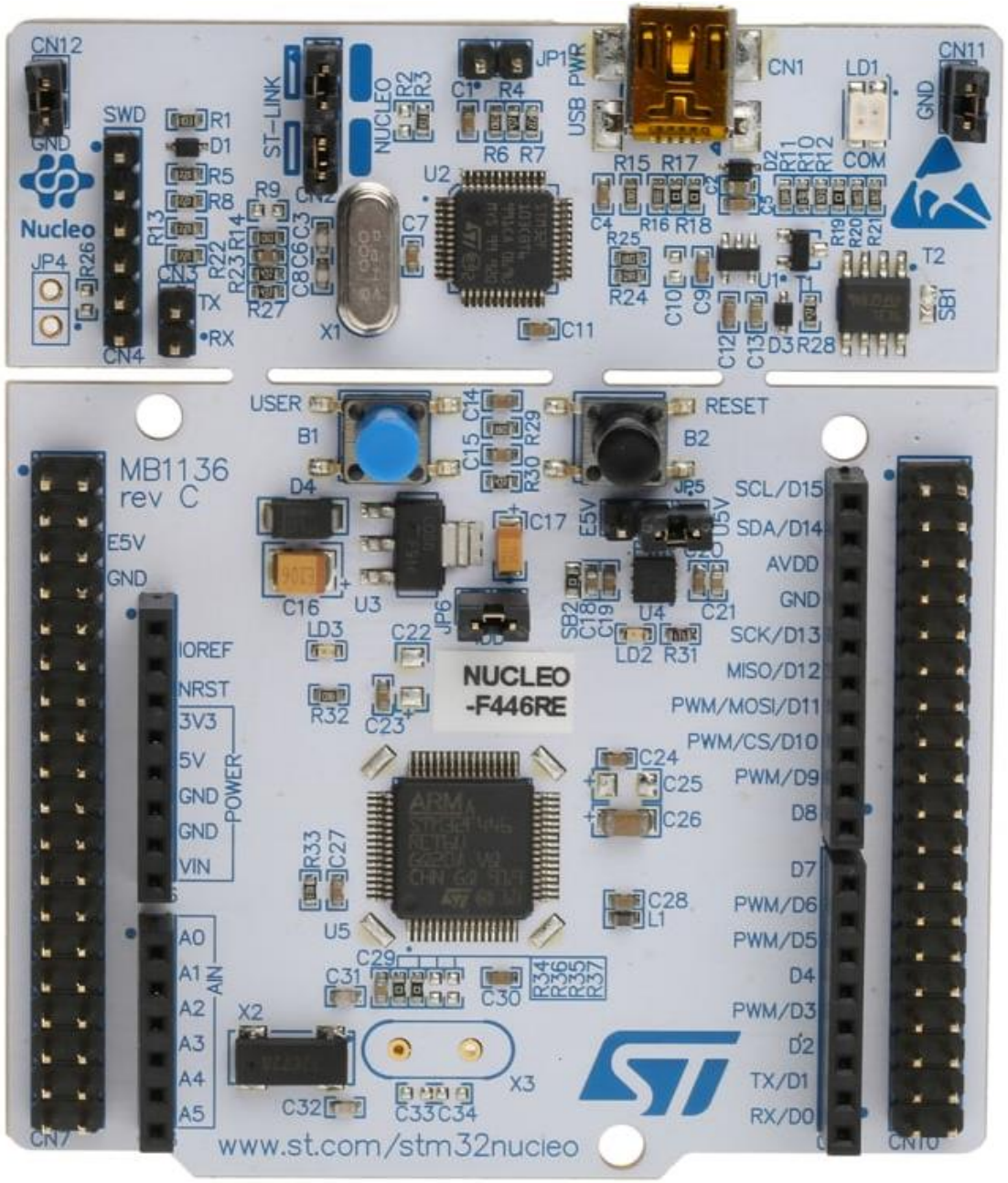
Development toolchains

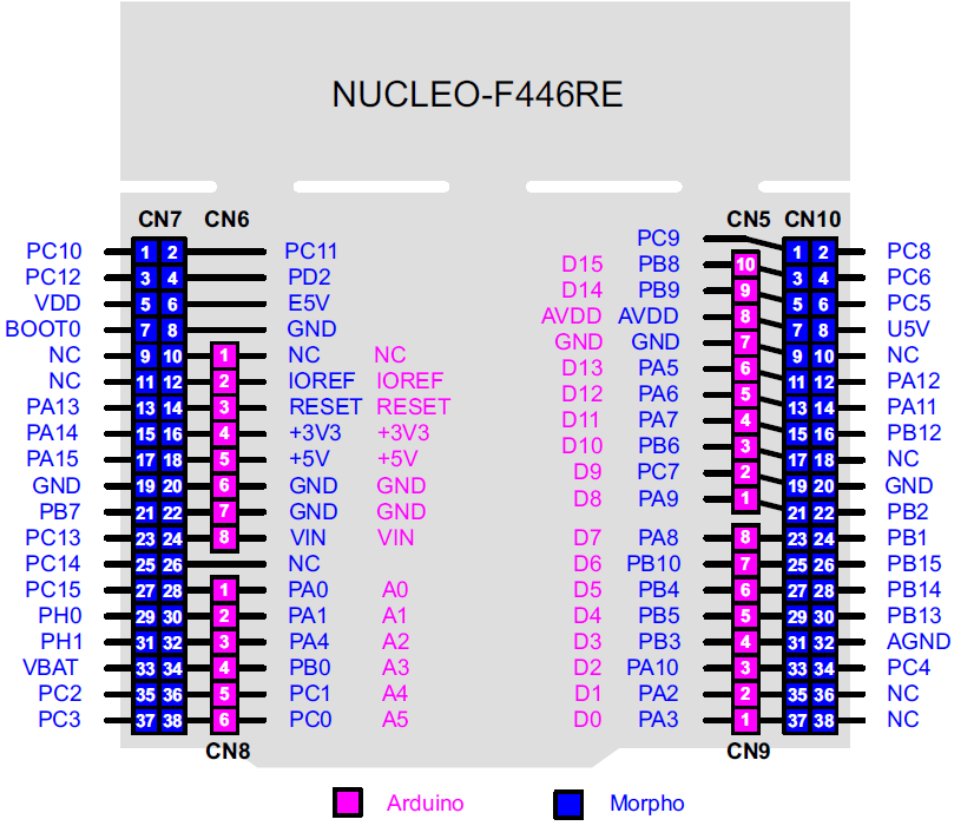
- IAR Systems® - IAR Embedded Workbench®
- Keil® - MDK-ARM
- STMicroelectronics - STM32CubeIDE

The first two are Windows only. We will use STM32CubeIDE.

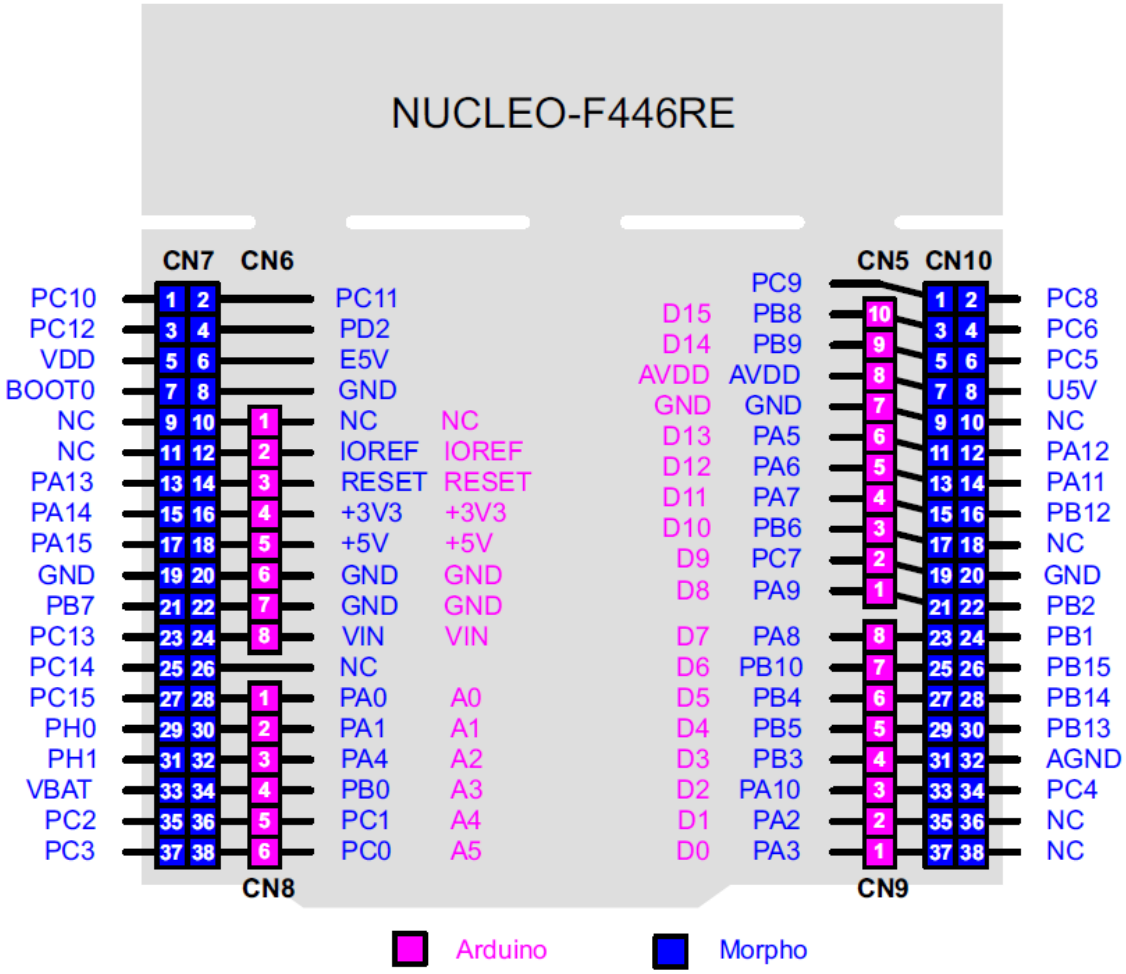
STM32F446RE NUCLEO

- Find these elements on the board
  - B1 (User button), B2 (Reset button)
  - LD1 (three-color led), LD2 (green led), LD3 (red led)





Understand the Pin labels



- PA10 means
  - GPIO Port A (GPIOA)
  - GPIO_PIN_10
- So, to write HIGH to PA10, call this function:
  - HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
- More on the functions later

Arduino connectors on NUCLEO-F446RE

CN5, CN6, CN8, and CN9 are female connectors compatible with ARDUINO® standard. Most shields designed for ARDUINO® can fit the STM32 Nucleo boards.

Connector	Pin	Pin name	STM32 pin	Function
Left connectors				
CN6 power	1	NC	-	-
	2	IOREF	-	3.3V Ref
	3	RESET	NRST	RESET
	4	+3.3V	-	3.3V input/output
	5	+5V	-	5V output
	6	GND	-	ground
	7	GND	-	ground
	8	VIN	-	Power input
CN8 analog	1	A0	PA0	ADC123_IN0
	2	A1	PA1	ADC123_IN1
	3	A2	PA4	ADC12_IN4
	4	A3	PB0	ADC12_IN8
	5	A4	PC1 or PB9(1)	ADC123_IN11 (PC1) or I2C1_SDA (PB9)
	6	A5	PC0 or PB8(1)	ADC123_IN10 (PC0) or I2C1_SCL (PB8)
Right connectors				
CN5 digital	10	D15	PB8	I2C1_SCL
	9	D14	PB9	I2C1_SDA

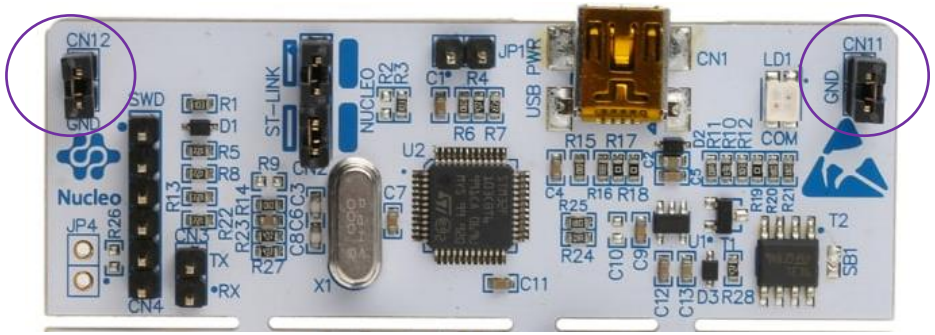
Connector	Pin	Pin name	STM32 pin	Function
CN5 digital	8	AREF	-	AVDD
	7	GND	-	ground
	6	D13	PA5	SPI1_SCK
	5	D12	PA6	SPI1_MISO
	4	D11	PA7	TIM14_CH1 || SPI1_MOSI
	3	D10	PB6	TIM4_CH1 || SPI1_CS
	2	D9	PC7	TIM8_CH2
	1	D8	PA9	-
CN9 digital	8	D7	PA8	-
	7	D6	PB10	TIM2_CH3
	6	D5	PB4	TIM3_CH1
	5	D4	PB5	-
	4	D3	PB3	TIM2_CH2
	3	D2	PA10	-
	2	D1	PA2	USART2_TX
	1	D0	PA3	USART2_RX

ST morpho connector on NUCLEO-F446RE

CN7 odd pins		CN7 even pins		CN10 odd pins		CN10 even pins	
Pin	Name	Name	Pin	Pin	Name	Name	Pin
1	PC10	PC11	2	1	PC9	PC8	2
3	PC12	PD2	4	3	PB8	PC6	4
5	VDD	E5V	6	5	PB9	PC5	6
7	BOOT0(1)	GND	8	7	AVDD	U5V(2)	8
9	-	-	10	9	GND	-	10
11	-	IOREF	12	11	PA5	PA12	12
13	PA13(3)	RESET	14	13	PA6	PA11	14
15	PA14(3)	+3.3V	16	15	PA7	PB12	16
17	PA15	+5V	18	17	PB6	-	18
19	GND	GND	20	19	PC7	GND	20
21	PB7	GND	22	21	PA9	PB2	22
23	PC13	VIN	24	23	PA8	PB1	24
25	PC14	-	26	25	PB10	PB15	26
27	PC15	PA0	28	27	PB4	PB14	28
29	PH0	PA1	30	29	PB5	PB13	30
31	PH1	PA4	32	31	PB3	AGND	32
33	VBAT	PB0	34	33	PA10	PC4	34
35	PC2	PC1 or PB9(4)	36	35	PA2	-	36
37	PC3	PC0 or PB8(4)	38	37	PA3	-	38

1. The default state of BOOT0 is LOW. It can be set to HIGH when a jumper is on pin5-7 of CN7. Two unused jumpers are available on CN11 and CN12 (bottom side of the board).
2. U5V is 5 V power from ST-LINK/V2-1 USB connector and it rises before +5V.
3. PA13 and PA14 share with SWD signals connected to ST-LINK/V2-1, it is not recommended to use them as IO pins if the ST-LINK part is not cut.

Spare jumpers



LEDs on the NUCLEO board

The tricolor LED (green, orange, red) LD1 (COM) provides information about ST-LINK communication status. LD1 default color is red. LD1 turns to green to indicate that communication is in progress between the PC and the ST-LINK/V2-1, with the following setup:

- Slow blinking Red/Off: at power-on before USB initialization
- Fast blinking Red/Off: after the first correct communication between the PC and ST-LINK/V2-1 (enumeration)
- Red LED On: when the initialization between the PC and ST-LINK/V2-1 is complete
- Green LED On: after a successful target communication initialization
- Blinking Red/Green: during communication with the target
- Green On: communication finished and successful
- Orange On: Communication failure

User LD2: the green LED is a user LED connected to ARDUINO® signal D13 corresponding to STM32 I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to *Table 11* to *Table 23* when:

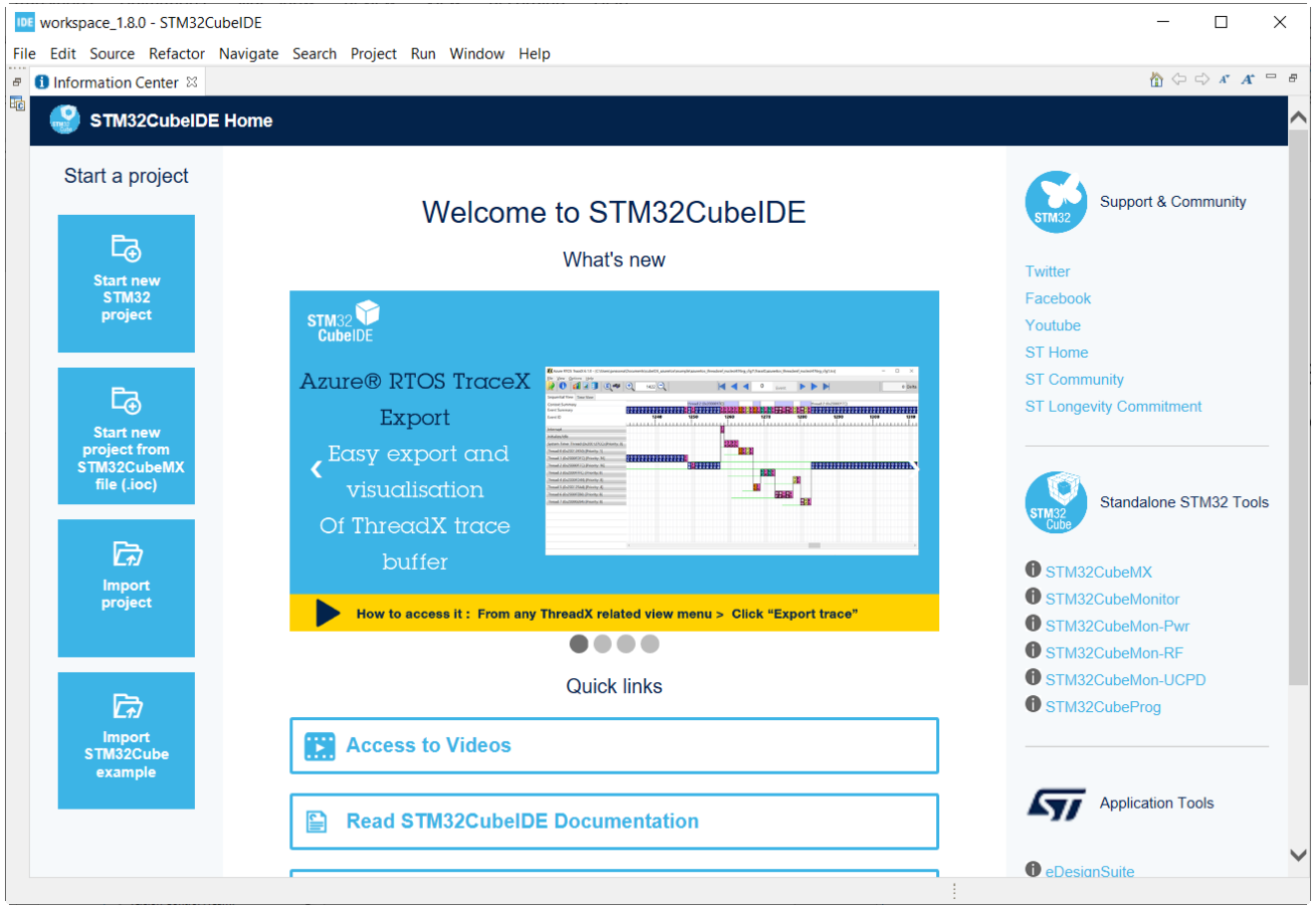
- the I/O is HIGH value, the LED is on
- the I/O is LOW, the LED is off

LD3 PWR: the red LED indicates that the STM32 part is powered and +5V power is available.

Toolchain installation

- We need to install the following software on the PC
  - STM32CubeProgrammer
    - https://www.st.com/en/development-tools/stm32cubeprog.html
  - STM32CubeIDE
    - https://www.st.com/en/development-tools/stm32cubeide.html

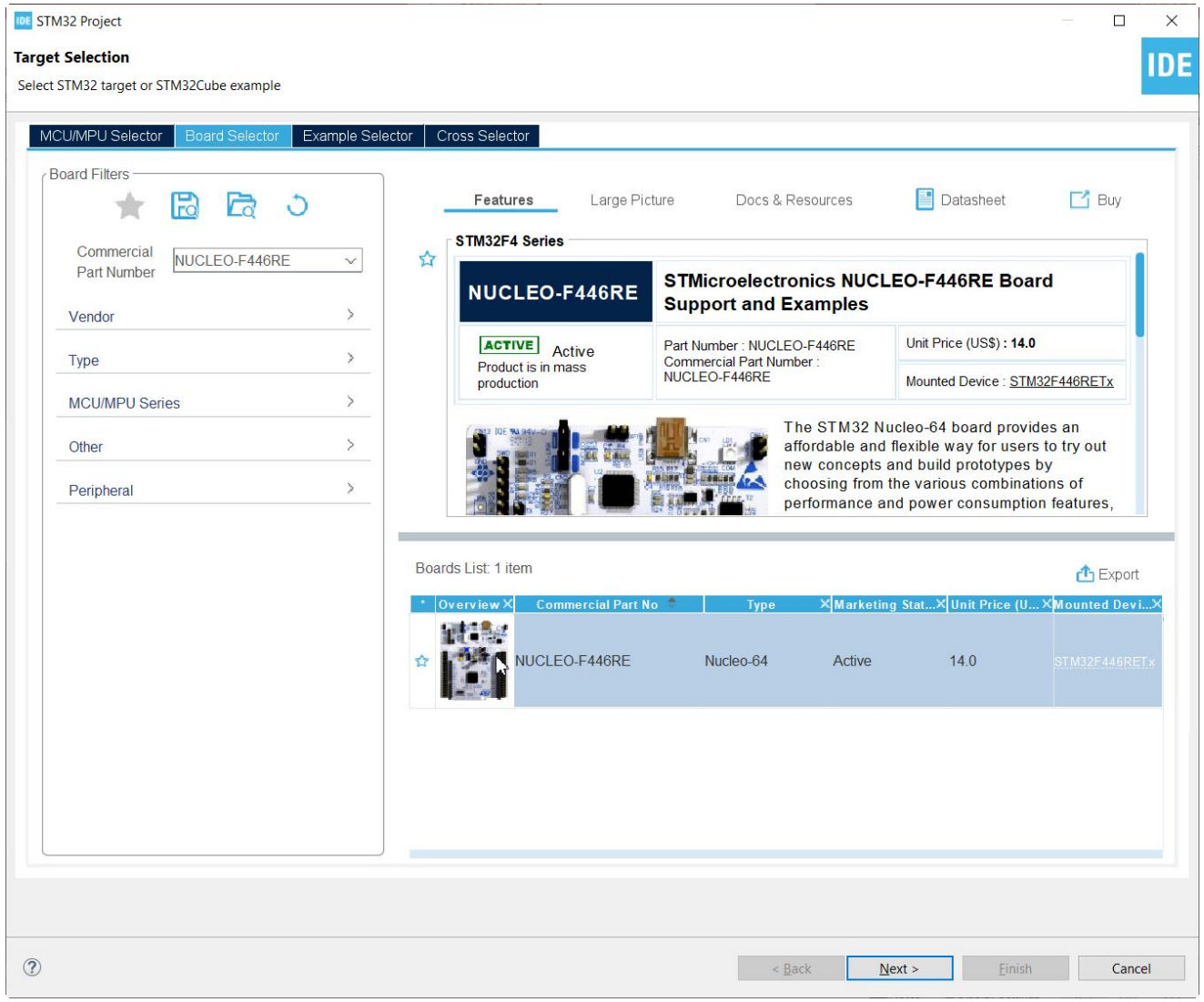
STM32CubeIDE



- Click "Start new STM32 project"
- Some additional downloads will happen

Create a project using STM32CubeIDE

- Click the “Board Selector” tab
- Select the board NUCLEO-F446RE from the dropdown menu
- Before clicking “Next”, download the manuals from Docs & Resources
  - The docs will be saved to {HOME}/STM32Cube/Repository
  - Take a quick scan of the docs to see what info is in there, for future reference
- Click “Next”

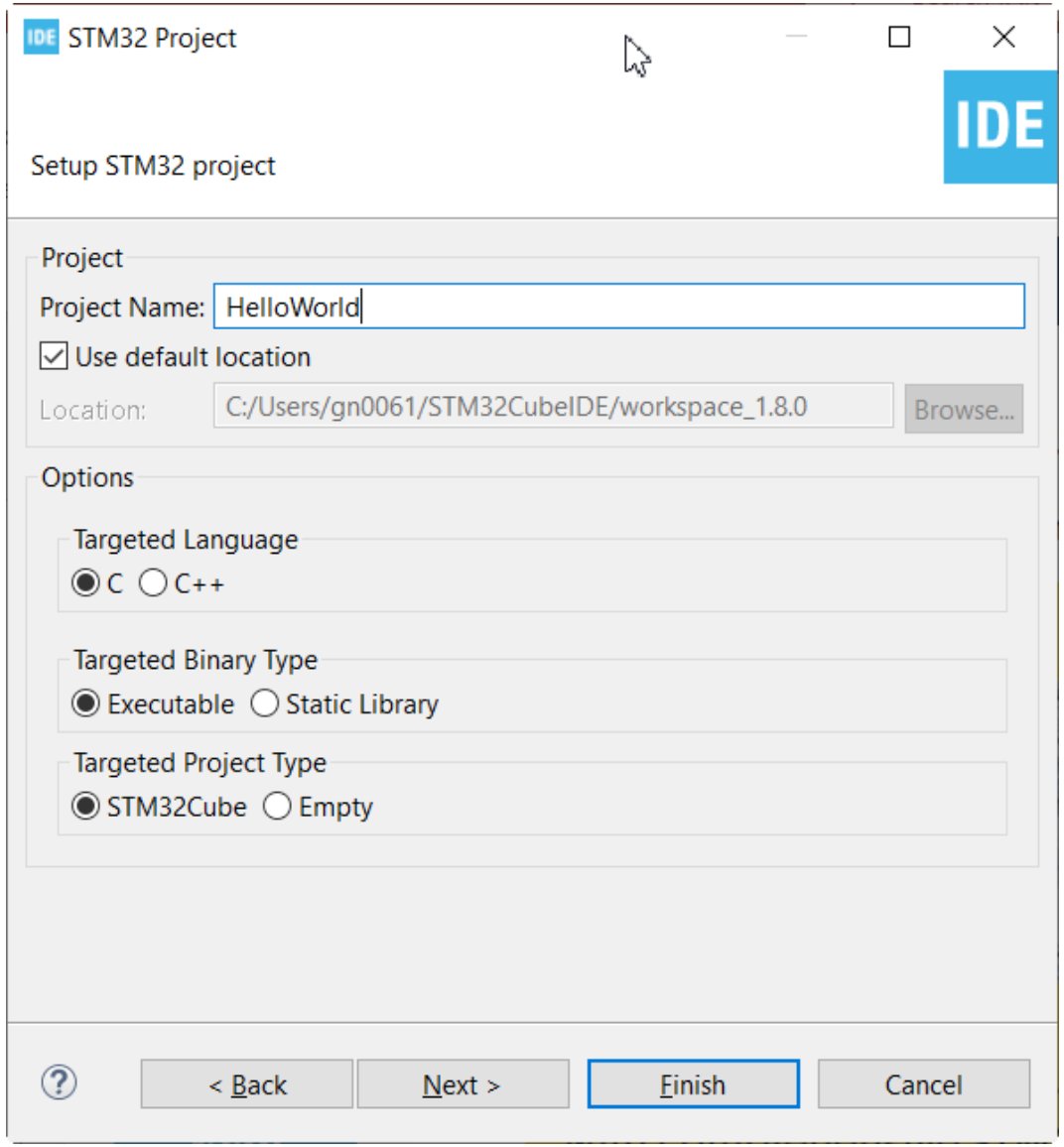


Some important points from the documents

- The STM32 Nucleo board and its shield can consume a maximum of 300 mA current, not more.
- The USART2 interface available on PA2 and PA3 of the STM32 microcontroller can be connected to ST-LINK MCU, ST morpho connector, or to ARDUINO® connector. The choice can be changed by setting the related solder bridges. By default, the USART2 communication between the target STM32 and ST-LINK MCU is enabled, in order to support virtual COM port for Mbed™ (SB13 and SB14 ON, SB62 and SB63 OFF).
- If the communication between the target STM32 PA2 (D1) or PA3 (D0) and shield or extension board is required, SB62 and SB63 must be ON, while SB13 and SB14 must be OFF.

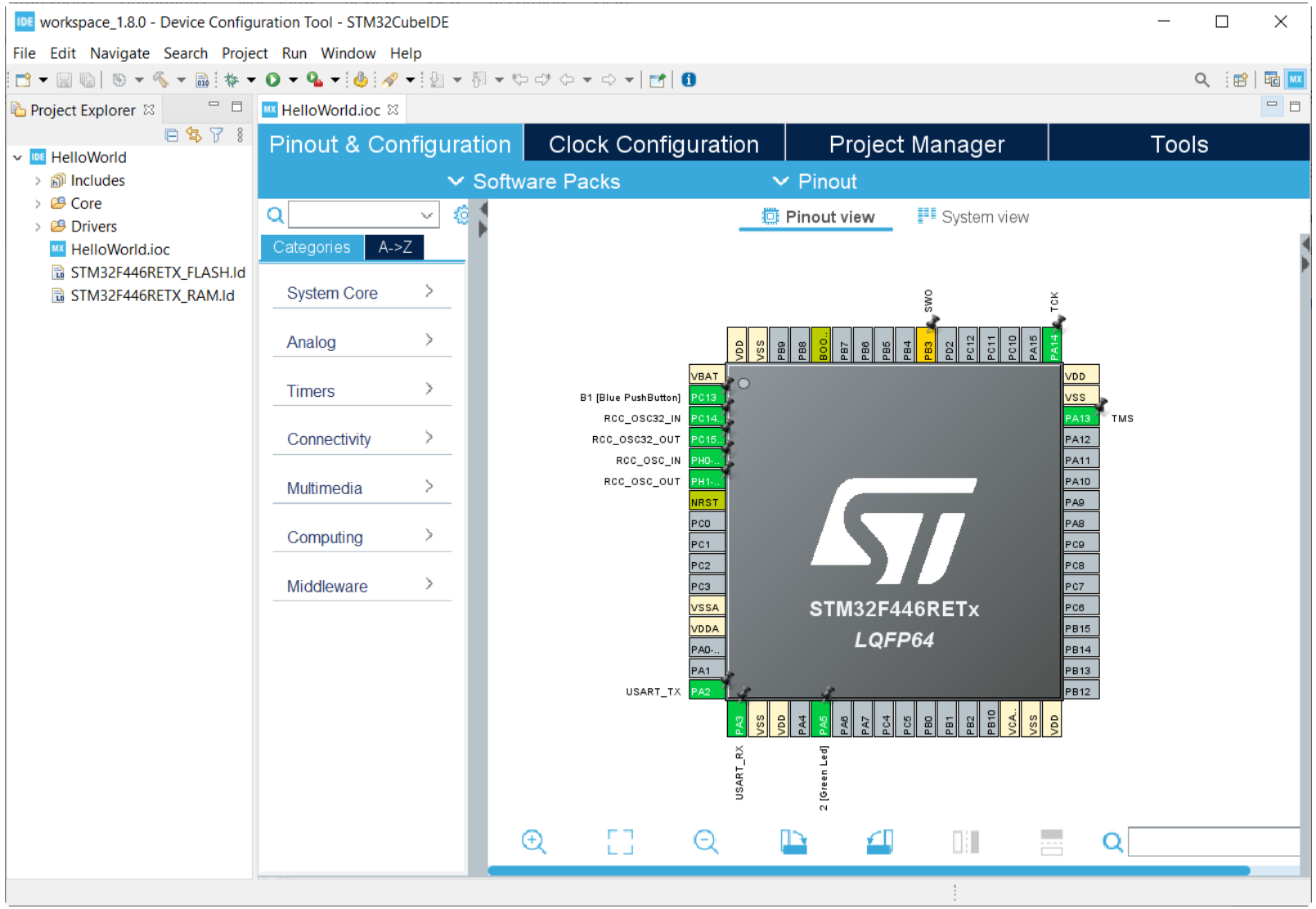
HelloWorld project

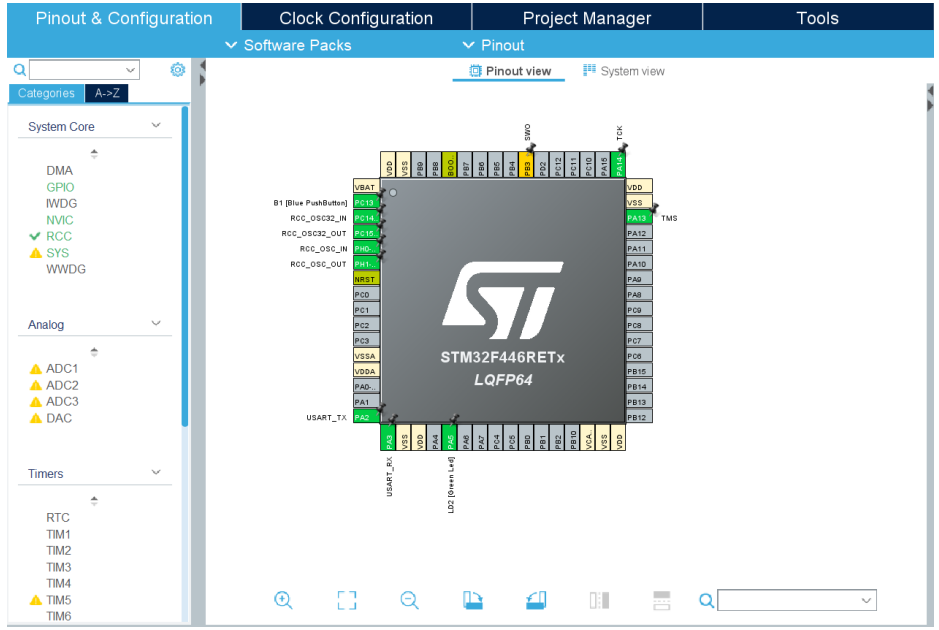
- Type in the project name, e.g., HelloWorld
- Click "Finish"
- Click "Yes" to initialize peripherals with their default mode.
- Click "Yes" to open the Device Configuration Tool perspective.
  - Before the perspective is loaded, some additional downloads (~700MB) will occur

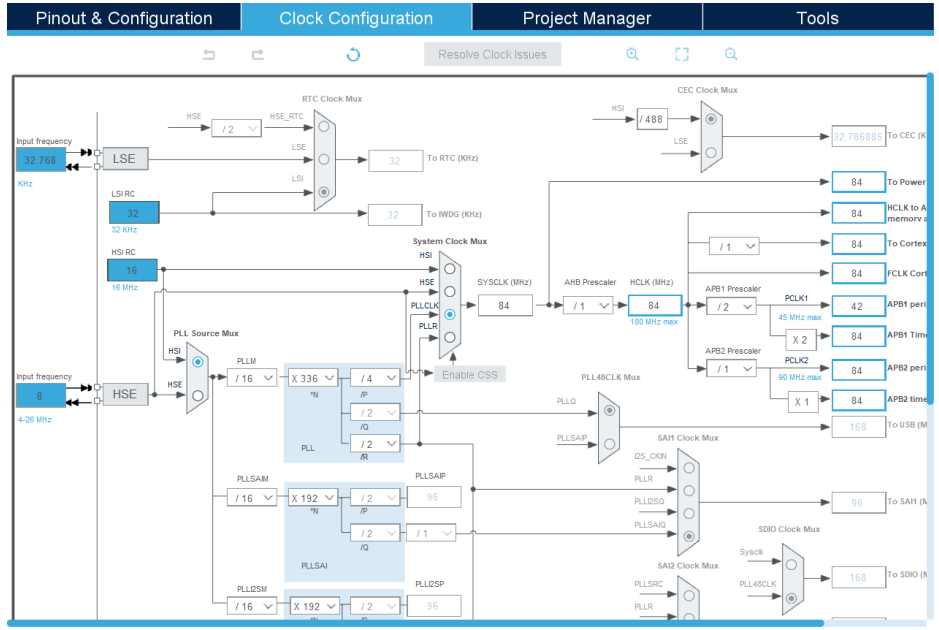


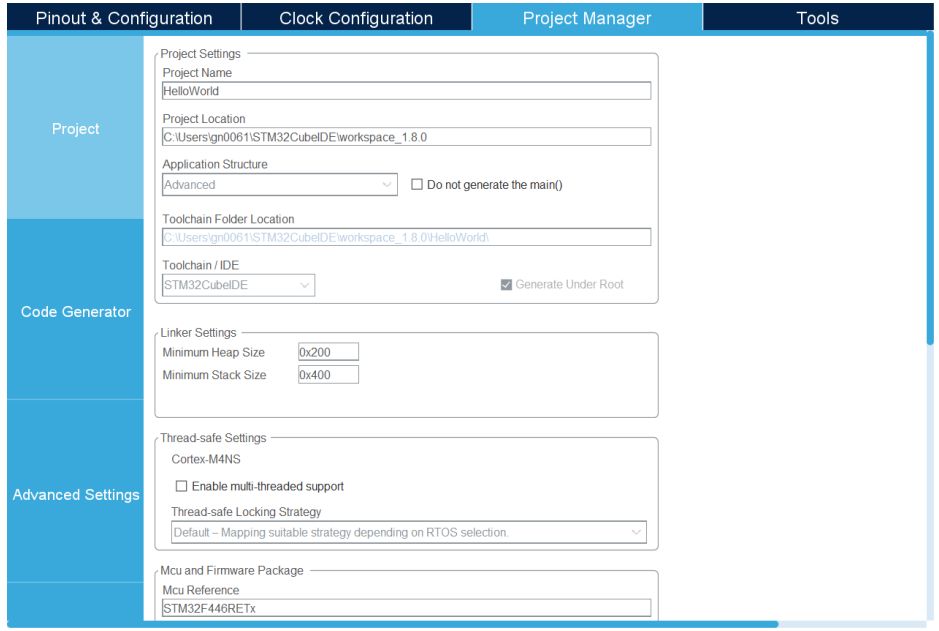
Configure and generate code

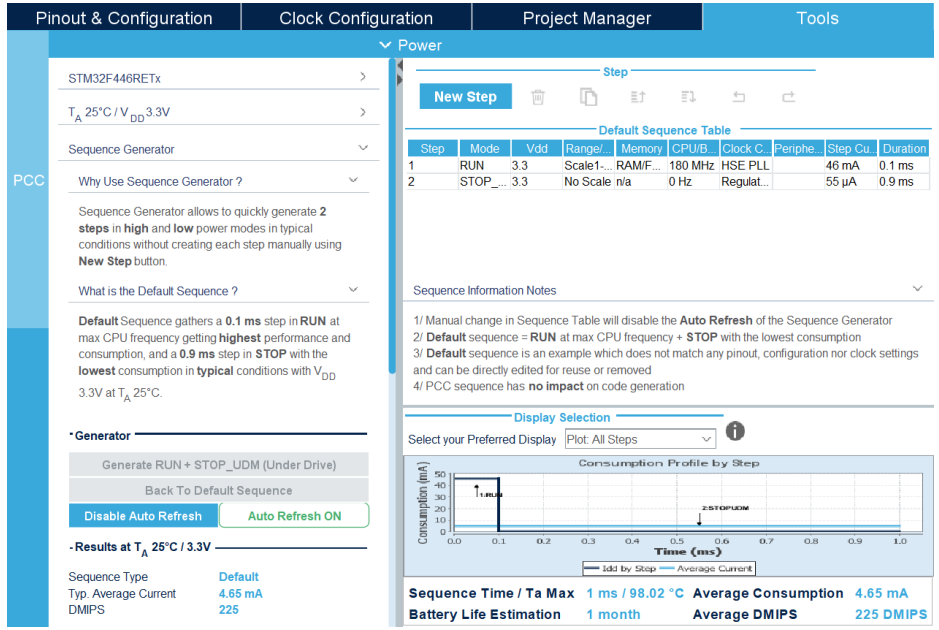
- Feel free to navigate the four tabs:
  - Pintout & Configuration
  - Clock Configuration
  - Project Manager
  - Tools
- Next slide show the four tabs
- Don't change anything
- In menu, click Project -> Generate Code
- Click "Yes" to open the code perspective




Pinout & Configuration
Clock Configuration
Project Manager
Tools
Software Packs
Pinout
Pinout view
System view
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
ADC1
ADC2
ADC3
DAC
Timers
RTC
TIM1
TIM2
TIM3
TIM4
TIM5
TIM6
STM32F446RETx
LQFP64


Pinout & Configuration
Clock Configuration
Project Manager
Tools
Resolve Clock Issues
RTC Clock Mux
CEC Clock Mux
System Clock Mux
PLL Source Mux
PLL48CLK Mux
SAI1 Clock Mux
SAI2 Clock Mux
SDIO Clock Mux
Enable CSS


Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
Project Settings
Project Name
HelloWorld
Project Location
C:\Users\gn0061\STM32CubeIDE\workspace_1.8.0
Application Structure
Advanced
Do not generate the main()
Toolchain Folder Location
C:\Users\gn0061\STM32CubeIDE\workspace_1.8.0\HelloWorld\
Toolchain / IDE
STM32CubeIDE
Generate Under Root
Linker Settings
Minimum Heap Size
0x200
Minimum Stack Size
0x400
Thread-safe Settings
Cortex-M4NS
Enable multi-threaded support
Thread-safe Locking Strategy
Default – Mapping suitable strategy depending on RTOS selection.
Mcu and Firmware Package
Mcu Reference
STM32F446RETx


Pinout & Configuration
Clock Configuration
Project Manager
Tools
Power
PCC
STM32F446RETx
Sequence Generator
Why Use Sequence Generator ?
Sequence Generator allows to quickly generate 2 steps in high and low power modes in typical conditions without creating each step manually using New Step button.
What is the Default Sequence ?
Default Sequence gathers a 0.1 ms step in RUN at max CPU frequency getting highest performance and consumption, and a 0.9 ms step in STOP with the lowest consumption in typical conditions with VDD 3.3V at TA 25°C.
Generator
Generate RUN + STOP_UDM (Under Drive)
Back To Default Sequence
Disable Auto Refresh
Auto Refresh ON
Results at TA 25°C / 3.3V
Sequence Type Default
Typ. Average Current 4.65 mA
DMIPS 225
Step
New Step
Default Sequence Table
Step Mode Vdd Range/ Memory CPU/B... Clock C... Periphe... Step Cu. Duration
1 RUN 3.3 Scale1-... RAM/F... 180 MHz HSE PLL 46 mA 0.1 ms
2 STOP_... 3.3 No Scale n/a 0 Hz Regulat... 55 µA 0.9 ms
Sequence Information Notes
1/ Manual change in Sequence Table will disable the Auto Refresh of the Sequence Generator
2/ Default sequence = RUN at max CPU frequency + STOP with the lowest consumption
3/ Default sequence is an example which does not match any pinout, configuration nor clock settings and can be directly edited for reuse or removed
4/ PCC sequence has no impact on code generation
Display Selection
Select your Preferred Display
Plot: All Steps
Consumption Profile by Step
Consumption (mA)
Time (ms)
Idd by Step
Average Current
Sequence Time / Ta Max 1 ms / 98.02 °C
Average Consumption 4.65 mA
Battery Life Estimation 1 month
Average DMIPS 225 DMIPS

Code perspective

- Always write user code within the

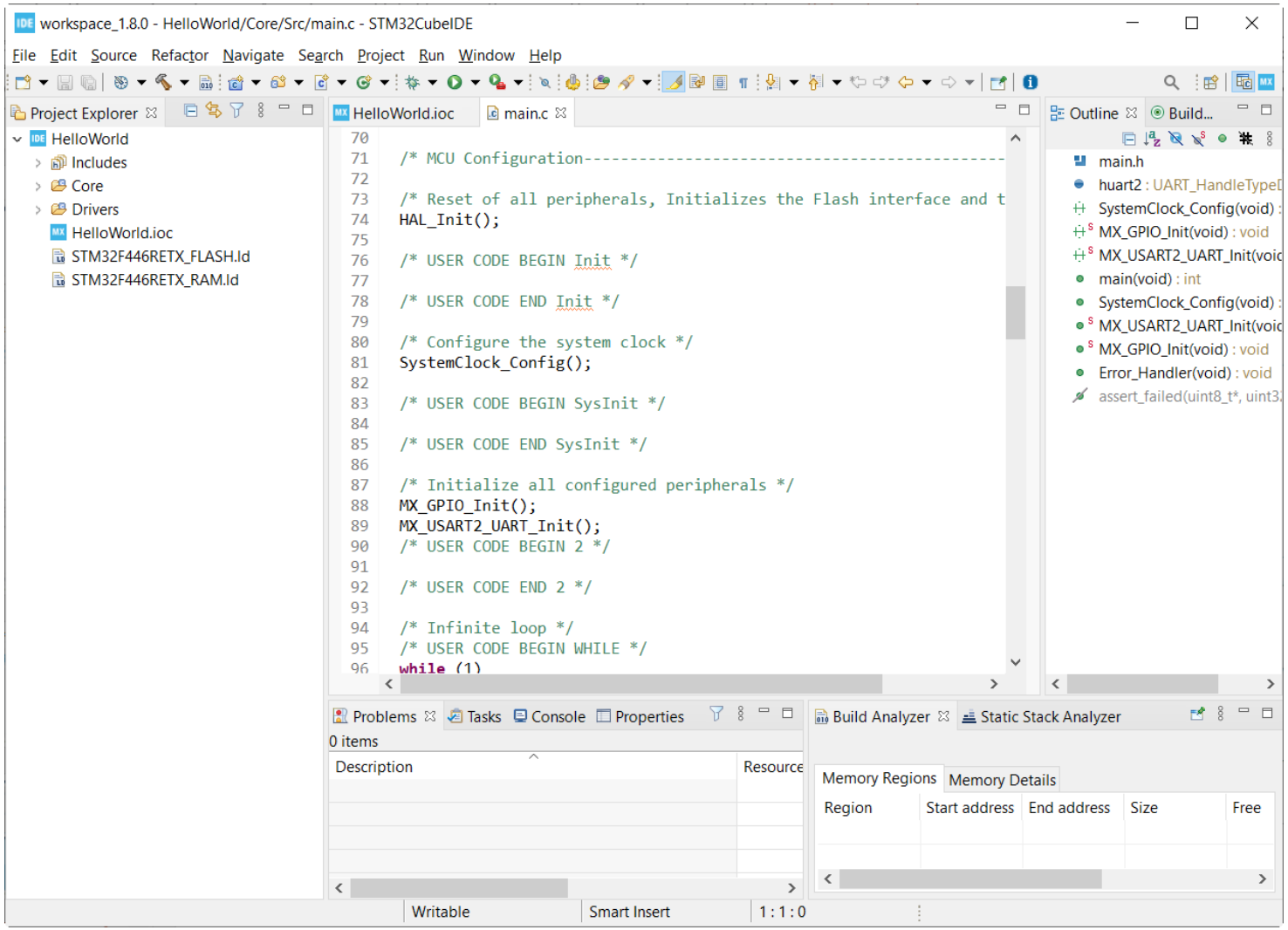
  /* USER CODE BEGIN .. */

  …

  /* USER CODE END .. */

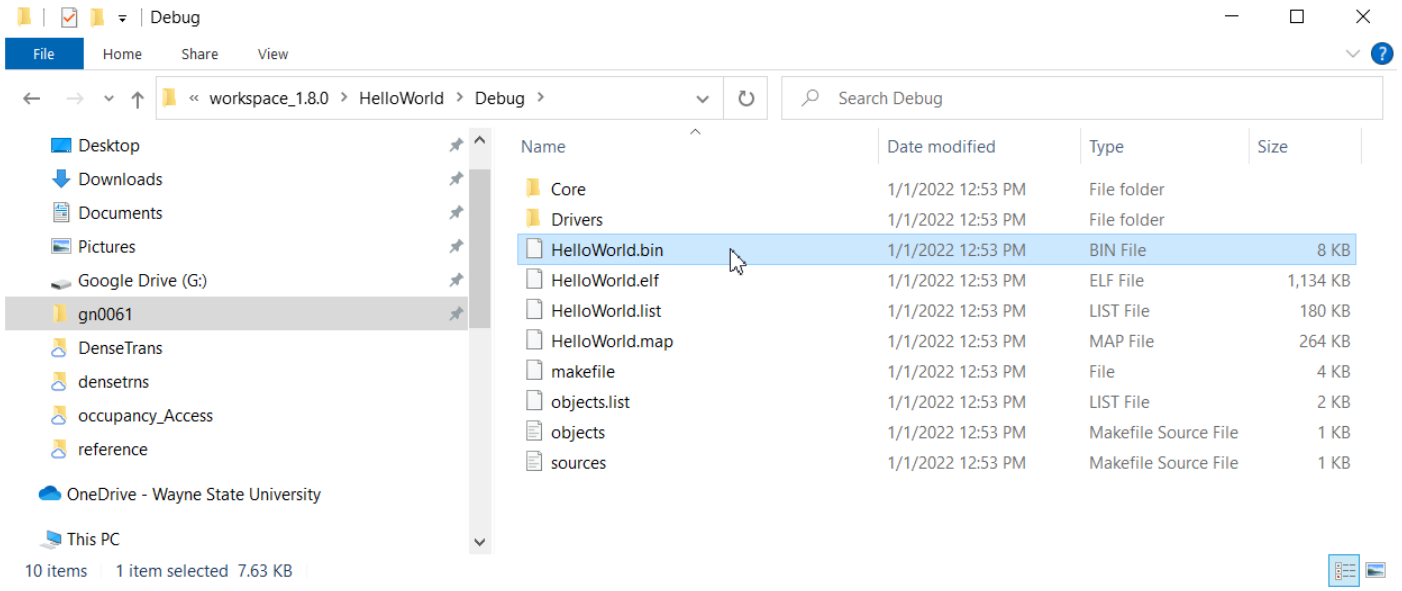
blocks.

- User code written outside such blocks will be wiped next time "Generate Code" is run after some configuration change.



Build the project

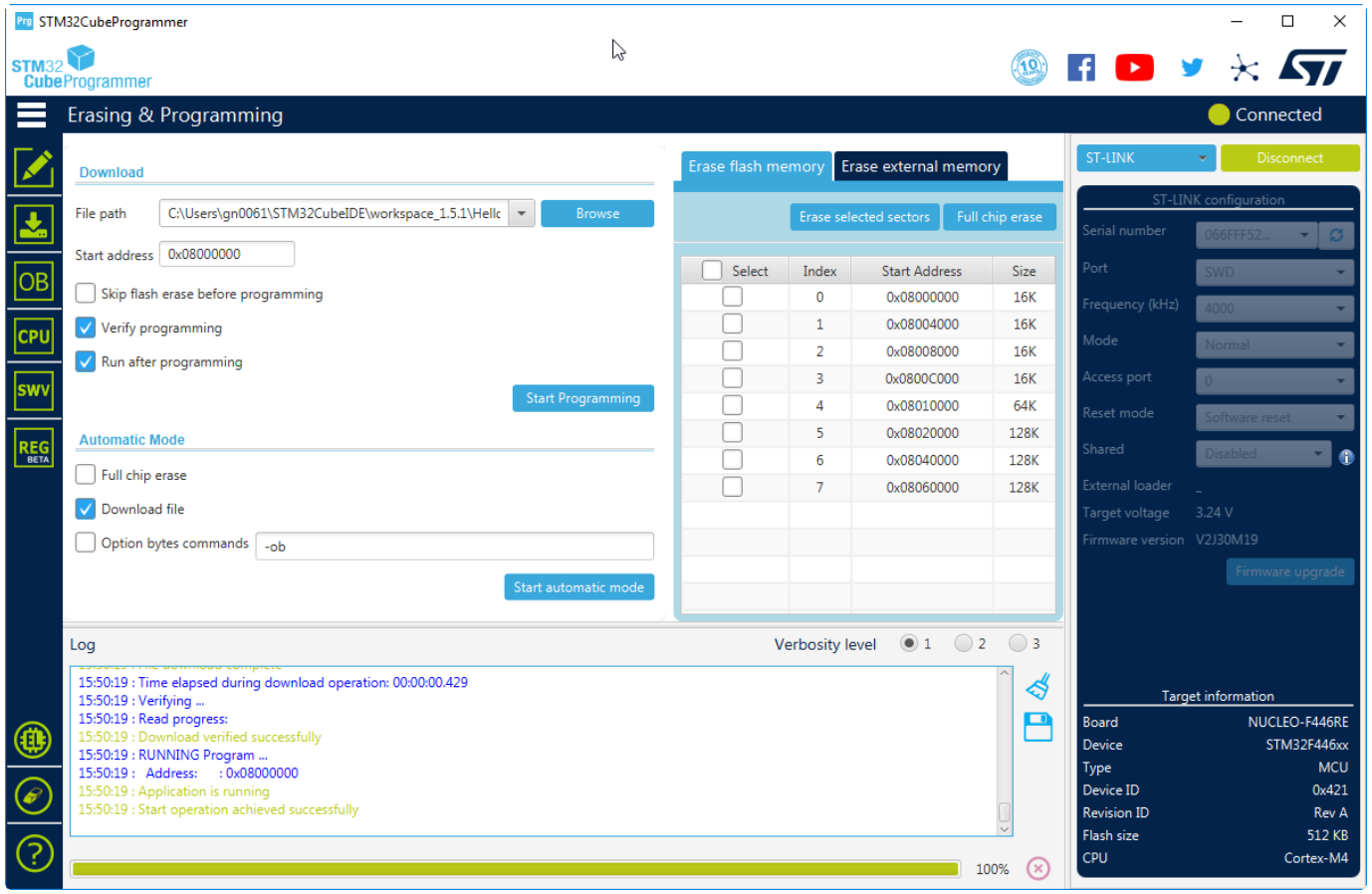
- For now, don't make any change to the code
- In the menu, click "Project -> Build Project"
- The build should finish quickly with no error or warning.
- In File Explore, go to the project folder
  - E.g., C:\Users\gn0061\STM32CubeIDE\workspace_1.8.0\HelloWorld\
- **HelloWorld.bin** is the file to send to the board, through the STM32CubeProgrammer



Download the program to the board

- Open the Programmer
- Connect to board (ST-LINK)
- Go to “Erasing & Programming”
- Browse and select the HelloWorld.bin file.
- Click “Start Programming”

Note: whatever content that was previously on the board will be erased.
After the programming, the board is reset and the new program will run automatically.
In this case, the program does nothing.

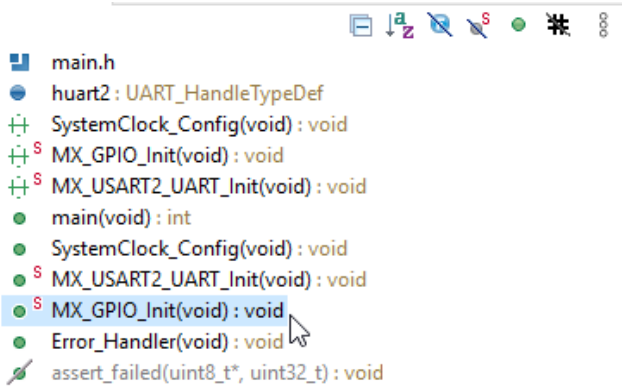


We’ve successfully created our first program on the board.
Whenever the board is powered, the program will run automatically.

main.c

- We can see that B1 (the user button) and LD2 (the green led) are enabled, and the LD2 is reset

```c
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
```



Note how the **GPIO_InitStruct** is configured. We will see details of this later.

GPIO_PIN_RESET means 0
GPIO_PIN_SET means 1

main.c

- The USART2 is enabled (connected to PA2 and PA3)

```c
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
```

main.h
huart2 : UART_HandleTypeDef
SystemClock_Config(void) : void
MX_GPIO_Init(void) : void
MX_USART2_UART_Init(void) : void
main(void) : int
SystemClock_Config(void) : void
MX_USART2_UART_Init(void) : void
MX_GPIO_Init(void) : void
Error_Handler(void) : void
assert_failed(uint8_t*, uint32_t) : void

GPIO Mode and Configuration

Configuration

Group By Peripherals

SYS | USART | NVIC
GPIO | Single Mapped Signals | RCC

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin ...	Signal on Pin	GPIO ou...	GPIO m...	GPIO P...	Maximu...	User ...	Modif...
PA2	USART2_TX	n/a	Alternat...	No pull-u...	Very High	USA...	☑
PA3	USART2_RX	n/a	Alternat...	No pull-u...	Very High	USA...	☑

main.c

- Check out the main function
- Add some functionality
  - Add the following two lines in the while loop

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

  /* USER CODE BEGIN 3 */
  HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  HAL_Delay(500);
}
/* USER CODE END 3 */
```

```c
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 */

  /* USER CODE END 2 */

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

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

The names such as “LD2_GPIO_Port” and “LD2_Pin” are defined in main.h file (which was generated automatically)

Save, Build Project, then download the HelloWorld.bin file to the board. The green led should blink (toggle on/off every 500 ms). If the program does not run automatically, press the Reset button on the board.

Definitions

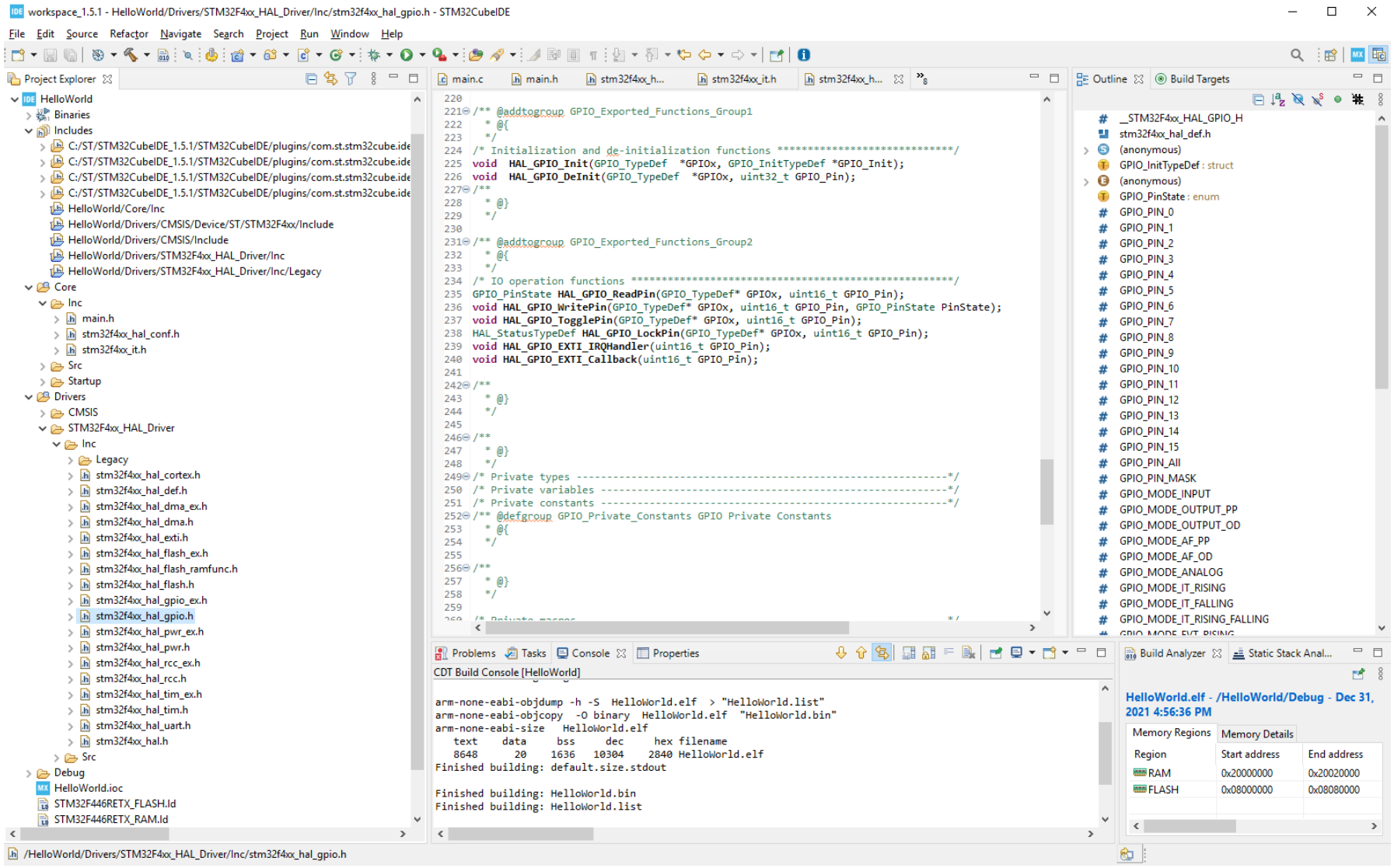
```
/* Private defines ------------------------
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */
```

- Some shorthand names are defined in main.h
- For example:
  - LD2_Pin is alias for GPIO_PIN_5
  - But where is GPIO_PIN_5 define?
  - It is defined in the stm34f4xx_hal_gpio.h

HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

- Where is the HAL_ function defined?
- What other GPIO functions are available?

Check out the Drivers/STM32F4xx_HAL_Driver folder in the source code



Some other GPIO functions

```
/* IO operation functions *****************************************************/
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);
```

```
/** @defgroup GPIO_pins_define GPIO pins define
  * @{
  */
#define GPIO_PIN_0                 ((uint16_t)0x0001)  /* Pin 0 selected    */
#define GPIO_PIN_1                 ((uint16_t)0x0002)  /* Pin 1 selected    */
#define GPIO_PIN_2                 ((uint16_t)0x0004)  /* Pin 2 selected    */
#define GPIO_PIN_3                 ((uint16_t)0x0008)  /* Pin 3 selected    */
#define GPIO_PIN_4                 ((uint16_t)0x0010)  /* Pin 4 selected    */
#define GPIO_PIN_5                 ((uint16_t)0x0020)  /* Pin 5 selected    */
#define GPIO_PIN_6                 ((uint16_t)0x0040)  /* Pin 6 selected    */
#define GPIO_PIN_7                 ((uint16_t)0x0080)  /* Pin 7 selected    */
#define GPIO_PIN_8                 ((uint16_t)0x0100)  /* Pin 8 selected    */
#define GPIO_PIN_9                 ((uint16_t)0x0200)  /* Pin 9 selected    */
#define GPIO_PIN_10                ((uint16_t)0x0400)  /* Pin 10 selected   */
#define GPIO_PIN_11                ((uint16_t)0x0800)  /* Pin 11 selected   */
#define GPIO_PIN_12                ((uint16_t)0x1000)  /* Pin 12 selected   */
#define GPIO_PIN_13                ((uint16_t)0x2000)  /* Pin 13 selected   */
#define GPIO_PIN_14                ((uint16_t)0x4000)  /* Pin 14 selected   */
#define GPIO_PIN_15                ((uint16_t)0x8000)  /* Pin 15 selected   */
#define GPIO_PIN_All               ((uint16_t)0xFFFF)  /* All pins selected */
```

```
typedef enum
{
  GPIO_PIN_RESET = 0,
  GPIO_PIN_SET
}GPIO_PinState;
```

This means
GPIO_PIN_RESET is alias for 0 (Low)
GPIO_PIN_SET is alias for 1 (High)

So, another way of blinking the green led is:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
    HAL_Delay(500);
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

Exercise 1: Modify the content in the while (1) loop to realize this:

- When the blue button (B1_Pin) is pressed (PinState is 0), the green led is on; otherwise, the green led is off.

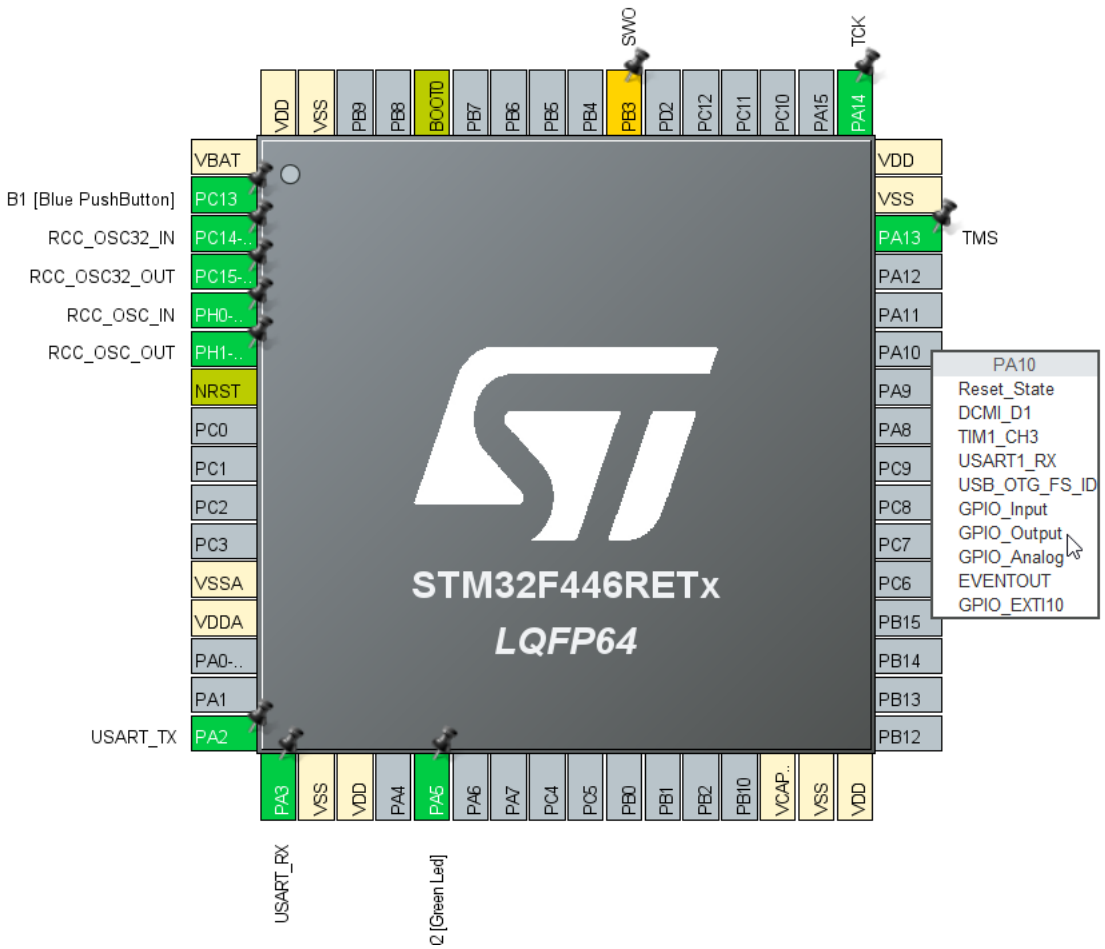
Hint: Use the HAL_GPIO_ReadPin function.

GPIO_InitStruct.Mode, .Pull, .Speed defs

```
#define  GPIO_MODE_INPUT                        0x00000000U   /*!< Input Floating Mode                   */
#define  GPIO_MODE_OUTPUT_PP                    0x00000001U   /*!< Output Push Pull Mode                 */
#define  GPIO_MODE_OUTPUT_OD                    0x00000011U   /*!< Output Open Drain Mode                */
#define  GPIO_MODE_AF_PP                        0x00000002U   /*!< Alternate Function Push Pull Mode     */
#define  GPIO_MODE_AF_OD                        0x00000012U   /*!< Alternate Function Open Drain Mode    */

#define  GPIO_MODE_ANALOG                       0x00000003U   /*!< Analog Mode  */

#define  GPIO_MODE_IT_RISING                    0x10110000U   /*!< External Interrupt Mode with Rising edge trigger detection          */
#define  GPIO_MODE_IT_FALLING                   0x10210000U   /*!< External Interrupt Mode with Falling edge trigger detection         */
#define  GPIO_MODE_IT_RISING_FALLING            0x10310000U   /*!< External Interrupt Mode with Rising/Falling edge trigger detection  */

#define  GPIO_MODE_EVT_RISING                   0x10120000U   /*!< External Event Mode with Rising edge trigger detection              */
#define  GPIO_MODE_EVT_FALLING                  0x10220000U   /*!< External Event Mode with Falling edge trigger detection             */
#define  GPIO_MODE_EVT_RISING_FALLING           0x10320000U   /*!< External Event Mode with Rising/Falling edge trigger detection      */
/**
  * @}
  */

/** @defgroup GPIO_speed_define  GPIO speed define
  * @brief GPIO Output Maximum frequency
  * @{
  */
#define  GPIO_SPEED_FREQ_LOW         0x00000000U  /*!< IO works at 2 MHz, please refer to the product datasheet */
#define  GPIO_SPEED_FREQ_MEDIUM      0x00000001U  /*!< range 12,5 MHz to 50 MHz, please refer to the product datasheet */
#define  GPIO_SPEED_FREQ_HIGH        0x00000002U  /*!< range 25 MHz to 100 MHz, please refer to the product datasheet  */
#define  GPIO_SPEED_FREQ_VERY_HIGH   0x00000003U  /*!< range 50 MHz to 200 MHz, please refer to the product datasheet  */
/**
  * @}
  */

 /** @defgroup GPIO_pull_define GPIO pull define
   * @brief GPIO Pull-Up or Pull-Down Activation
   * @{
   */
#define  GPIO_NOPULL        0x00000000U   /*!< No Pull-up or Pull-down activation  */
#define  GPIO_PULLUP        0x00000001U   /*!< Pull-up activation                  */
#define  GPIO_PULLDOWN      0x00000002U   /*!< Pull-down activation                */
```

Use other GPIO pins (e.g., PA_10)

- Double click HelloWorld.ioc in the Project Explore pane to open the Device Configuration perspective
- Change the role of PA10 to GPIO_Output in the Pinout & Configuration view (see pic)
- Click the save button , click "Yes" to the question "Do you want generate code?", then click "Yes" to open the code perspective.



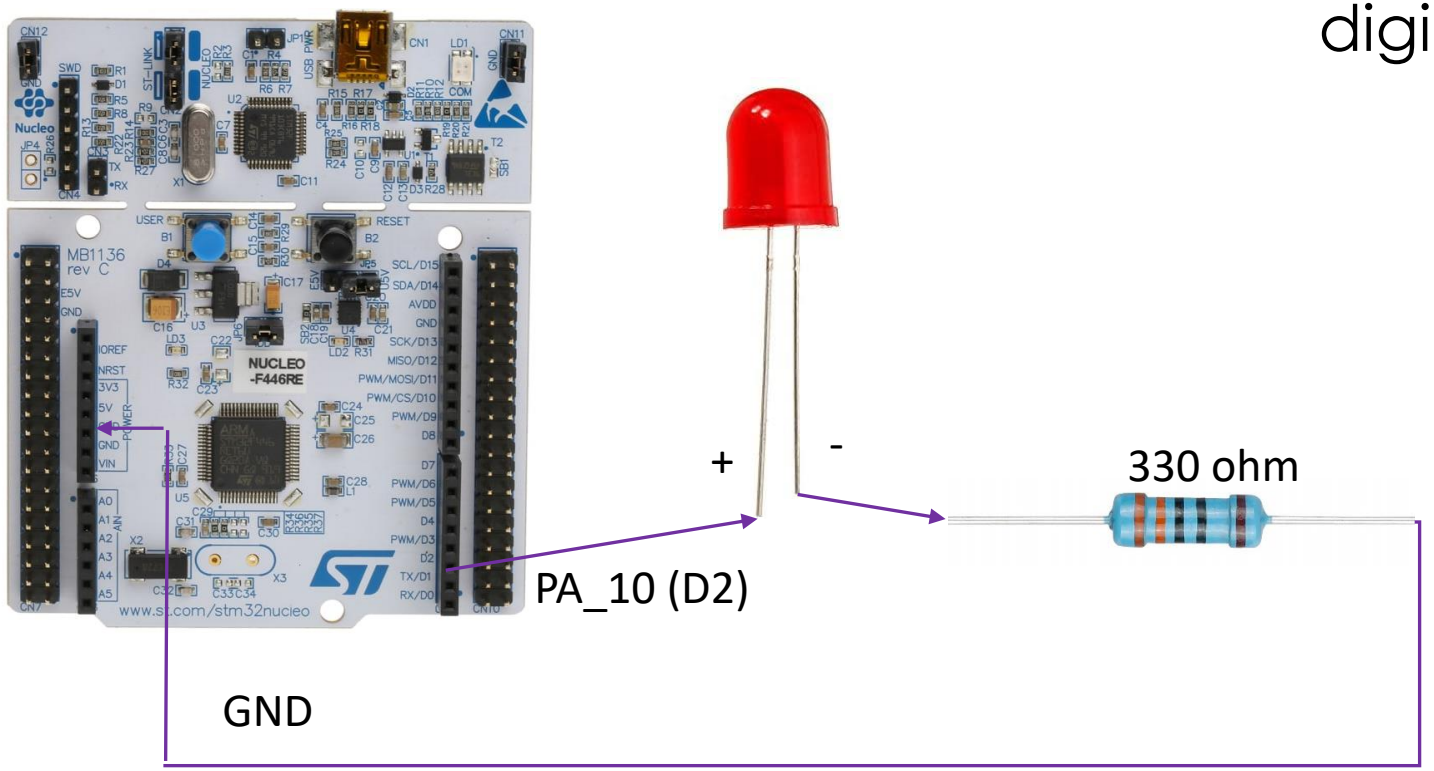
Regenerated MX_GPIO_Init code in main.c

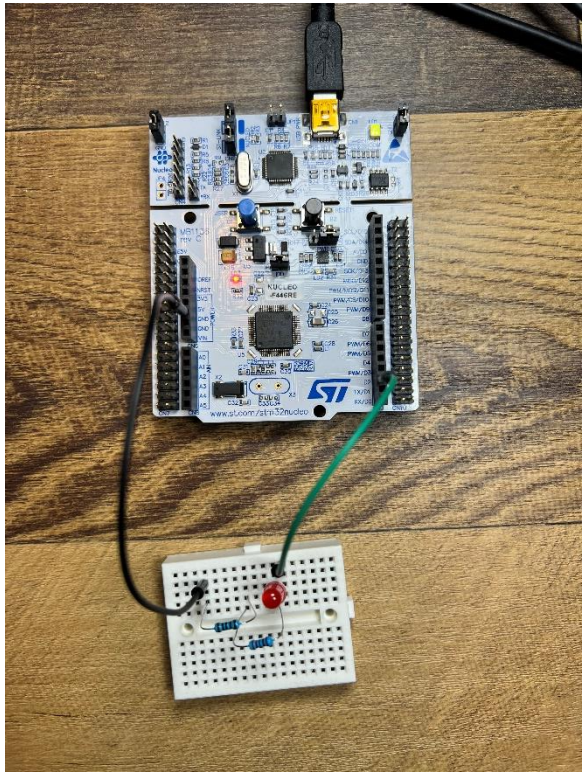
```c
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LD2_Pin|GPIO_PIN_10, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD2_Pin PA10 */
  GPIO_InitStruct.Pin = LD2_Pin|GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
```

- LD2_Pin | GPIO_PIN_10
- Both LD2_Pin (alias for GPIO_PIN_5) and GPIO_PIN_10 are enabled on GPIO Port A (GPIOA)

Hardware connection

- Let PA_10 (now configured as a digital output pin) to drive a led

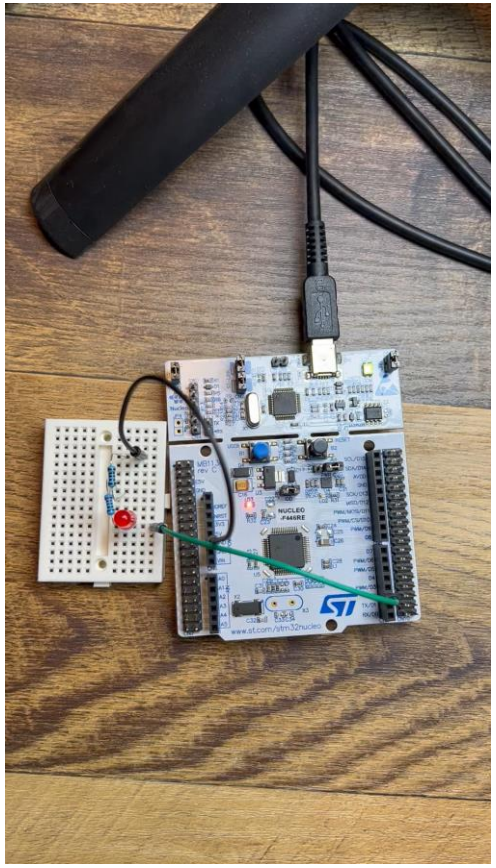




Write user code

- If the blue button (B1_Pin, or equivalently GPIO_Pin_13) is pressed, then light up the led connected to PA_10; otherwise, turn off the led

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

  /* USER CODE BEGIN 3 */
  if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET){
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_SET);
  } else {
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
  }
}
/* USER CODE END 3 */
```



Exercises

1. Use the blue button to toggle the led connected to PA_10.
   - Toggle means: click -> turn on, click again -> turn off
2. Connect a tactile switch to PA_8, set PA_8 as digital input, and use the switch to toggle the led connected to PA_10

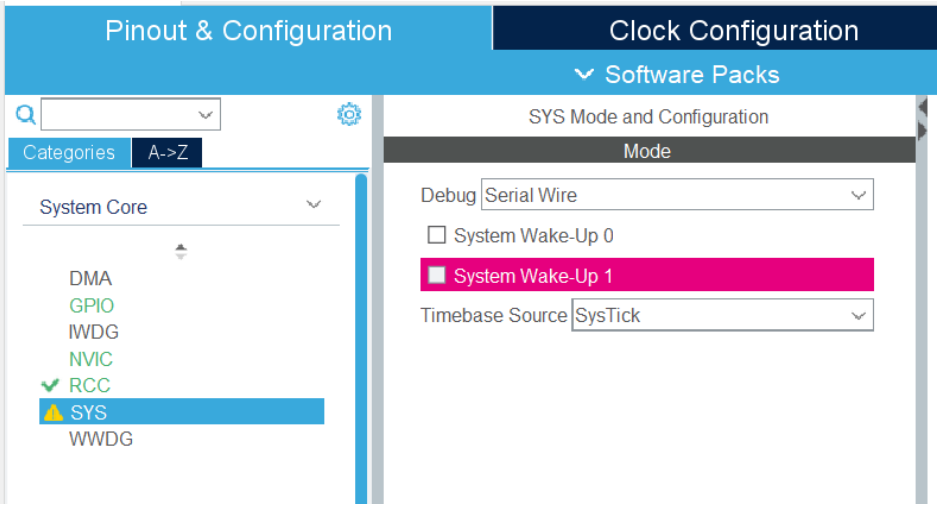
Tactile switch

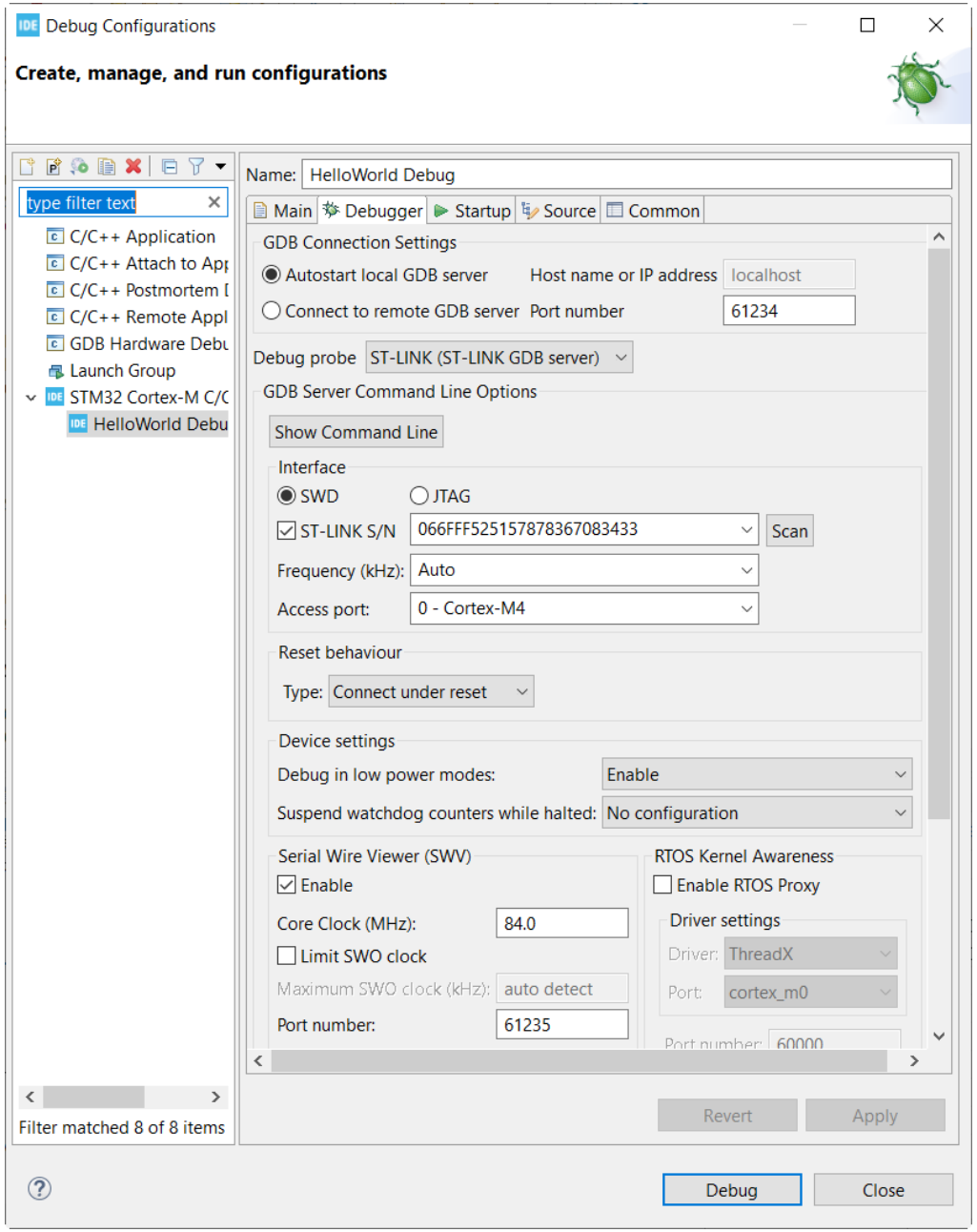
Debug functions

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int counter = 0;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */
```

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

    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    HAL_Delay(500);
    counter++;
    if(counter > 20){
        counter = 0;
    }
  }
  /* USER CODE END 3 */
}
```

- We will monitor a global variable
  - int counter = 0
- Increment the counter every 500 ms, and reset when hit 20
- We are going to monitor the value of this variable while the program is running
- Make sure Debug is "Serial Wire"
  - PA13 and PA14 will be used automatically

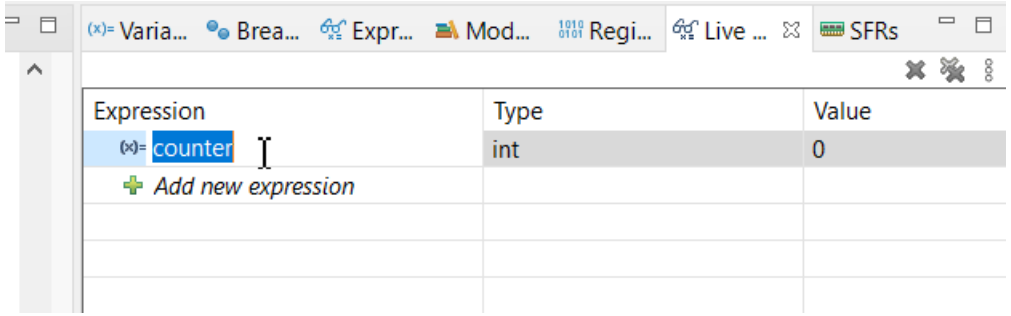


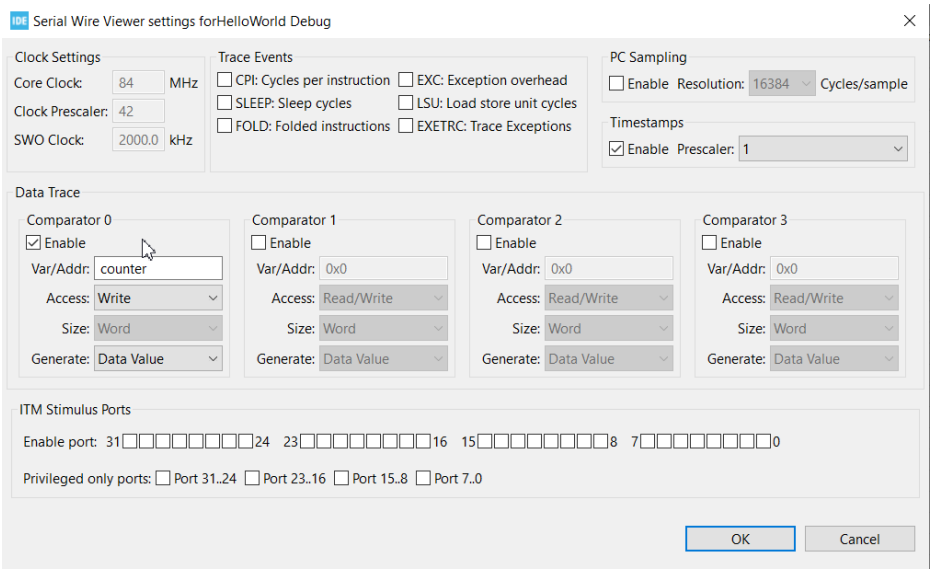


Debug Configuration

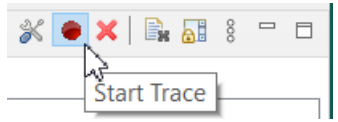
- Make sure the board is connected to the PC via USB
  - It should not be in active connection elsewhere, i.e., STM32CubeProgrammer
- Go to Run -> Debug Configuration
- Click the Debugger tab
- Check the box "ST-LINK S/N", click Scan, and select the board ID
- Check "Enable" under SWV
- Set Core Clock (MHz) to 84.0
  - This value is read from the HCLK value from the Clock Configuration page
- "Enable live expressions" is checked
- Click Debug

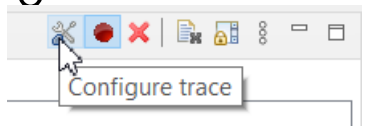
The Debug perspective





- In the Live Expressions tab, add the variable name "counter" as a new expression
- Hit the play button (or F8), the program will run on board, and the counter value is displayed in the Live Expression view in real-time
- Hit the suspend button to pause the program
- Go to Window -> Show View -> SWV -> SWV Data Trace
  - Click "Configure trace"
  - Add "counter" to comparator 0
  - Turn on "Start Trace" button





workspace_1.8.0 - HelloWorld/Core/Src/main.c - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

- New Window
- Editor
- Appearance
- Show View
  - ThreadX
  - FreeRTOS
  - SWV
    - SWV Trace log
    - SWV Exception Trace log
    - SWV Data Trace
    - SWV Data Trace Timeline Graph
    - SWV ITM Data Console
    - SWV Statistical Profiling
  - Breakpoints Alt+Shift+Q, B
  - Build Analyzer
  - Console Alt+Shift+Q, C
  - Debug
  - Debugger Console
  - Debug Sources
  - Disassembly
  - Error Log Alt+Shift+Q, L
  - Executables
  - Expressions
  - Fault Analyzer
  - Live Expressions
  - Memory
  - Memory Browser
  - Modules
  - Outline Alt+Shift+Q, O
  - Problems Alt+Shift+Q, X
  - Progress
  - Project Explorer
  - Registers
  - SFRs
  - Signals
  - Static Stack Analyzer
  - Templates
  - Trace Control
  - Variables Alt+Shift+Q, V
  - Other... Alt+Shift+Q, Q
- Perspective
- Navigation
- Preferences

Debug | Project Explorer

- HelloWorld Debug [STM32 Cortex-M C/C++ Application]
  - HelloWorld.elf [cores: 0]
    - Thread #1 [main] 1 [core: 0] (Suspended : Signal : SIG
      - HAL_Delay() at stm32f4xx_hal.c:400 0x8000a22
      - main() at main.c:102 0x8000520
  - arm-none-eabi-gdb (8.3.1.20191211)
  - ST-LINK (ST-LINK GDB server)

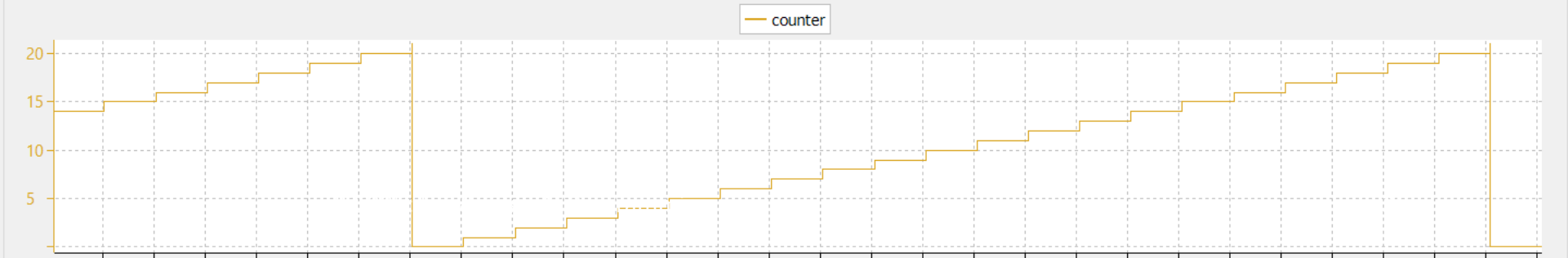
tup_stm... | stm32f4xx_hal.c | stm32f4xx_ha... | core_cm4.h

```
75
76  /* USER
77
78  /* USER
79
80  /* Conf
81  SystemC
82
83  /* USER
84
85  /* USER
86
87  /* Init
88  MX_GPIO
89  MX_USAR
90  /* USER
91
92  /* USER
93
94  /* Infi
95  /* USER
96  while (
97  {
98    /* US
99
100   /* US
```

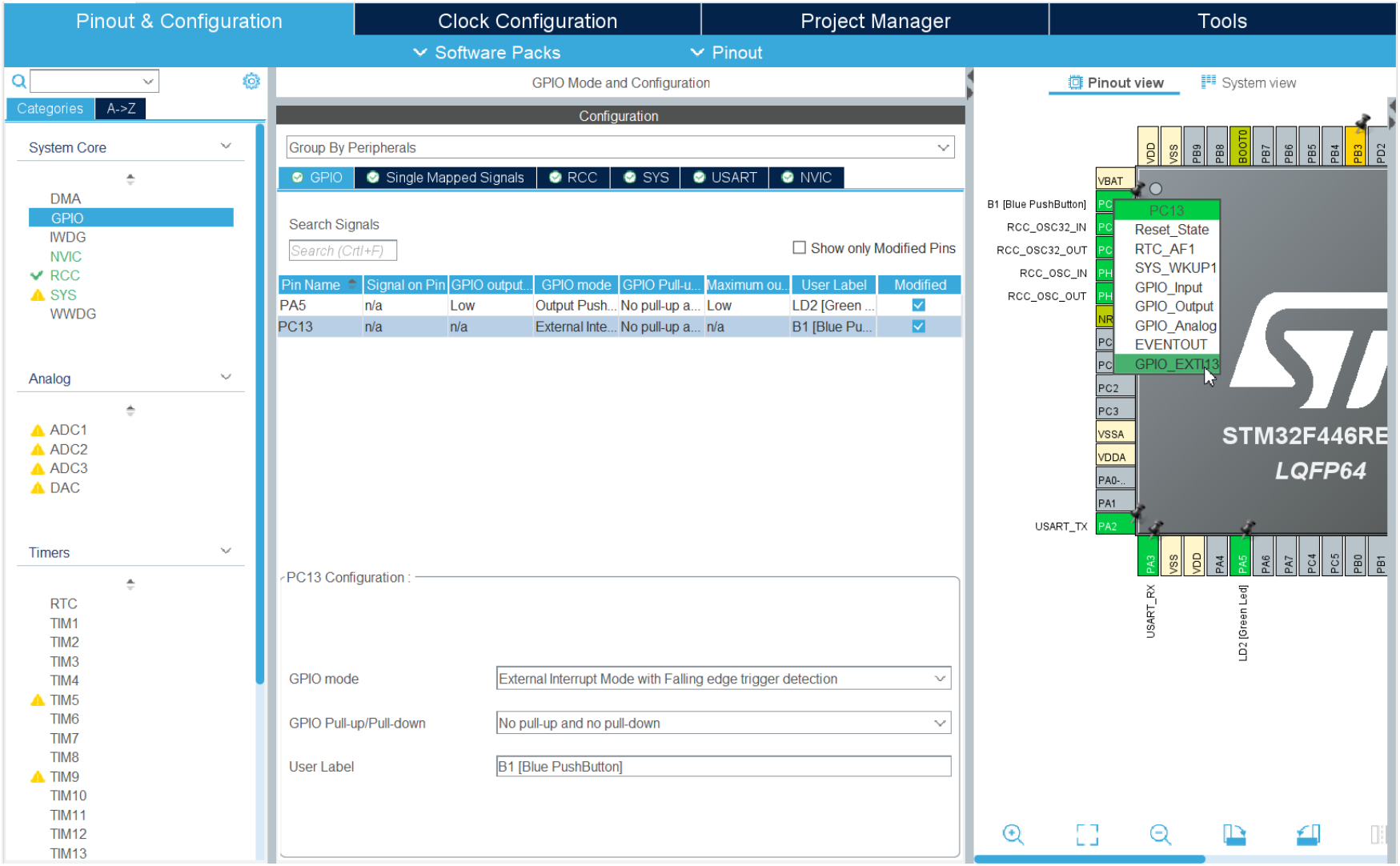
Varia... | Brea... | Expr... | Mod... | Regi... | Live ... | SFRs

Expression	Type	Value
counter	int	1
Add new expression		

Console | Problems | Executables | Debugger Console | Memory | SWV Data Trace | SWV Statistical Profiling | SWV Data Trace Timeline Graph

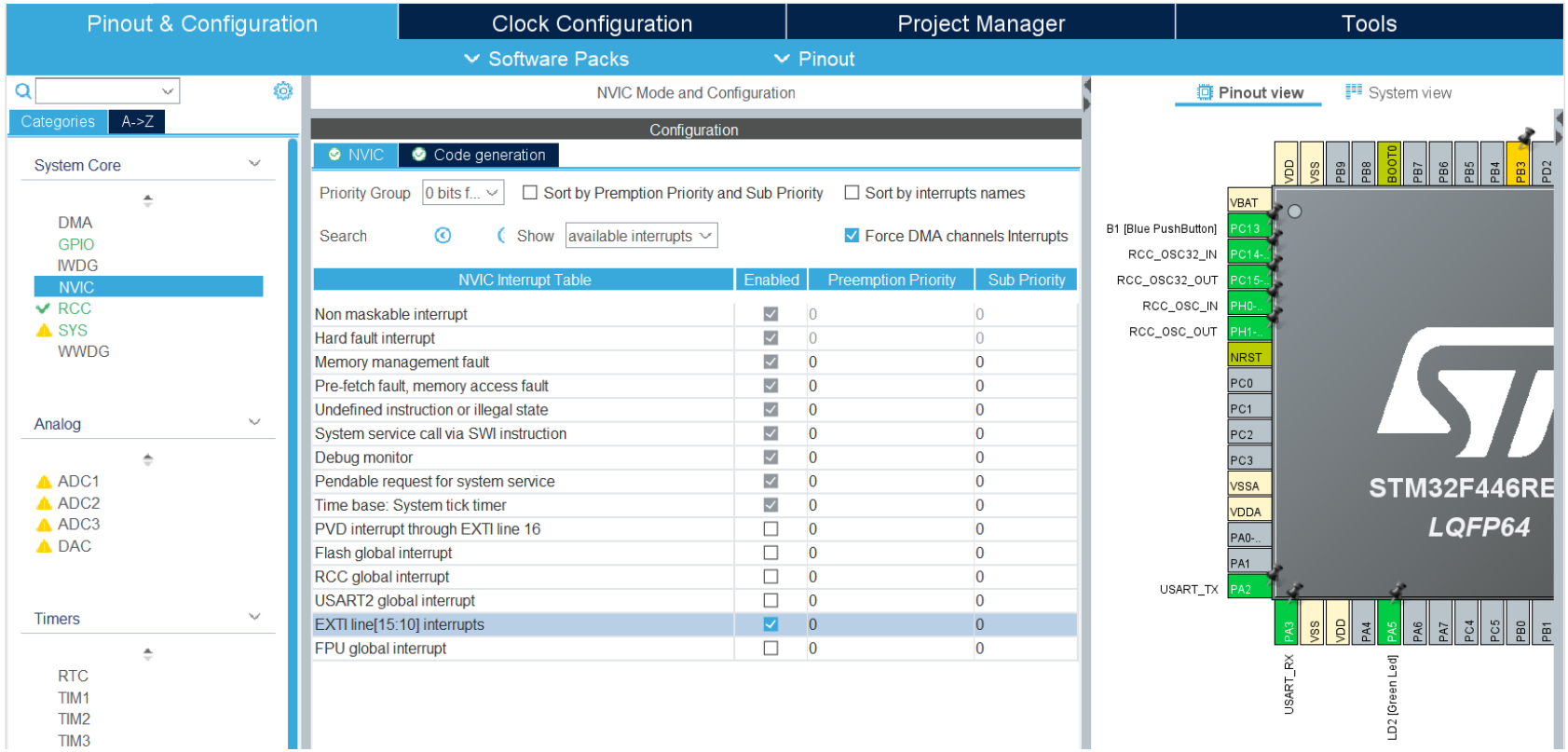


Use Interrupts



- Configure PC13 (the blue button) as GPIO_EXTI13
- Set GPIO mode (Falling edge trigger)

Enable EXTI 15:10 interrupts in NVIC



- PC13 is grouped together with many other pins into a common NVIC IRQ named EXTI15_10_IRQ.
- We have to enable it to use PC13 pin for interrupt.
- Disregard the priority setting, just use the default 0.
  - Smaller value means higher priority.

IRQ Handler invocation chain

- Core
  - Inc
  - Src
    - main.c
    - stm32f4xx_hal_msp.c
    - stm32f4xx_it.c
    - syscalls.c
    - sysmem.c
    - system_stm32f4xx.c
  - Startup
- Drivers
  - CMSIS
  - STM32F4xx_HAL_Driver
    - Inc
    - Src
      - stm32f4xx_hal_cortex.c
      - stm32f4xx_hal_dma_ex.c
      - stm32f4xx_hal_dma.c
      - stm32f4xx_hal_exti.c
      - stm32f4xx_hal_flash_ex.c
      - stm32f4xx_hal_flash_ramfunc.c

```c
/* STM32F4xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f4xx.s).                   */
/******************************************************************************/

/**
  * @brief This function handles EXTI line[15:10] interrupts.
  */
void EXTI15_10_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI15_10_IRQn 0 */

  /* USER CODE END EXTI15_10_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(B1_Pin);
  /* USER CODE BEGIN EXTI15_10_IRQn 1 */

  /* USER CODE END EXTI15_10_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

- HAL_GPIO_IRQHandler(B1_Pin); is called in EXTI15_10_IRQHandler
- Where is this function defined?
- IN the file stm32f4xx_hal_gpio.c

HelloWorld.ioc | main.c | stm32f4xx_it.c | stm32f4xx_hal_gpio.c

```c
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
    __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}

/**
  * @brief  EXTI line detection callbacks.
  * @param  GPIO_Pin Specifies the pins connected EXTI line
  * @retval None
  */
__weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
}
```

- We have to implement the **HAL_GPIO_EXTI_Callback** function in main.c

Implement the callback function

HelloWorld.ioc | main.c | stm32f4xx_it.c | stm32f4xx_hal_gpio.c

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

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

- Do nothing in the infinite loop
- The green led is to be toggled by the B1_Pin interrupt

HelloWorld.ioc | main.c | stm32f4xx_it.c | stm32f4xx_hal_gpio.c

```c
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if(GPIO_Pin == B1_Pin){
        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    }
}
/* USER CODE END 4 */
```

- User implementation of the HAL_GPIO_EXTI_Callback function
- If the interrupt is trigger by B1_Pin, then toggle the led

- After making these changes, save, build project, and debug.

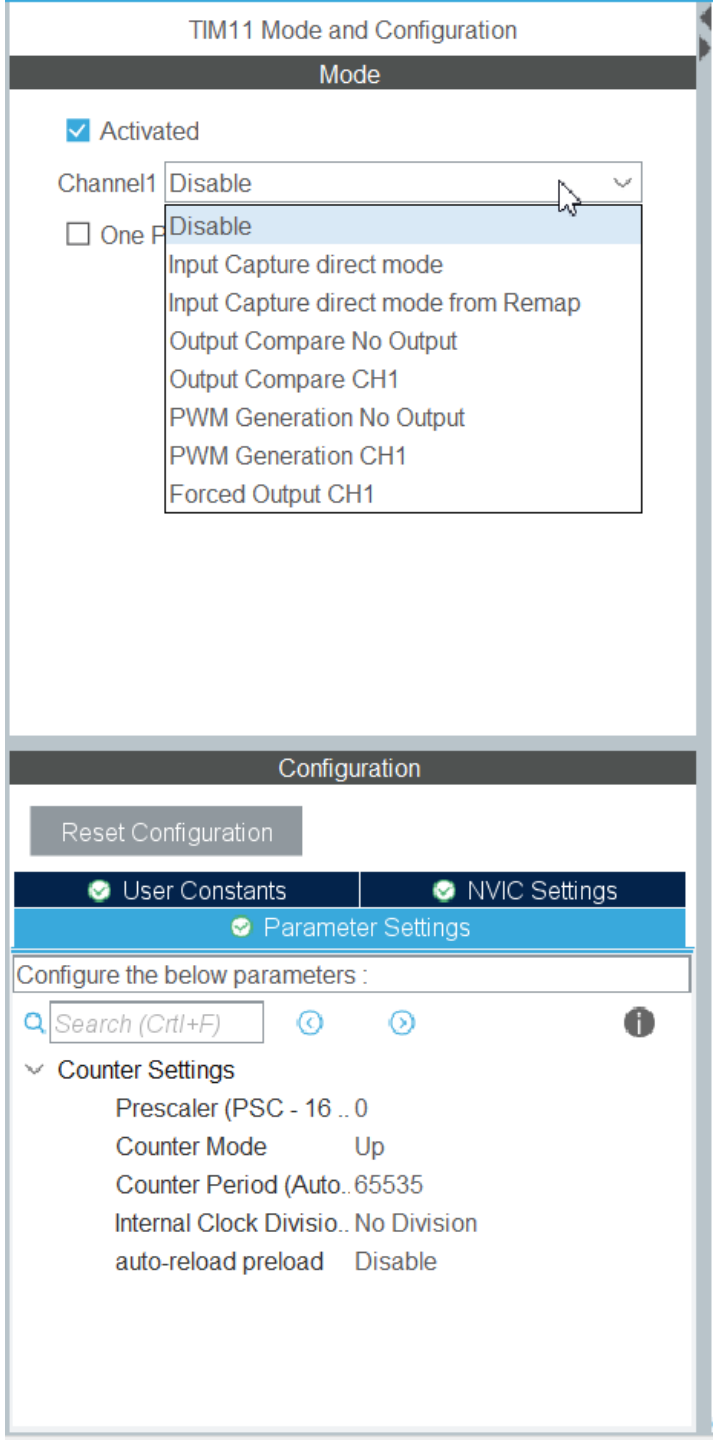
- **Exercise:** Change the interrupt trigger to the Rising edge, and try again
- Which one (falling or rising) is better?

Timers

- Many embedded applications are time dependent
- Timers are the primary means by which the MCU controls the application
- It would be a great waste of the processing power if functions like HAL_Delay() are pervasively used to time processes
- Use hardware timers along with interrupts is the right way to implement embedded time-dependent applications
- Use case examples:
  - Measure the frequency of an incoming digital pulse train
  - Measure elapsed time on an output signal (called output compare for STM timers)
  - Generate precise PWM signals for servo and motor control
  - Generate single pulse with programmable length and delay
  - ADC (e.g., oscilloscope, audio signal) sampling at a given frequency

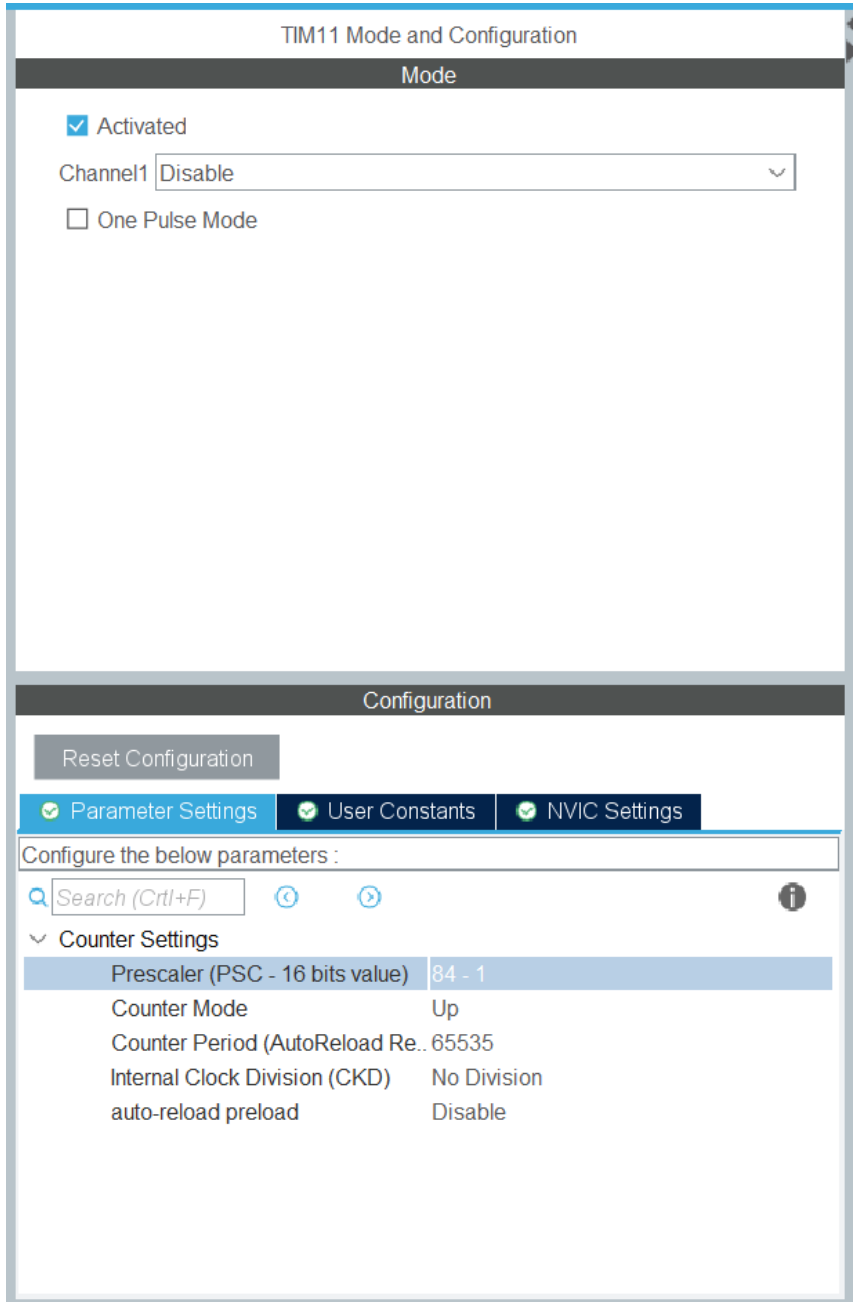
Timer setup

- A timer is essentially a counter
  - The clock source Frequency and the **Prescaler** controls how fast it counts
  - The timer's **Counter Period** determines how many counts before it's reset to 0, i.e., the timer's maximum counting value
    - A value of 0 turns off the timer - the timer will not run at all



Timer modes

- **Input Capture**: allows us to automatically store the timer's value when an event happens on a pin
- **Output Compare**: allows us to toggle a pin whenever the value in the timer reaches a certain amount
- **PWM Generation**: toggles a pin at different duty cycles based on a timer's counter value



84 MHz / 84 = 1 MHz → 1 tick / microsecond

Max time: 65535 * 1 microsecond = 65.535 ms

The maximum time we can measure with this timer is about 65 milliseconds.

We will use this timer (TIM11) to measure how long something takes to process in the MCU

Save configuration, generate code, and go to main.c

Use hardware timer to time a block of code

```c
/* Private includes -------------
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */

```

- Use UART2 to output messages
- Use Putty at the STLink Virtual COM Port to receive
- Cannot use physical TX/RX pins on board unless solder bridge SB62 or SB63 is closed.

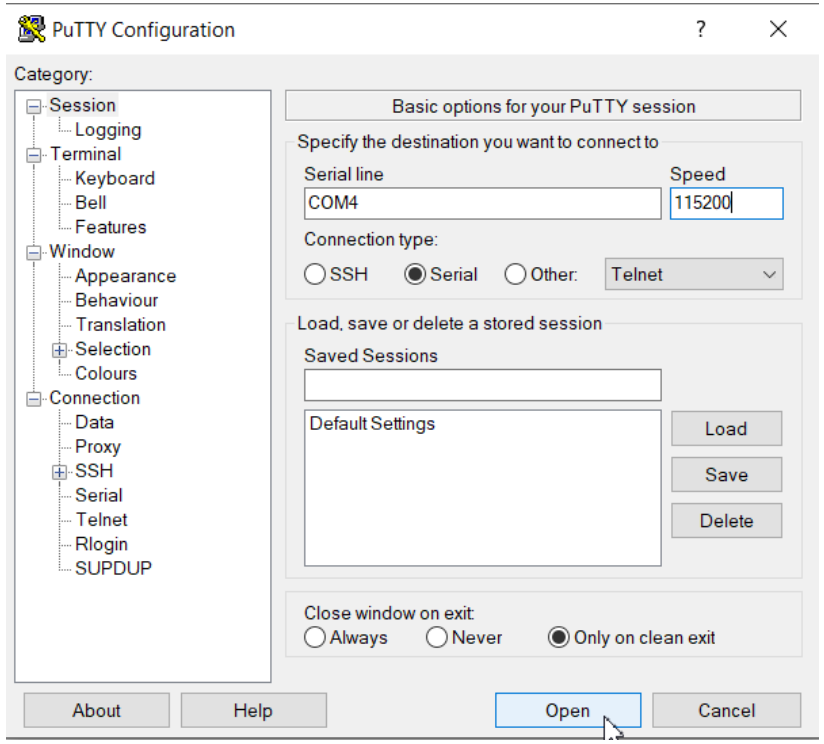
```c
int main(void)
{
  /* USER CODE BEGIN 1 */
  char uart_buf[50];
  int uart_buf_len;
  uint16_t timer_val;
  /* USER CODE END 1 */

```

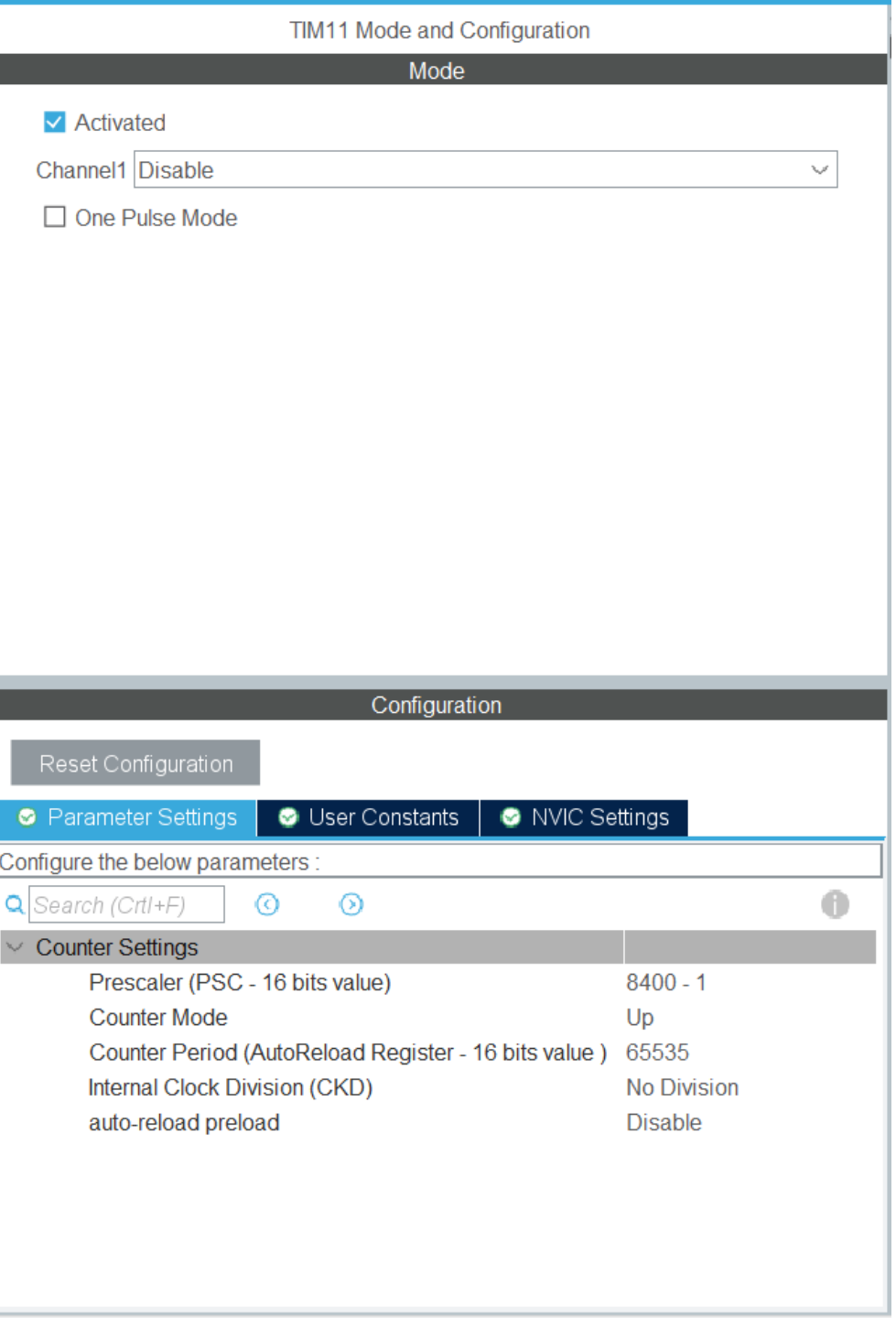
```c
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM11_Init();
  /* USER CODE BEGIN 2 */

  uart_buf_len = sprintf(uart_buf, "Timer test\r\n");
  HAL_UART_Transmit(&huart2, (uint8_t*)uart_buf, uart_buf_len, 100);
  HAL_TIM_Base_Start(&htim11);

  /* USER CODE END 2 */
```

```c
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    // Get current time (microseconds)
    timer_val = __HAL_TIM_GET_COUNTER(&htim11);
    // any code we want to measure how long it takes to execute
    // we will put a dummy delay here: wait for 50 ms
    // Given the prescaler and period set up, TIM11 cannot measure anything longer than 65 ms
    HAL_Delay(50);
    // Get time elapsed
    timer_val = __HAL_TIM_GET_COUNTER(&htim11) - timer_val;
    // Show elapsed time
    uart_buf_len = sprintf(uart_buf, "%u us\r\n", timer_val);
    HAL_UART_Transmit(&huart2, (uint8_t *)uart_buf, uart_buf_len, 100);
    // Wait again so we don't flood the Serial terminal
    HAL_Delay(1000);
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);

    /* USER CODE END WHILE */

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

Use Serial port to receive uart2 messages

- Do nothing in the infinite loop
- The green led is to be toggled by the B1_Pin interrupt





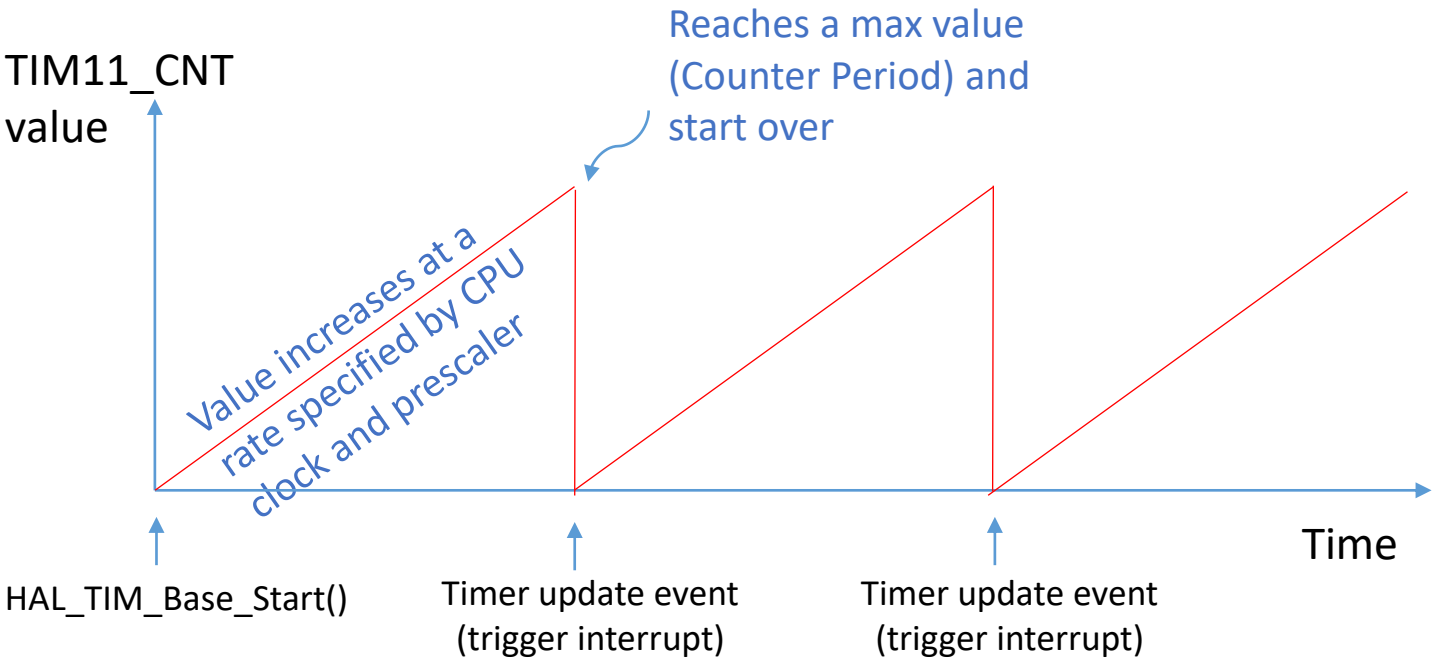
Non-blocking blinky using Timer

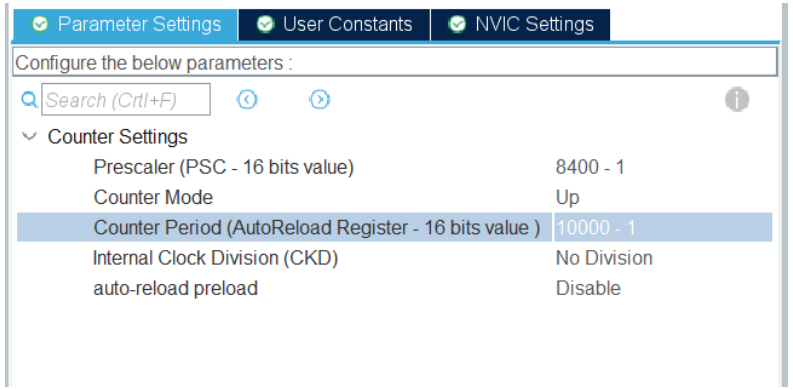
- 84 MHz / 8400 = 10 kHz → 1 tick every 0.0001 sec
- Can count up to 0.0001 * 65536 = 6.5536 sec

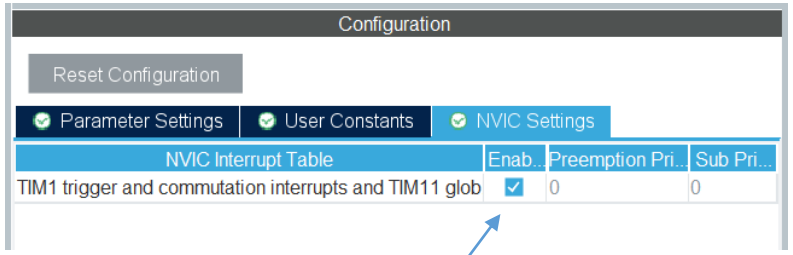
```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_TIM11_Init();
/* USER CODE BEGIN 2 */

HAL_TIM_Base_Start(&htim11);
// Get current time
timer_val = __HAL_TIM_GET_COUNTER(&htim11);

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  // If enough time has passed (1 second, 10000 ticks, = 84 MHz/8400), toggle LED
  // Below way to calculate elapsed time will work even if the timer rolls over to 0
  // This creates non-blocking code to blink the LED every second
  if(__HAL_TIM_GET_COUNTER(&htim11) - timer_val >= 10000){
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
    timer_val = __HAL_TIM_GET_COUNTER(&htim11);
  }

  /* USER CODE END WHILE */

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

Timer triggered interrupt

Target event rate: 1 Hz
Time between target event: 1 sec
Clock speed: 84 MHz
Prescaler: 8400
Desired Period value:
1 sec * (84 MHz / 8400) = 10000







Enable this, or interrupt won't trigger

Timer functions to use

68.2.3 Time Base functions

This section provides functions allowing to:

- Initialize and configure the TIM base.
- De-initialize the TIM base.
- Start the Time Base.
- Stop the Time Base.
- Start the Time Base and enable interrupt.
- Stop the Time Base and disable interrupt.
- Start the Time Base and enable DMA transfer.
- Stop the Time Base and disable DMA transfer.

This section contains the following APIs:

- *HAL_TIM_Base_Init()*
- *HAL_TIM_Base_DeInit()*
- *HAL_TIM_Base_MspInit()*
- *HAL_TIM_Base_MspDeInit()*
- *HAL_TIM_Base_Start()*
- *HAL_TIM_Base_Stop()*
- *HAL_TIM_Base_Start_IT()*
- *HAL_TIM_Base_Stop_IT()*
- *HAL_TIM_Base_Start_DMA()*
- *HAL_TIM_Base_Stop_DMA()*

Use this function to start timer if we are using interrupts

- TIM Period elapsed callback
- TIM Output Compare callback
- TIM Input capture callback
- TIM Trigger callback
- TIM Error callback

This section contains the following APIs:

- *HAL_TIM_PeriodElapsedCallback()*
- *HAL_TIM_PeriodElapsedHalfCpltCallback()*
- *HAL_TIM_OC_DelayElapsedCallback()*
- *HAL_TIM_IC_CaptureCallback()*
- *HAL_TIM_IC_CaptureHalfCpltCallback()*
- *HAL_TIM_PWM_PulseFinishedCallback()*
- *HAL_TIM_PWM_PulseFinishedHalfCpltCallback()*
- *HAL_TIM_TriggerCallback()*
- *HAL_TIM_TriggerHalfCpltCallback()*
- *HAL_TIM_ErrorCallback()*

This function gets called whenever the timer resets

https://www.st.com/resource/en/user_manual/dm00105879-description-of-stm32f4-hal-and-ll-drivers-stmicroelectronics.pdf

Code to blink LED using Timer triggered Interrupt

```c
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM11_Init();
  /* USER CODE BEGIN 2 */
  // Start timer in interrupt mode
  HAL_TIM_Base_Start_IT(&htim11);

  /* USER CODE END 2 */

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

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

Implement the Callback function in main.c

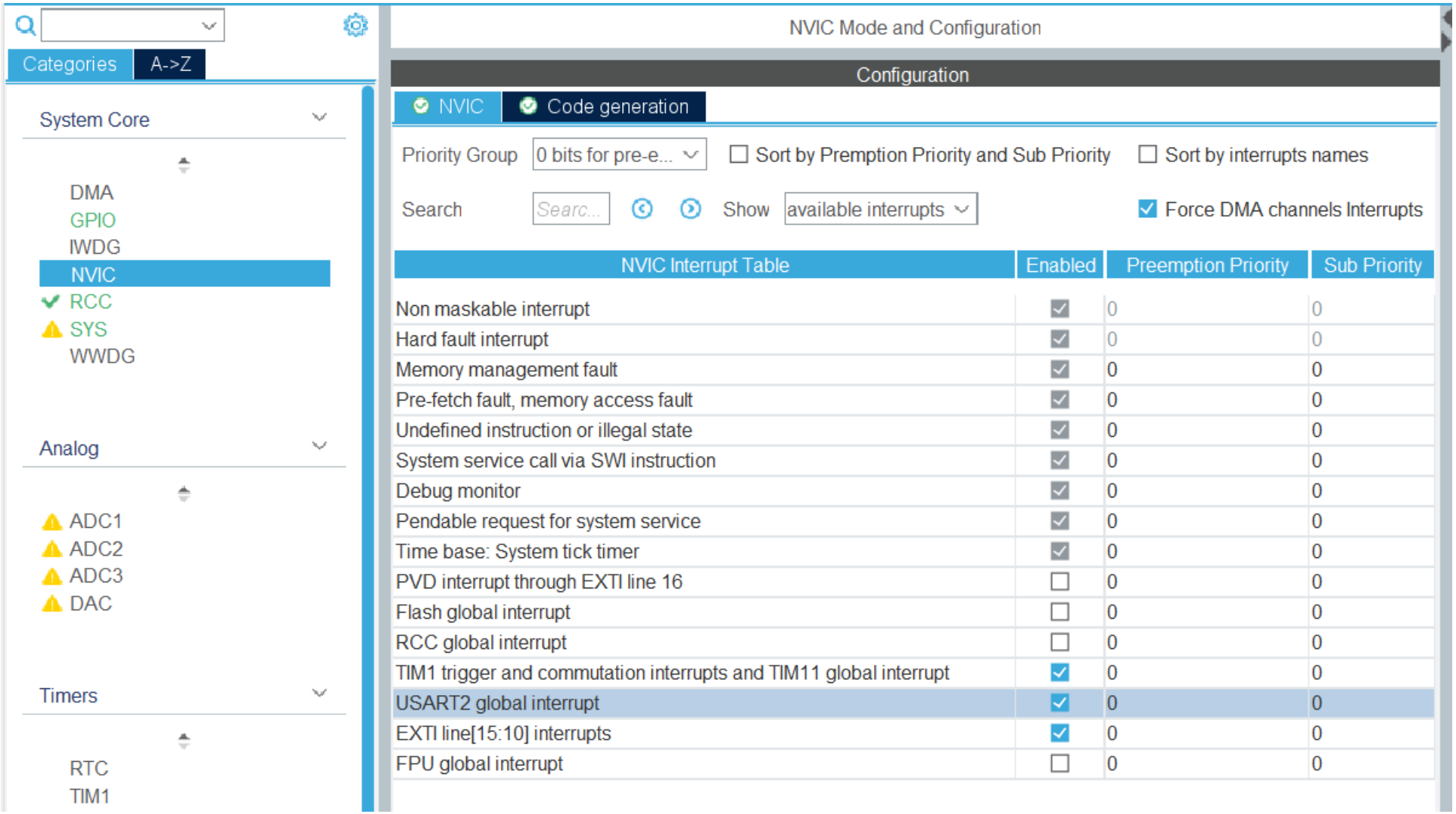
```c
/* USER CODE BEGIN 4 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
  // Check which version of the timer triggered this callback and toggle LED
  if (htim == &htim11){
    HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
  }
}
/* USER CODE END 4 */
```

Be careful with the function name here. If it's misspelled, no error will occur, but the LED will not light

Use UART Rx to receive data

- There are several ways.
- **Polling**: call HAL_UART_Receive() in while(1) loop
  - Other executions are blocked until the data is received or the timeout happens.
- **Interrupt**: call HAL_UART_Receive_IT() before while(1) loop, and implement user callback function HAL_UART_RxCpltCallback() to handle event (e.g., process the received data)
  - USART2 global interrupt must be enabled in NVIC, see next slide
  - Inside the callback function, need to make a call to HAL_UART_Receive_IT() to restart, i.e., continue receiving data in interrupt mode
  - At beginning of the callback function body, use this to respond to the right UART:
    - if(huart->Instance == USART2){ /* Do everything in this block */ } else {/* Do nothing */}
- **DMA**: call HAL_UART_Receive_DMA() before while(1) loop, and implement callback functions
  - HAL_UART_RxCpltCallback()
  - HAL_UART_RxHalfCpltCallback() // perform some tasks when half the Rx buffer is filled

Enable USART2 global interrupt



Useful UART functions

```
/* IO operation functions *****************************************************/
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
HAL_StatusTypeDef HAL_UART_Transmit_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_Receive_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UART_DMAPause(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_DMAResume(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_DMAStop(UART_HandleTypeDef *huart);

HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint16_t *RxLen,
                                           uint32_t Timeout);
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);
HAL_StatusTypeDef HAL_UARTEx_ReceiveToIdle_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);

/* Transfer Abort functions */
HAL_StatusTypeDef HAL_UART_Abort(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortTransmit(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortReceive(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_Abort_IT(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortTransmit_IT(UART_HandleTypeDef *huart);
HAL_StatusTypeDef HAL_UART_AbortReceive_IT(UART_HandleTypeDef *huart);

void HAL_UART_IRQHandler(UART_HandleTypeDef *huart);
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortTransmitCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_AbortReceiveCpltCallback(UART_HandleTypeDef *huart);

void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size);
```

Use DMA to transfer data to UART Tx (Polling)

DMA Mode and Configuration

Configuration

DMA1 | DMA2 | MemToMem

DMA Request	Stream	Direction	Priority
USART2_TX	DMA1 Stream 6	Memory To Peripheral	Low

NVIC Interrupt Table	Enabled	Preemption Priority	Sub P
Non maskable interrupt	☑	0	0
Hard fault interrupt	☑	0	0
Memory management fault	☑	0	0
Pre-fetch fault, memory access fault	☑	0	0
Undefined instruction or illegal state	☑	0	0
System service call via SWI instruction	☑	0	0
Debug monitor	☑	0	0
Pendable request for system service	☑	0	0
Time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
DMA1 stream6 global interrupt	☑	0	0
TIM2 global interrupt	☐	0	0
USART2 global interrupt	☐	0	0
EXTI line[15:10] interrupts	☐	0	0
FPU global interrupt	☐	0	0

- Before the while(1) loop, add
  - `HAL_UART_MspInit(&huart2);`
- In the while(1) loop:

```
huart2.Instance->CR3 |= USART_CR3_DMAT;

HAL_DMA_Start(&hdma_usart2_tx, (uint32_t)msg,
(uint32_t)&huart2.Instance->DR,
(uint32_t)strlen(msg));

HAL_DMA_PollForTransfer(&hdma_usart2_tx,
HAL_DMA_FULL_TRANSFER, HAL_MAX_DELAY);

huart2.Instance->CR3 &= ~USART_CR3_DMAT;

HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

HAL_Delay(1000);
```

Use DMA to transfer data to UART Tx (Interrupt)

```
/* USER CODE BEGIN 2 */
HAL_UART_MspInit(&huart2);
HAL_DMA_RegisterCallback(&hdma_usart2_tx, HAL_DMA_XFER_CPLT_CB_ID, &DMATransferComplete);
```

Register a user callback function, let us name it DMATransferComplete

In the while(1) loop:

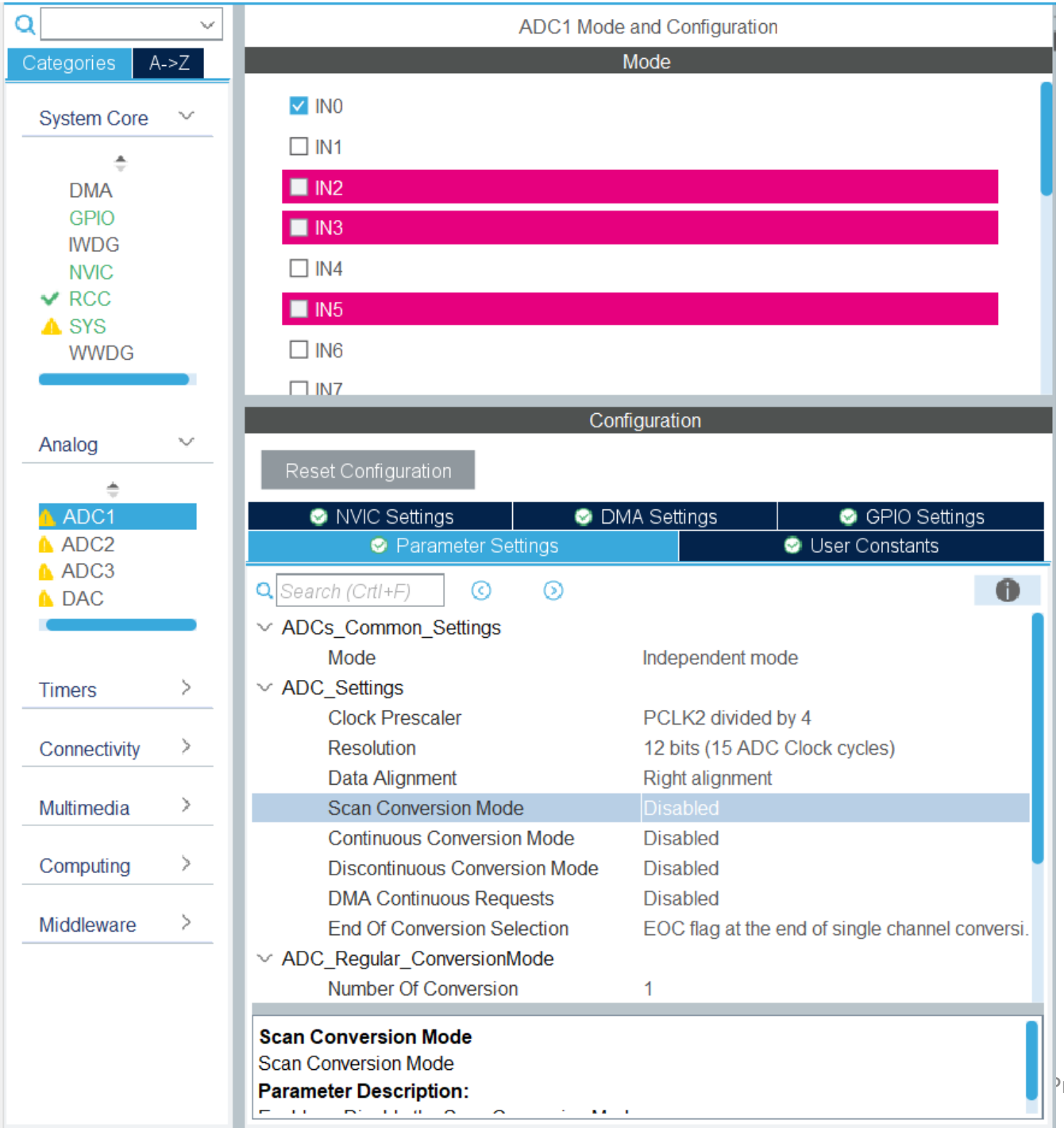
```
// USART Control register 3 bitwise or with DMA enable Transmitter
// Enable the DMA transmit bit which tells the usart peripheral that
// we plan to send out data via DMA
huart2.Instance->CR3 |= USART_CR3_DMAT;
// Tell the DMA instance hdma_usart2_tx to start the Interrupt mode
// to use the msg in memory as the source, the destination address is
// the transmit data register DR, the message length is strlen(msg)
HAL_DMA_Start_IT(&hdma_usart2_tx, (uint32_t)msg, (uint32_t)&huart2.Instance->DR, (uint32_t)strlen(msg));
HAL_Delay(1000);
```

```
void DMATransferComplete(DMA_HandleTypeDef *hdma){
  // Disable UART DMA mode, so UART can be used for other purposes
  huart2.Instance->CR3 &= ~USART_CR3_DMAT;
  // Toggle LED
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
}
```

Definition of the callback function

Compared to the Polling method, the interrupt method does not block the process, especially when the message (msg) size is large.

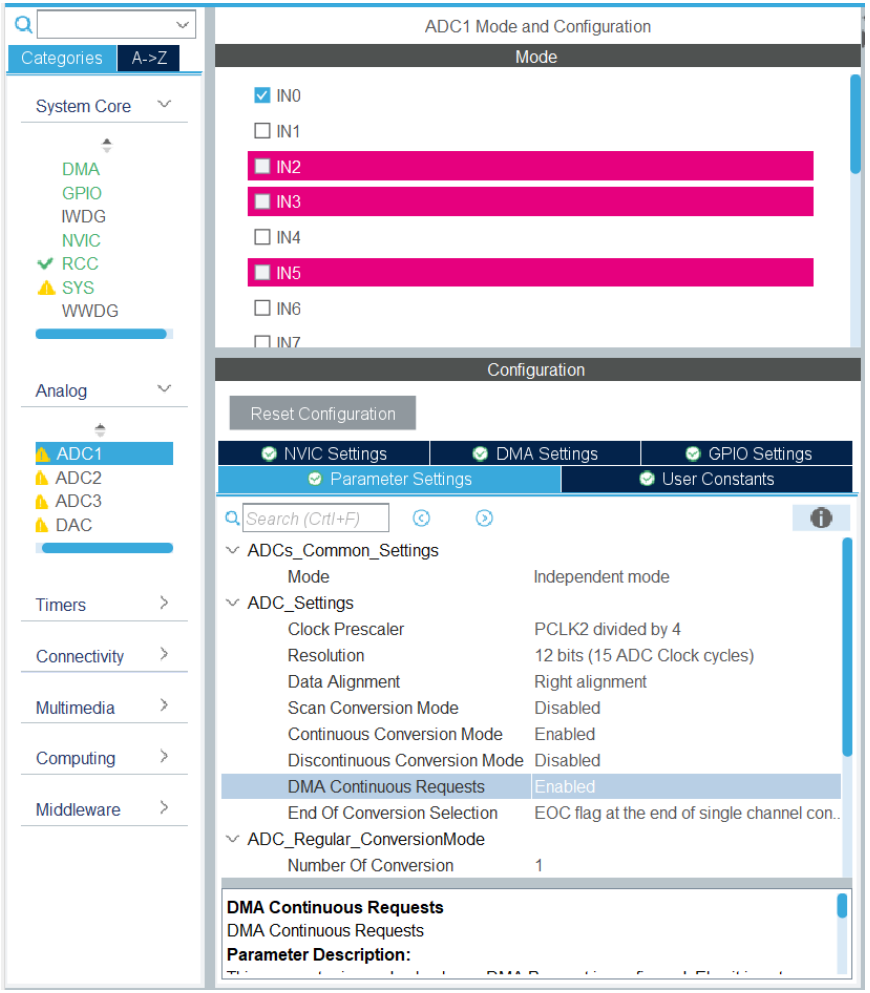
ADC (Polling)




```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    // set LED pin high
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    // Get ADC value
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    adc_val = HAL_ADC_GetValue(&hadc1);
    // set LED pin low
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    // Transmit value via uart2
    buflen = sprintf(buf, "%hu\r\n", adc_val);
    HAL_UART_Transmit(&huart2, (uint8_t*)buf, buflen, HAL_MAX_DELAY);
    // Add a 1 ms delay
    HAL_Delay(1);
    /* USER CODE END WHILE */

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

ADC (DMA)

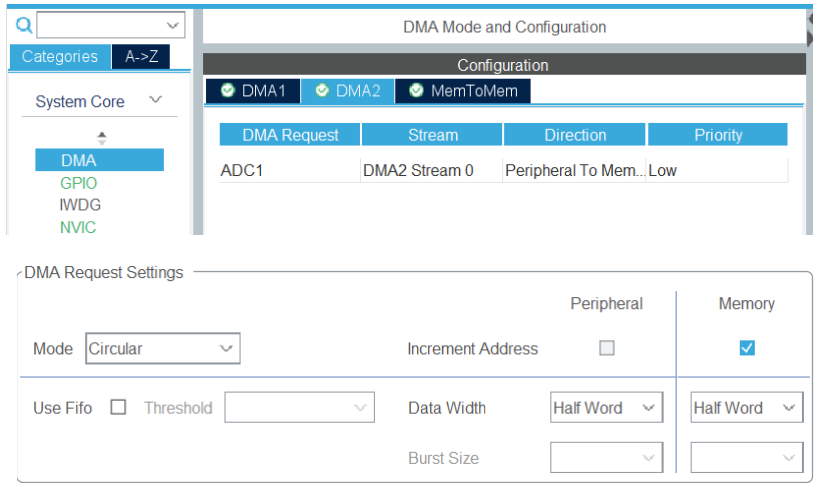
- Analog samples continuously enter memory (buf) without CPU intervention
- Each time half and full buf is filled, a respective event is triggered. Write callback to handle them, e.g., process / copy the data




```
/* Private user code ---------
/* USER CODE BEGIN 0 */
uint16_t buf[4096];
/* USER CODE END 0 */

/* USER CODE BEGIN 2 */
HAL_ADC_Start_DMA(&hadc1, (uint32_t*)buf, 4096);
/* USER CODE END 2 */
```

Empty while (1) loop will work

```
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc){
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc){
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
}

/* USER CODE END 4 */
```

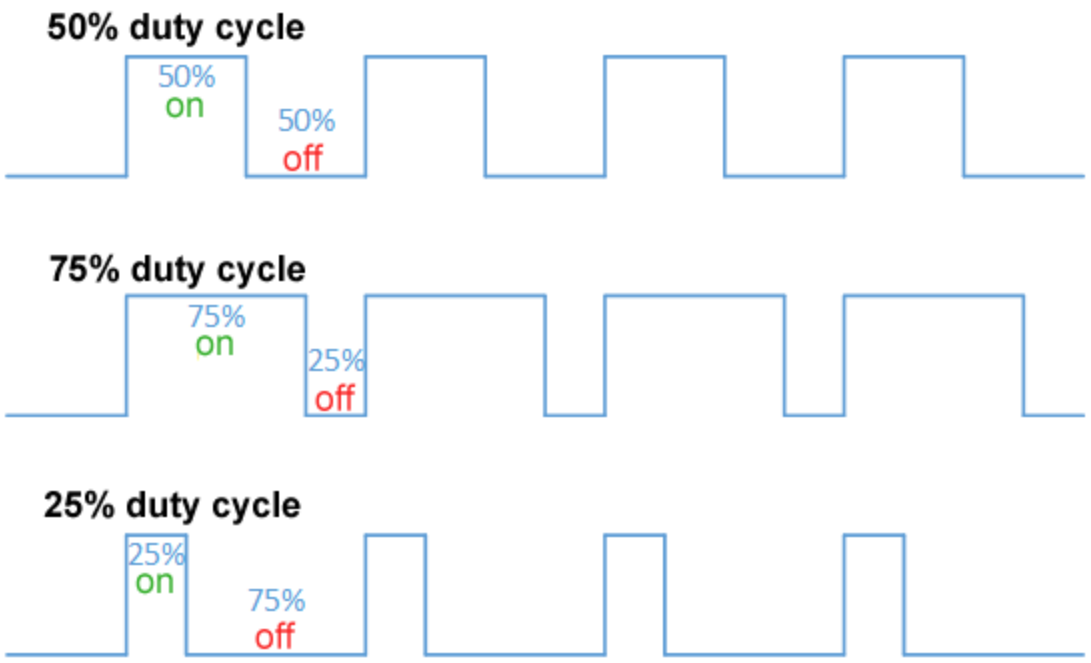


- Sampling frequency is determined by clock and prescaler
- Useful for audio sampling

Pulse width modulation (PWM)

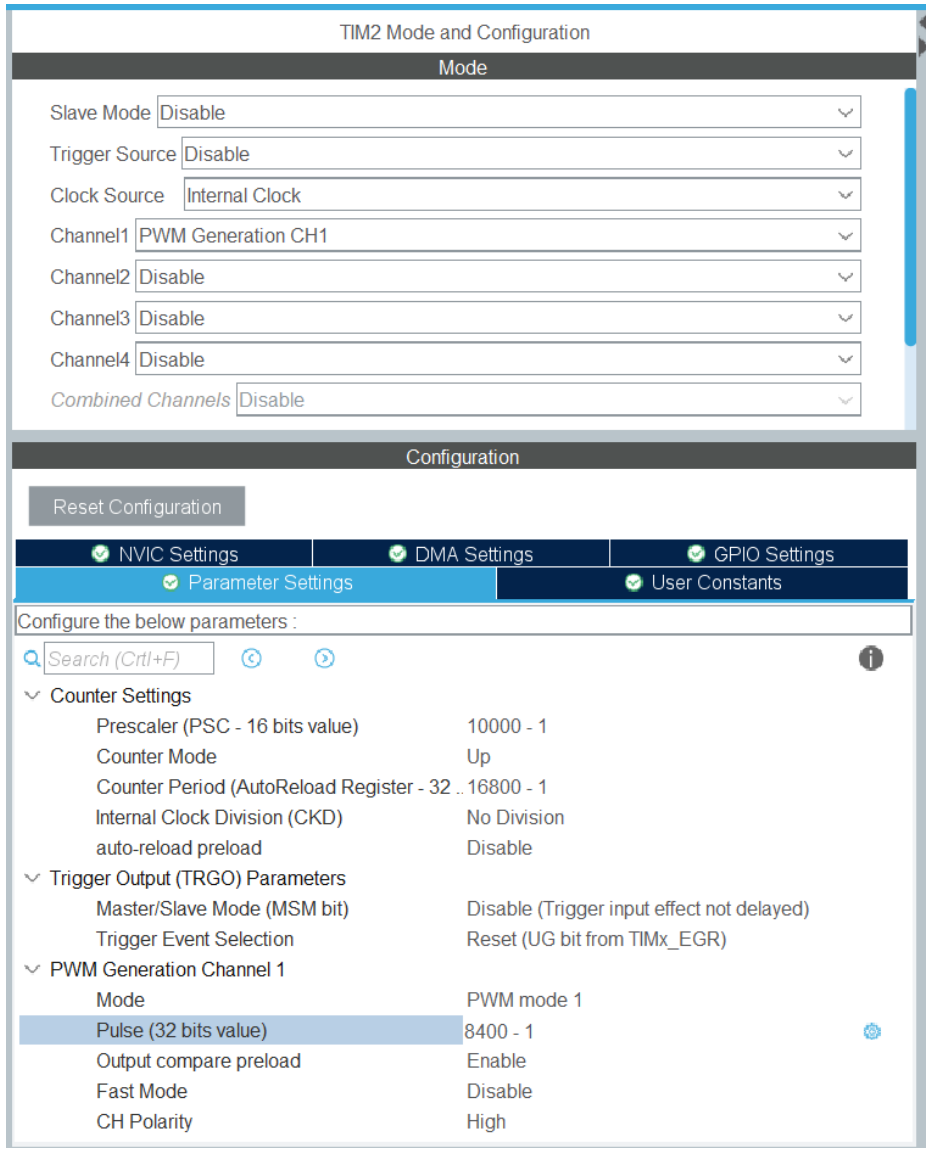
- A digital signal generated by the MCU for a variety of purposes, e.g.,
  - Set the position of a standard servo motor
  - Control the rotational speed of a continuous rotation motor
  - Control the luminescence of a light emitter
  - Control the rotational speed of a standard electric motor with external driver circuitry
  - Control a photovoltaic solar battery charger's charging rate
  - Generate a waveform such as sine, triangle and square
  - Generate acoustic sound waveform

PWM configuration

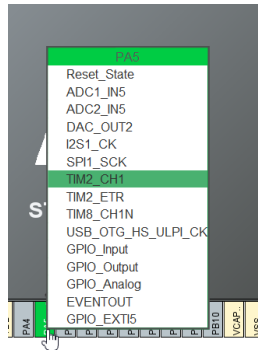


- Frequency
  - 50 Hz (or equivalently 20 ms period) is common for servo control
- Duty cycle
  - The percentage of the amount of time the signal is in the high state during a period
- PWM signals are generated by a Timer

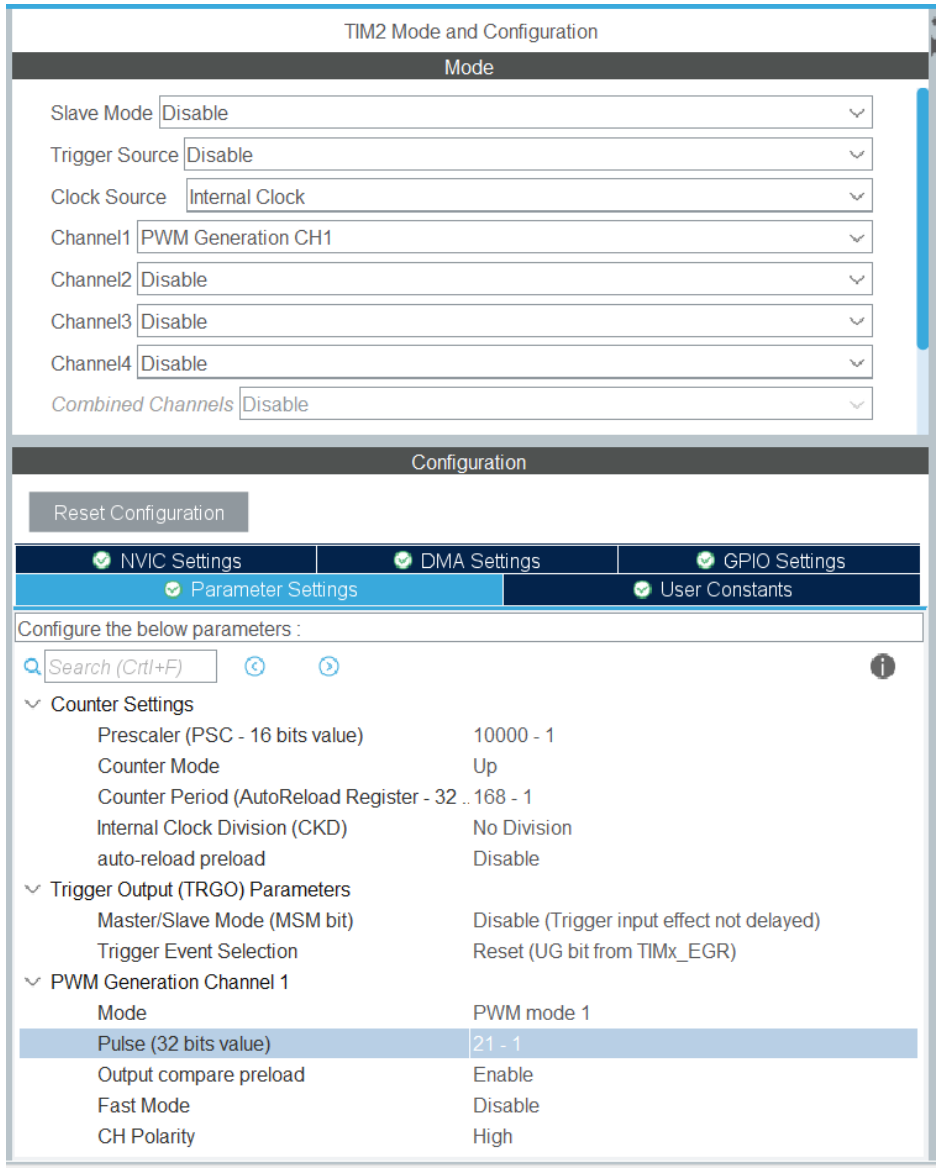
Generate PWM to toggle the LED



- Configure TIM2 for PWM
  - TIM2 Channel 1 (CH1), assign to PA5
  - Clock Source: Internal Clock (84 MHz)
  - Channel 1: PWM Generation CH1
  - Prescaler: 10000
    - So, counter frequency = 84 MHz/10000 = 8400 Hz
  - Counter Period: 16800
    - So, timer resets every 16800/8400 Hz = 2 sec
    - The PWM signal frequency is therefore 2 sec
  - Pulse: 8400
    - Duty cycle = Pulse / Period = 8400 / 16800 = 50%
  - `HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);` in `main.c` before `while(1) loop`
- Result: the green led (PA5) toggles at 1Hz



Use PWM to dim the LED



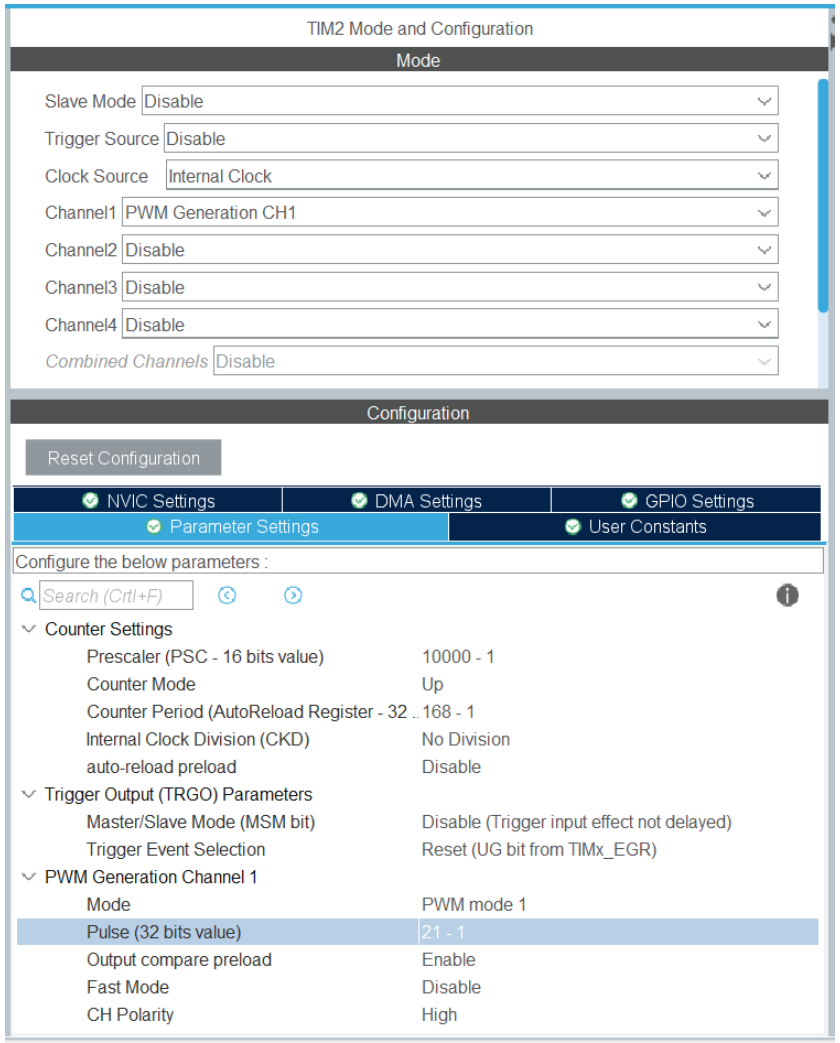
- Based on the previous example, modify
  - Prescaler: 168
    - So, the counter frequency is 50 Hz
  - Pulse: 21
    - So, the duty cycle is 12.5%
- Result
  - The green led is on but looks dim
- Exercise
  1. Try some other Prescaler and Pulse values
  2. Assign TIM2_CH1 to PA0, and use an oscilloscope to visualize the signal

Use PWM to control a servo motor

- A typical servo motor expects to be updated every 20 ms with a pulse between 1 ms and 2 ms, or in other words, between a 5 and 10% duty cycle on a 50 Hz waveform. With a 1.5 ms pulse, the servo motor will be at the natural 90 degree position. With a 1 ms pulse, the servo will be at the 0 degree position, and with a 2 ms pulse, the servo will be at 180 degrees.

https://learn.sparkfun.com/tutorials/pulse-width-modulation/all

Update the duty cycle in the program



- The Pulse value is stored in the register
  - TIM2->CCR1
  - Timer 2 Channel 1's Capture Compare Register
  - This register value can be updated in user code

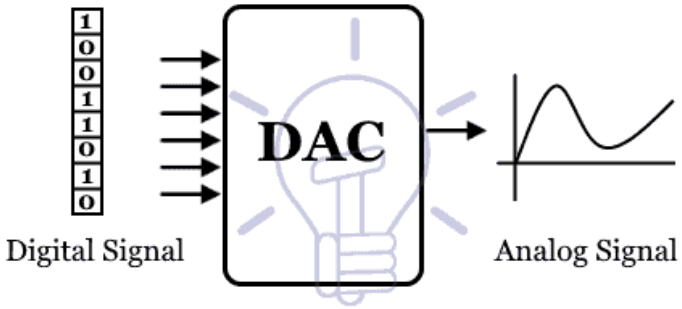
```c
int main(void)
{
  /* USER CODE BEGIN 1 */
  int pulse = 20 - 1;
  uint8_t dir = 1;
  /* USER CODE END 1 */
```

Experiment: Vary the CCR1 value in the while(1) loop
Result: the green led will gradually light up and dim down.

```c
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    if(pulse==80) dir = 0;
    if(pulse==1) dir = 1;
    if(dir == 1){
      pulse++;
    } else {
      pulse--;
    }
    TIM2->CCR1 = pulse;
    HAL_Delay(20);
```

Exercise: This execution is blocking. Change it to non-blocking mode by using a different timer (e.g., TIM11) to trigger the adjustment of TIM2->CCR1, every 20 ms.

DAC



Digital To Analog Converter

https://www.st.com/resource/en/application_note/cd00259245-audio-and-waveform-generation-using-the-dac-in-stm32-products-stmicroelectronics.pdf

Figure 3. DAC interaction without DMA

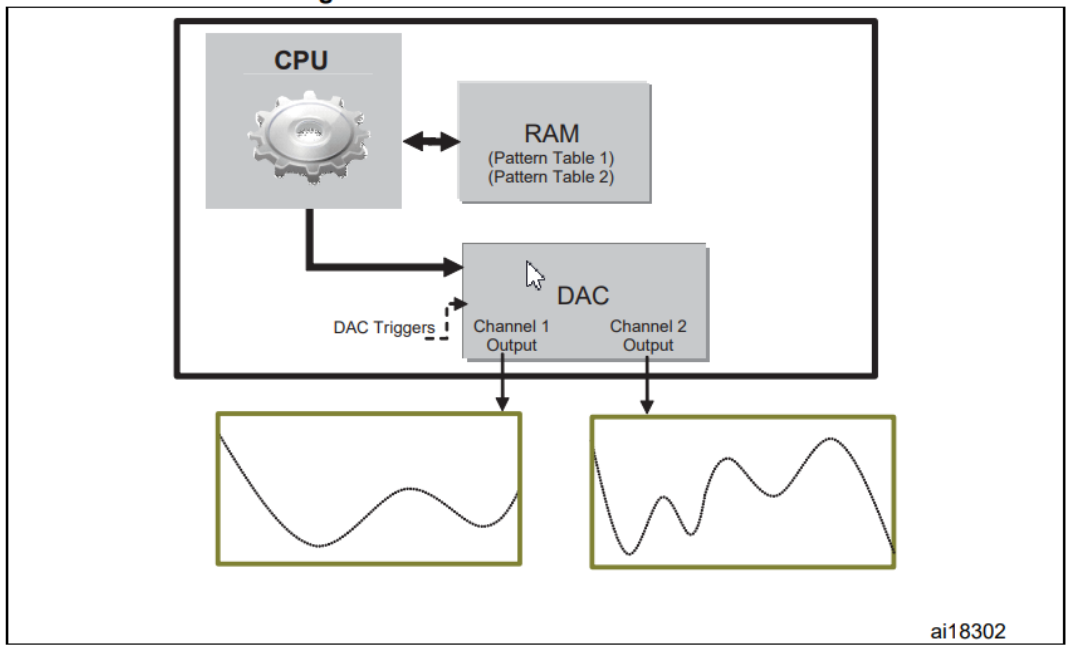
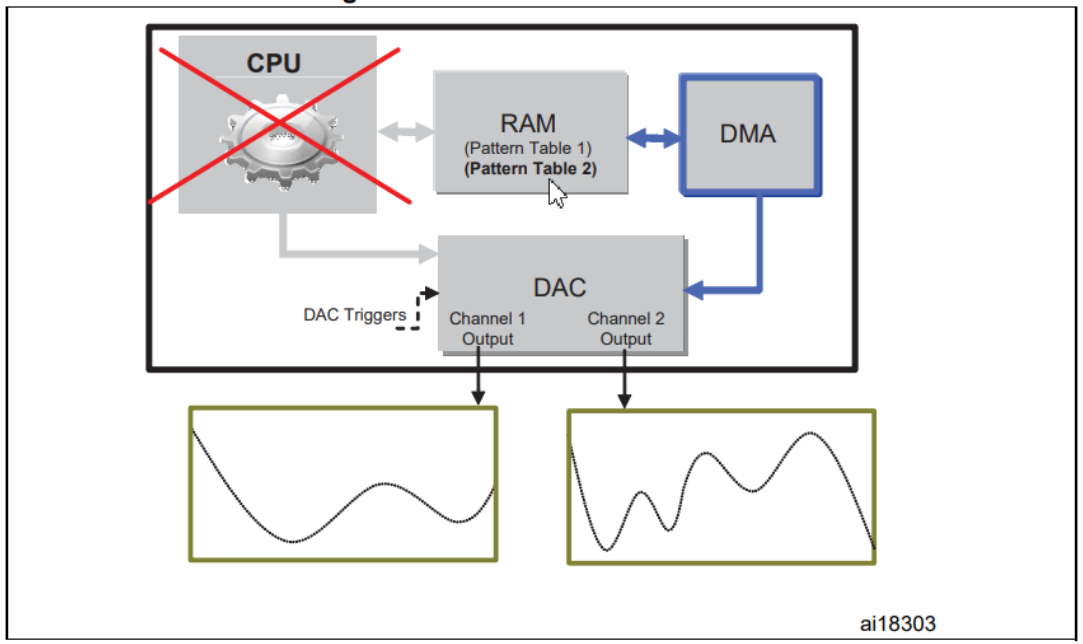
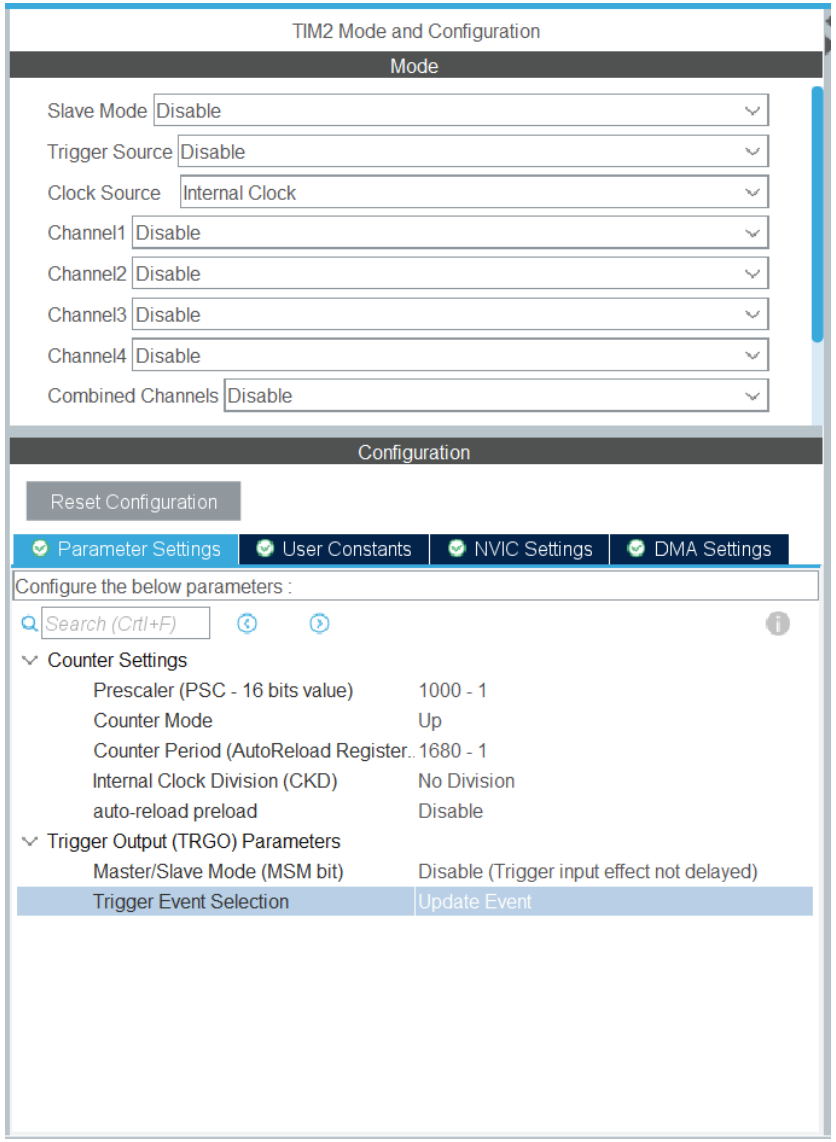
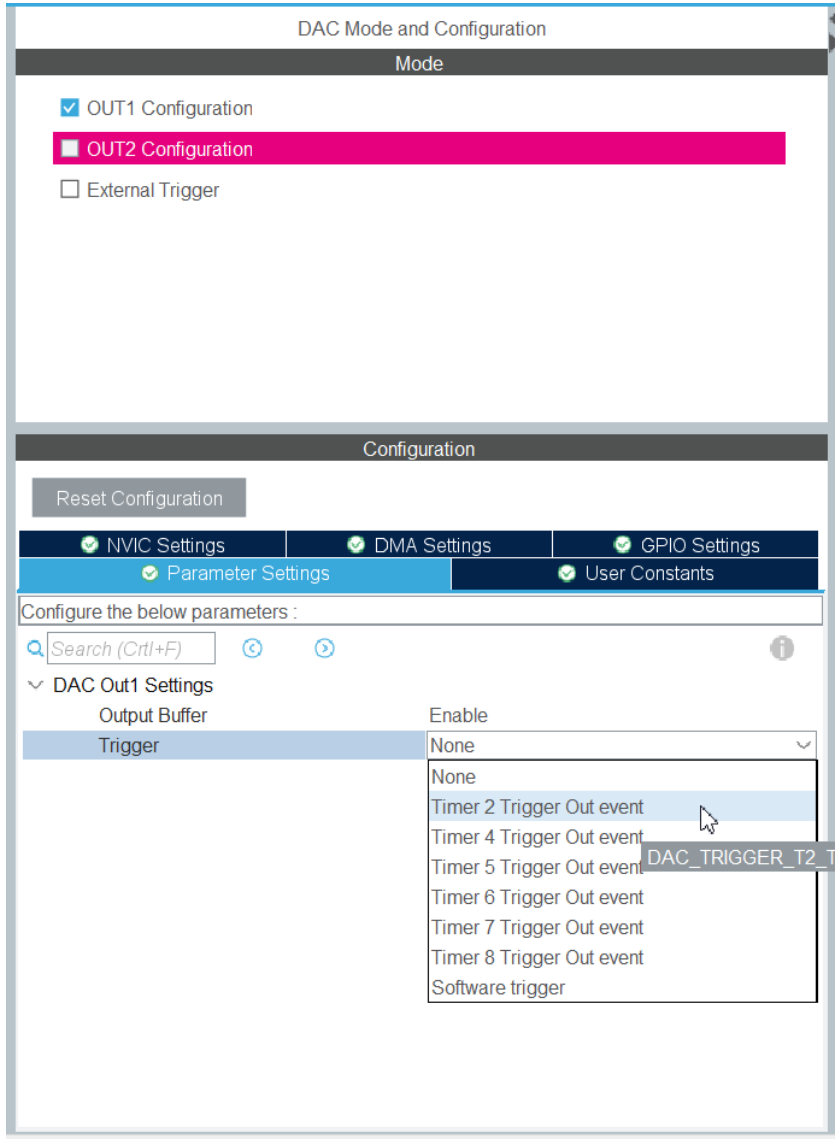


Figure 4. DAC interaction with DMA



DAC with DMA





- Output a sine wave through DAC Channel 1, assigned to Pin PA4
- The DAC is triggered by Timer 2
- TIM2 Trigger Event Selection is set to Update Event
- TIM2's clock source is 84 MHz
- Prescaler: 1000
- Period: 1680
- So, 84 MHz/1000/1680 = 50 Hz
- This means there are 50 DAC conversions per second
- The sine wave is represented by NSAMP = 100 samples, meaning 100 DAC conversions per sine cycle
- So, the sine wave frequency is 50/100 = 0.5 Hz
- The analog signal output is used for driving a LED
- See video clip

DAC with DMA

```
/* Private define --------
/* USER CODE BEGIN PD */
#define NSAMP 100
#define PI 3.1415926
/* USER CODE END PD */

uint16_t sine_data[NSAMP];
for(int i=0; i<NSAMP; i++){
  sine_data[i] = (0xFFF+1)/2*(sin(i*2*PI/NSAMP)+1);
}

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_TIM2_Init();
MX_DMA_Init();
MX_DAC_Init();
/* USER CODE BEGIN 2 */

HAL_TIM_Base_Start(&htim2);
HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t *)sine_data, NSAMP, DAC_ALIGN_12B_R)
```

The while(1) loop is empty

- Video clip

The sampling step is $2\pi / n_s$ (number of samples).

The value of sin(x) varies between -1 and 1, we have to shift it up to have a positive sine wave with samples varying between 0 and 0xFFF (corresponding to the 0 to 3.3 V voltage range, where V_{REF} is set to 3.3 V).

$$y_{SineDigital}(x) = \left(\sin\left(x \cdot \frac{2\pi}{n_s}\right) + 1\right)\left(\frac{(0xFFF + 1)}{2}\right)$$

Digital inputs are converted to output voltages on a linear conversion between 0 and V_{REF+}.

The analog output voltages on each DAC channel pin are determined by the equation

$$DAC_{fOutput} = V_{REF}\frac{DOR}{DAC_MaxDigitalValue + 1}$$

For right-aligned 12-bit resolution: DAC_MaxDigitalValue = 0xFFF
For right-aligned 8-bit resolution: DAC_MaxDigitalValue = 0xFF

So the analog sine waveform $y_{SineAnalog}$ can be determined by the following equation

$$y_{SineAnalog}(x) = 3.3Volt\frac{y_{SineDigital}(x)}{0xFFF + 1}$$

To set the frequency of the sine wave signal, the user has to set the frequency ($f_{TimerTRGO}$) of the timer trigger output.

The frequency of the produced sine wave is

$$f_{Sinewave} = \frac{f_{TimerTRGO}}{n_s}$$

Final notes

- In all experiments, we used the default Clock Configuration (HCLK = 84 MHz)
  - For control projects, the lowest frequency that is high enough to meet the application's response requirement should be used. Lower CPU speed means lower power consumption.
  - For digital processing, the clock frequency may be configured so that some timer (e.g., the trigger timer for DAC) can produce the desired signal frequency (e.g., 44.1 kHz for audio).

References

- STM32CubeIDE Advanced Debug Features: Part 3
  - https://youtu.be/Eg_GLvLHM1o
- Programming with STM32 Getting Started with the Nucleo Board and C/C++, by Donald Norris, McGraw Hill Education, 2018
  - https://www.amazon.com/Programming-STM32-Getting-Started-Nucleo/dp/1260031314
- Getting Started with STM32 and Nucleo Part 6: Timers and Timer Interrupts | Digi-Key Electronics
  - https://youtu.be/VfbW6nfG4kw
- Receive data using UART in STM32 || Poll || Interrupt || DMA
  - https://youtu.be/l5msGz2tZP0