life.augmented

STM32C0 MCU Series

Your next 8-bit MCU is a 32-bit – STM32C0!

Workshop

Nicolas Fillon – Field Application Engineer

Tim Nakonsut– Product Marketing Engineer

STM32

Agenda

- Overview of STM32C0 family
- Lab 1 Blinky : Blink an LED by software
- Lab 2 PWM : Use hardware (PWM timer) to blink an LED
- Lab 3 EXTI : External Interrupt using a user push button
- Lab 4 Printf : Printf debugging using UART communication
- Lab 5 LL Driver : Example using Low Layer Driver
- Optional Lab 6 ADC : ADC example using DMA

life.augmented

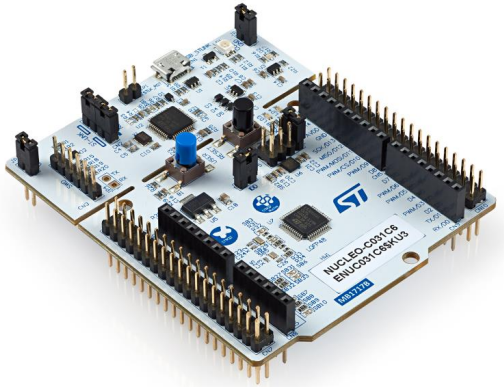
STM32C0 Labs: Introduction

STM32C0 Labs Introduction

- You have received by email a link to download a file containing:
  - The pre-requisites (host machine, operating system…)
  - The Installation Procedure (STM32CubeIDE, STM32Cube_FW_C0…)
  - The source code to be added (to copy and paste code to the Labs)
- Make sure you are meeting the Pre-requisite criteria as explained in the document
- Make sure you have installed the tools as described in the document
- Have the document open in the “Code to be added section” at the end of the document to copy and paste source code when needed during the Labs.

STM32C0 Labs Introduction

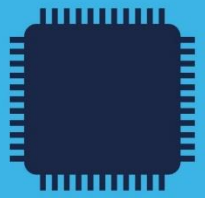
- For these Labs you will need:
- The NUCLEO-C031C6:



- A Micro–USB Cable:

- Optional: one single jumper cable for the ADC Lab:

life.augmented



Your next 8-bit MCU is a 32-bit!

Just call it STM32C0!

Introducing the STM32C0
ST's most affordable 32-bit MCU

Streamline costs without comprising performance

$	**Affordability**	Helps you reduce costs thanks to an attractive price point and an optimized BOM
APPROVED QUALITY CONTROL	**Reliability**	Benefits from proven STM32 quality & reliability
	Continuity	Consistent pinout with STM32G0 Shares same technological platform

life.augmented

STM32 MCUs and MPUs portfolio

MPU

STM32MP1
4158 CoreMark
Up to 800 MHz Cortex-A7
209 MHz Cortex-M4

High Perf MCUs

STM32**F2**	STM32**F4**	STM32**H5**	STM32**F7**	STM32**H7**
Up to 398 CoreMark 120 MHz Cortex-M3	Up to 608 CoreMark 180 MHz Cortex-M4	1023 CoreMark 250 MHz Cortex-M33	1082 CoreMark 216 MHz Cortex-M7	Up to 3224 CoreMark Up to 550 MHz Cortex -M7 240 MHz Cortex -M4

Mainstream MCUs

STM32**F3**	STM32**G4**
245 CoreMark 72 MHz Cortex-M4	569 CoreMark 170 MHz Cortex-M4

Mixed-signal MCUs

STM32**C0**	STM32**F0**	STM32**G0**	STM32**F1**
114 CoreMark 48MHz Cortex M0+	106 CoreMark 48 MHz Cortex-M0	142 CoreMark 64 MHz Cortex-M0+	177 CoreMark 72 MHz Cortex-M3

Ultra-low Power MCUs

STM32**L0**	STM32**L1**	STM32**L4**	STM32**L4+**	STM32**L5**	STM32**U5**
75 CoreMark 32 MHz Cortex-M0+	93 CoreMark 32 MHz Cortex-M3	273 CoreMark 80 MHz Cortex-M4	409 CoreMark 120 MHz Cortex-M4	443 CoreMark 110 MHz Cortex-M33	651 CoreMark 160 MHz Cortex-M33

Wireless MCUs

STM32**WL**	STM32**WB**	STM32**WBA**
162 CoreMark 48 MHz Cortex-M4 48 MHz Cortex-M0+	216 CoreMark 64 MHz Cortex-M4 32 MHz Cortex-M0+ ●	407 CoreMark 100 MHz Cortex-M33

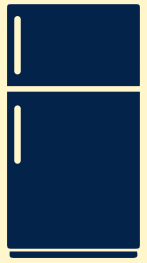
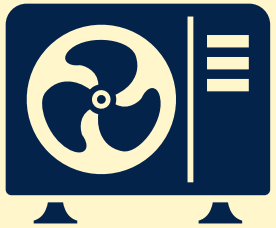
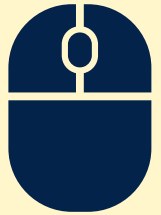
life.augmented

■ Latest product generation

● Radio co-processor only

More than **60,000 customers**
Over **10 billion STM32** shipped since 2007

Perfect fit for applications typically served by 8-bit/16-bit MCUs

Smart homes	Industrial devices	Consumer devices	
			
Fridges Ovens Coffee machines	Industrial pumps Fan control Circuit breakers	Smoke detectors Fire detectors Alarms	PC peripherals & accessories

life.augmented

Affordability

Attractive price point

Most cost-effective STM32 MCU

Compact

9 tiny packages down to:

- 3 x 3 mm 20-pin QFN
- WLCSP12
- 8-pin SO8N

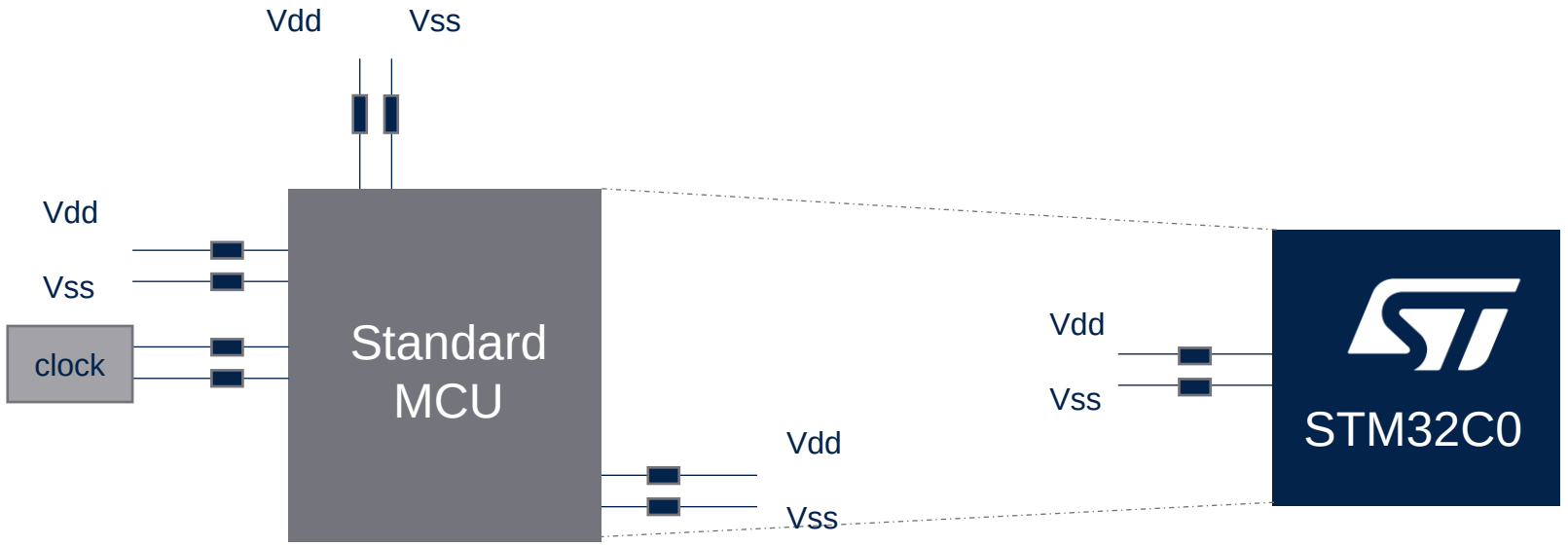
Reduced BOM costs

- Smallest package: max I/O count
- Fewer surrounding components:
  - accurate internal high-speed clock 1% RC
  - only one power supply pair

life.augmented

Optimized BOM cost

The STM32C0 series lets designers do more with less



- Smaller package: **fewer surrounding components**
- Platform optimized with **1 power supply pair only**
- Embedded **high-speed clock with high accuracy**

life.augmented

Compact
Multiple packages

Easy handling

SO8N
TSSOP-20
LQFP32/ 48

Low thickness and tiny

20-pin UFQFPN 3 x 3 mm

28/32/48-pin UFQFPN

Lowest thickness, tiniest form factor

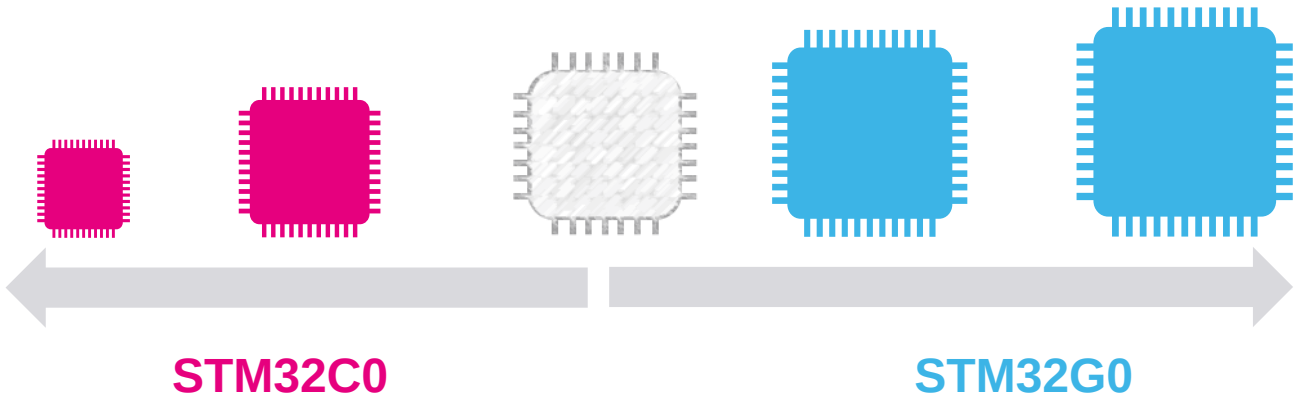
WLCSP12
1.70 x 1.42 mm

9 packages

life.augmented

The STM32 Continuum

The STM32C0 series uses the same 90nm technology as STM32G0, ensuring high quality standards



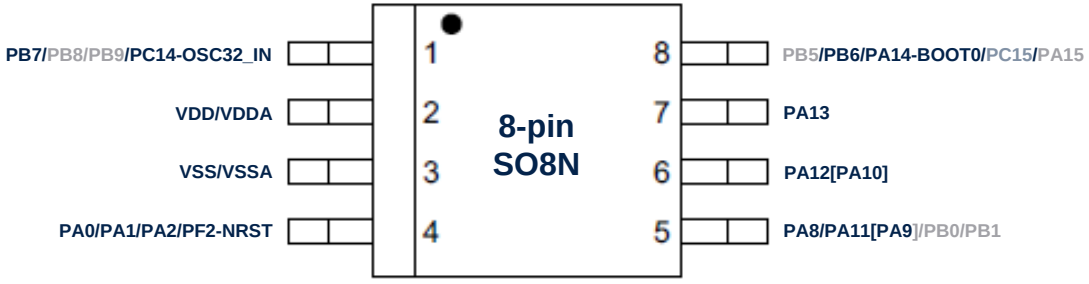
- Arm® Cortex®-M0+ running at 48MHz
- Delivers 44DMIPS instruction throughput with 114CoreMark performance
- Continuum with STM32G0 series
  - Consistent pinout
  - Same IP platform
  - Same technology platform

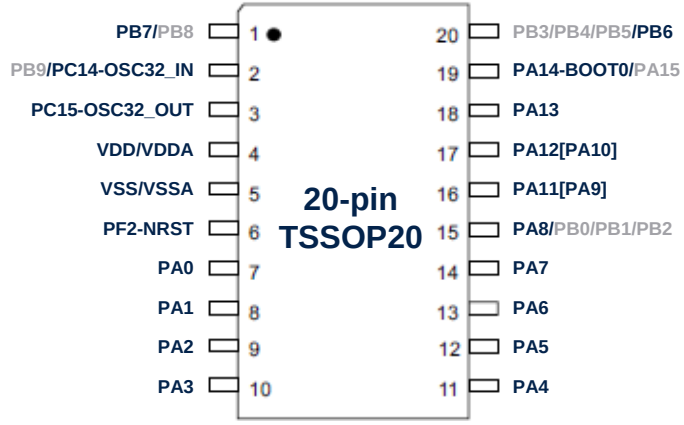
Safe in deliveries: 10-year longevity program
Renewed commitment every year

life.augmented

Easy porting with STM32G0

Consistent pinout with STM32G0 leaves room for future product upgrades





Consistent I/O footprint

Common pin location
for alternate functions & system

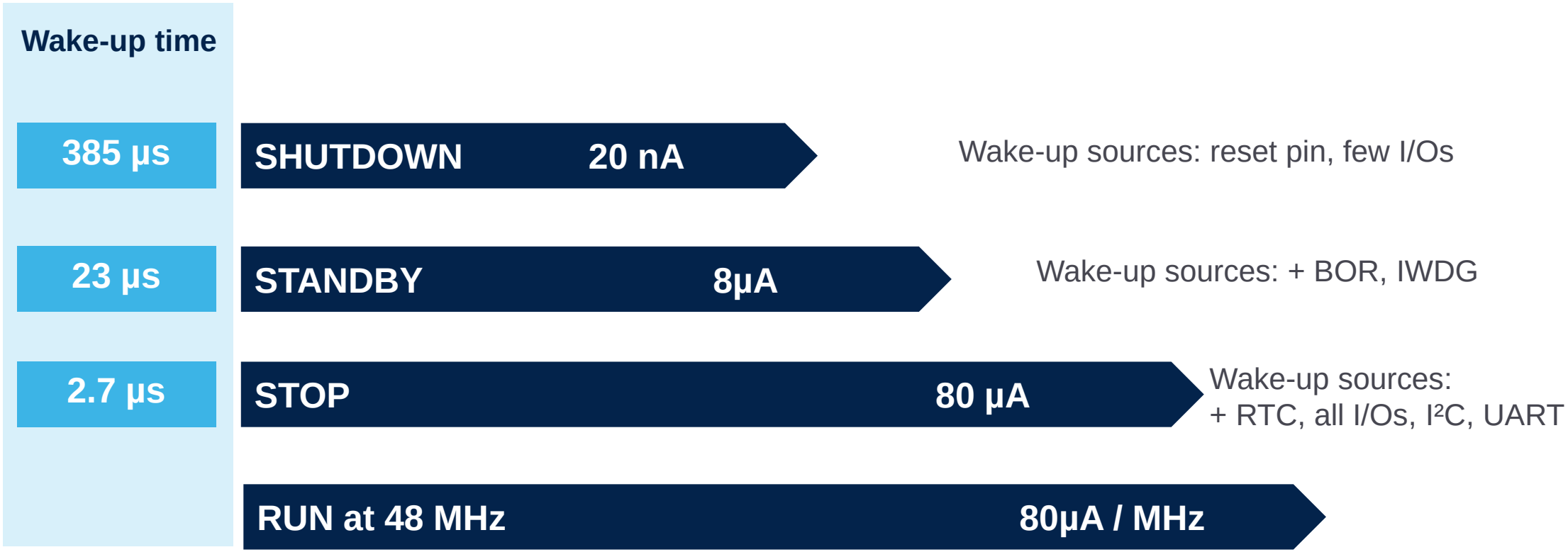
Maximum I/O ratio vs pin count

Common signals on STM32C011 and STM32G031

Legend: Common signals - STM32G031 only - STM32C011 only

Low-power modes for better efficiency

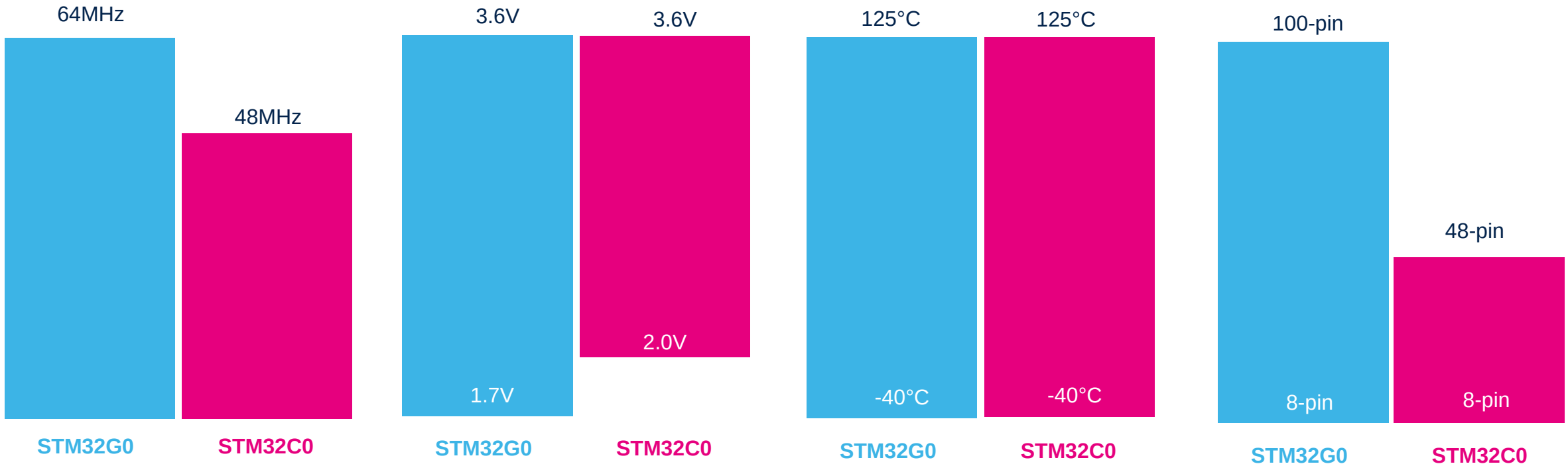
Excellent dynamic consumption



Conditions: 25°C, V_{DD} = 3V

life.augmented

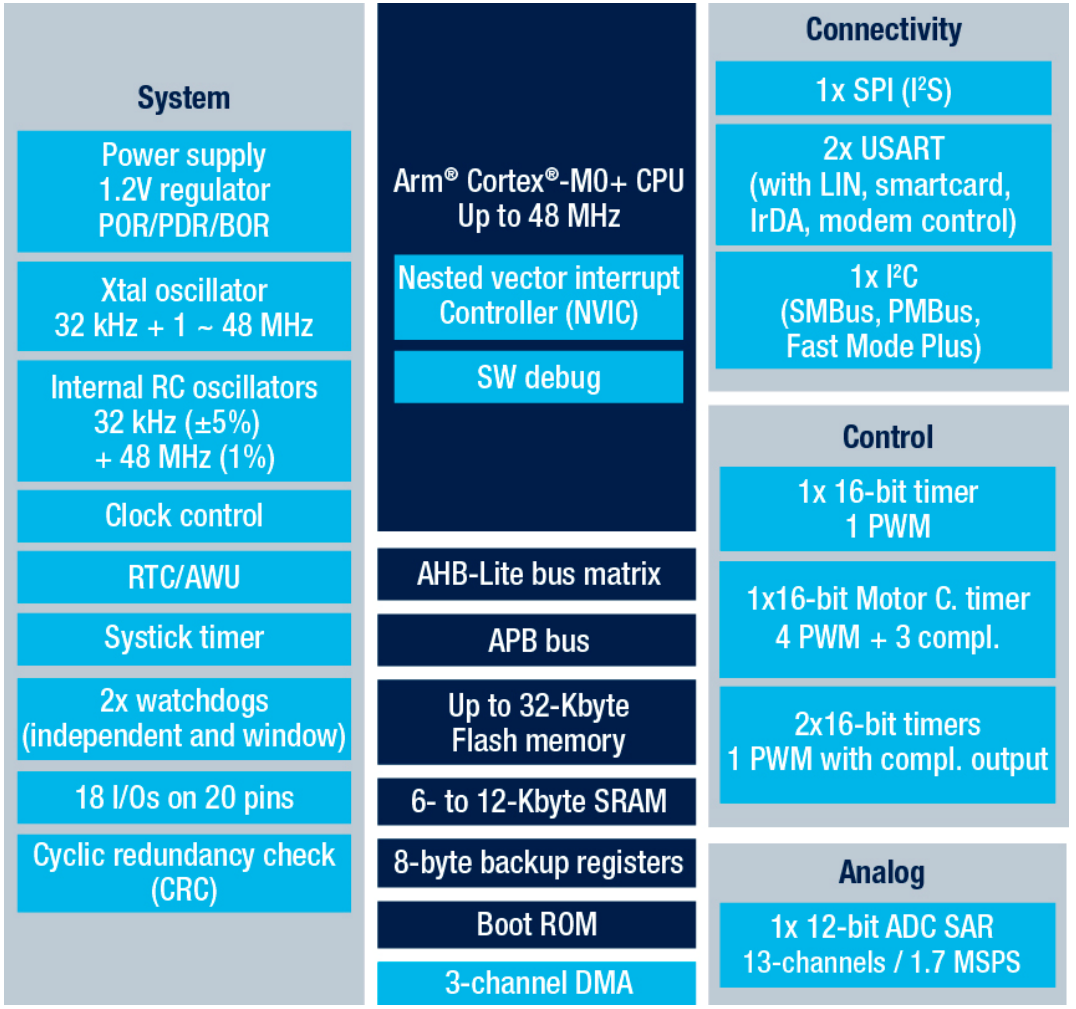
Performance benchmark STM32C0 & STM32G0



life.augmented

STM32C011 / C031 Block Diagram

- 32-bit Arm Cortex-M0+ core
- 2 to 3.6V power supply
- I/O ports maximization
- One supply pair
- 1% internal clock
- All clock sources
  - Low speed 32kHZ
  - High speed
  - Internal / External
- Direct Memory Access (DMA)

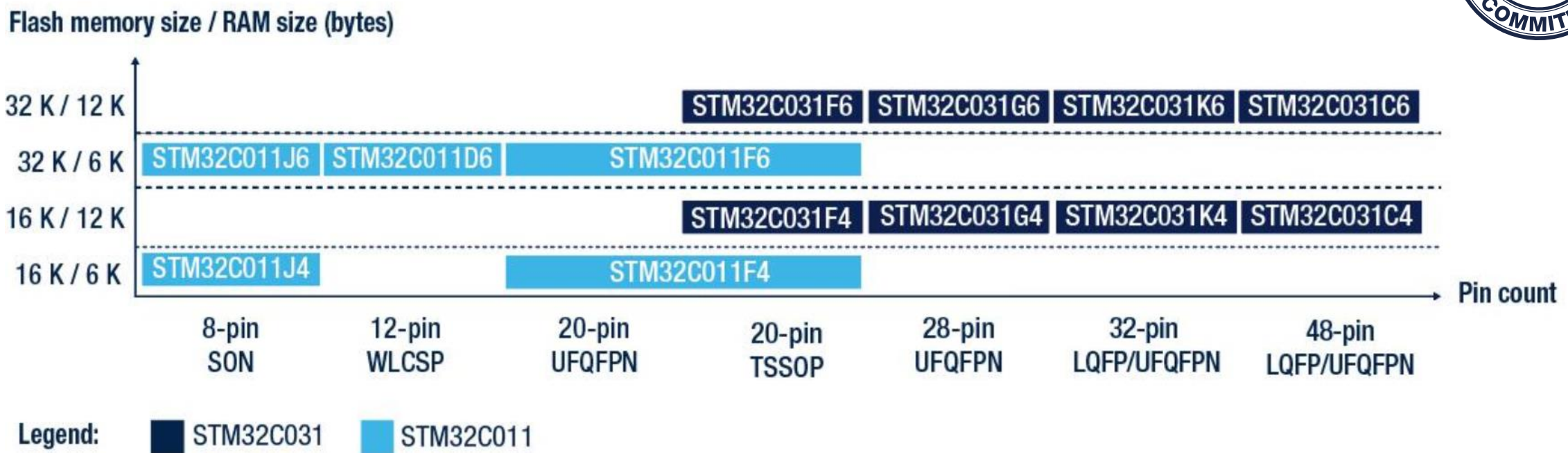


- Timers 16-bit with Motor Control features
- Communication peripherals incl.
  - 2xUSART
- Real-time Clock
- 12-bit Ultra-fast ADC
- Safety features
- Excellent dynamic consumption 80µA/MHz
- SRAM size:
  - STM32C011: 6Kbytes
  - STM32C031: 12Kbytes

life.augmented

STM32C0 portfolio

Same feature-set, different RAM size and packages

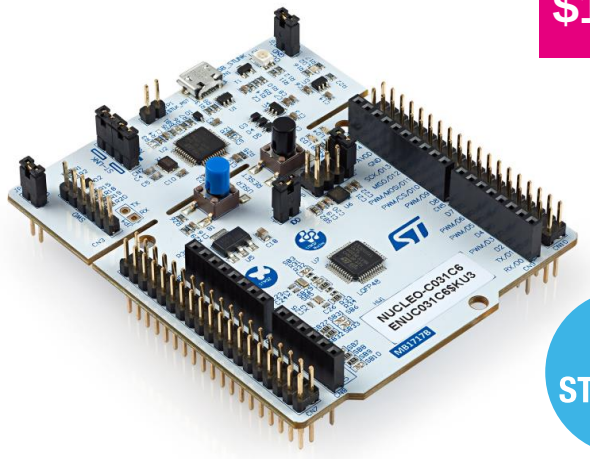


life.augmented

Development tools for STM32C0 series

Speed-up evaluation, prototyping and design

$10.32



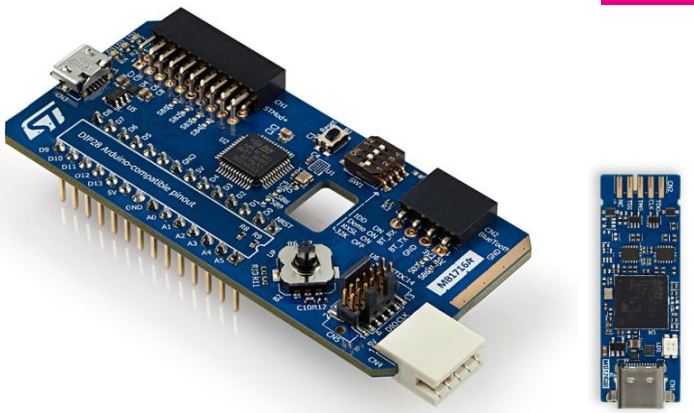
STM32 Nucleo

STM32 Nucleo with C031

Prototyping QFP48

NUCLEO-C031C6

$17

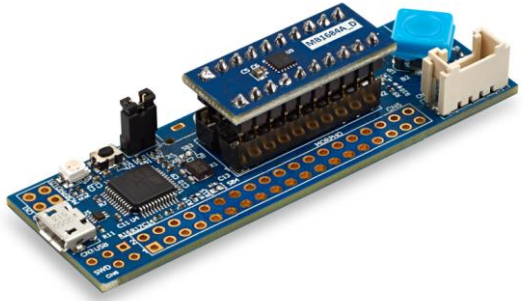


Discovery kit for C031

Mini evaluation board
Full voltage range 2.0 ~ 3.6 V
Standalone fast STLinkV3-Minie

STM32C0316-DK

$11



Discovery kit for C011

Ready to use wired sample
Daughter board QFN20/DIP20

STM32C0116-DK

Recommended Resell Price (RRP)

life.augmented

Leveraging STM32Cube software suite

Software Tools

Embedded Software

Complemented with Microsoft Azure RTOS

STM32 CubeMX
Configuration

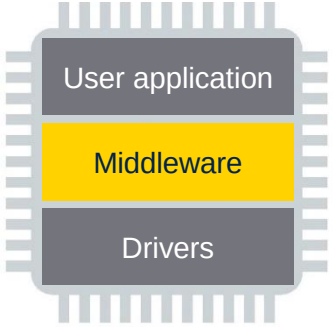
STM32 CubeIDE
Development

STM32 CubeProgrammer
Programming

STM32 CubeMonitor
Monitor

STM32 CubeMCU Packages
Packages

Github



Expansions

STM32 CubeExpansion

Azure RTOS

Azure RTOS ThreadX
Real-time operating system

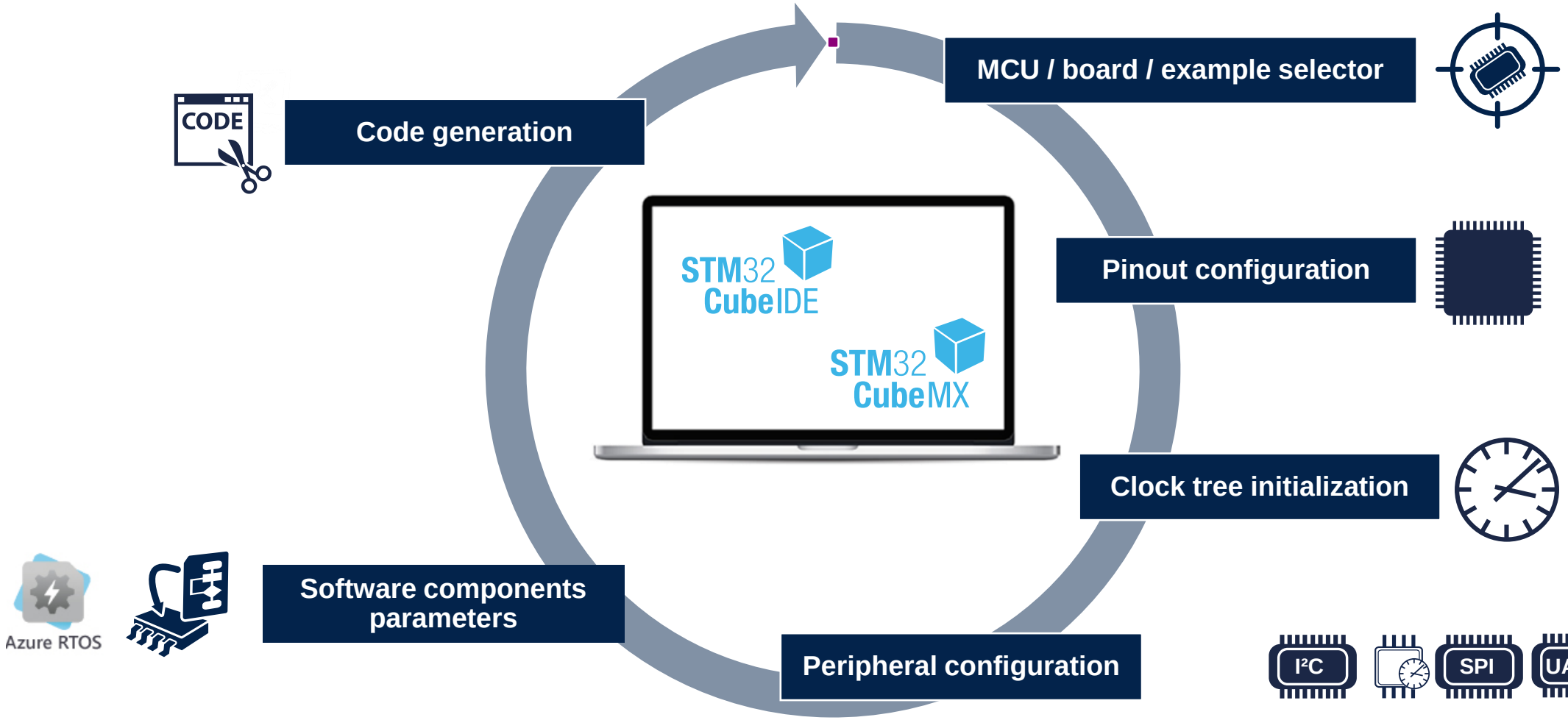
Azure RTOS FileX
FAT file system, fault tolerant

Azure RTOS NetX/NetX Duo
TCP/IP stacks

Azure RTOS USBX
USB stack, host and device

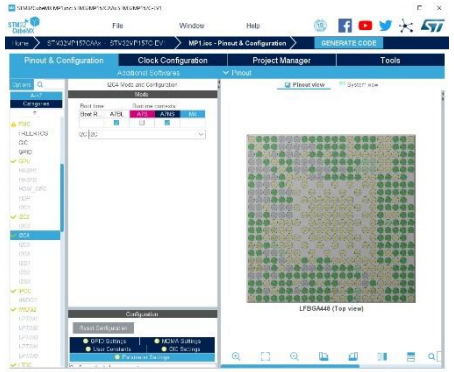
Free-of-charge production License

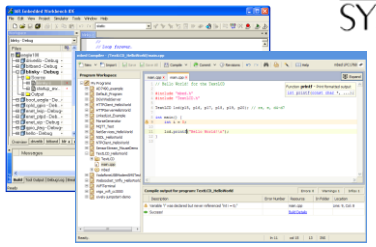
STM32Cube configuration tool



Software tools for STM32C0

Complete support of Arm Cortex-M0+ architecture





STM32CubeMX

Graphical tool for easy configuration

- Configure and generate code
- Peripherals and middleware configuration

IDEs Compile and Debug

Simple, powerful solutions

- Partners IDE (Arm KEIL) FREE
- IDE based on Eclipse FREE
- RTOS aware debug

STM32 Programming & Monitoring tools

STM32CubeProg
STM32CubeMonitor

- Device and memory configuration
- Program the application
- Monitor variables at run-time

life.augmented

Releasing your creativity

/STM32

@ST_World

community.st.com

www.st.com/STM32C0

wiki.st.com/stm32mcu

github.com/stm32-hotspot

www.st.com/stm32-mcu-developer-zone

life.augmented

life.augmented

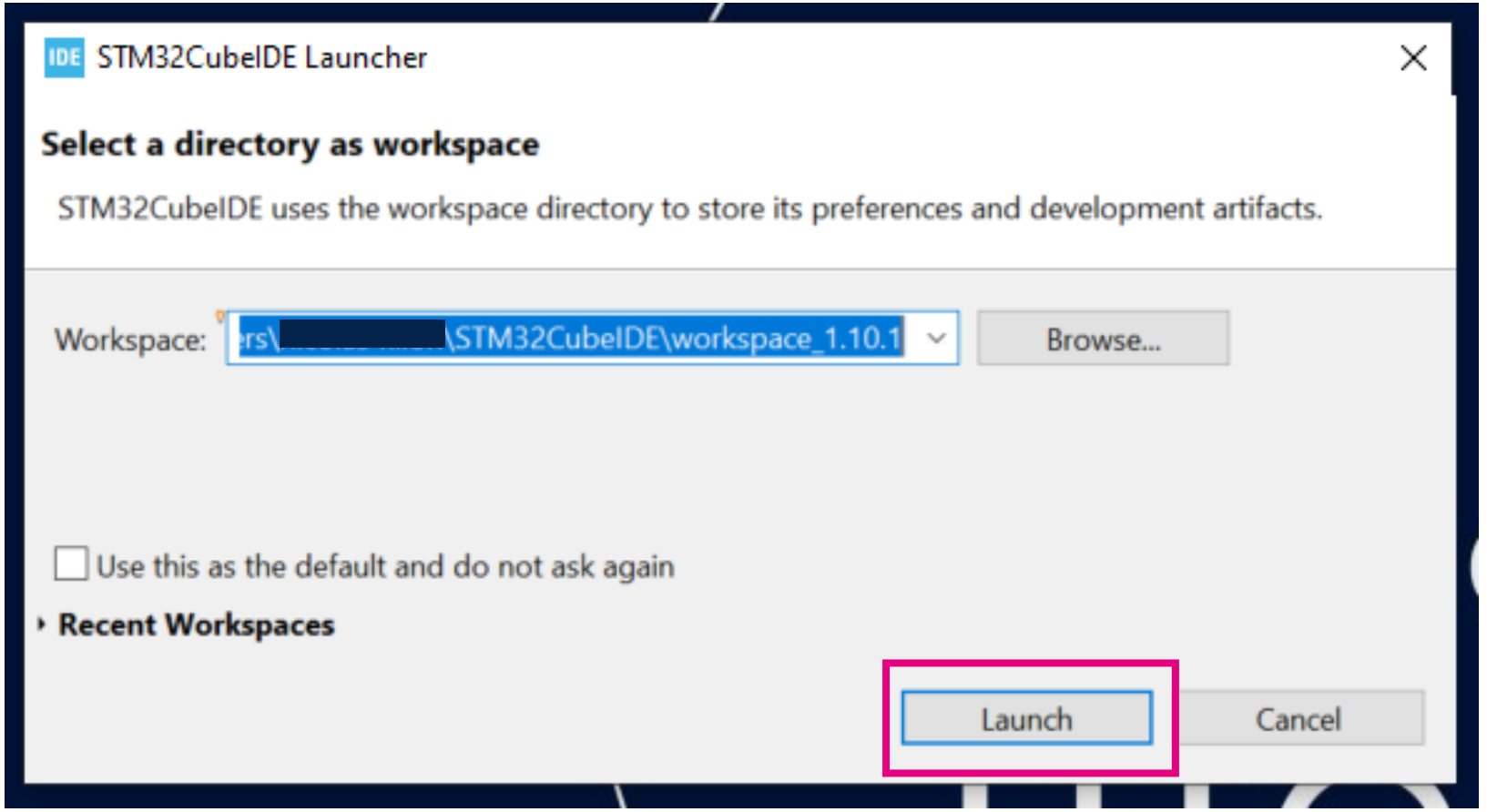
STM32C0 Lab 1: Blinky

Lab: blinky

Objective:

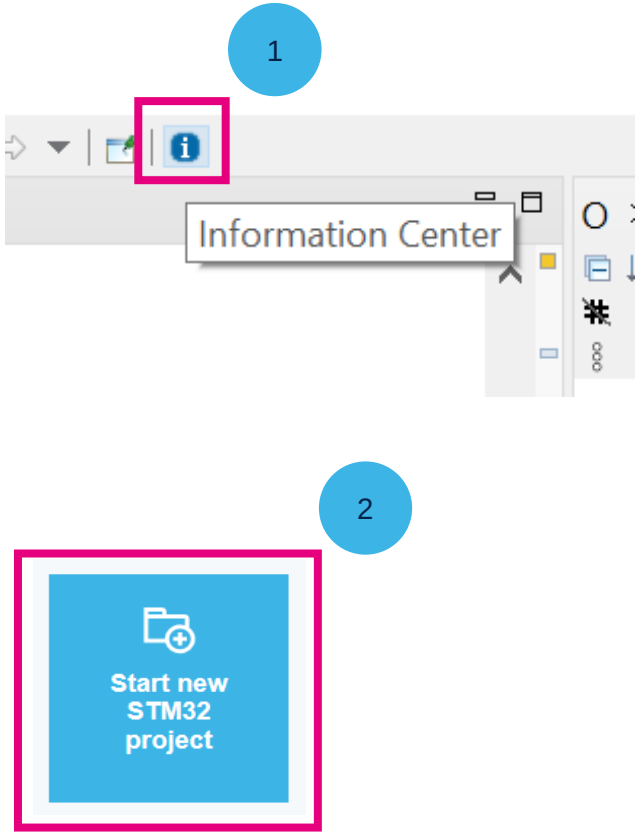
- The objective of this lab is to generate a simple project using STM32CubeIDE Software.
- In this example we are going to blink one of the LEDs present on the STM32C031 Nucleo board, connected to PA5 of the STM32C0 MCU.

Run STM32CubeIDE



Create new project

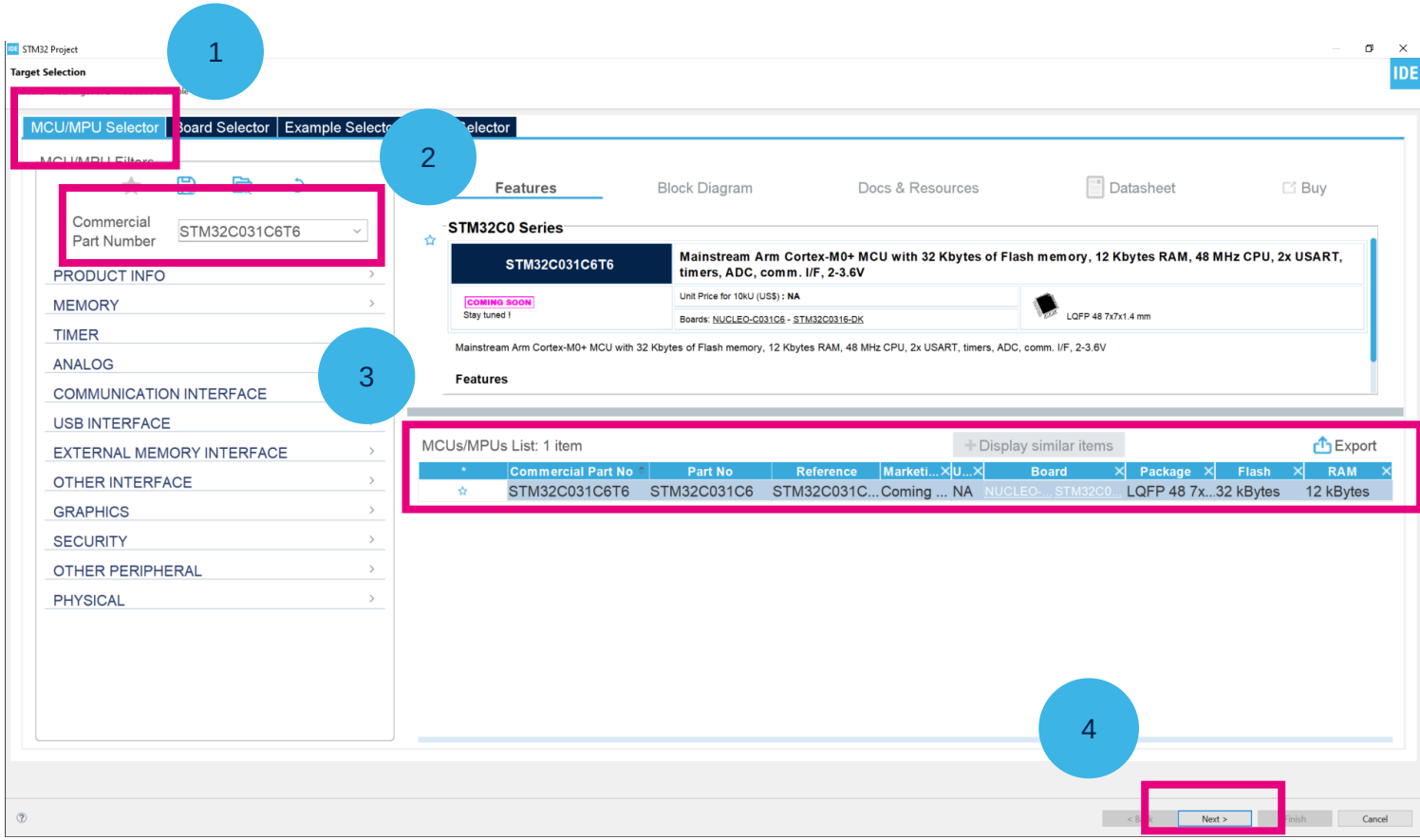
- Click the Information Center icon 1
- Start new STM32 project 2



life.augmented

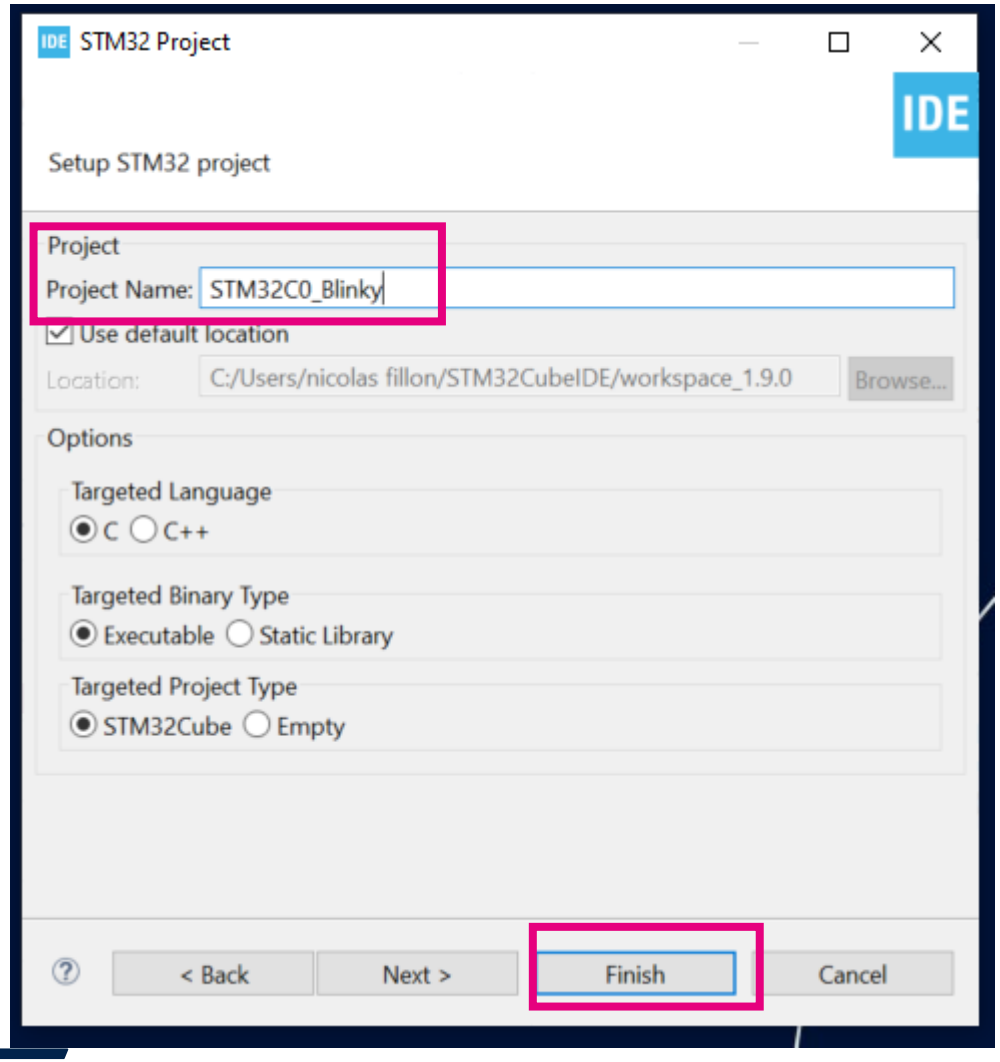
Create new project

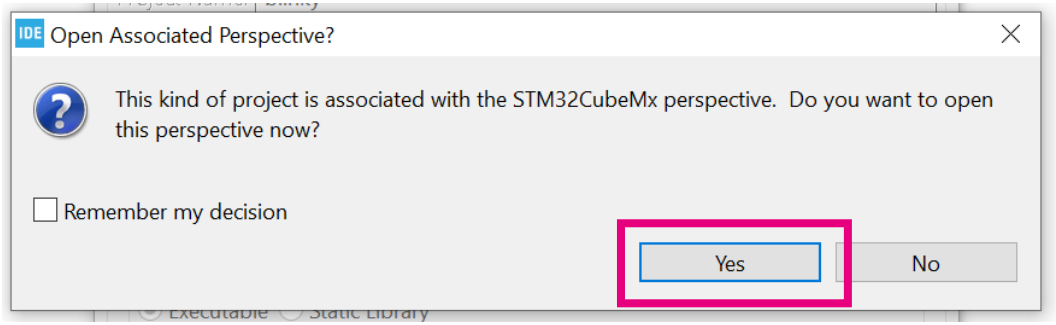
- Click Access To MCU Selector 1
- Type: “STM32C031C6T6” in the Part Number Search 2
- Then Select STM32C031C6T6
  - LQFP48, 32 kB Flash 3
- Click “Next” 4



life.augmented

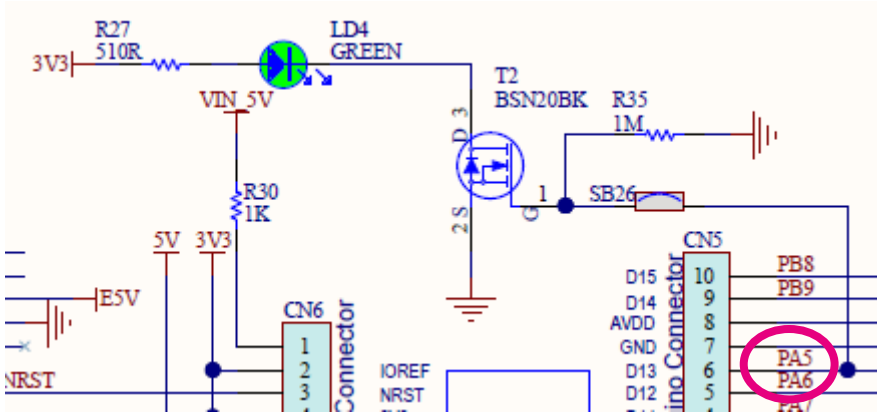
Give a name to the project



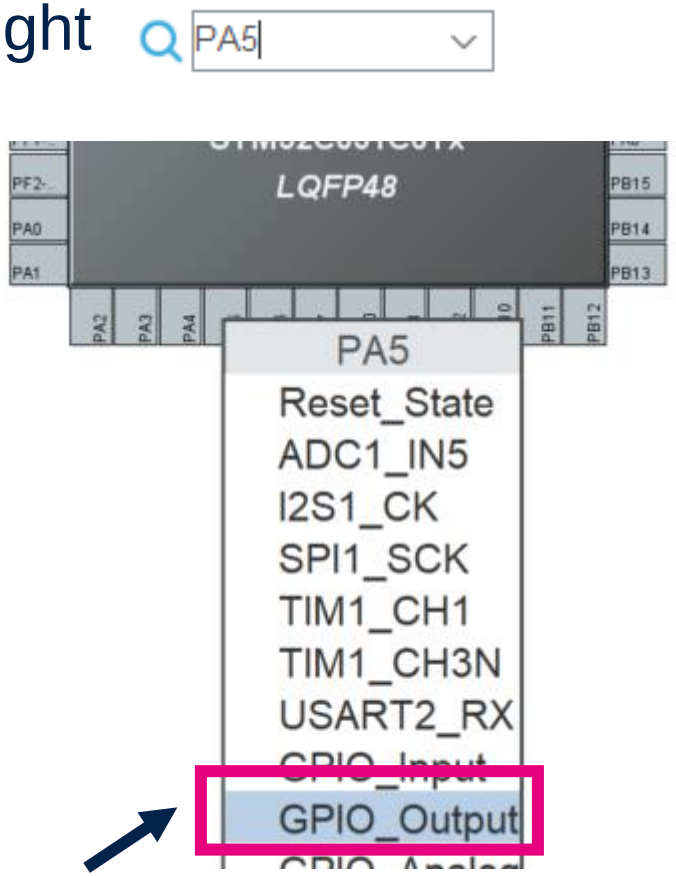


Pin configuration

- In this example we are going to use one of the LEDs present on the STM32C031 Nucleo board (connected to PA5 as seen in the schematic below)
- Search for PA5 in the search window at the bottom right
- Left-click PA5 and set it to GPIO_Output mode



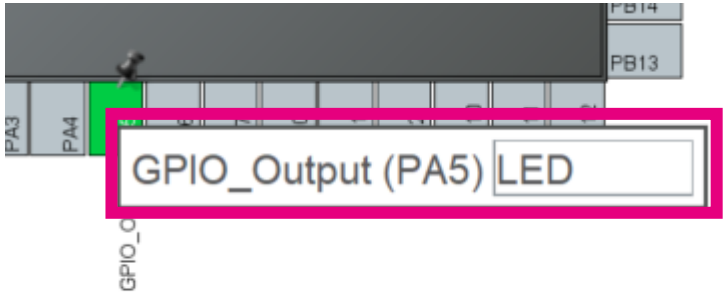
From Nucleo schematics



Enter a User Label name for PA5

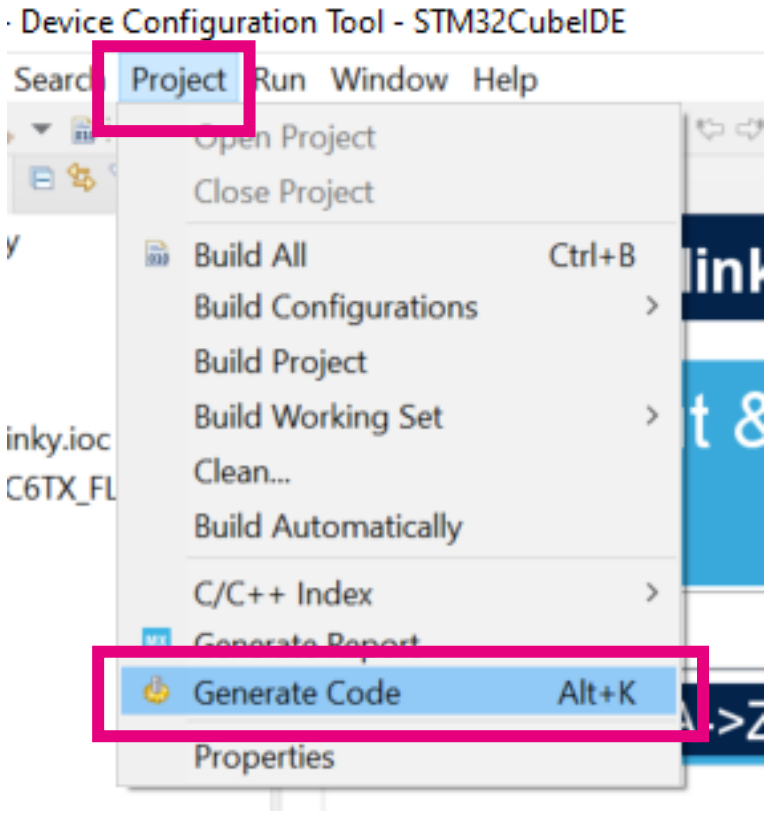
- Add a label to PA5 like “LED”
- Right click on PA5 on the Pinout View like this and click “Enter User Label”:

- Enter “LED” and press Enter:



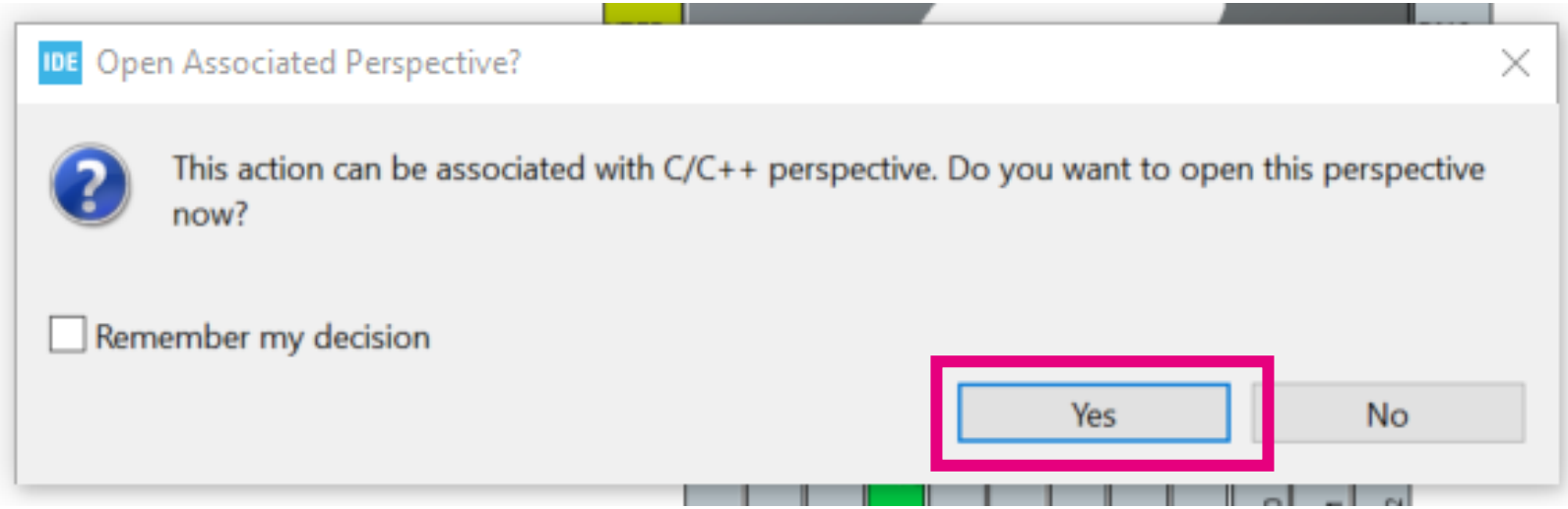
Generate source code

- Project -> Generate Code or ALT + K or Save the Project



Change Perspective

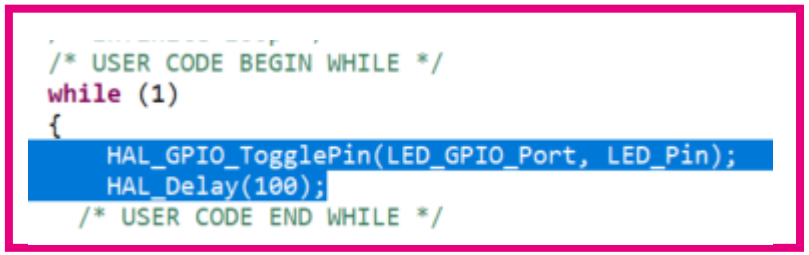
- Click Yes

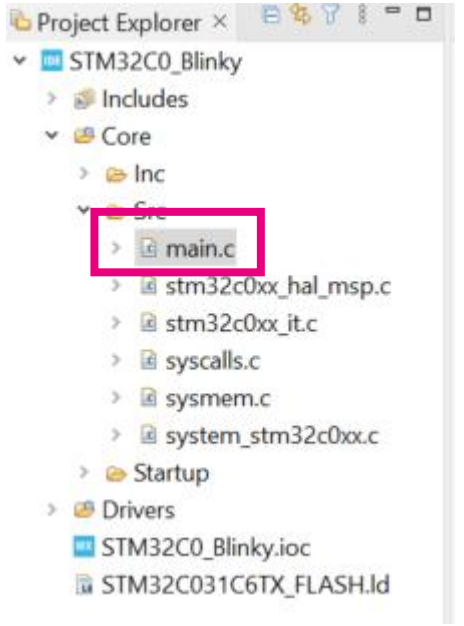


Step 4: toggle the LED

- In the Project Explorer Window:
- Expand the file tree and open the main.c file
- Add the following code inside the while(1) loop in “main.c” between the “USER CODE BEGIN WHILE” and “USER CODE END WHILE”

```
HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
HAL_Delay(100);
```



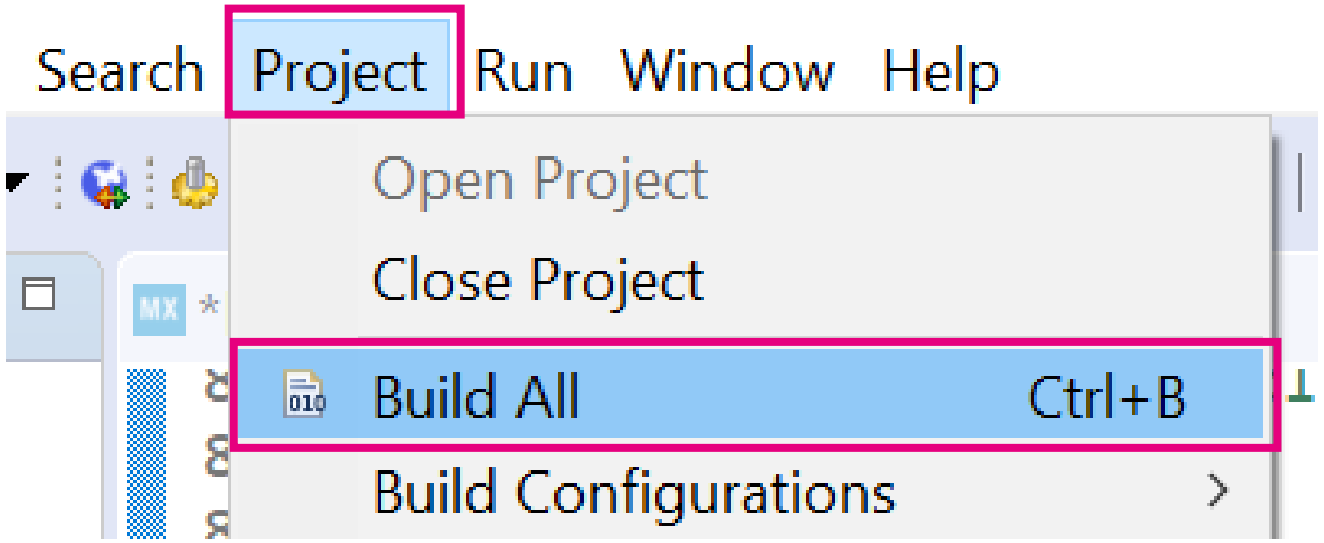


Note: Code within the “USER CODE BEGIN WHILE” / “USER CODE END WHILE” section will be preserved after regeneration by STM32CubeIDE

life.augmented

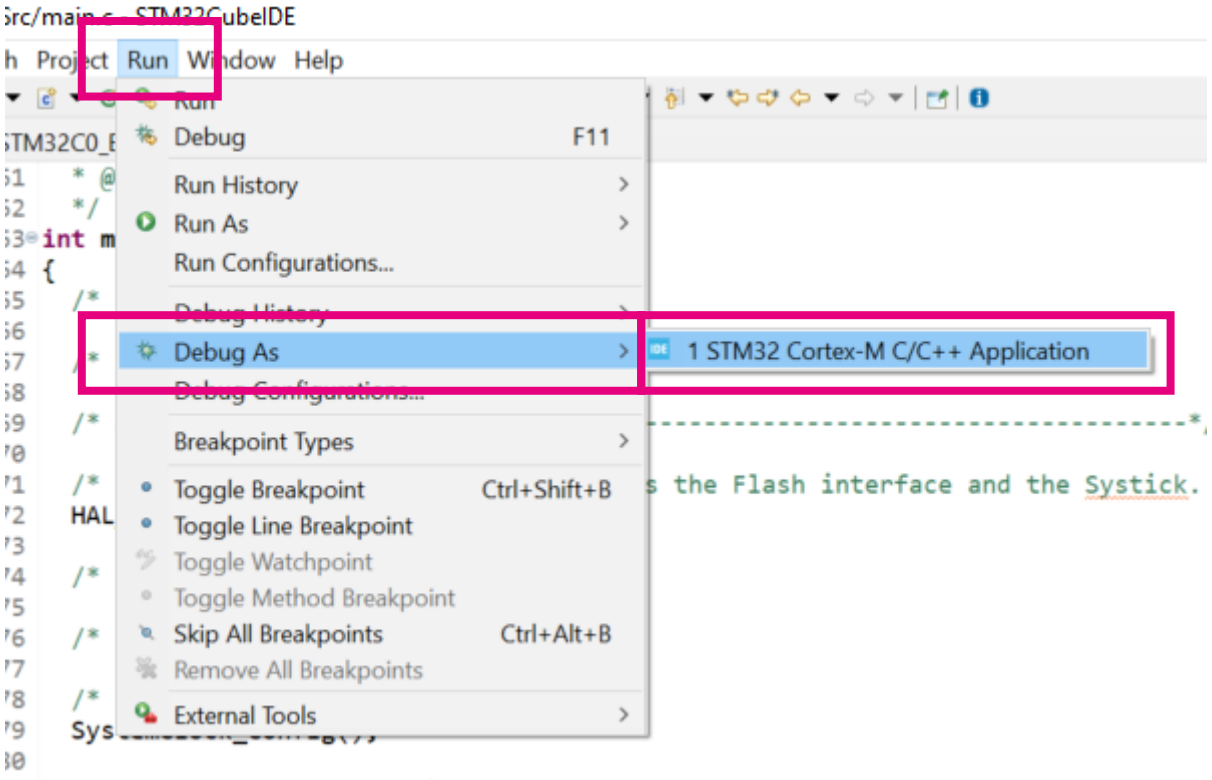
Build the project

- Project -> Build All or CTR +B

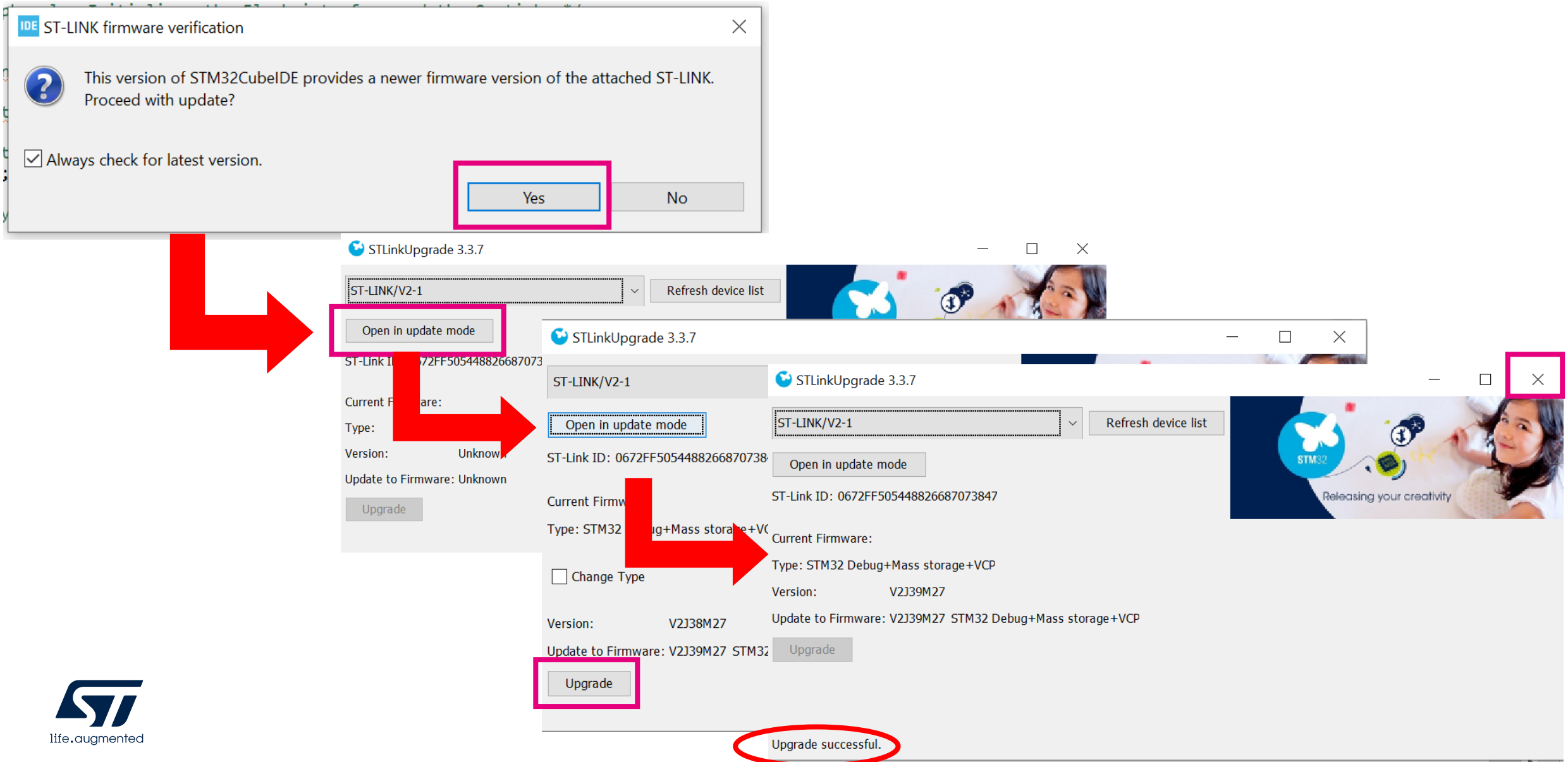


Debug

- Run -> Debug As -> STM32 Cortex-M C/C++ Application

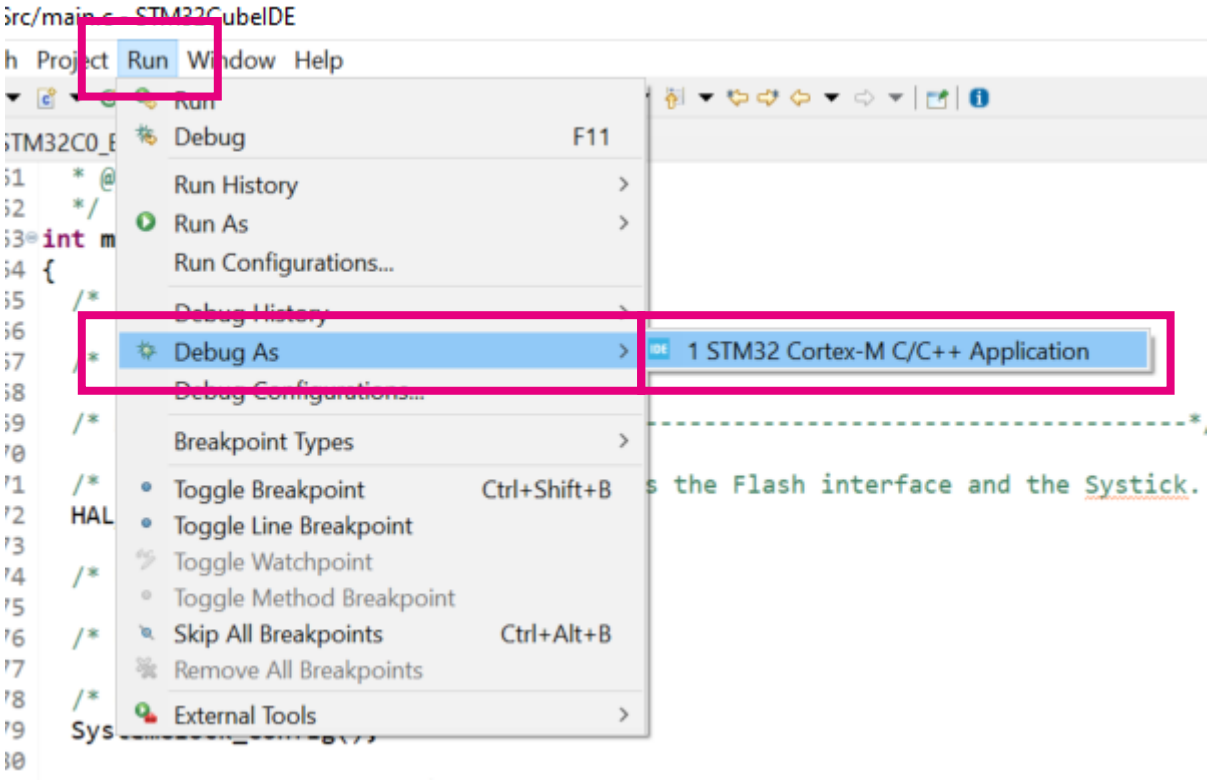


ST-LINK Firmware Update (if needed)

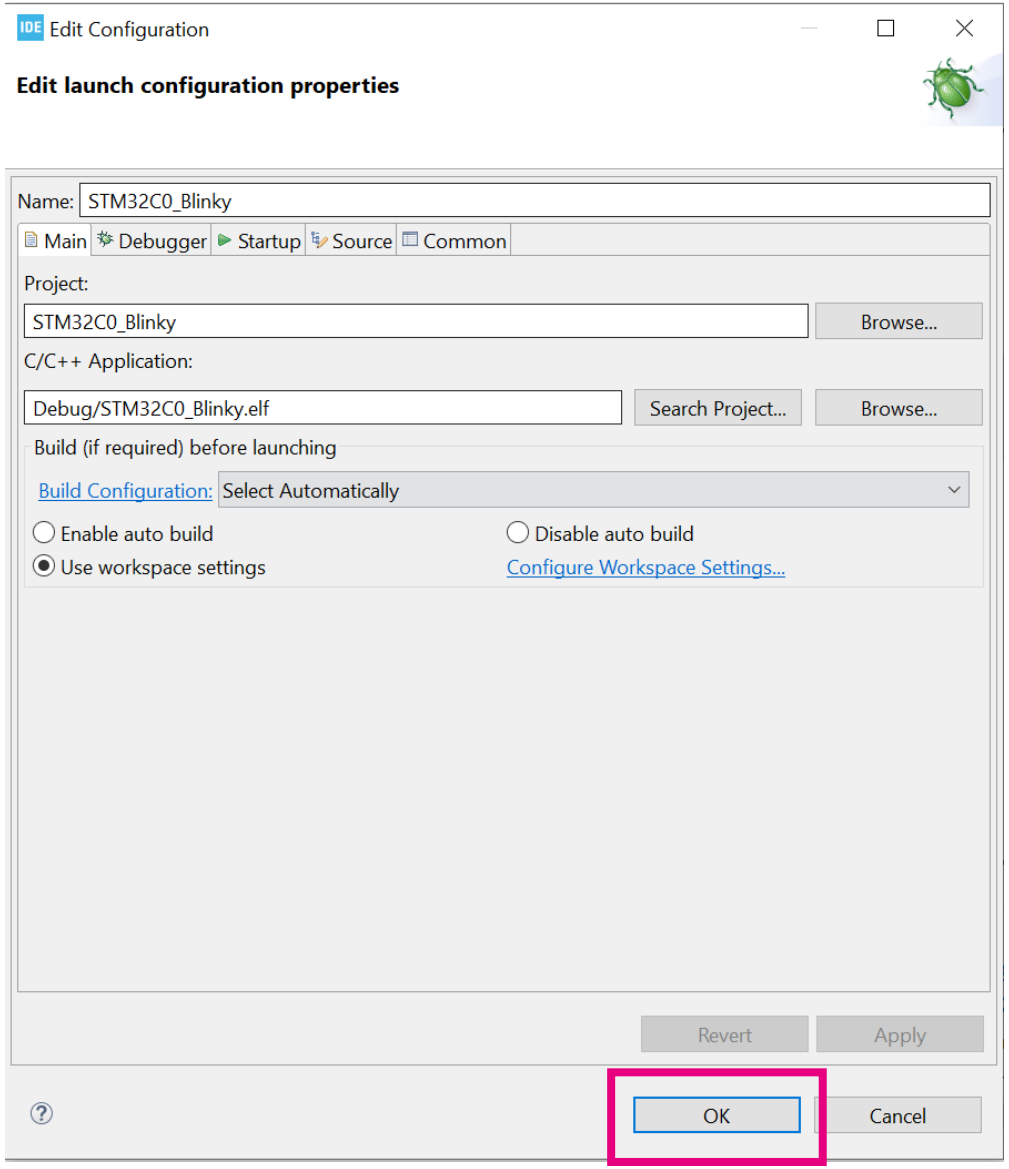


Debug

- Run -> Debug As -> STM32 Cortex-M C/C++ Application



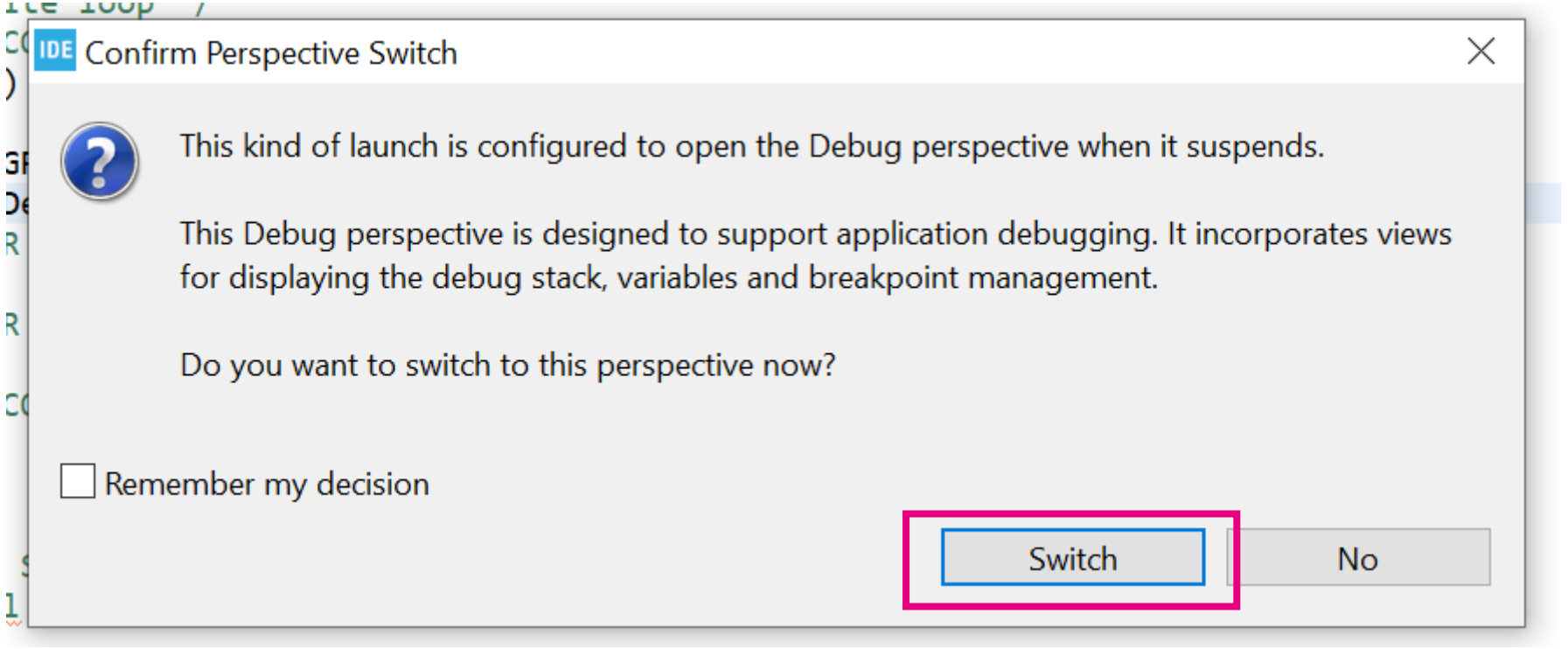
Check configuration



Note: It might ask you to upgrade the ST-LINK Driver once you press OK, do the upgrade as requested

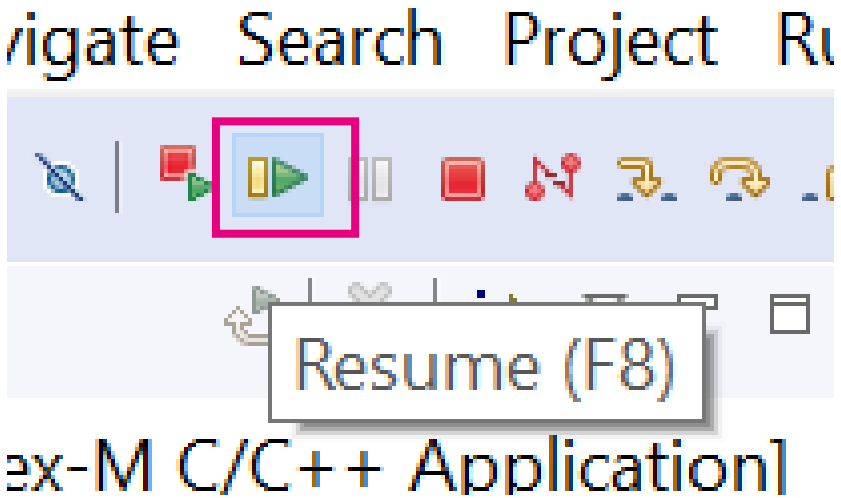
Switch the perspective

- Click Switch



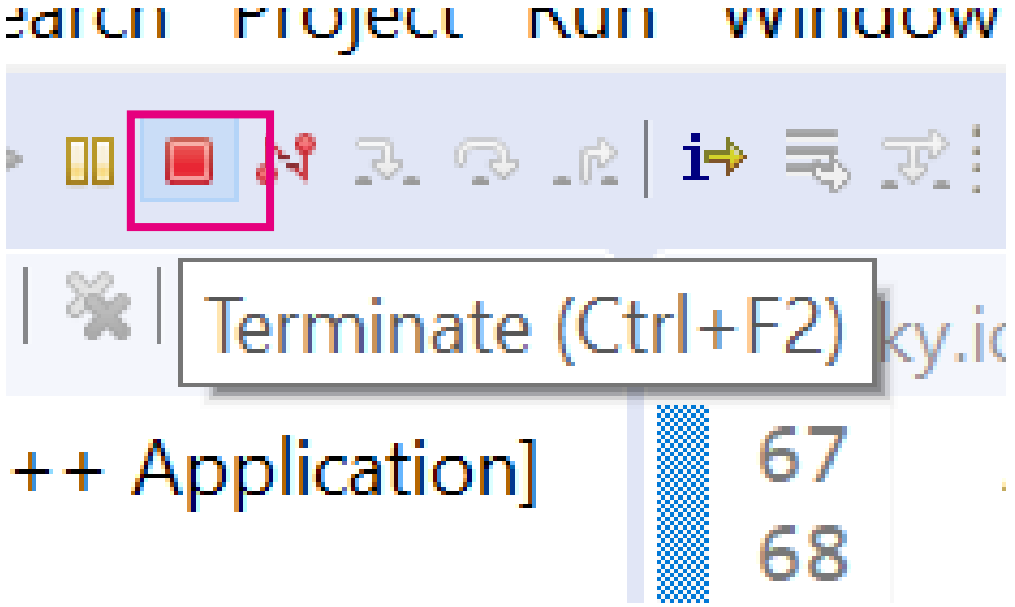
Run the code

- Click on the Resume Icon or F8

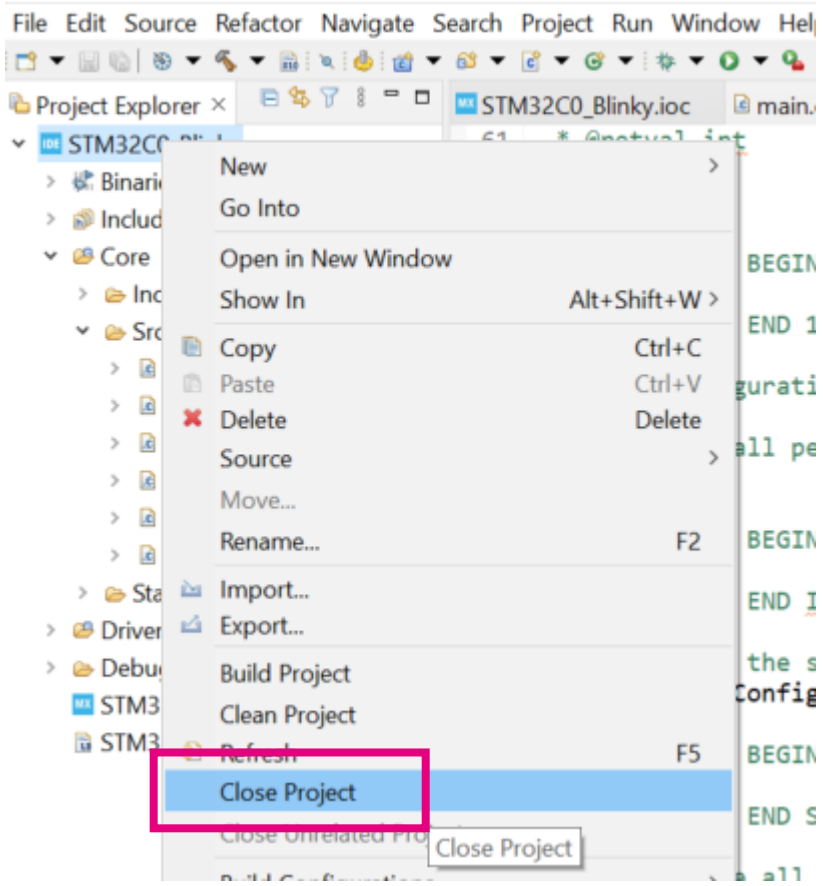


Enjoy the LED (LD4) blinking!

Exit the debugger



Close the project



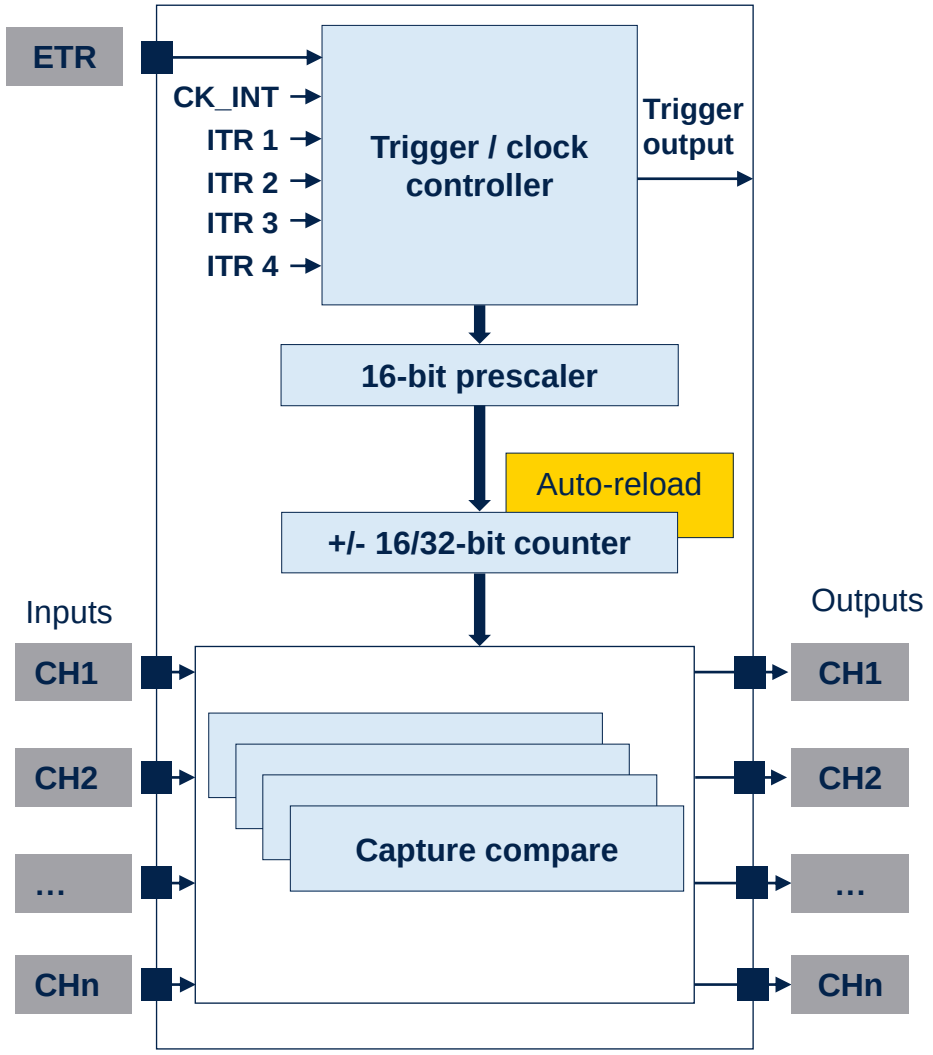
STM32C0 Lab 2: PWM

Lab: PWM timer

Objective:

- Now let’s use a more advanced peripheral like the Timer.
- In this lab we are going to configure a Timer in a PWM (Pulse Width Modulation) mode to blink the LED that we previously controlled with a GPIO.
- PA5 has an alternate Timer channel alternate function which is Timer 1 Channel 1: TIM1_CH1 that we will be using.

Timer - overview



- Multiple timer units providing timing resources
  - Internally (triggers and time-bases)
  - Externally, for outputs or inputs:
    - For waveform generation (PWM)
    - For signal monitoring or measurement (frequency or timing)

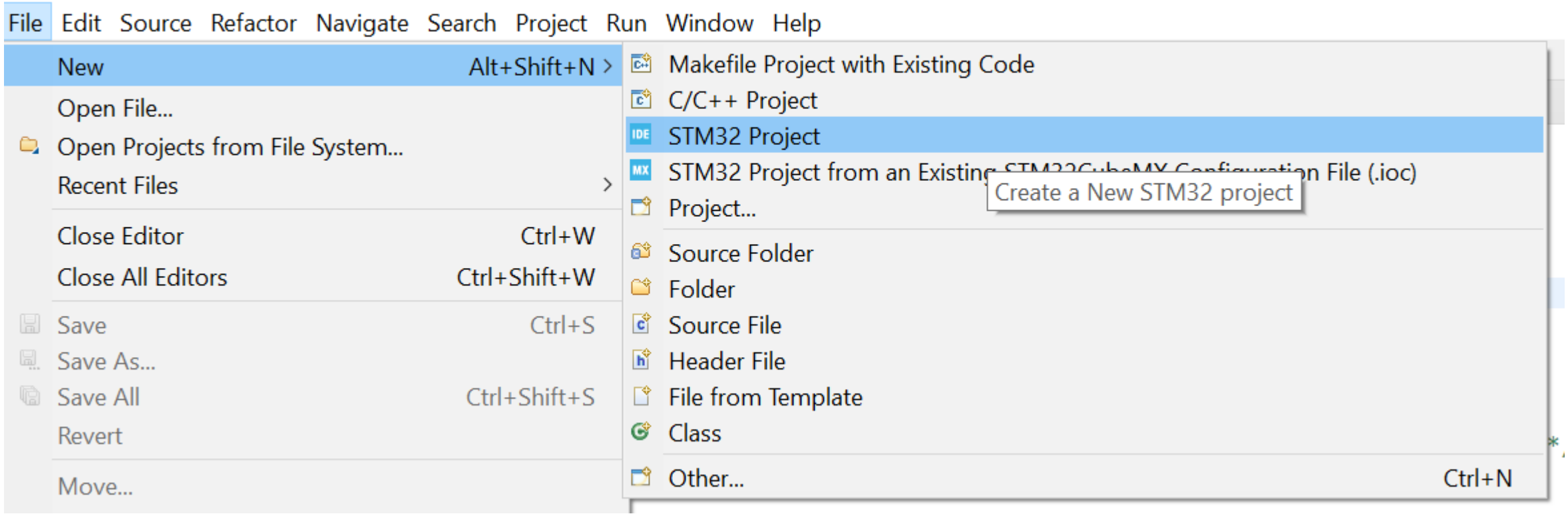
Application benefits

- Versatile operating modes reducing CPU burden and minimizing interfacing circuitry needs
- A single architecture for all timer instances offers scalability and ease-of-use
- Also fully featured for motor control and digital power conversion applications

life.augmented

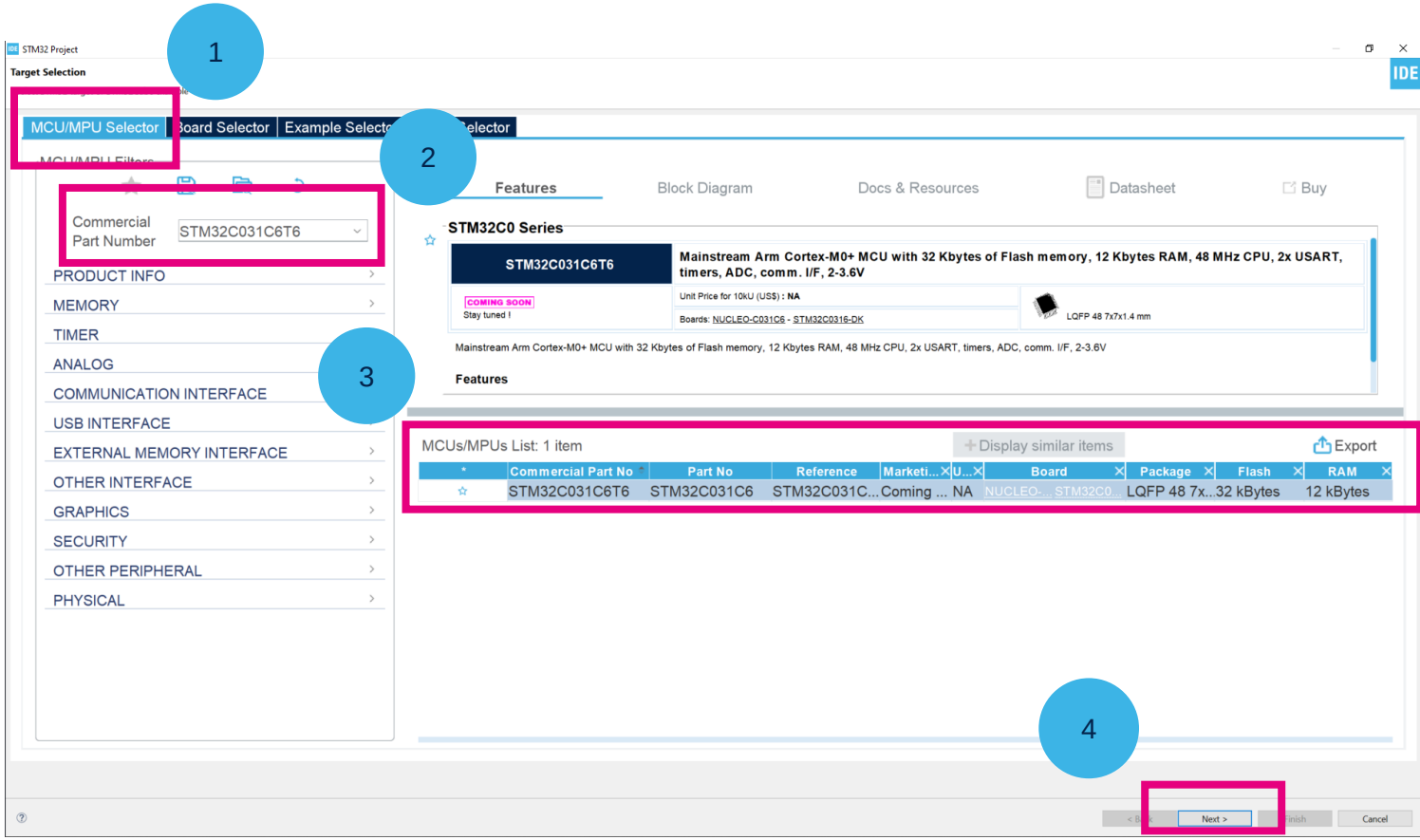
Lab: new project

- In STM32CubeIDE Create a New Project



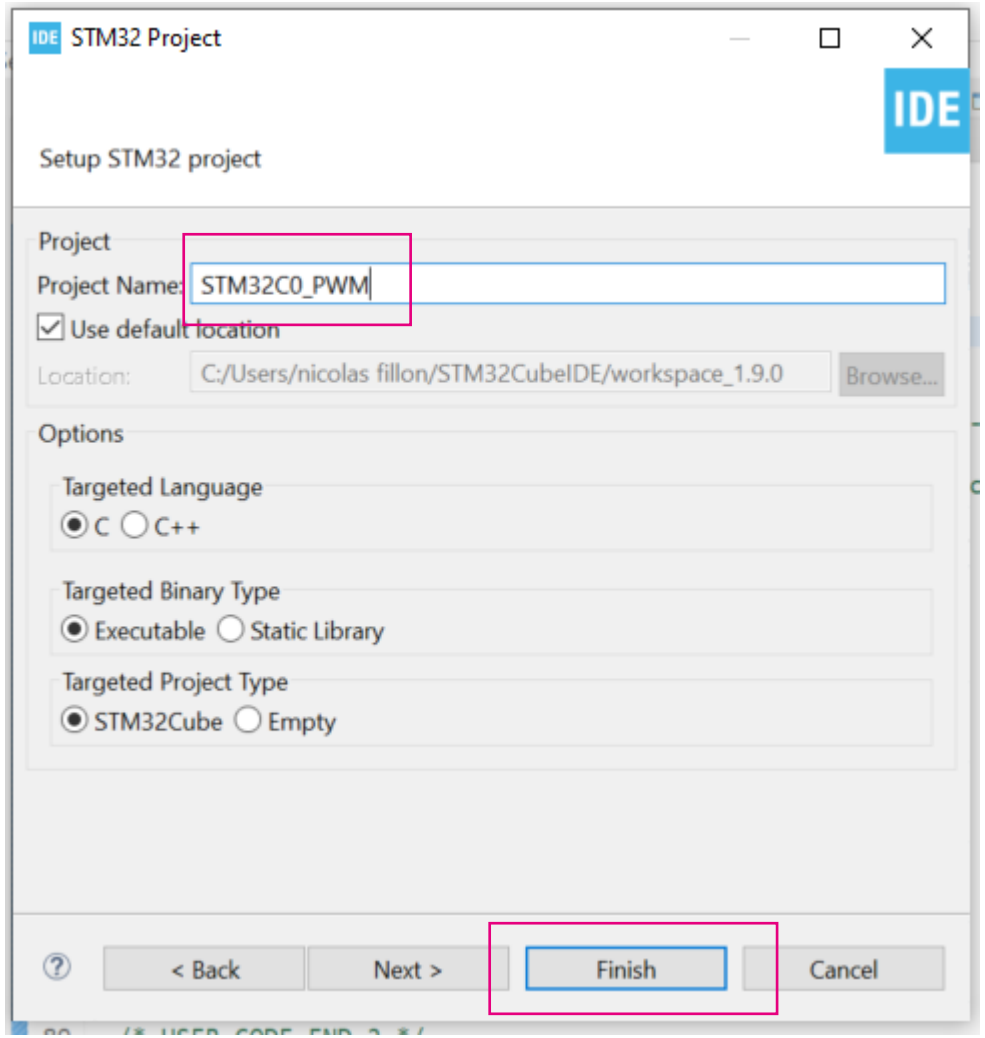
Create new project

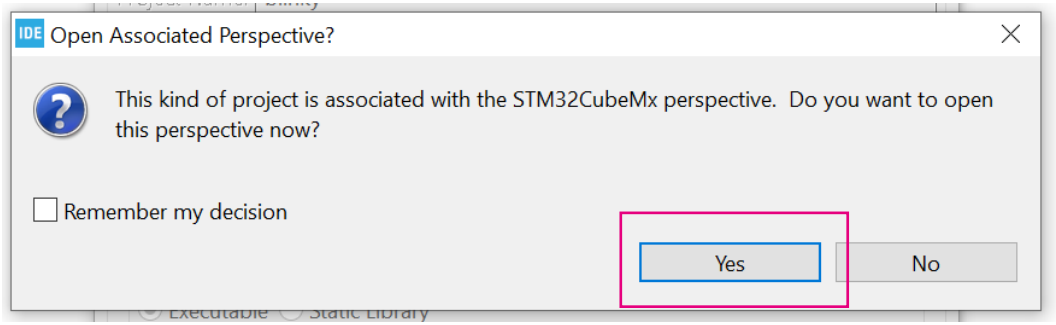
- Click Access To MCU Selector 1
- Type: “STM32C031C6T6” in the Part Number Search 2
- Then Select STM32C031C6T6
  - LQFP48, 32 kB Flash 3
- Click “Next” 4



life.augmented

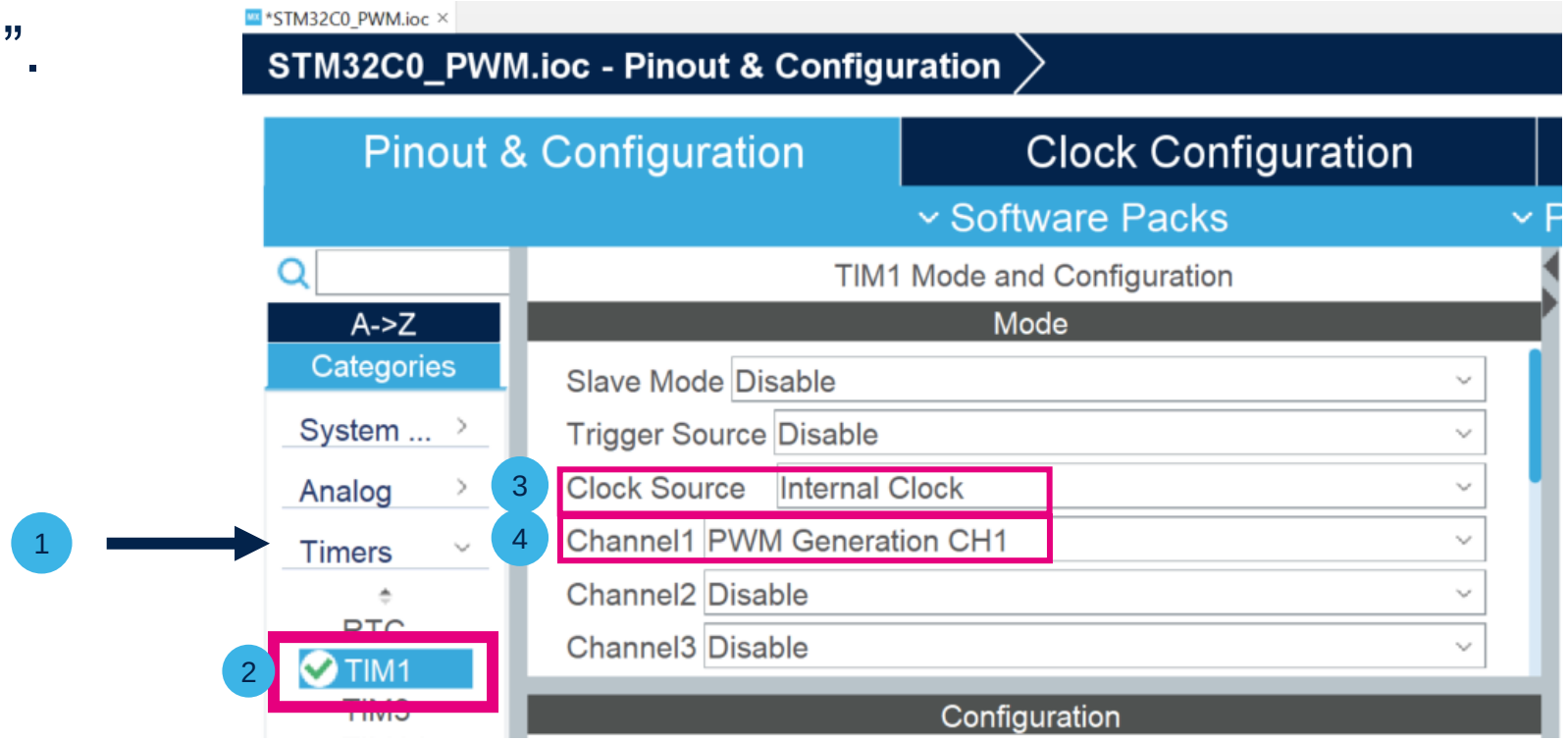
Give a name to the project





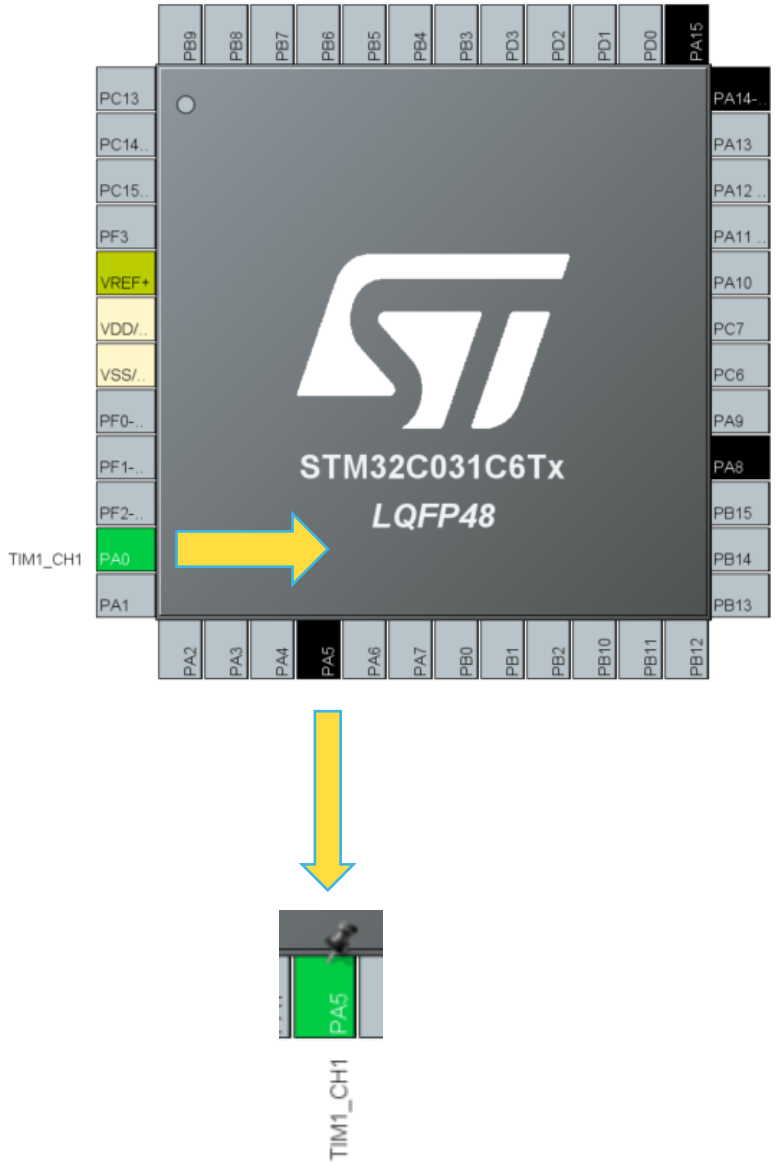
Lab: Timer 1 CH1 configuration

- In this STM32CubeIDE project we are going to add Timer 1 Channel 1 to blink LD4 (PA5) on the Nucleo board.
- In the Pinout & Configuration tab, Expand Timers Categories, then click on TIM1 peripheral and select “Internal Clock” for Source Clock and set Channel1 to “PWM Generation CH1”.



Remapping Timer 1 CH1 output to PA5

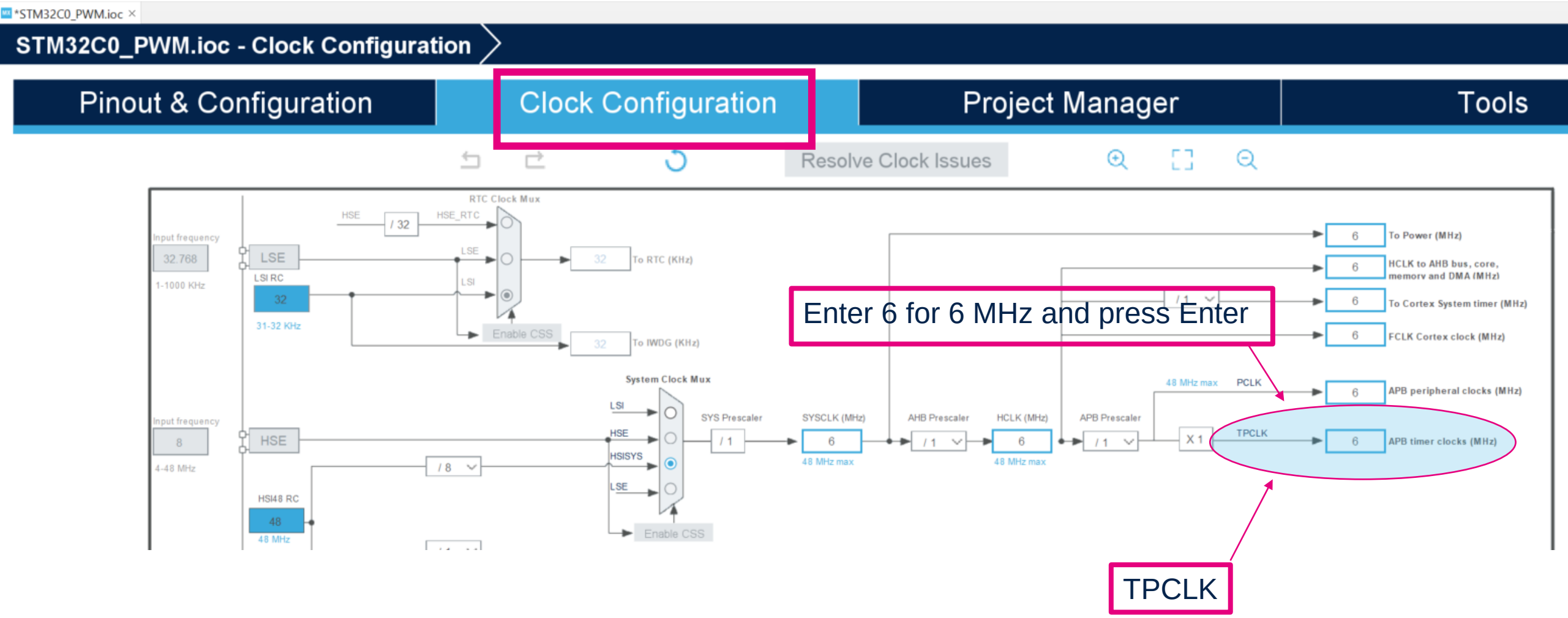
- By default, the tool will configure Timer 1 CH1 to PA0
- We want to remap it to PA5
  - NOTE: PA5 is connected to LD4 (Green LED)
- Hold “Ctrl” button and left mouse click on PA0
- Then drag the mouse pointer to PA5 and then release



Timer parameters calculation

- We want the Timer's PWM output channel to be:
  - T = 1 second period (1 Hz)
  - D = 50% duty cycle (0.5)
- Timer input clock frequency (TPCLK) is set to 6 MHz.
- Prescaler for the Timer is set to 128. The resulting timer counter clock is:
  - CK_CNT = TPCLK / Prescaler = 6 MHz / 128 = 46875 Hz
- To get T=1 Hz (or 1 sec period) the Counter Period needs to be set to: 46875
  - Counter Period = CK_CNT / T = 46875 / 1 = 46875
- To get D=50% duty cycle the Pulse needs to be set to: 23437
  - Pulse = Counter Period / 2 = 46875 / 2 = 23437

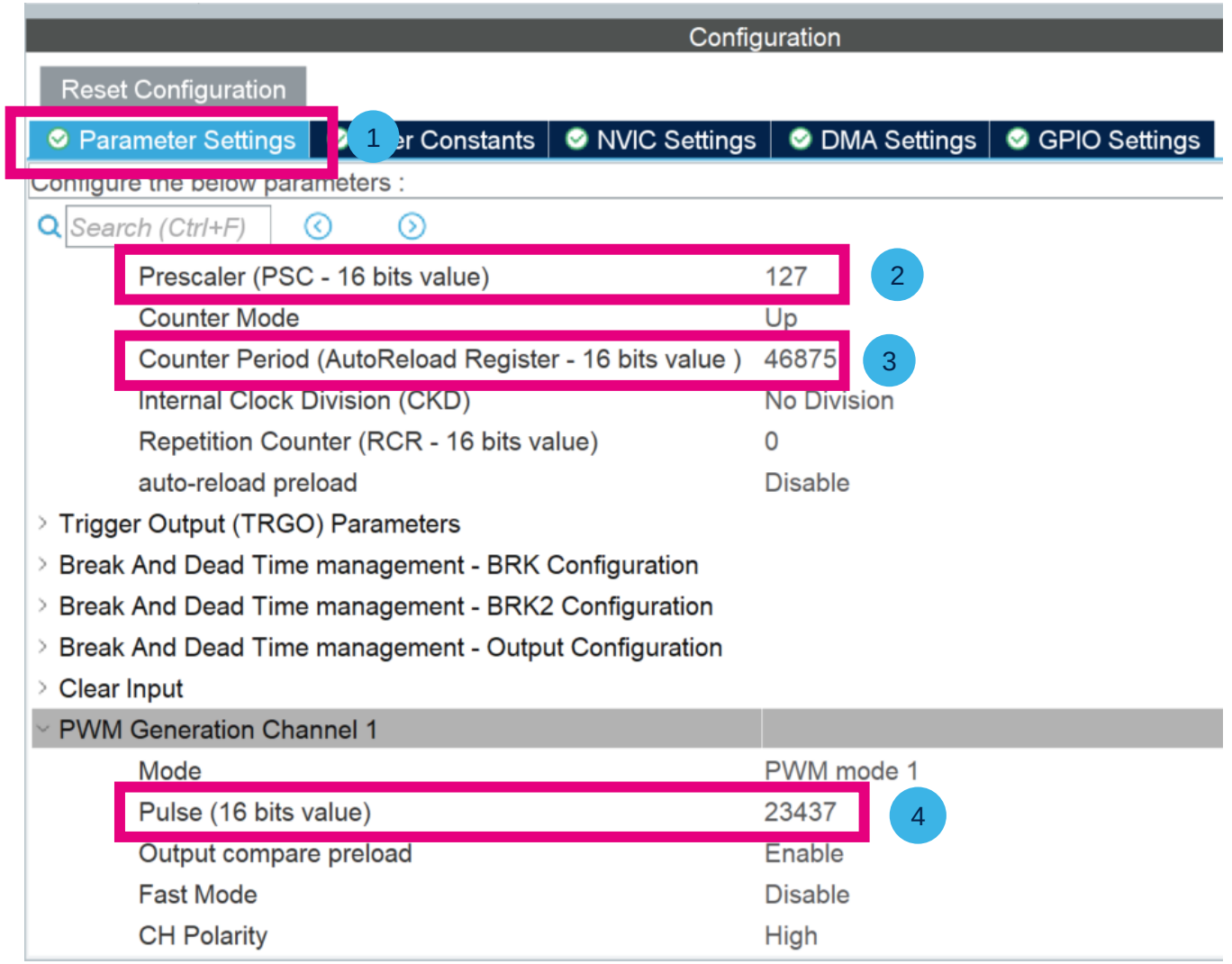
Clock tree configuration



TIM1 configuration – 4 steps

- Select the Pinout & Configuration
- In Parameters Settings of the TIM1 (1)
- Configure 1 Hz timer
  - PSC Prescaler -1 = 127 (2)
  - Counter Period = 46875 (3)
- Set CH1 PWM
  - Pulse = 23437 (4)

Note about Prescaler: prescaler = PSC +1



life.augmented

Generate source code and add code

- Generate Code by saving the project (File -> Save)
- Open the main.c, Add the following code before the while(1) loop in order to start the PWM Timer:

  Note : within “USER CODE BEGIN 2” / “USER CODE END 2” section

```
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
```

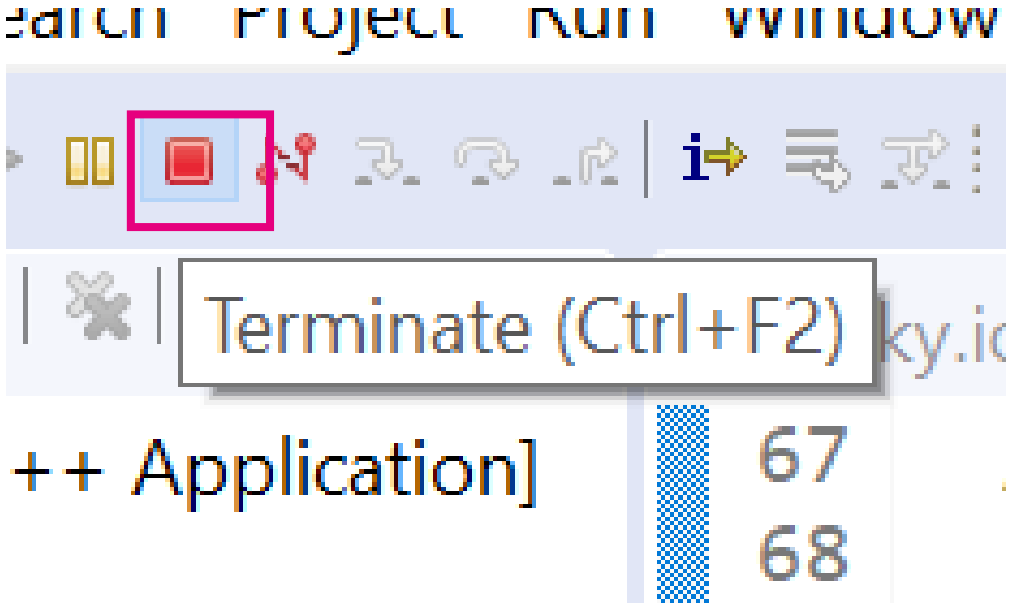
```
/* USER CODE BEGIN 2 */
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
/* USER CODE END 2 */
```

life.augmented

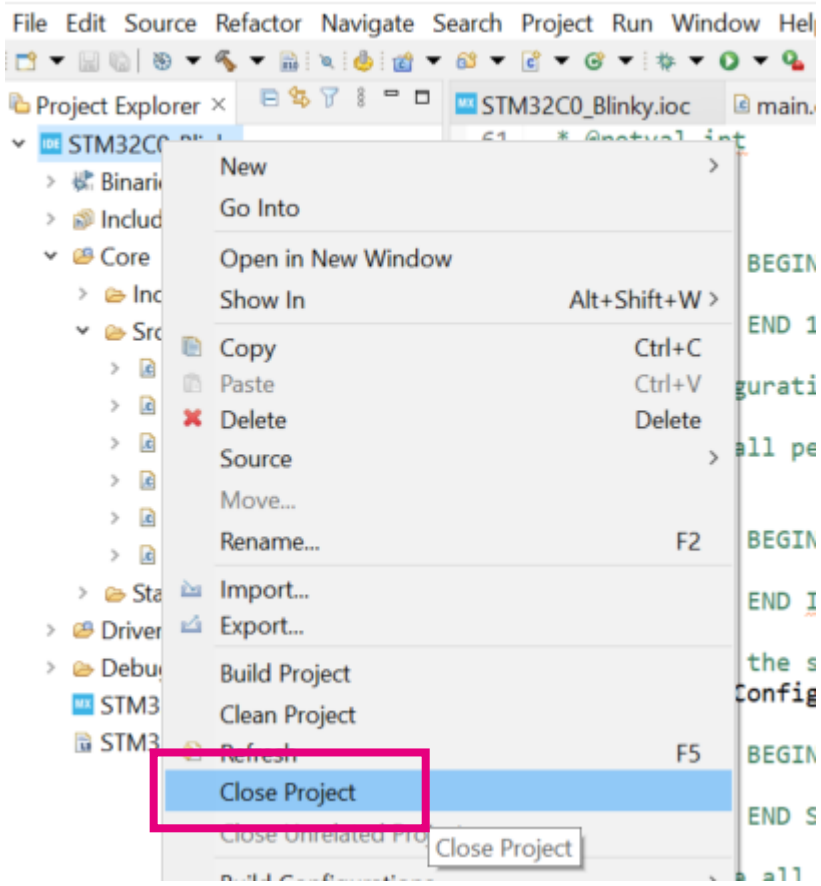
Build and run the code

- Build
- Enter debug session
- Run the code
- Enjoy the flashing LED (LD4)!
  - LD4 is flashing using the PWM Timer

Exit the debugger



Close the project



STM32C0 Lab 3: EXTI

Lab: NVIC + EXTernal Interrupts

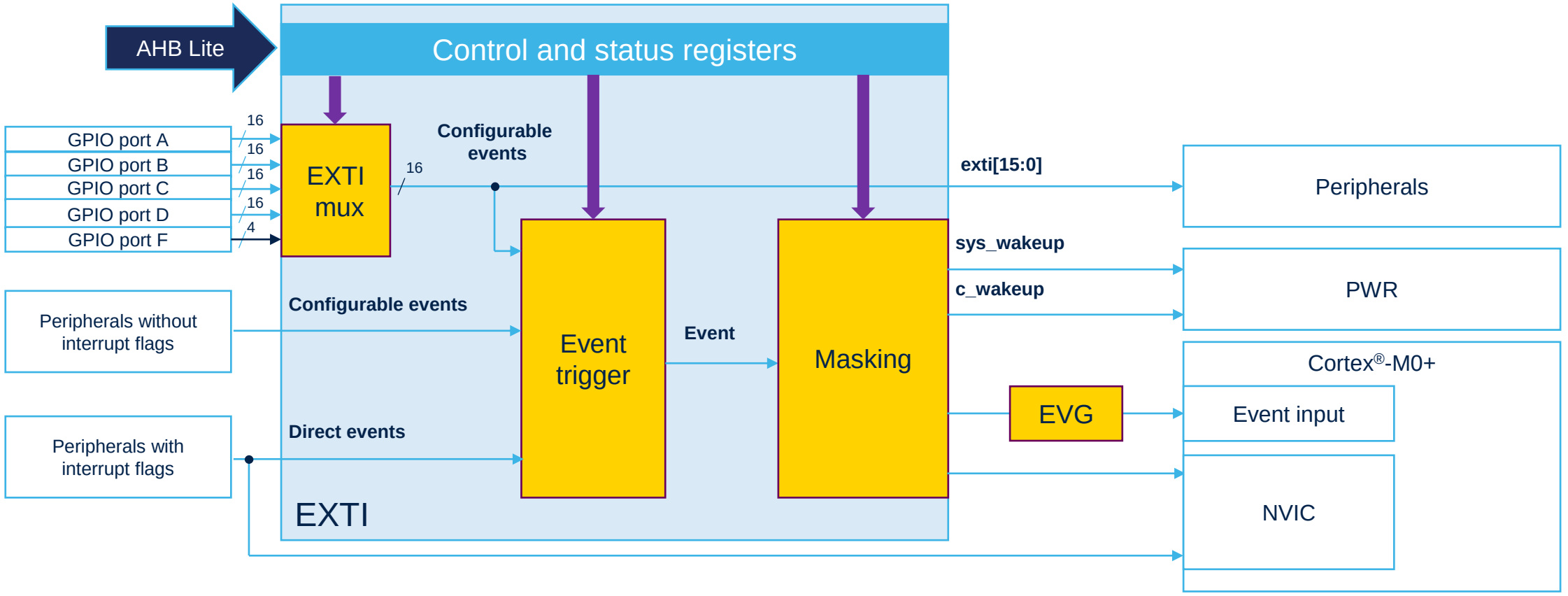
Objective:

- In this project we are going to configure the GPIO that is connected to the user button as External Interrupt (EXTI) with rising edge trigger.
- We will also configure the Interrupt Controller: the NVIC.

EXTI - key features

- Wake-up from Stop mode, interrupts and events generation
  - Independent interrupt and event masks
- Configurable events
  - Active edge selection
  - Dedicated pending flag
  - Trigger-able by software
  - Linked to:
    - GPIO
- Direct events
  - Status flag provided by related peripheral
  - Linked to:
    - RTC, TAMP, I2C1, USARTx, LSE_CSS
- P[x]y (where x = {A,B,C …} like as PA0, PB0, PC0… are linked to the same interrupt event EXTIy
- Groups of EXTI inputs with the same interrupt vector: (EXTI0_1, EXTI2_3, EXTI4_15)

EXTI - block diagram



EVG: EVent Generator

NVIC - overview

- The NVIC (Nested vector Interrupt Controller) is integrated in the Cortex®-M0+ CPU:
  - 32 maskable interrupt channels
  - 4 programmable priority levels
  - Low-latency exception and interrupt handling
  - Power management control

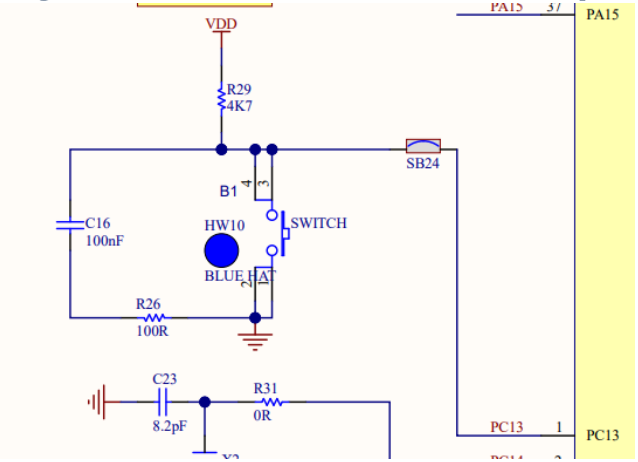
Application benefits

- Supports prioritization levels with dynamic control
- Fast response to interrupt requests
- Relocatable vector table

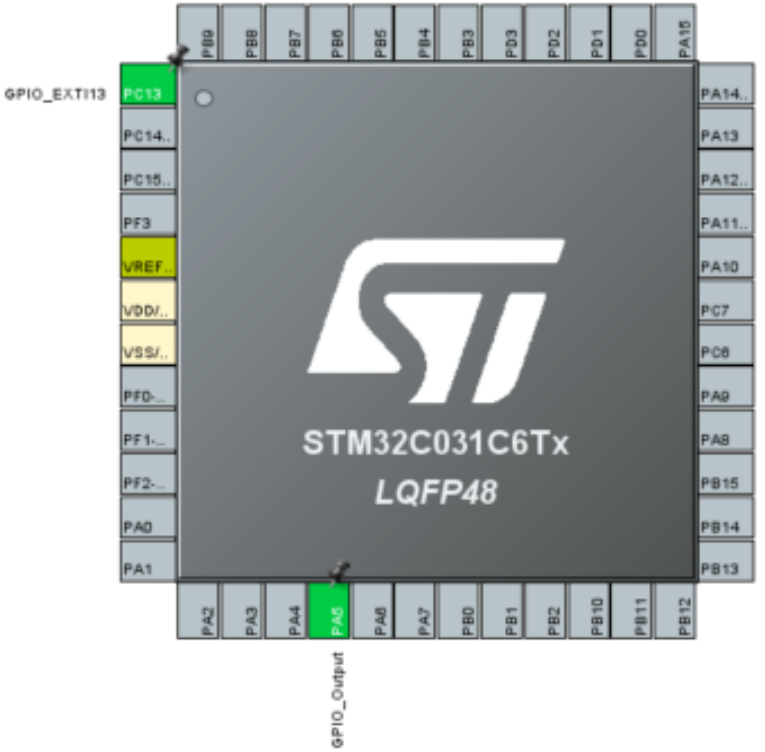
life.augmented

Lab: pinout configuration

- In STM32CubeIDE create a new project “STM32C0_EXTI”.
- Add configuration of the IO that is connected to the User Button (connected to PC13) and to toggle the LED LD4 (connected to PA5) on the STM32C0 Nucleo board.
- Left-click on PC13 and set it to GPIO_EXTI13 mode.
- PA5 to be configured as GPIO Output push-pull.



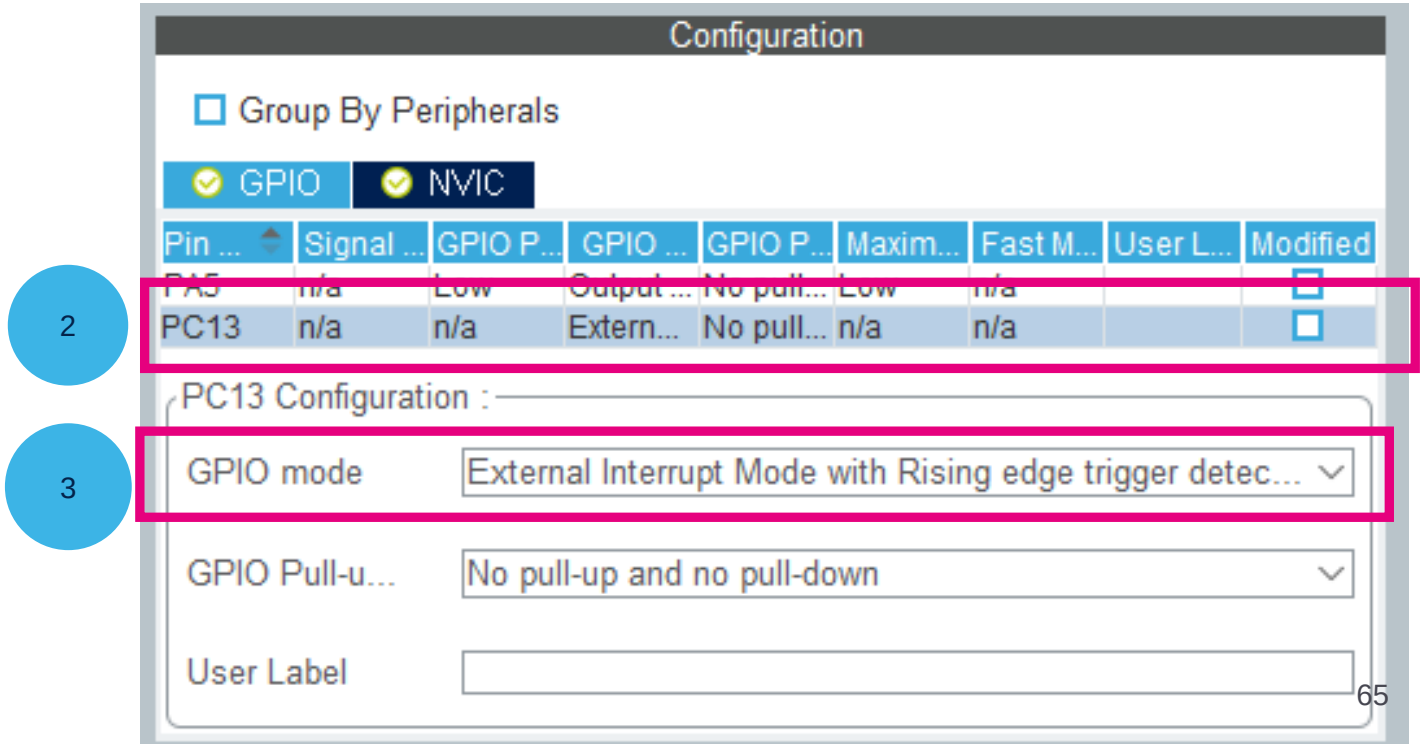
From Nucleo schematics



life.augmented

GPIO configuration

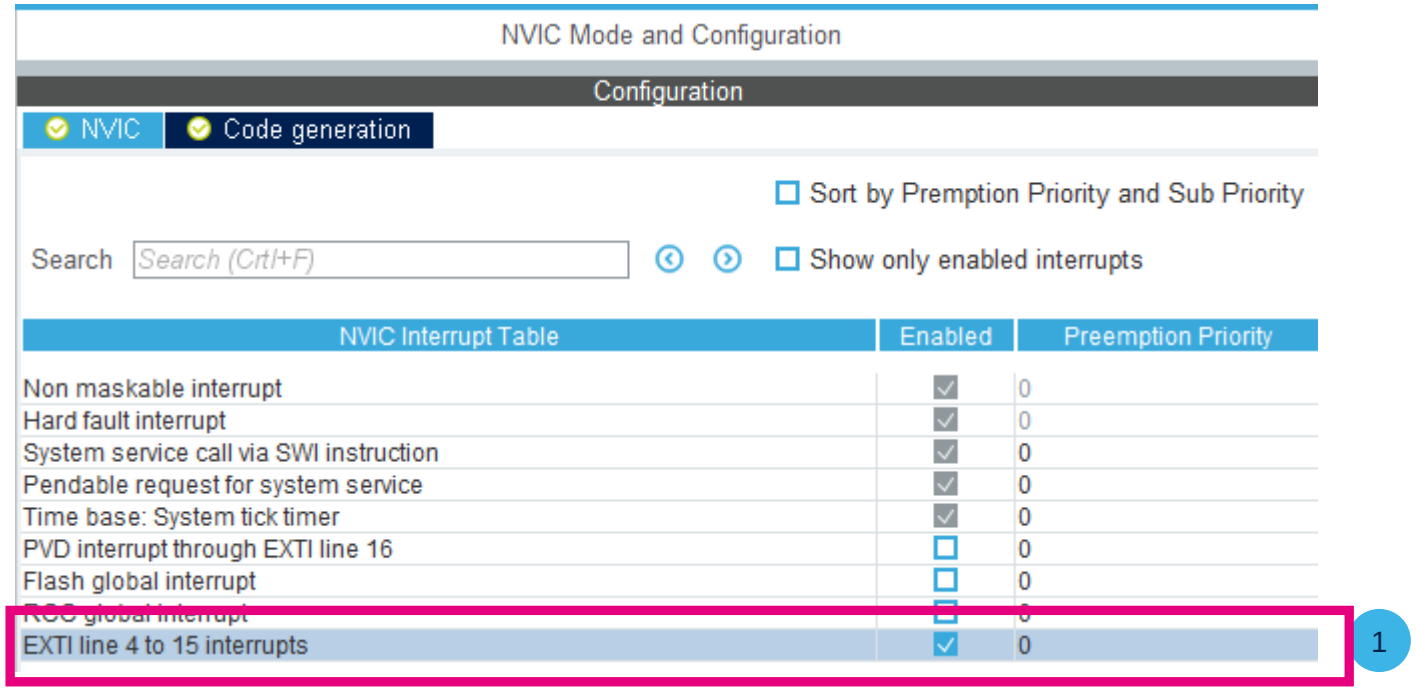
- Select GPIO under System View
- Click on Pin Name PC13
- Make sure GPIO mode is “External **Interrupt Mode** with Rising edge trigger detection”



NVIC configuration

- Select NVIC under System View

- Enable “EXTI line 4 to 15 interrupts” (by checking the box) 1



Generate source code

- Open main.c, add the following code:
  - within “USER CODE BEGIN PV” / “USER CODE END PV” section
    - **Generate Code**

```
uint8_t PC13_flag = 0;
```

```
/* USER CODE BEGIN PV */
    uint8_t PC13_flag = 0;
/* USER CODE END PV */
```

life.augmented

Add EXTI rising edge callback function

- Also, in main.c add the following code,
  - within “USER CODE BEGIN 4” / “USER CODE END 4” section

```
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  PC13_flag++;
  if ( ( PC13_flag & 0x01 ) == 0x01 )
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  }
}
```

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  PC13_flag++;
  if ( ( PC13_flag & 0x01 ) == 0x01 )
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  }
}
/* USER CODE END 4 */
```

life.augmented

Build and run the code

- Click the “Build” button

  - Push the Blue “USER” button to toggle the LED LD4!

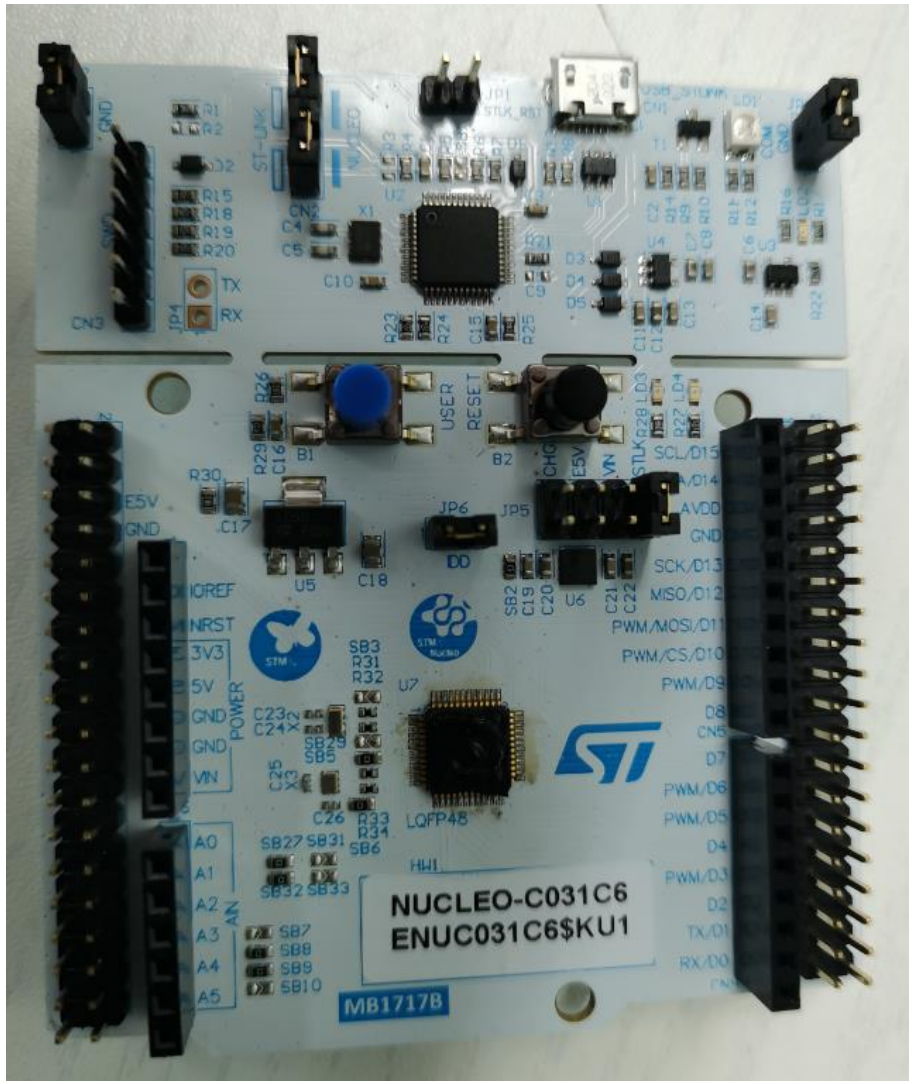
STM32C0 Lab 4: Printf debugging using UART

Lab: printf() debugging using UART

Objective:

- Redirect “printf “ output to USART2 which is connected to the ST-LINK Virtual COM port on the Nucleo board
- Using the Terminal in the STM32CubeIDE to view the printf output.

debugging settings



USART2 debug will be used via the ST-LINK Virtual-COM port

Set up additional GPIO / Clocks:
PA2 – USART2, “USART2-TX”
PA3 – USART2, “USART2-RX”
USART2 Clock = PCLK1 (48 MHz)

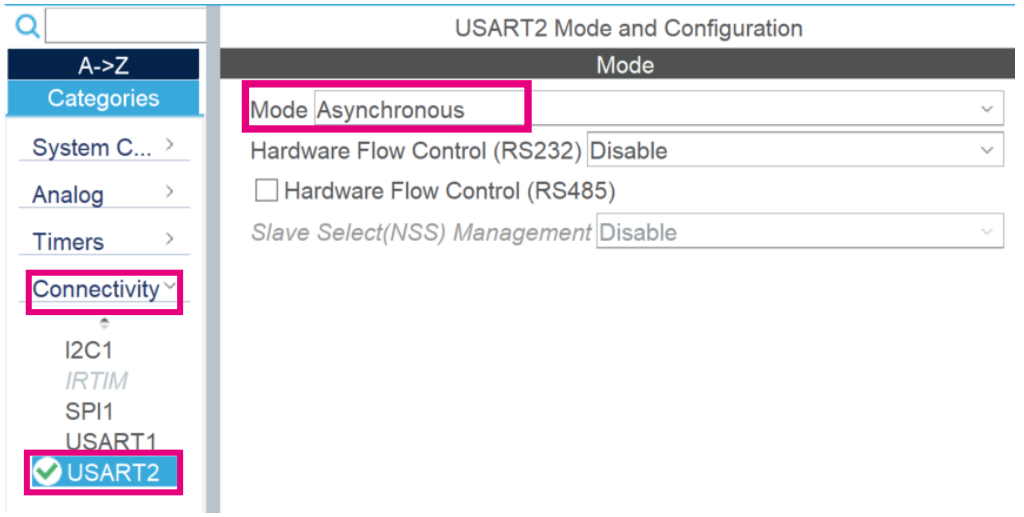
USART2 settings:
Asynchronous Mode
115200
8/N/1
No HW Flow control
Tx/Rx,
No advanced features

We will use the terminal in STM32CubeIDE.

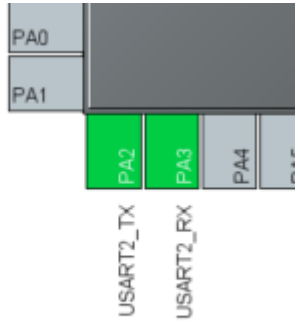
USART2 is routed to the ST-LINK’s USART, and brought via the USB Virtual-COM port class (SB27/32 located on the back on the board have been soldered)

Create a new project and GPIO configuration additions

- Click on USART2 dialog (under Connectivity), and select Asynchronous mode:

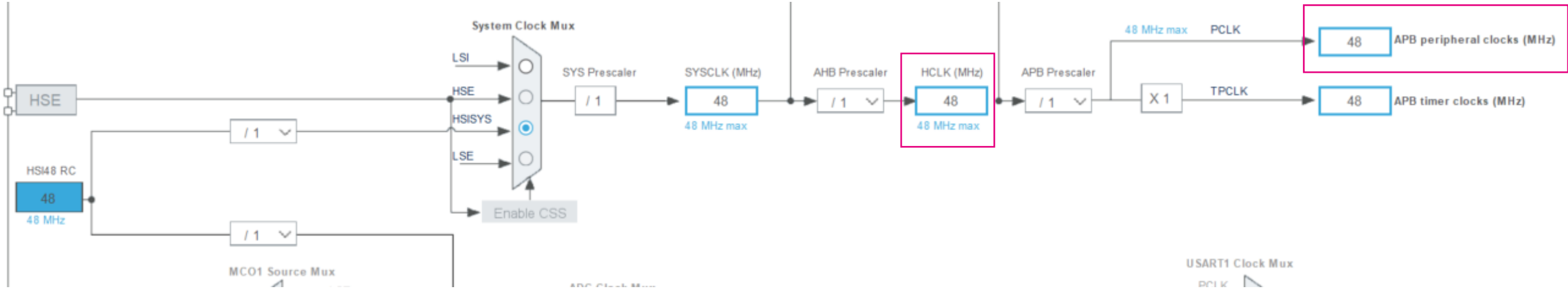


- Make sure PA2 & PA3 are selected for Tx / Rx pins:



Clock configuration

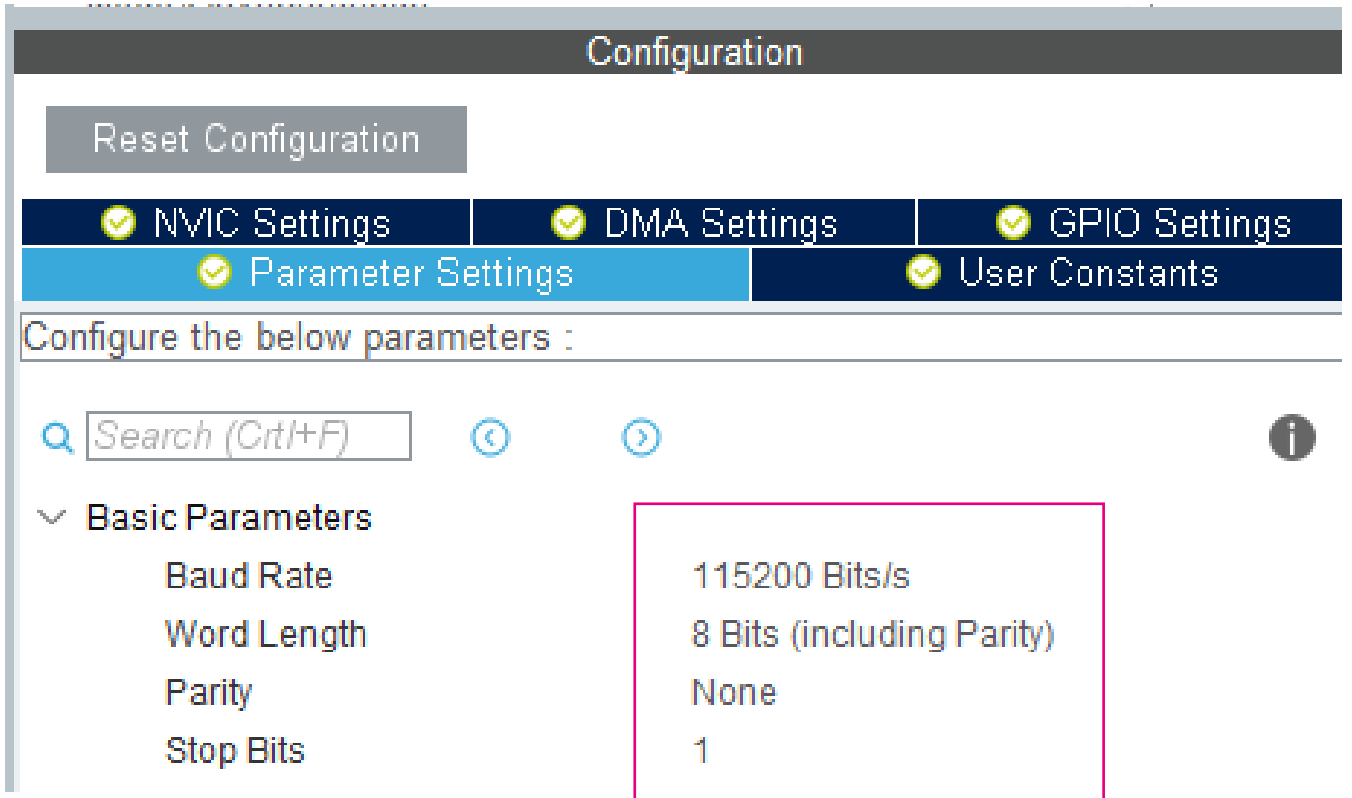
- Run the STM32C0 at 48 MHz for this lab, the USART2 clock also at 48 MHz.



USART2 configuration

- Click on the Configuration tab and select USART2

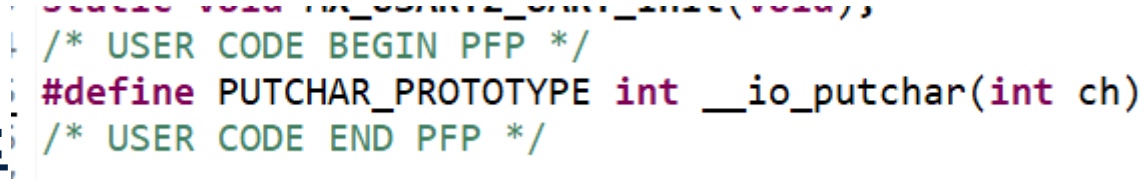
- Parameter Settings tab
  - 115200 Bits/s
  - 8-bit word length
  - No parity bit
  - 1 Stop bit
  - Keep Default settings for the rest



Generate code and adding printf redirecting code in main.C

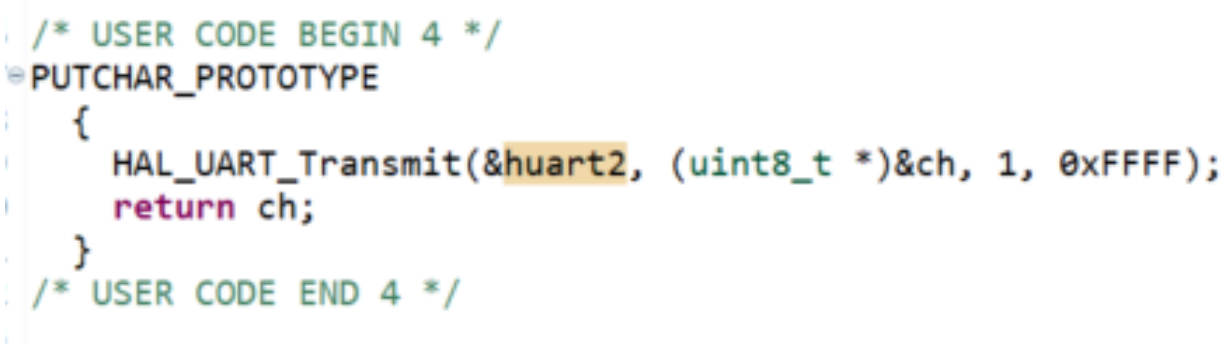
1- Add following code in the section below:

```
  /* USER CODE BEGIN PFP */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
/* USER CODE END PFP */
```



2- Add following function in the section below:

```
  /* USER CODE BEGIN 4 */
  PUTCHAR_PROTOTYPE
  {
    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
    return ch;
  }
  /* USER CODE END 4 */
```

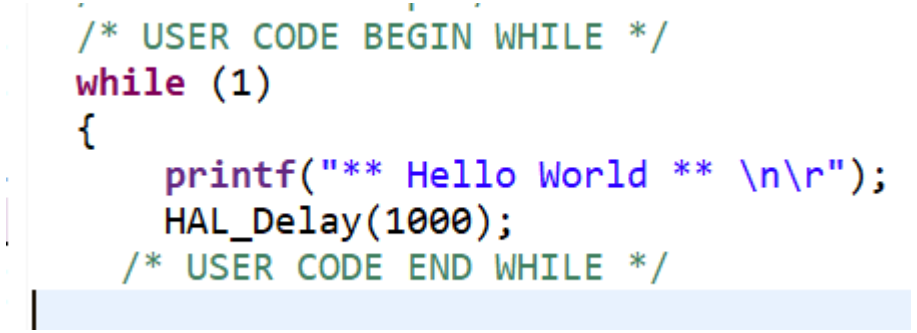


life.augmented

Adding application code in main.C

Add application code in main loop:

```
/* USER CODE BEGIN WHILE */
 while (1)
 {
     printf("** Hello World ** \n\r");
     HAL_Delay(1000);
 /* USER CODE END WHILE */
```



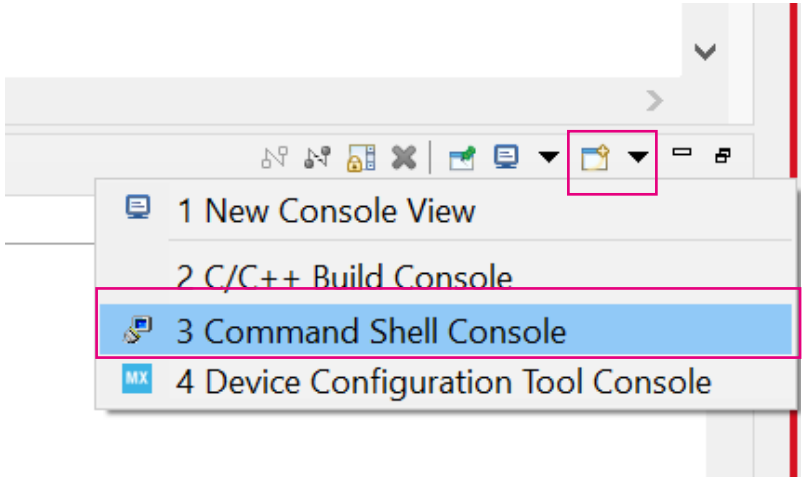
life.augmented

Build the project and run the application

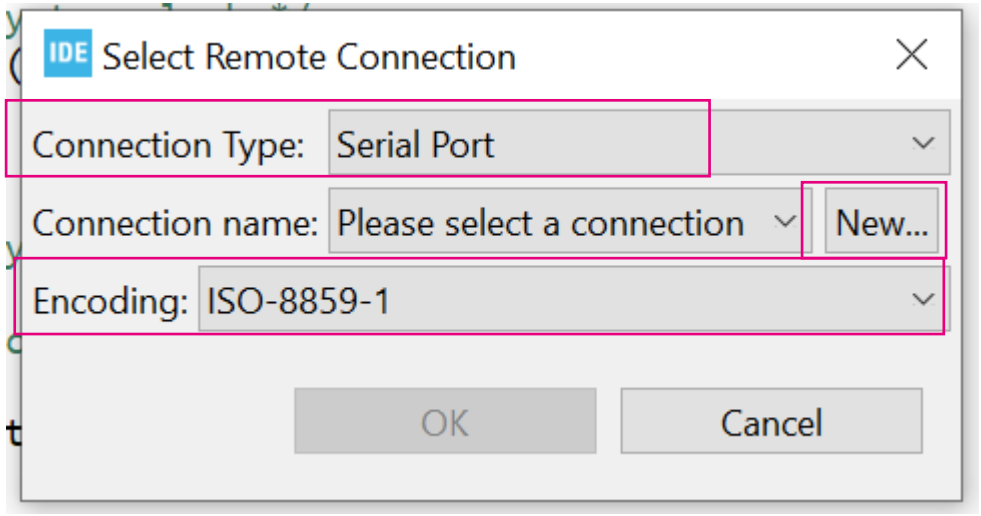
- Build Project
- Enter Debug session
- Click “Resume” button to run the code

Configure the Terminal inside STM32CubeIDE

- Open a Command Shell Console



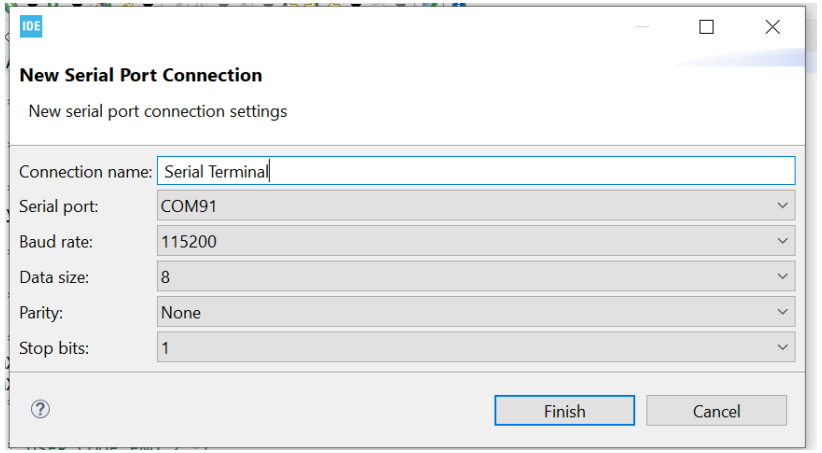
- In Select Remote Connection, select Serial Port for Connection Type, ISO for Encoding and then Press the “New…” icon below:



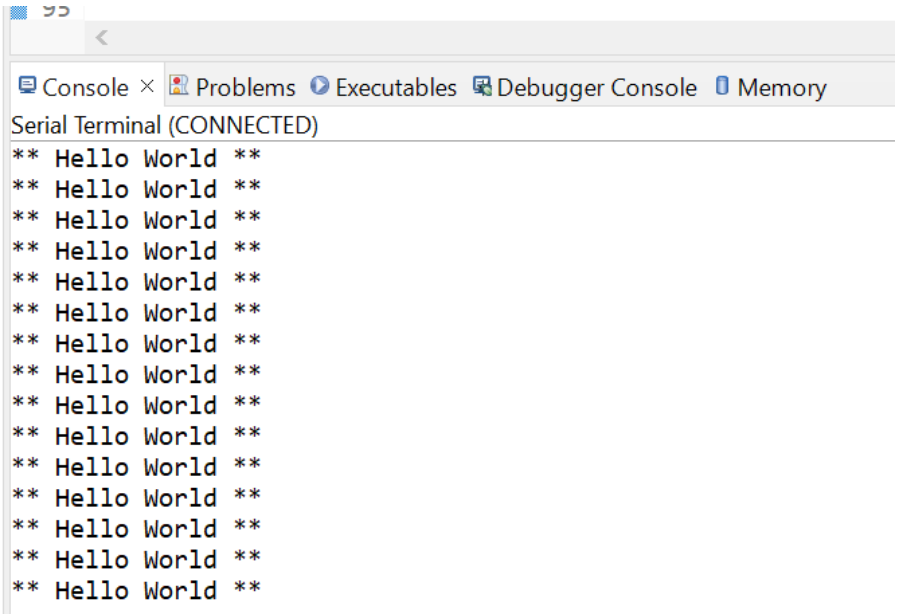
life.augmented

Configure the Terminal inside STM32CubeIDE

- In the New serial Port connection, give a name to the connection, select the COM port of your ST-LINK (COM91 for me) and use the following settings, then clock Finish:



- You should see the printf message being displayed in the Console Tab.



life.augmented

STM32C0 : LL Drivers

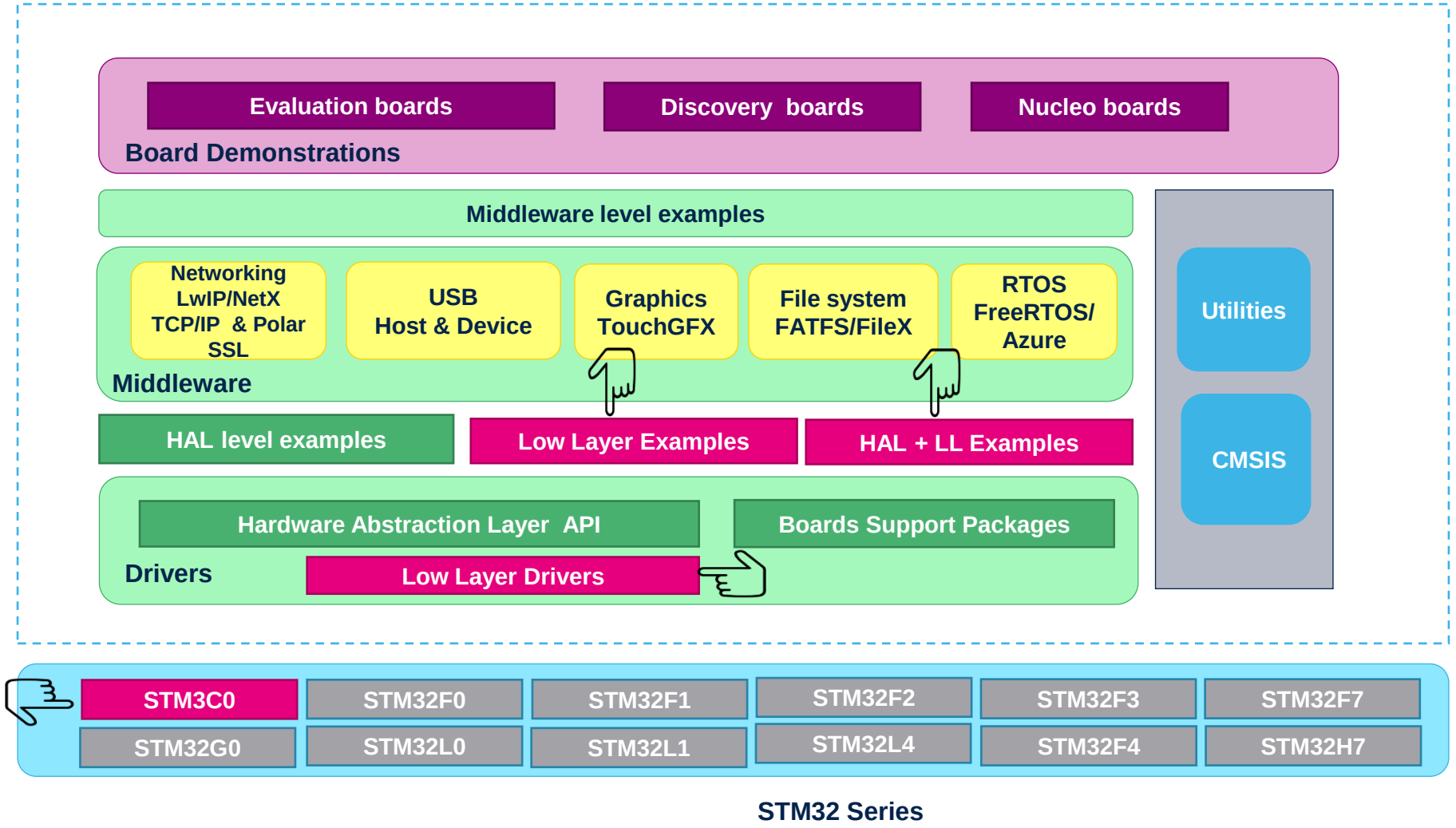
Overview

- STM32Cube HAL & LL are complementary and covers a wide range of applications requirements
- HAL offers high level and functionalities-oriented APIs, with high portability level and hide product/IPs complexity to end user
- LL offers low level APIs at registers level, w/ better optimization in code size and code execution but LL are less portability and require deep knowledge of the product specification

Low Layer (LL) library features

- The Low Layer (LL) Library offers the following services:
  - Set of static inline functions for direct register access (provided in *.h files only)
  - One-shot operations that can be used by the HAL drivers or from application level.
  - Independent from HAL and can be used standalone (without HAL drivers)
  - Full feature coverage of the supported peripherals

STM32Cube FW package block view



Benchmark- USART transmit example

- The below data are based on the “USART Transmitter IT” example:
  - Configure GPIO & USART peripheral for sending characters to HyperTerminal (PC) in Asynchronous mode using IT
  - Using below configuration:
    - Platform: STM32L486xx
    - Compiler : IAR
    - Optimization : High Size
    - Heap Size = 512 Bytes
    - Stack Size = 512 Bytes

	HAL Drivers	Low layer Drivers
read-only code memory (Bytes)	7206	2154
read-only data memory (Bytes)	204	94
read write data memory (Bytes) (*)	1408	1093

ROM Size divided by ~4

RAM Size reduction

- **LL** offer smaller footprint & high performance but less portability & require expertise
- **HAL** offer high level API (hide complexity) & portability but higher footprint & less performance

(*) to add Heap and Stack size for total RAM

life.augmented

STM32C0 Lab 5: Using the low layer (LL) drivers

Lab: Using the low layer drivers

Objective:

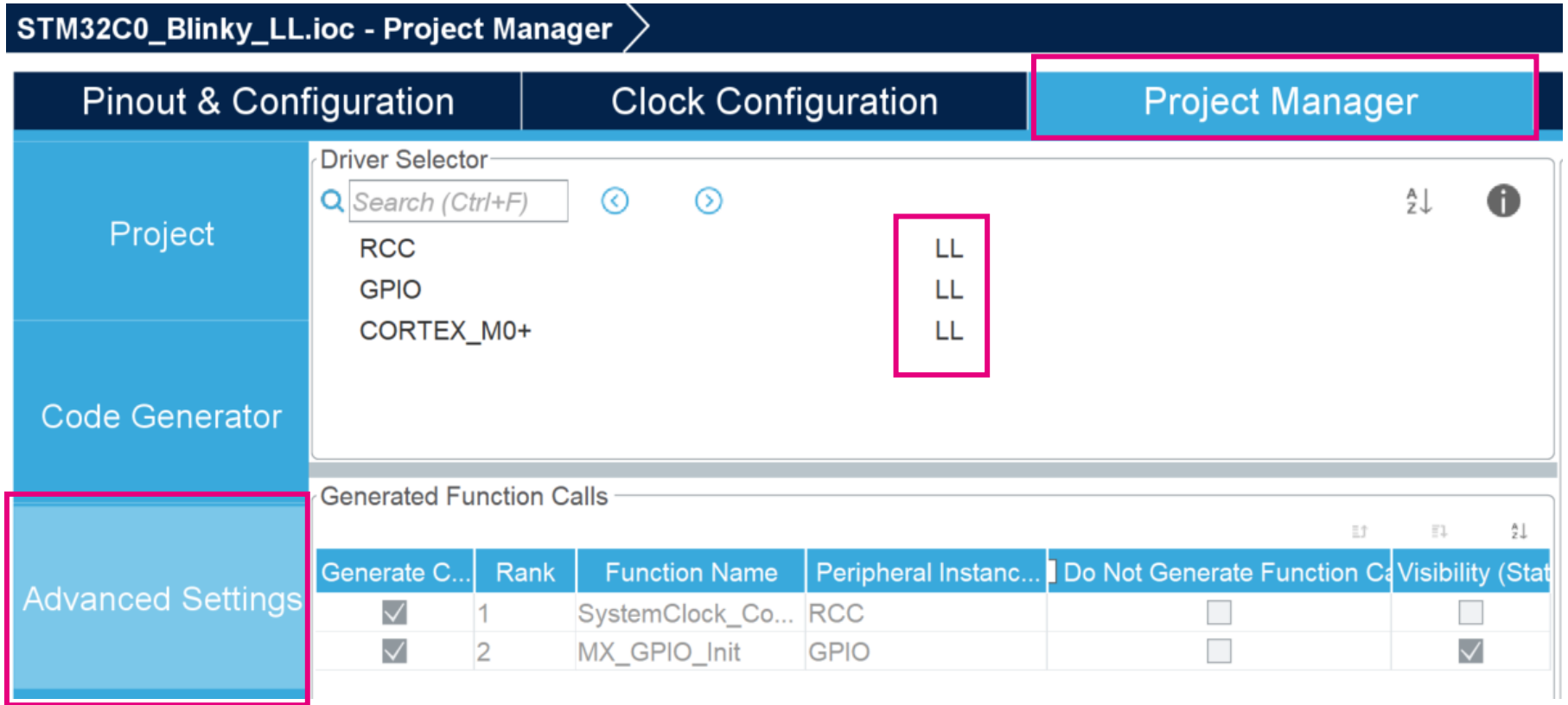
- Generate a project with STM32CubeIDE using the Low Layer Drivers and check how much improvement we get compared to a HAL project in term of Flash and RAM usage

Project

- In STM32CubeIDE create a project you can call “STM32C0_LL”
- Set GPIO PA5 as Output Push-Pull like we did for the “STM32C0_Blinky” lab

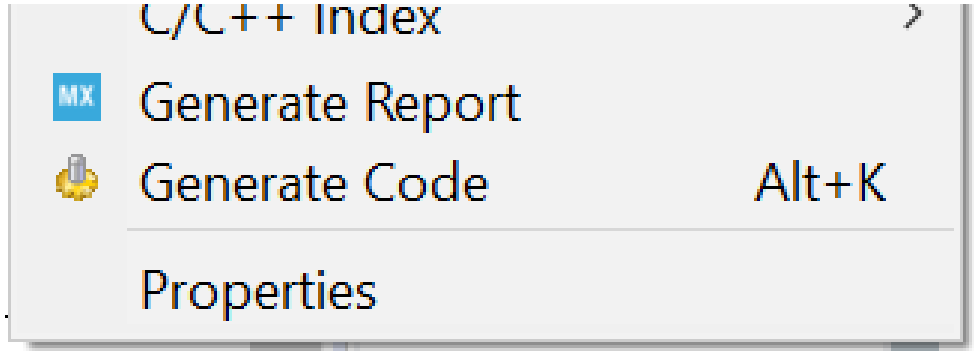
Configuration

- In Project Manager Tab
- In the Advanced Settings Tab
- Instead of HAL drivers select LL (Low Level) for RCC, GPIO and CORTEX_M0+



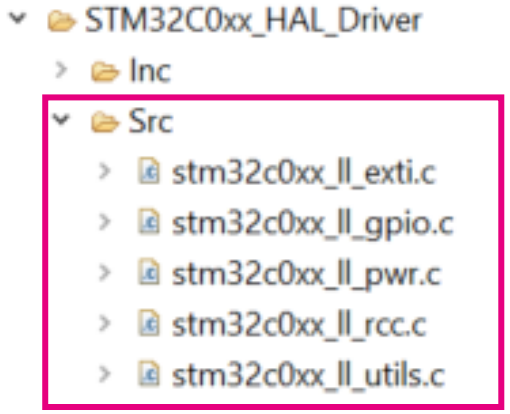
Generate source code

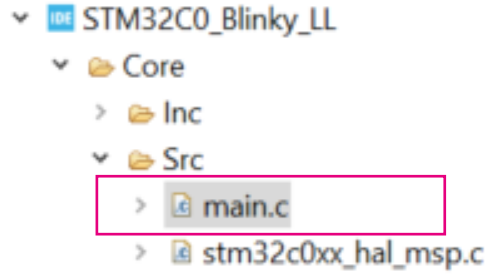
- **Generate Code (ALT + K)**



Add code to toggle the LED

- Expand the “Drivers” and notice that now the drivers are low layer (_ll):





- Expand the file tree and open the main.c file
- Add the following code inside the while(1) loop
  - Add within “USER CODE BEGIN WHILE” / “USER CODE END WHILE” section (this will preserve your code after code regeneration)

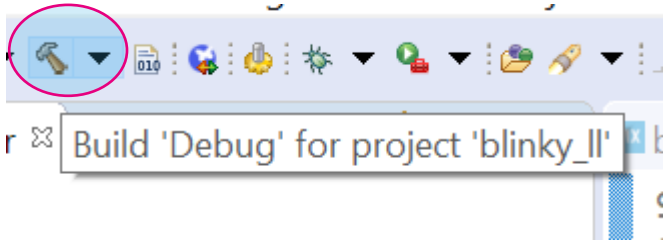
```
LL_GPIO_TogglePin(GPIOA, LL_GPIO_PIN_5);
// Delay 100 ms
LL_mDelay(100);
```

```
/* USER CODE BEGIN WHILE */
while (1)
{
    LL_GPIO_TogglePin(GPIOA, LL_GPIO_PIN_5);
    // Delay 100 ms
    LL_mDelay(100);
  /* USER CODE END WHILE */
```

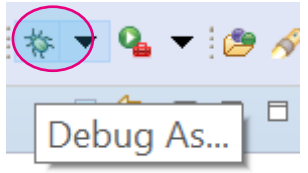
life.augmented

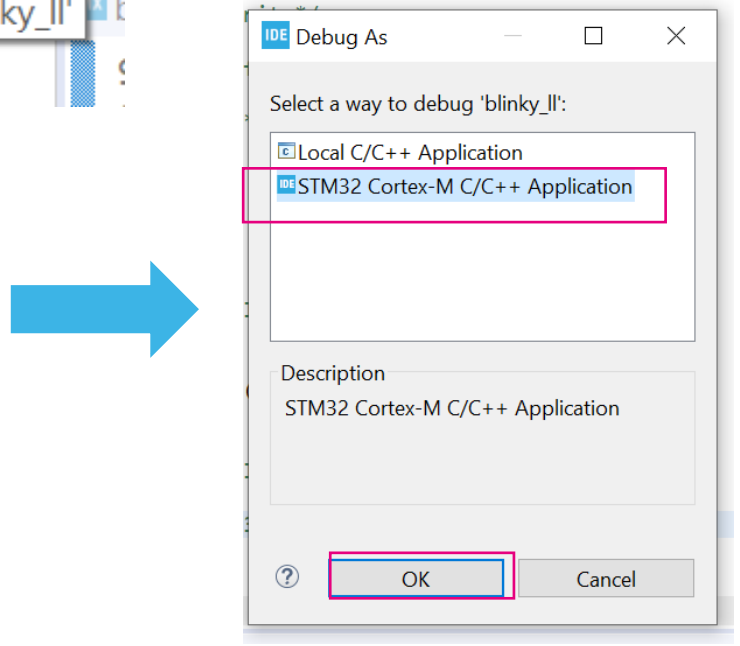
Build the project

- Click the “Build” button



  - Click the “Start/Stop Debug Session” button





  - Click “Run” button

  - The Green LED (LD4) should be blinking just like the blinky example with the HAL drivers

Let’s compare the differences between HAL and LL projects

- In the console window of the “STM32C0_Blinky” project that is using HAL drivers, after building we see:

```
CDT Build Console [STM32C0_Blinky]
20:36:24 **** Incremental Build of configuration Debug for project STM32C0_Blinky ****
make -j8 all
arm-none-eabi-size   STM32C0_Blinky.elf
   text    data     bss     dec     hex filename
   4440      20    1572    6032    1790 STM32C0_Blinky.elf
Finished building: default.size.stdout


20:36:25 Build Finished. 0 errors, 0 warnings. (took 487ms)
```

Let’s compare the map files between HAL and LL projects

- In the console window of the “STM32C0_Blinky_LL” project that is using LL drivers, after building we see:

```
CDT Build Console [STM32C0_Blinky_LL]
20:37:30 **** Incremental Build of configuration Debug for project STM32C0_Blinky_LL ****
make -j8 all
arm-none-eabi-size   STM32C0_Blinky_LL.elf
   text    data     bss     dec     hex filename
   3608      12    1564    5184    1440 STM32C0_Blinky_LL.elf
Finished building: default.size.stdout


20:37:30 Build Finished. 0 errors, 0 warnings. (took 424ms)
```

Comparison table between HAL and LL for our “blinky” code

- Using below configuration:
  - Heap Size = 512 Bytes / Stack Size = 512 Bytes
  - STM32Cube C0 1.0.0
  - CubeIDE 1.10.1

	HAL Drivers	Low layer Drivers
read-only code memory (Bytes)	4440	3608
read write data memory (Bytes)	20	12

ROM Size with 20% improvment

RAM Size reduction divided ~ by 2

life.augmented

STM32C0 Lab 6: ADC + DMA + TIM

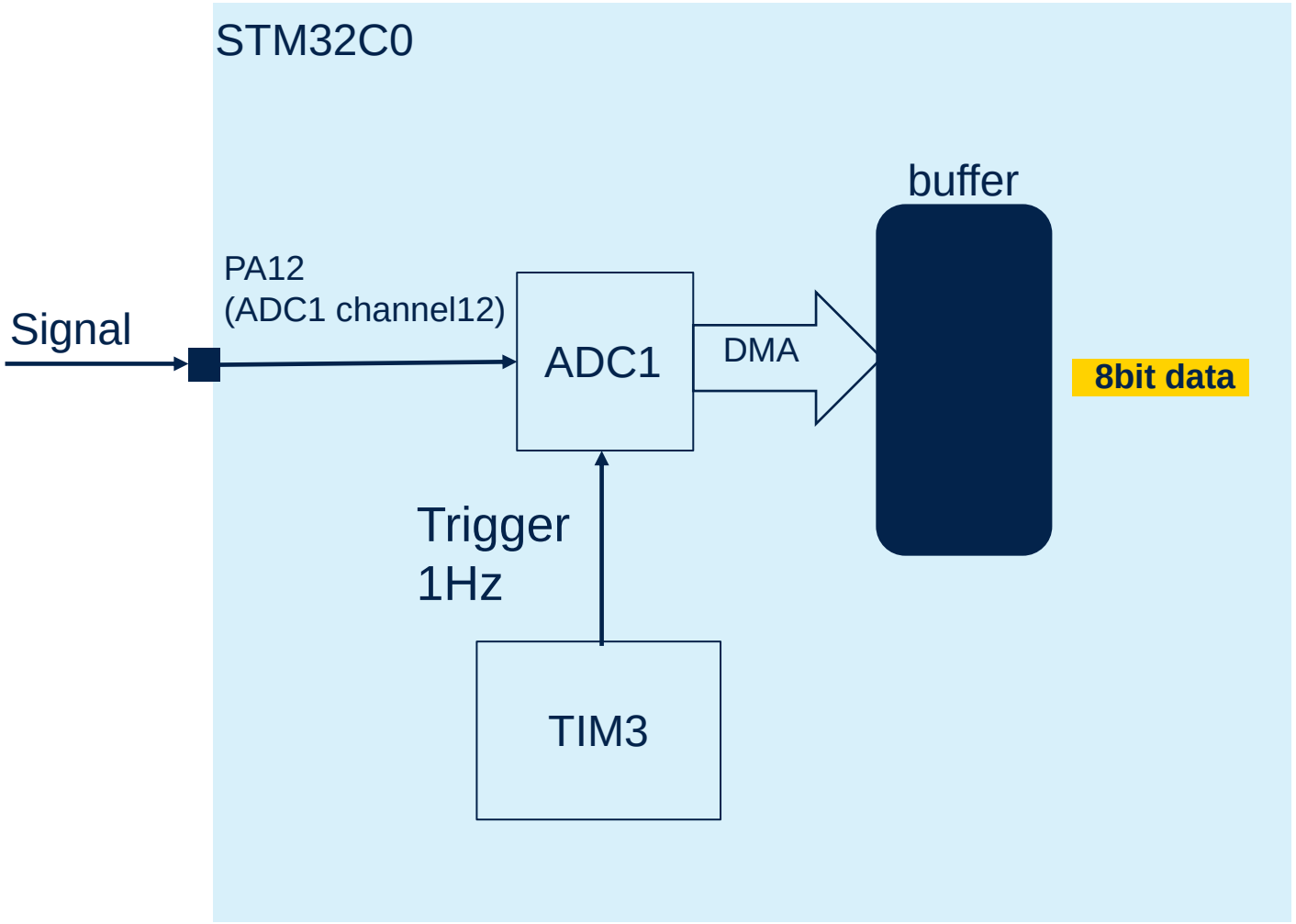
Lab: ADC+DMA+TIM

Objective:

- The objective of this lab is to learn about the ADC (Analog to Digital Converter) with the DMA (Direct Memory Access) and TIM (Timer) of the STM32C0.

- We will convert a signal on PA12 which is ADC1 Channel 12 and store the converted value in a buffer in RAM thanks to DMA and Timer.

Block Diagram



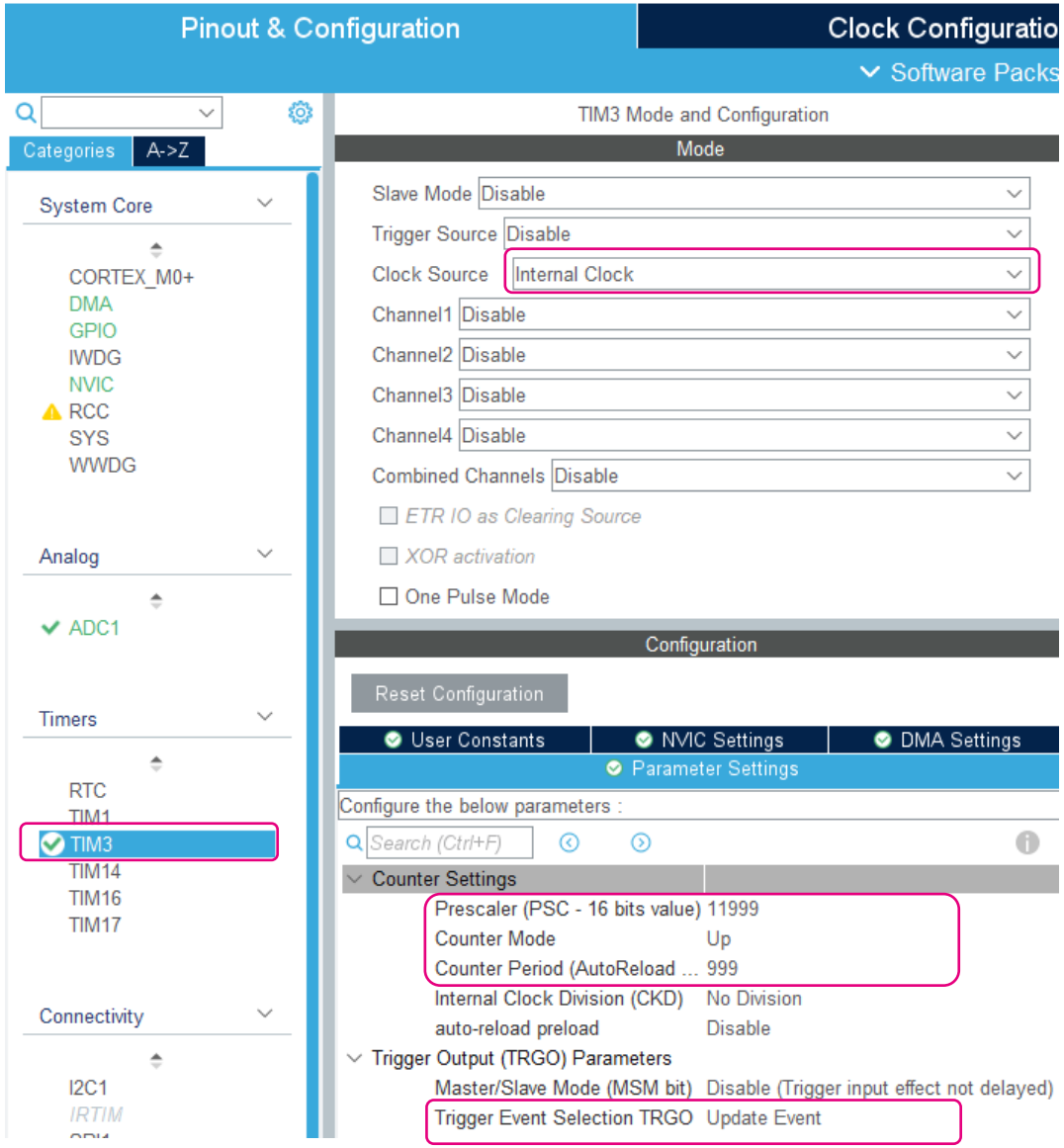
Example description

- ADC1 is measuring 8bit data on PA12 (ADC1_IN12 channel) each 1 second triggered by TIM3 (working as timebase) on its update event
- Data from ADC1 are transferred by DMA (DMA1_Channel2) to the buffer
- DMA1_Channel2 is configured in Circular mode using bytes (both sides)

TIM3 configuration

Goal: Configure TIM3 as upcounter using an internal clock which would overflow each 1 second with TRGO signal genaration on this event.

- Select TIM3 within Timers group
- In Mode part:
  - Clock Source = Internal Clock
- In Configuration part:
  - Prescaler = 11999
  - Counter Period = 999
  - Trigger Event Selection TRGO = Update Event

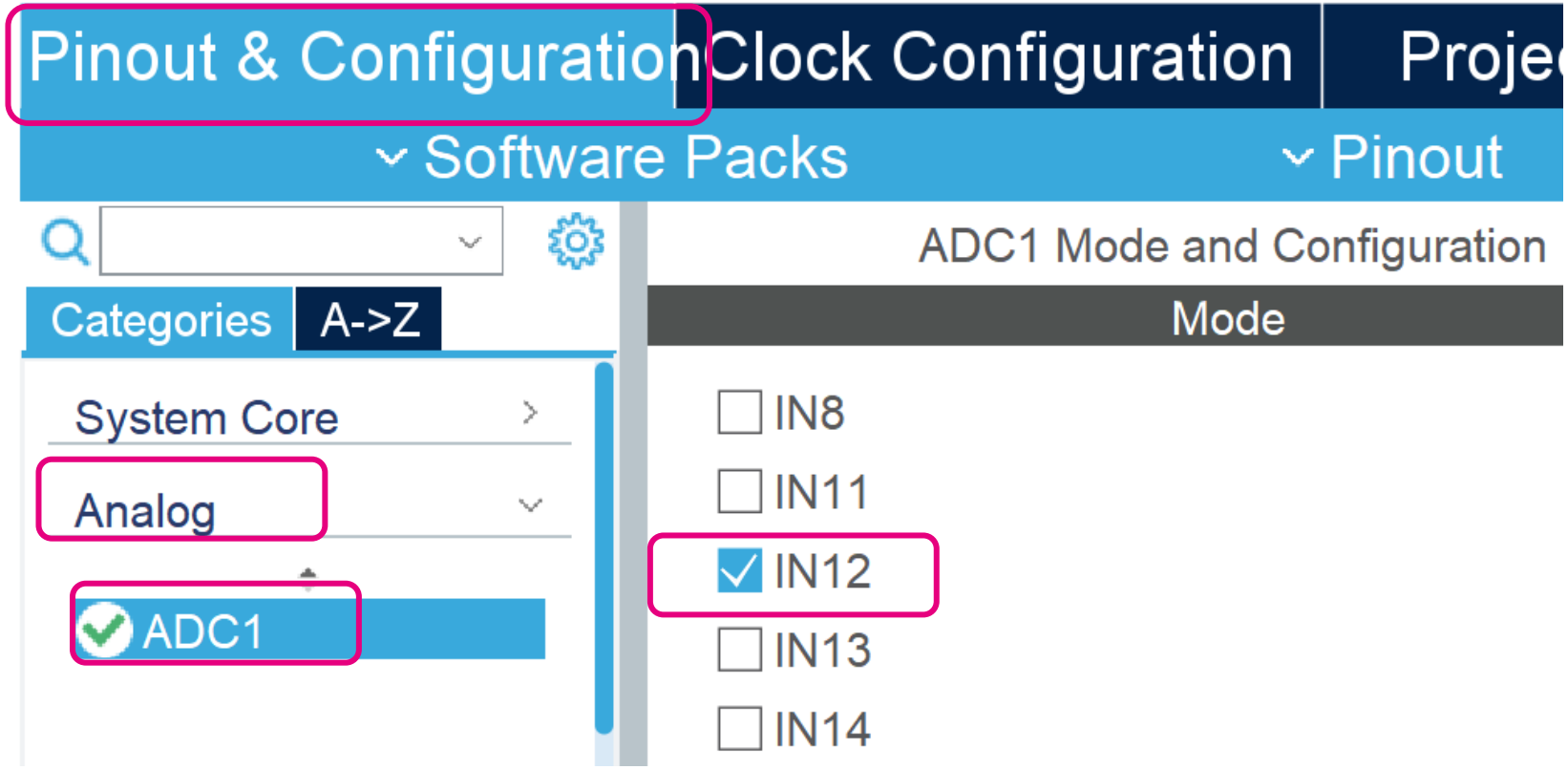


life.augmented

ADC1 configuration

Goal: Configure ADC1 to convert signal on ADC1_IN12 line (PA12) on each trigger coming from TIM3 TRGO line, using 12.5 cycles sample time, 8bit resolution and transferring data over DMA.

- Select ADC1 within Analog group
- Activate IN12: ADC1 Channel 12.

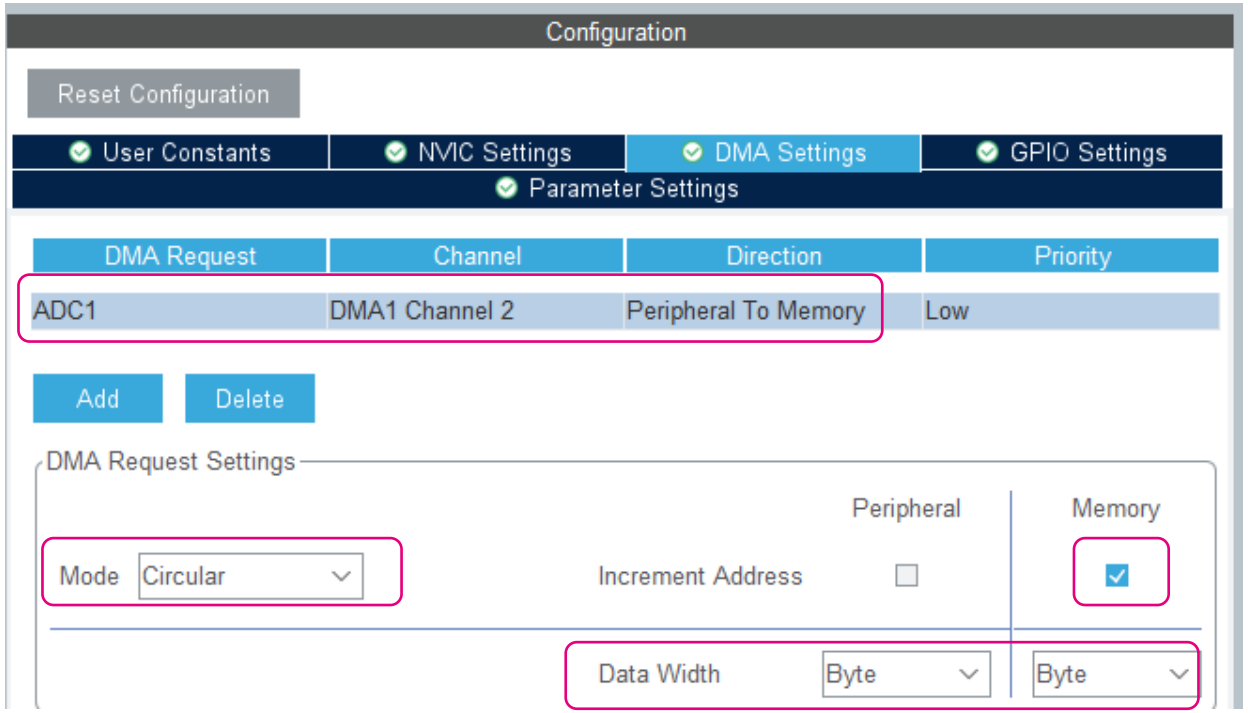


life.augmented

ADC1 – DMA configuration

Goal: Add DMA support on ADC1 transfer complete events. Transfer using bytes using circular buffer

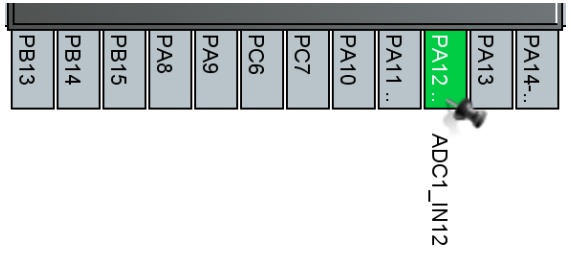
- Select DMA Settings tab within ADC Configuration
- Click on Add button and select ADC1 as DMA request
- Set the following DMA Settings:
  - Mode = Circular
  - Increment address = on Memory side
  - Data width = Byte on both sides

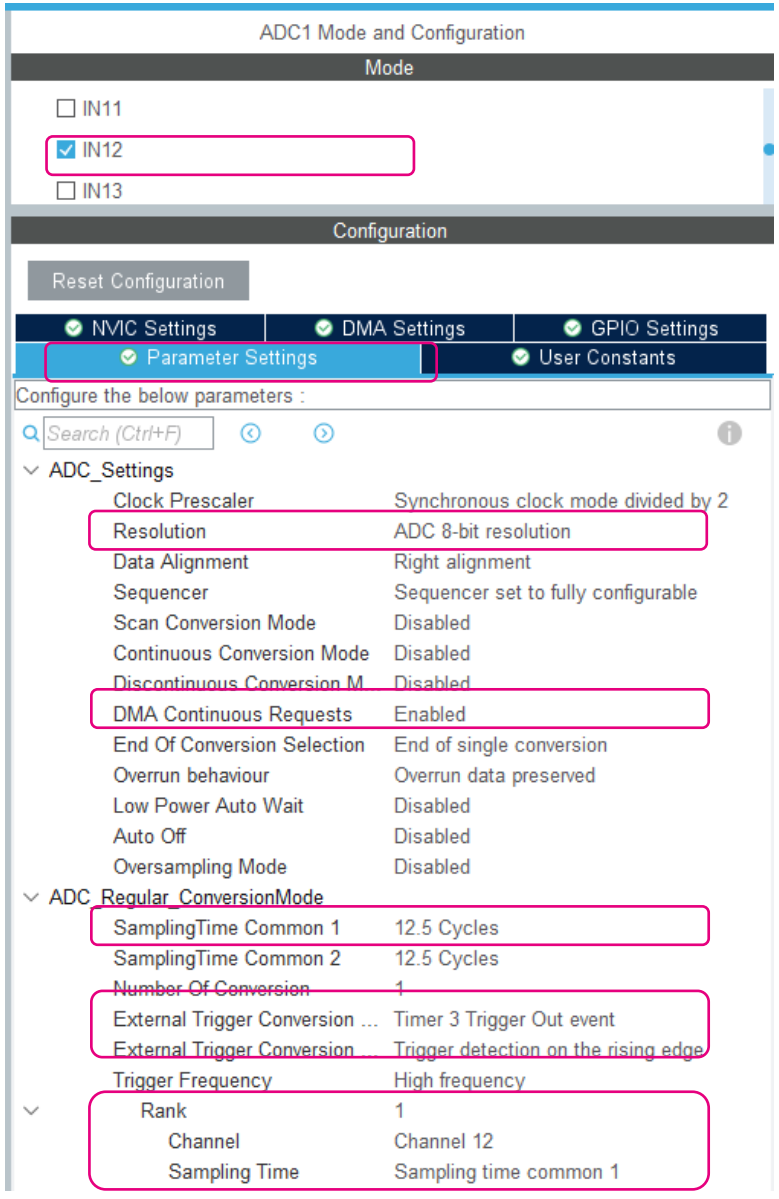


Goal: Configure ADC1 to convert signal on ADC1_IN12 line (PA12) on each trigger coming from TIM3 TRGO line, using 12.5 cycles sample time, 8bit resolution and transferring data over DMA.

ADC1 configuration

- In Configuration part select Parameter Settings tab and change:
  - Resolution = 8 bits
  - DMA Continuous Requests = Enabled
  - SamplingTime Common1= 12.5 Cycles
  - Ext Trigger Conv Source = Timer 3 Trigger Out event (TRGO)
  - Ext Trigger Conversion Edge = rising edge
  - Rank1, Sampling time = Sampling time common 1

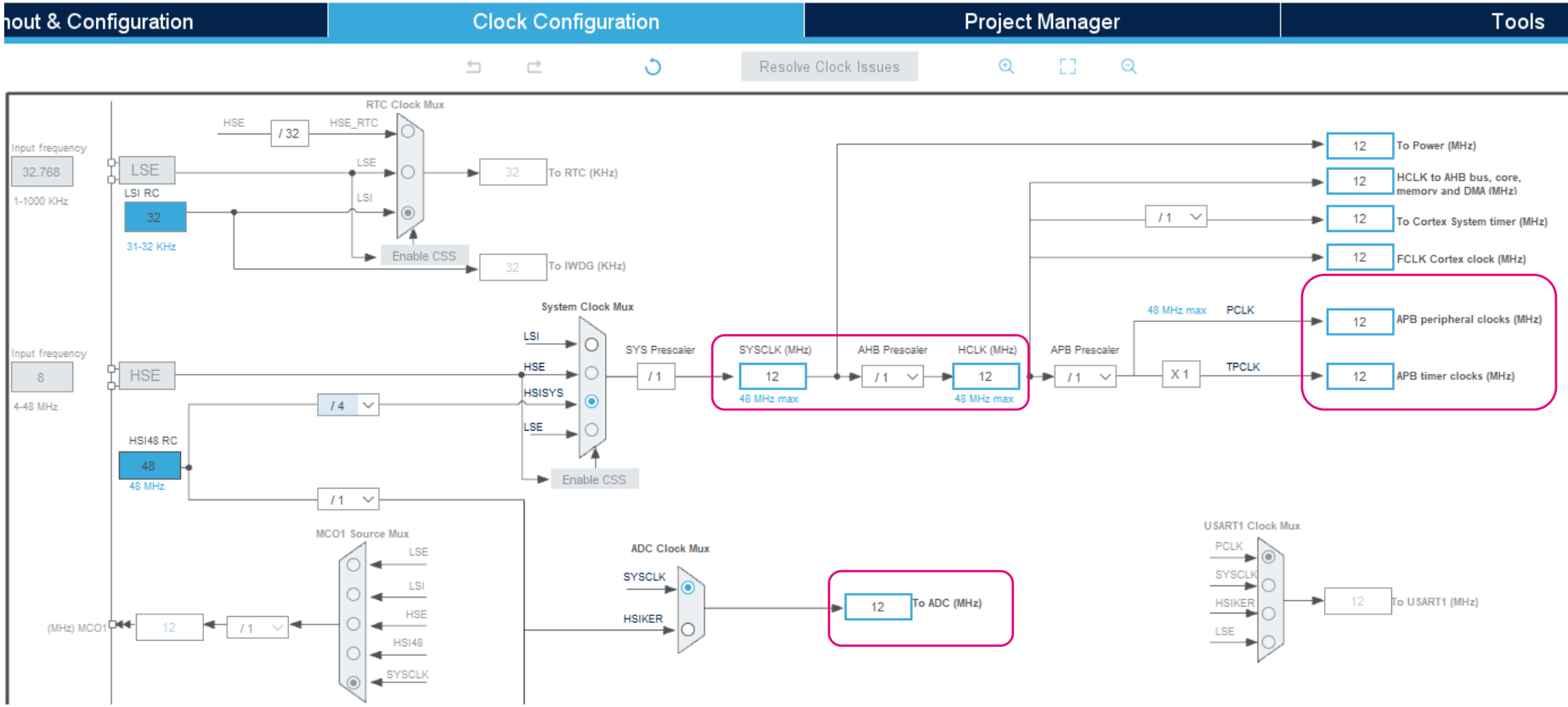




life.augmented

Clock Configuration
default settings

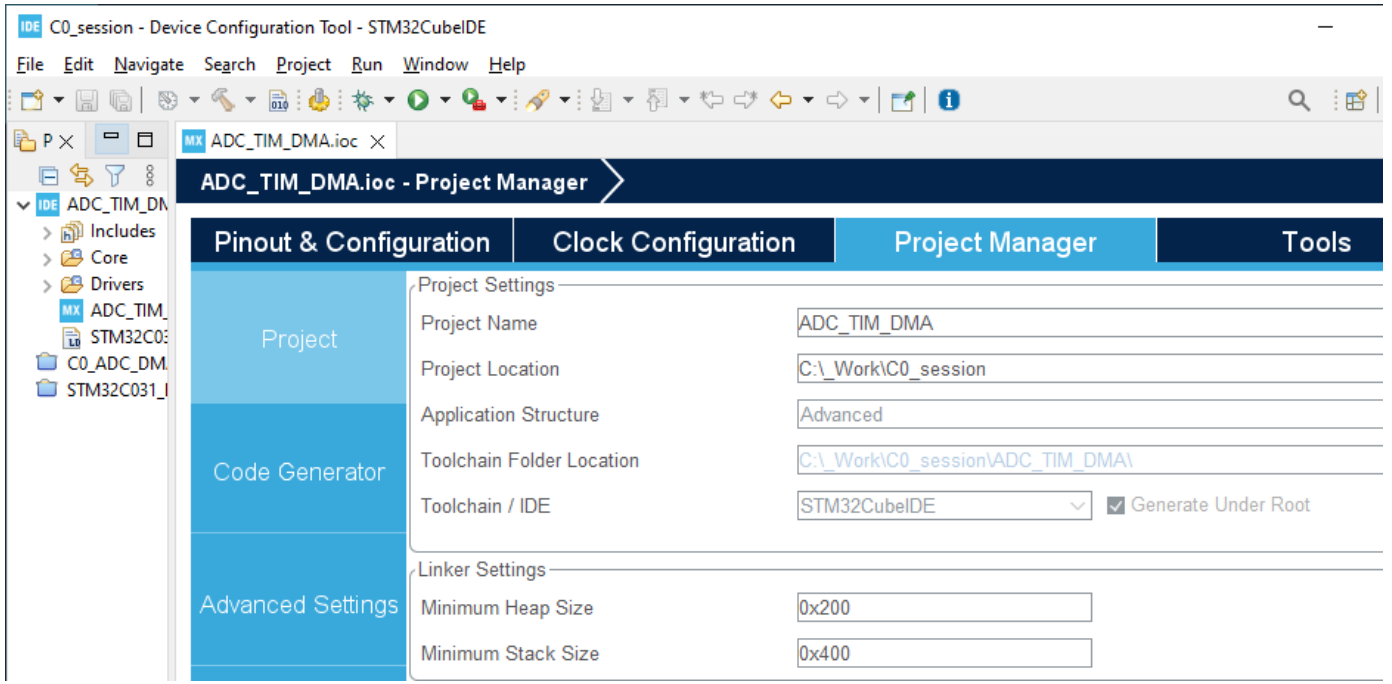
- Default configuration on HSI 12MHz
- No need to change
- System clock 12MHz common for ADC1, TIM3



life.augmented

Project generation
STM32CubeIDE

- Select „Project Manager” section
- Specify project location and project name if not yet done
- Generate the project by one of the following actions:
  - Pressing  icon
  - Saving project by „**Ctrl+S**”
  - Select **Project -> Generate project**
  - Click on **Alt+K**
- After project generation open main.c file



Coding
main.c file

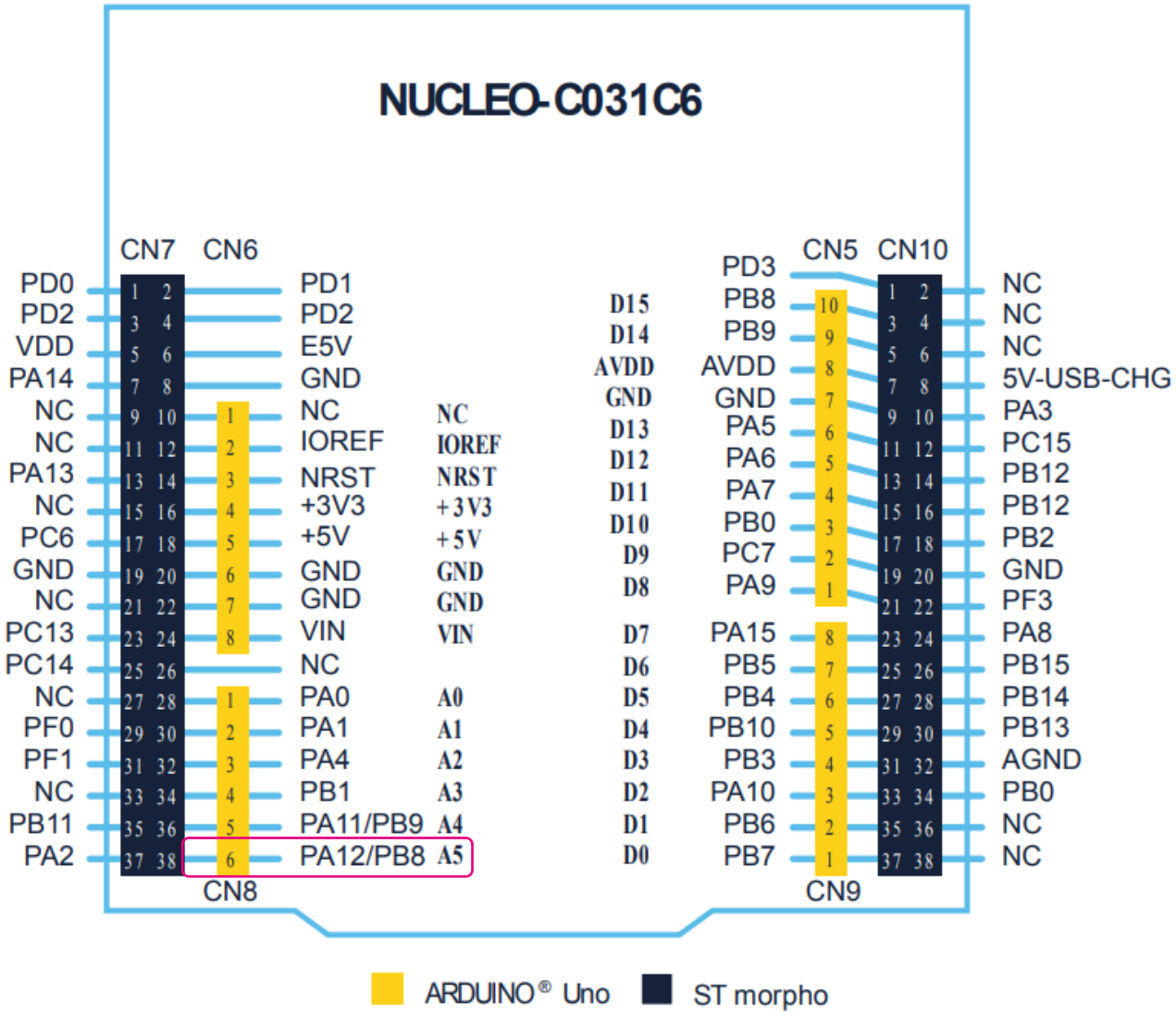
- Data buffer declaration

```
/* USER CODE BEGIN PV */
uint8_t buffer[8];
```

- All peripherals start

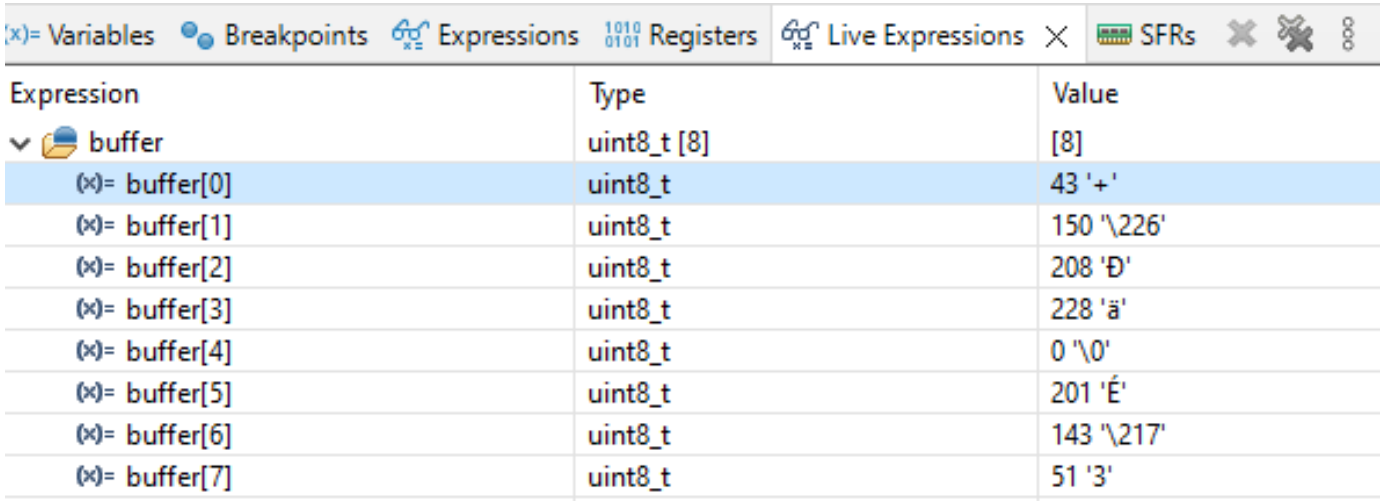
```
 /* USER CODE BEGIN 2 */
  HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_ADC_Start_DMA(&hadc1, (uint32_t *)buffer, 8);
  HAL_TIM_Base_Start(&htim3);
```

PA12 location on your Nucleo board



Run the application

- Compile the code (i.e using „hammer” icon)
- Start debug session (i.e using „bug” icon)
- Add **buffer** table into „live expressions”
- Run the application
- As a result you should see each second new data coming to **buffer** table within live expressions if you change PA12 input signal



In case it is not working, please have a look whether DMA is initialized BEFORE any other peripheral which is using it.

Our technology starts with You

life.augmented