STM32G0 and STM32CubeMX 5.0 Workshop

V1.3

Agenda

8:00 AM – 9:00 AM	**Registration and system check for pre-installed tools**
9:00 AM – 3:00 PM **(Lunch from 12 to 1 PM)**	STM32G0 Overview
	Overview of the STM32CubeProgrammer
	Lab: Upload and save a binary firmware image to a file
	STM32CubeMX 5.0 Overview
	STM32Cube Library
	Lab: Blink an LED by software
	Lab: Use hardware (PWM timer) to blink an LED
	Lab: External Interrupt
	Lab: Low power
	Lab: Power Consumption Estimation
	Optional Lab: Utilize a printf for console output
	Optional Lab: Low Layer Library usage
	Optional DMA presentation
	Optional Lab: DMA
	Lab: Restore original demo code firmware to the board

STM32G0 MCU series

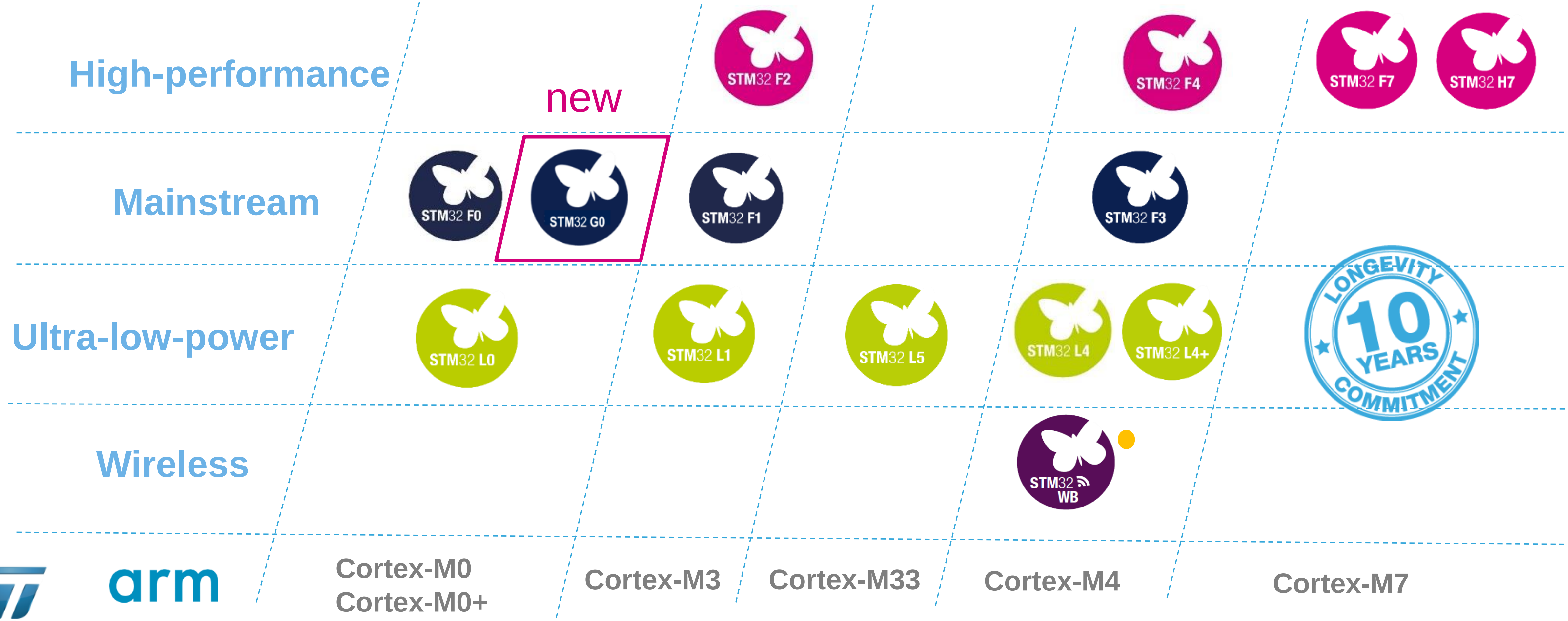
Efficiency at its best

life.augmented

STM32G0: great **investment**

Keep releasing your growing creativity

Note ●: Cortex-M0+ Radio Co-processor



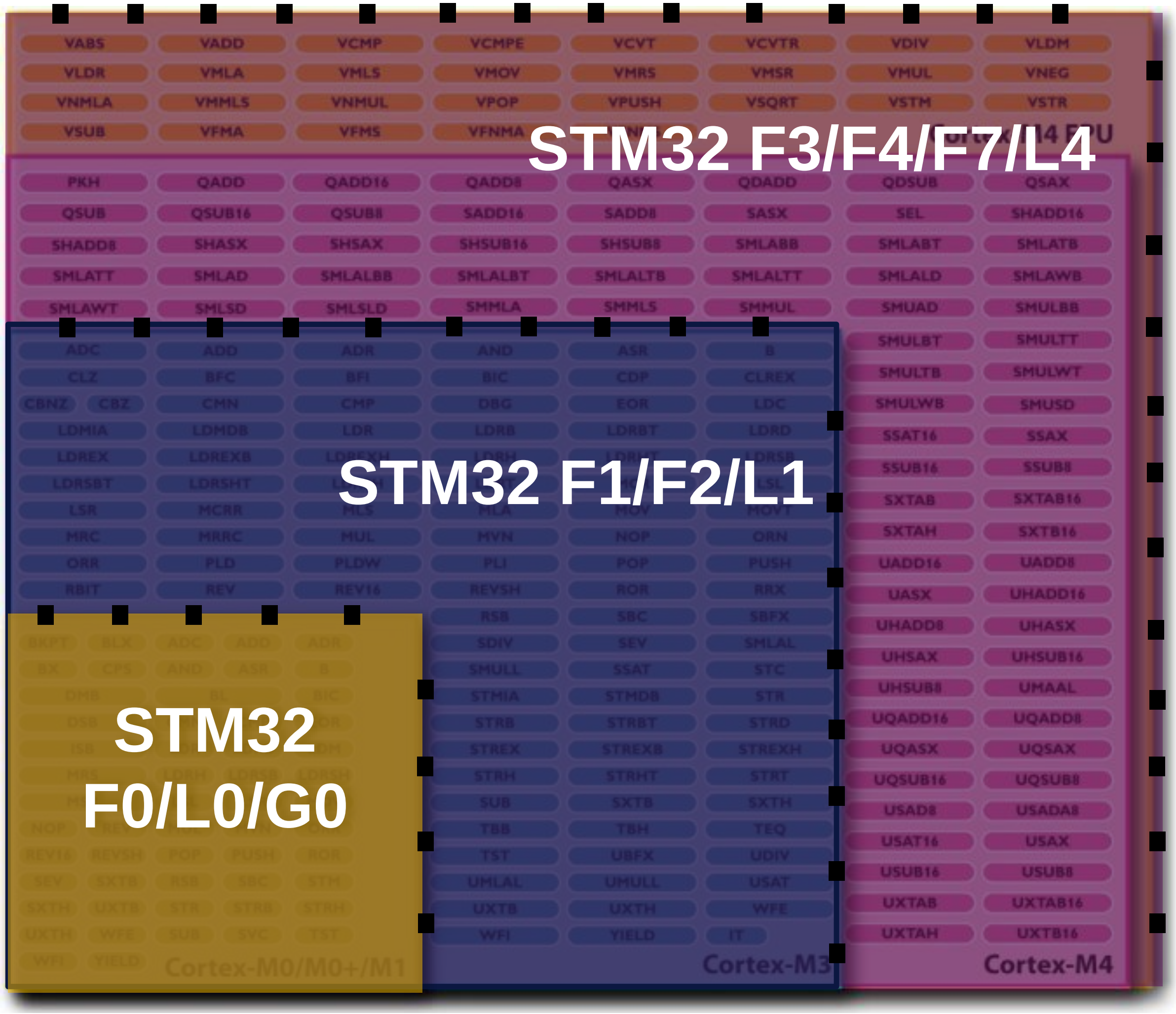
life.augmented

Cortex-M Seamless scalability

Extended Performance
Cortex-M4 / M7
Floating Point Unit (FPU)
DSP (SIMD, fast MAC)

Foundation
Cortex-M3
Advanced data processing
Bit field manipulations

Budget price
Cortex-M0/M0+
General data processing
I/O control tasks



life.augmented

Key messages of STM32G0 series

1

Efficient

- ARM Cortex M0+ at 64MHz
- Compact cost: maximum I/Os count
- Best RAM/Flash Ratio
- Smallest possible package down to 8-pin
- Very low power consumption (3µA in stop, <100µA/MHZ in Run)
- Accurate internal high-speed clock 1% RC
- Best optimization, down to each and every detail
- Offers the best value for money

2

Robust

- Low electromagnetic susceptibility, EMC
- Clock Monitoring and 2 Watchdogs
- Error correction on Flash
- IoT ready with embedded security
- Hardware AES-256 encryption
- New Securable Memory Area
- Safe Firmware upgrade / Install

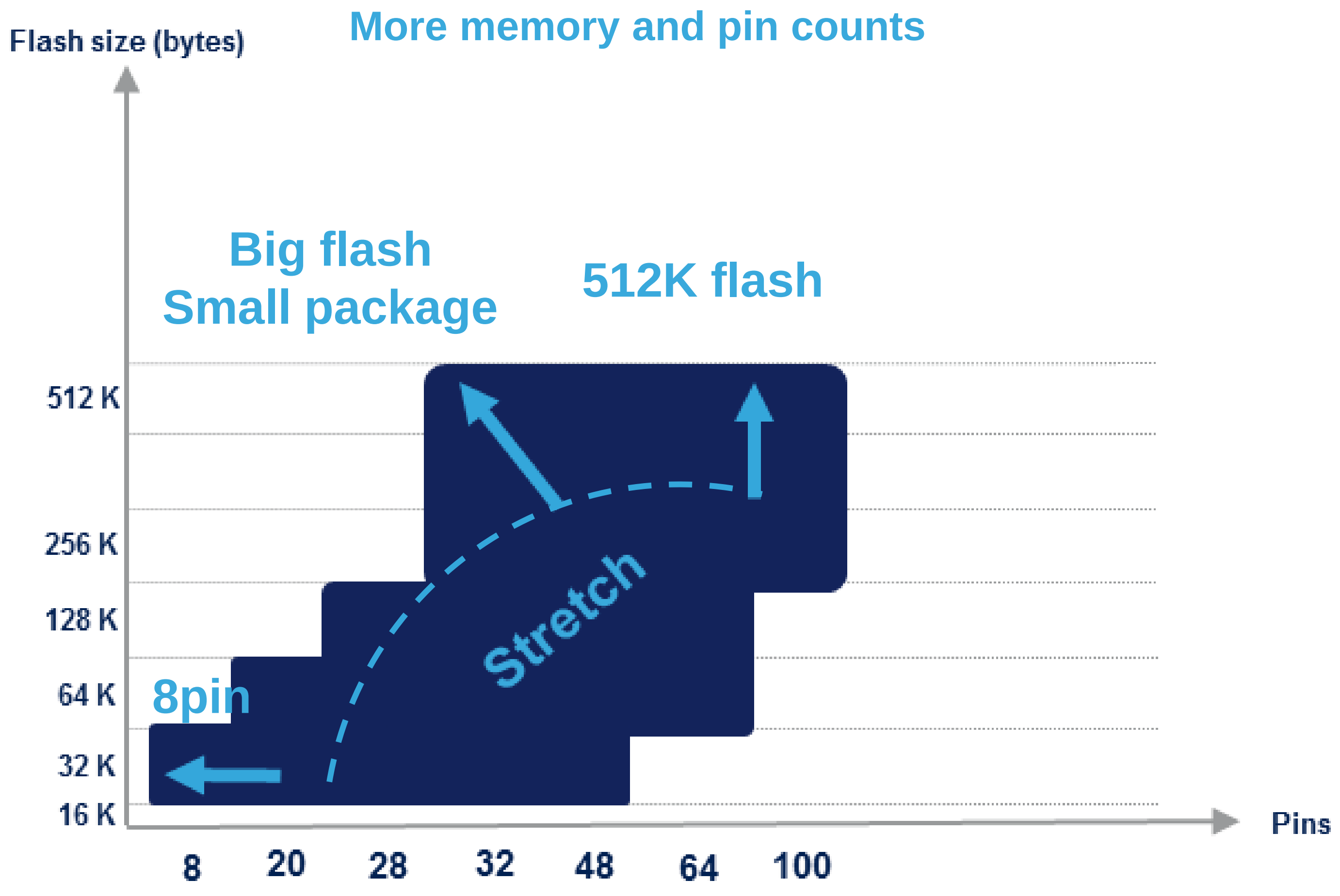
3

Simple

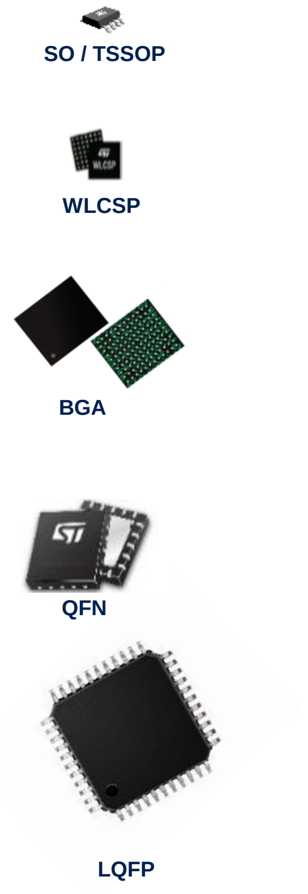
- Easy to configure thanks to the intuitive and graphical STM32CubeMX configuration tool
- Easy to develop based on the Hardware Abstraction Layer library (HAL) or the low-layer library (LL) allowing maximum re-use and faster time-to-market

Wider platform

Portfolio streeeeeeeeetched for efficient budget applications



More packages



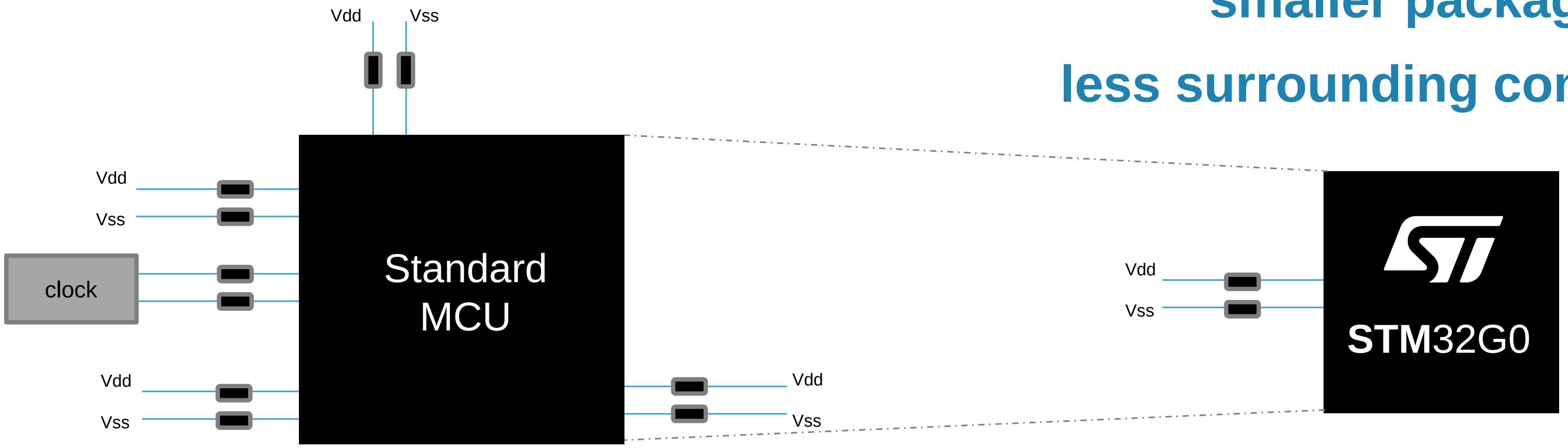
life.augmented

Reducing BOM cost

New platform optimized with 1 power supply pair only up to 64pin packages

smaller package

less surrounding components



Innovations for your benefit

- **No external clock -10cts**
  Accurate internal high speed clock +/-1% for 0 / 85°C
- **No decoupling capacitances -4cts**
  Remove up to 6 decoupling capacitors for supply and clocks
- **Smaller PCB -1cts**
  Smaller package, less components: save on PCB area

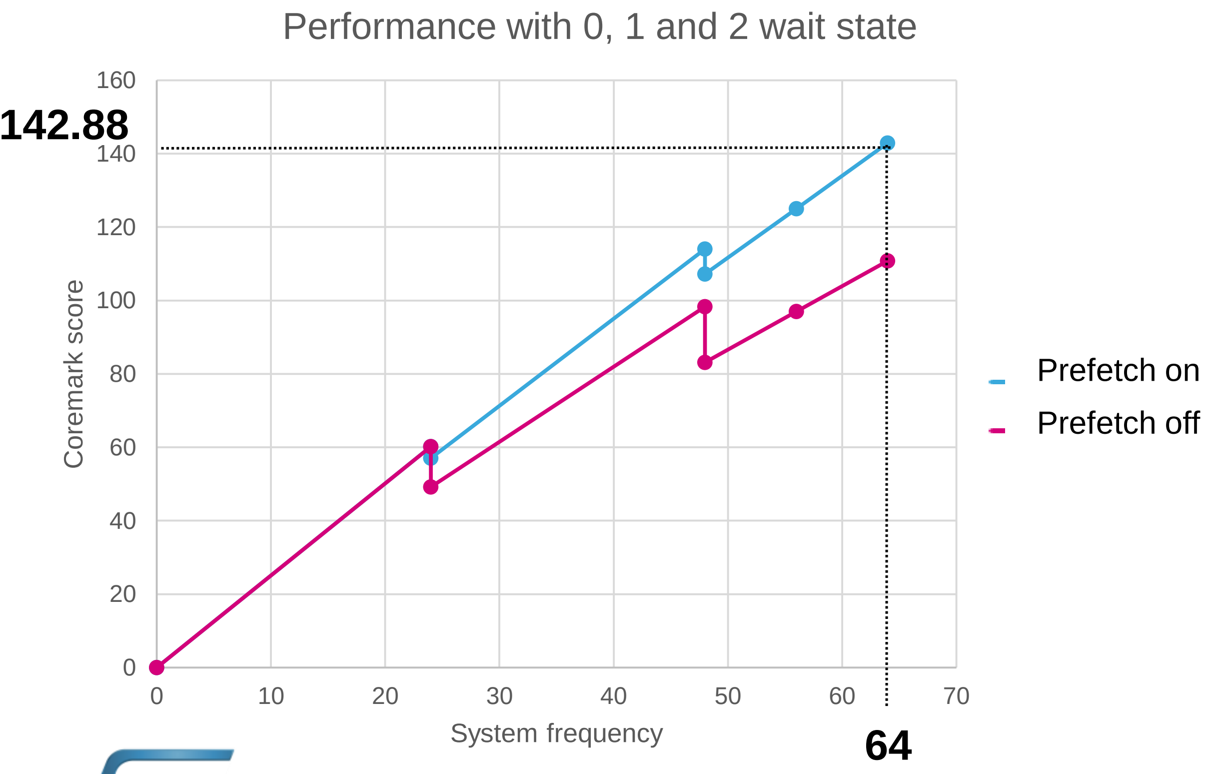
Additional benefits for your convenience:

- **USB-C power delivery -15cts**
  Integrated transceivers, pull-up/down resistors and digital
- **Secure programming -25cts**
  In house or at 3rd parties

life.augmented

Providing **more performance**

Do not compromise on performance with STM32G0



- **Up to 64** MHz/ **59 DMIPS**
- **Up to >142** CoreMark Result
- ARM Cortex-M0+ with Memory Protection Unit (MPU)
- Flexible **DMA** up to 7 channels

STM32 G0

Low-power modes efficiency

When Mainstream MCU Series meets low-power requirements

Wake-up time	Mode	Consumption	Wake-up sources
	VBAT	*10 nA / 400 nA	Tamper: few I/Os, RTC
258 µs	SHUTDOWN	*40 nA / 500 nA	Wake-up sources: reset pin, few I/Os, RTC
14 µs	STANDBY	*200 nA / 500 nA	Wake-up sources: + BOR, IWDG
5 µs	STOP Flash-RTC off-off/off-on/on-off	3.0µA / 5µA / 8µA	Wake-up sources: + all I/Os, PVD, COMPs, LPUART, LPTIM, I²C, UART, USB-PD
6 cycles	SLEEP 24MHz, Vdd=3V, PLL=on	800 µA	Wake-up sources: any interrupt or event
	RUN at 64 MHz	<100 µA / MHz	

Conditions: 25°C, Vdd = 3V

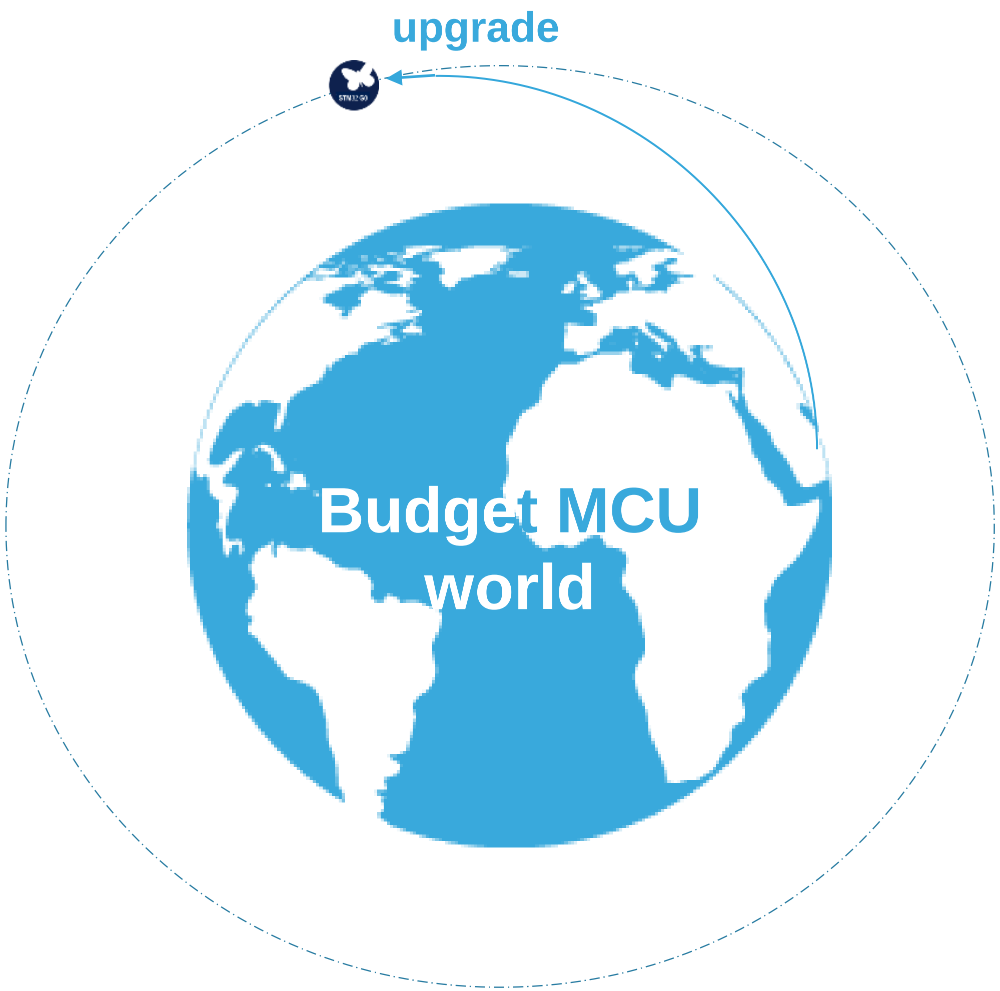
Note : * without RTC / with RTC

life.augmented

Ready for **tomorrow**

Faster, more accurate analog and digital functions

- More **RAM** for Flash
  - Up to 36KB SRAM for 128KB and 64KB Flash
- **Timers** frequency up to **128MHz** resolution (<**8ns)**
  - **Advanced control** capabilities
- **12-bit ADC** up to **2.5MSPS** (0.4µs) conversion time
  - **16-bit** oversampling by hardware
- **32Mbit/s SPI**, 7 Mbaud/s USART, 1Mbit/s I²C communication



Smart peripherals

STM32 G0

FD CAN
Up to 2 instances

OBD

Industrial

V_{BAT} with RTC
for battery backup

400 nA in V_{BAT} mode
for RTC and
20x 32-bit backup registers

TRNG & AES
for Security

128-/256-bit AES
key encryption hardware
accelerator

Comparators
2 instances
Down to 30ns propagation delay

DAC
2x 12-bit DAC,

ADC
16x12-bit, 16-bit oversampling
2.5MSPS (0.4µs)

Timers
8ns PWM resolution
Advanced control
16- and 32-bit

STM32 G0

USB-C Power Delivery
Up to 2 ports with dead-battery management

USB
USB 2.0
Full speed
Device / Host

SPI / UART / I²C
4x SPIs
8 USARTs (ISO 7816, LIN, IrDA, modem)
3 I²C

I/Os
Up to 92 fast I/Os

Smart **integration**

Save on battery life

Low consumption process and design
Low-Power UART: wake-up on frame
Low-Power Timer: counts and generate signals
I²C wake-up on address

Save on BOM cost

+/-1% high speed clock internal from 0 to 85°C
+/-2% high speed clock internal from **-40 to 125°C**
IO maximization: **smaller package footprint**

More flexibility

More RAM or **more safety** with parity enable/disable
Dynamic DMA assignement on **DMAMUX**
All IOs with external interrupt capability

STM32 G0

Always keep control
Diagnose, react

Main Clock monitoring
Backup clock and interrupts
Voltage monitoring: programmable interrupts and reset
Window watchdog on CPU clock
Independant watchdog on independant clock
Checksum by hardware
ECC on Flash, **Parity** on RAM

High temperature

from -40°C
up to + **125°C**

High robustness

Highly immune to fast-transients
Robust IOs against negative injections

Smart applications

Lighting

- High temperature 125°C
- Fast CPU 64MHz
- Advanced timers with high-resolution 7.8ns
- Fast comparators
- ADC-12bit, DAC-12bit
- Low-thickness packages
- AES & security for secure upgrades

Consumer objects

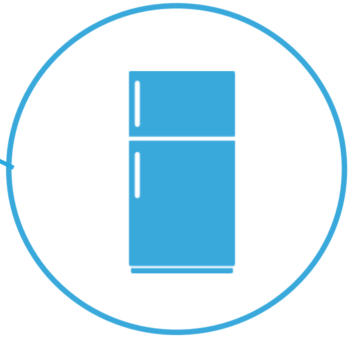
Smartphones, IoT devices, rechargeable connected devices, drones, toys

- Low-thickness, small form-factor
- 64MHz CPU with DMA
- Low consumption in run and low-power, fast wake-up
- USB type-C Power Delivery 3.0
- USB FS 2.0 dev/host crystal-less

Industrial devices
Motor control
Advanced control

Air conditioning, e-bikes, industrial equipments

- High temperature 125°C
- CANFD support
- SPI, USART, I²C
- Advanced timers with high-resolution 7.8ns
- Real Time Clock with backup registers
- AES & security for secure upgrades



Smart Home

Home appliances, alarms and safety, advanced user interfaces

- High temperature 125°C
- Safety monitoring features
- More RAM for flash
- Low consumption <100µA/MHz in run

STM32 G0

life.augmented

STM32 G0 product lines

Common peripherals and architecture:

- ARM Cortex-M0+ 64MHz 0.93 DMIPS/MHz
- MPU
- Communication Peripheral: USART, SPI, I²C
- Multiple general-purpose 16-bit timers
- Integrated reset and brown-out waming
- DMA channels
- 2x watchdogs Real-time clock (RTC)
- Integrated regulator PLL and clock circuit
- Main oscillator and 32 kHz oscillator
- **Internal RC oscillators** 32kHz , 16 MHz
- -40 to +125 °C
- Low voltage 1.65 to 3.6 V (Value Line: 2.0 to 3.6V)
- Temperature sensor

Up to +125°C

STM32G0x0 – Value Line (ex: STM32G070)

Up to 512KB Flash	Up to ç-80KB SRAM	12-bit ADC 2.5MSPS

STM32G0x1 – Access Line (ex: STM32G071)

Up to 512KB Flash	Up to 80KB SRAM	12-bit ADC 2.5MSPS	2x Comp 2x 12-bit DAC	1x 32-bit Timer	1x 16-bit MC Timer >100MHz	1x16-bit Timer >100MHz	USB-PD	USB OTG 2.0 FS	CAN-FD	Securable Memory Area

STM32G0+11 – Access Line & Encryption (ex: STM32G081)

Up to 512KB Flash	Up to 80KB SRAM	12-bit ADC 2.5MSPS	2x Comp 2x 12-bit DAC.	1x 32-bit Timer	1x 16-bit MC Timer >100MHz	1x16-bit Timer >100MHz	USB-PD	USB OTG 2.0 FS	CAN-FD	Securable Memory Area	AES TRNG

On some part-numbers

life.augmented

STM32G0 Portfolio

	8pin SON	20pin TSSOP	28pin UFQFPN	32pin LQFP UFQFPN	48pin LQFP UFQFPN	64pin LQFP UFBGA	100pin LQFP
512K / 256K				G0C1 G0B1	G0C1 G0B1	G0C1 G0B1 G0B0	G0C1 G0B1 G0B0
128K			G081 G071	G081/ G0C1 G071/ G0B1 G070	G081/ G0C1 G071/ G0B1 G070	G081/ G0C1 G071/ G0B1 G070	
64K		G041 G031	G041/ 81 G031/ 71	G041/ 81 G031/ 71 G030	G041/ 81 G031/ 71 G030	G081 G071	
32K	G041 G031 G030	G041 G031 G030	G041 G031	G041 G031 G030	G041 G031 G030		

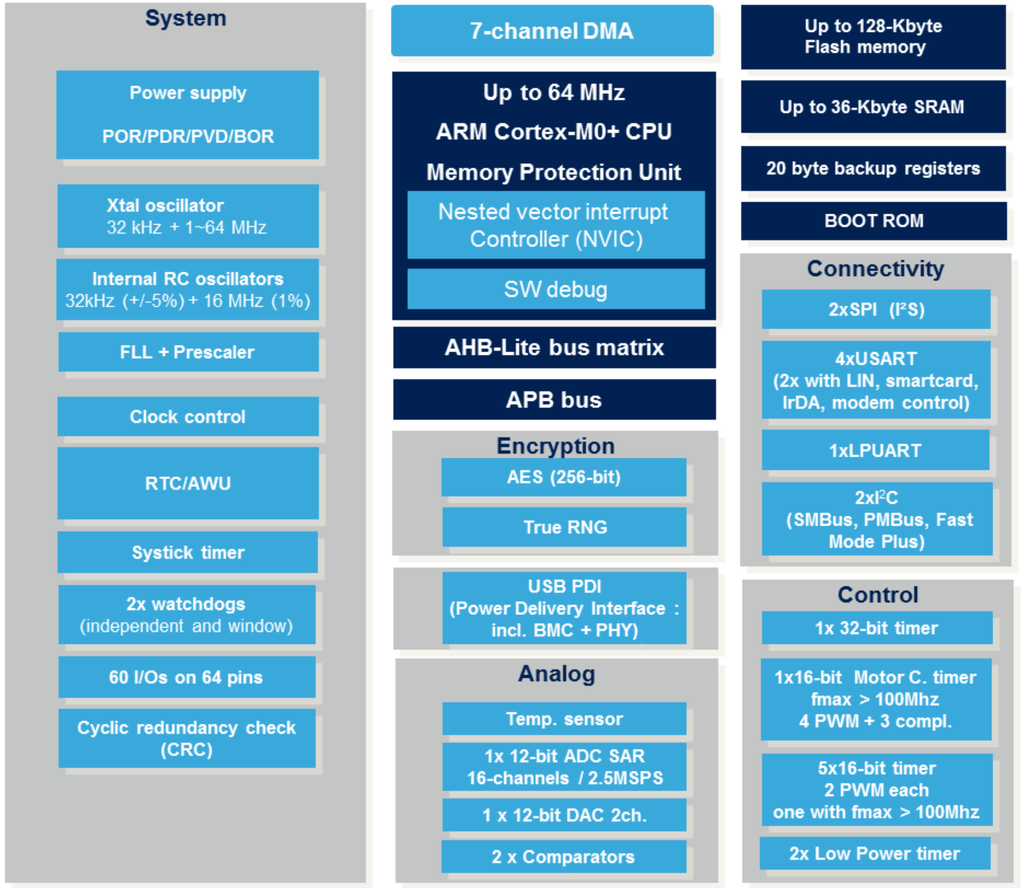
+ WLCSP25 STM32G081
+ WLCSP18 STM32G041

Access AES
Access
Value

Access Line

Advanced features and solutions

- Arm 32-bit Cortex-M0+ core
- 1.7 to 3.6V power supply
- RAM maximization
- 1% internal clock
- Direct Memory Access (DMA)
- Communication peripherals
- USB-C Power Delivery



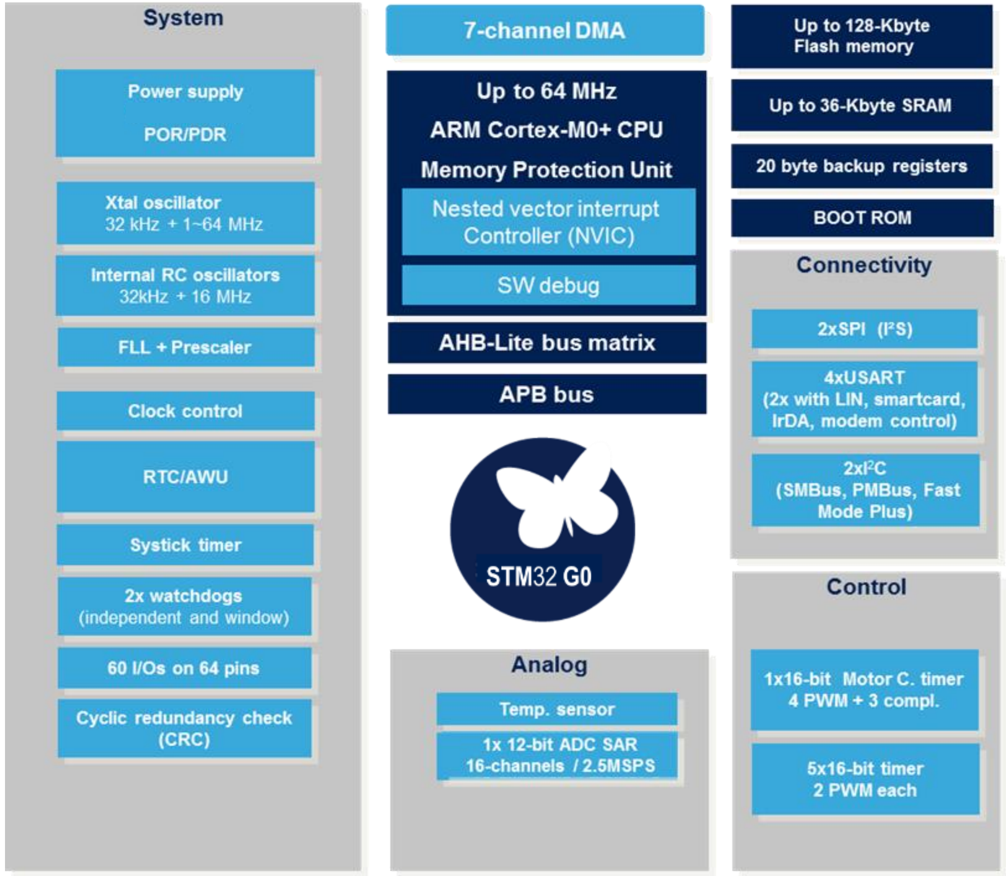
- Timers up to 2xfcpu resolution
- Real Time Clock
- I/O ports maximization
- ADC 12-bit Ultra-fast
- DAC 12-bit
- Comparators
- Safety features
- Advanced Security features

life.augmented

Value Line

No compromise on what matters

- Arm 32-bit Cortex-M0+ core
- 2.0 to 3.6V power supply
- RAM maximization
- 1% internal clock
- Direct Memory Access (DMA)
- Communication peripherals



- Timers
- Real Time Clock
- I/O ports maximization
- ADC 12-bit Ultra-fast
- Safety features

STM32 G0

life.augmented

More **security**

Integrated security features, ready for tomorrow's needs

Firmware IP protection

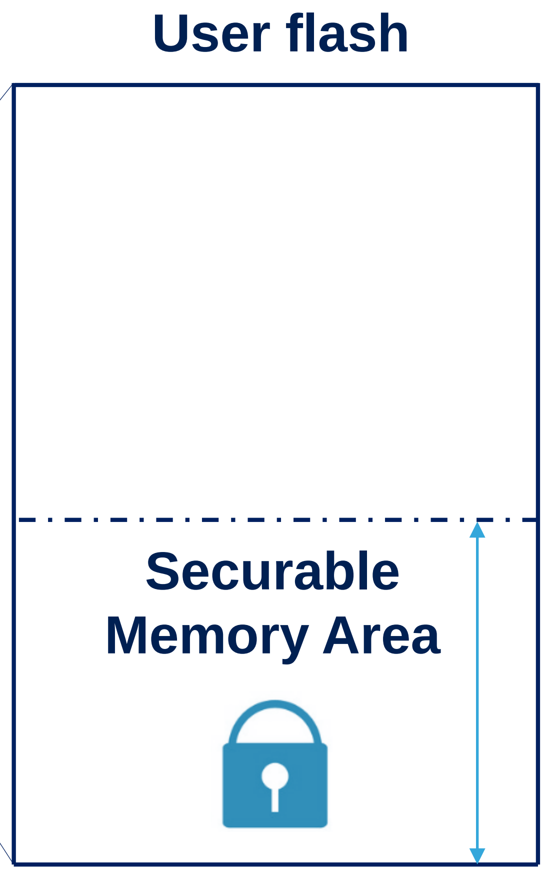
Mutual distrustful

Secret key storage

Authentication

Secure firmware upgrade

Securable Memory Area
Execute-only Protection
Read-out Protection
Write Protection
Memory Protection Unit (MPU)
AES-256 / SHA-256 Encryption
True Random Number Generator
Unique ID



Standard user flash by default

Can be secured once exiting
No more access nor debug

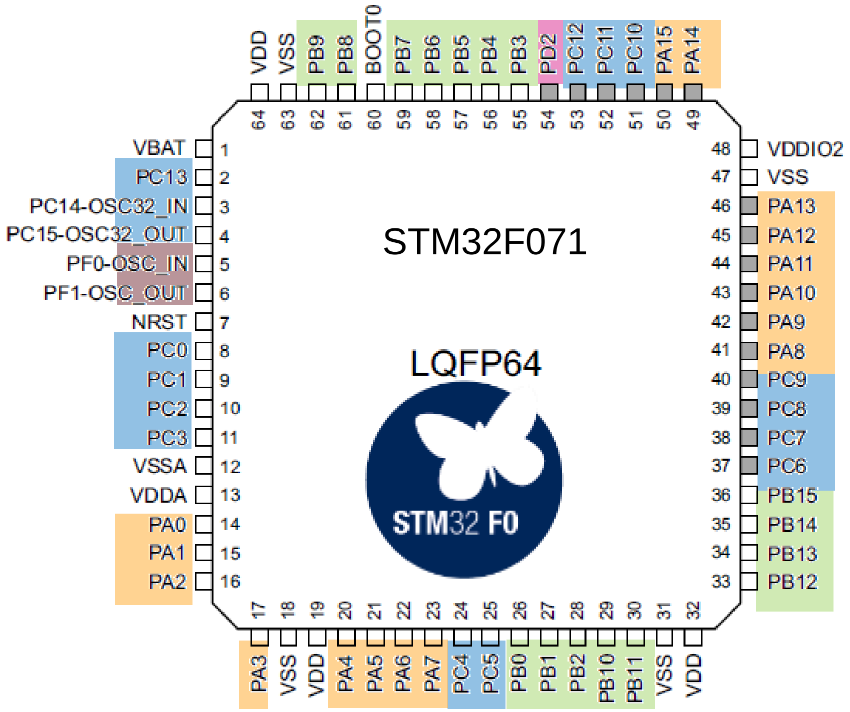
Configurable size

Good fit to store critical data
- **Critical routines**
- **Keys**

life.augmented

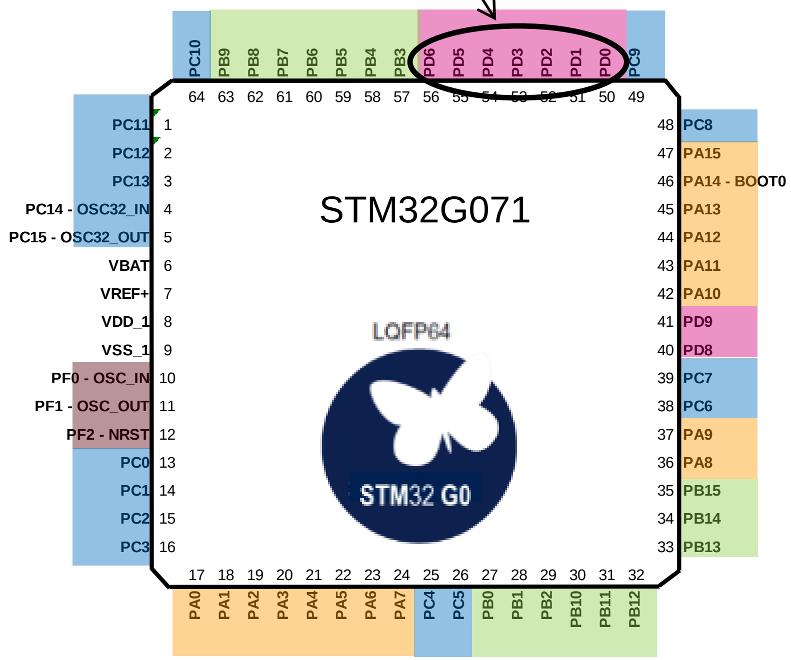
LQFP64 pin-to-pin comparison

- **9 IOs more on STM32G071 vs STM32F071**
- 5 IOs more on STM32G071 vs STM32F051



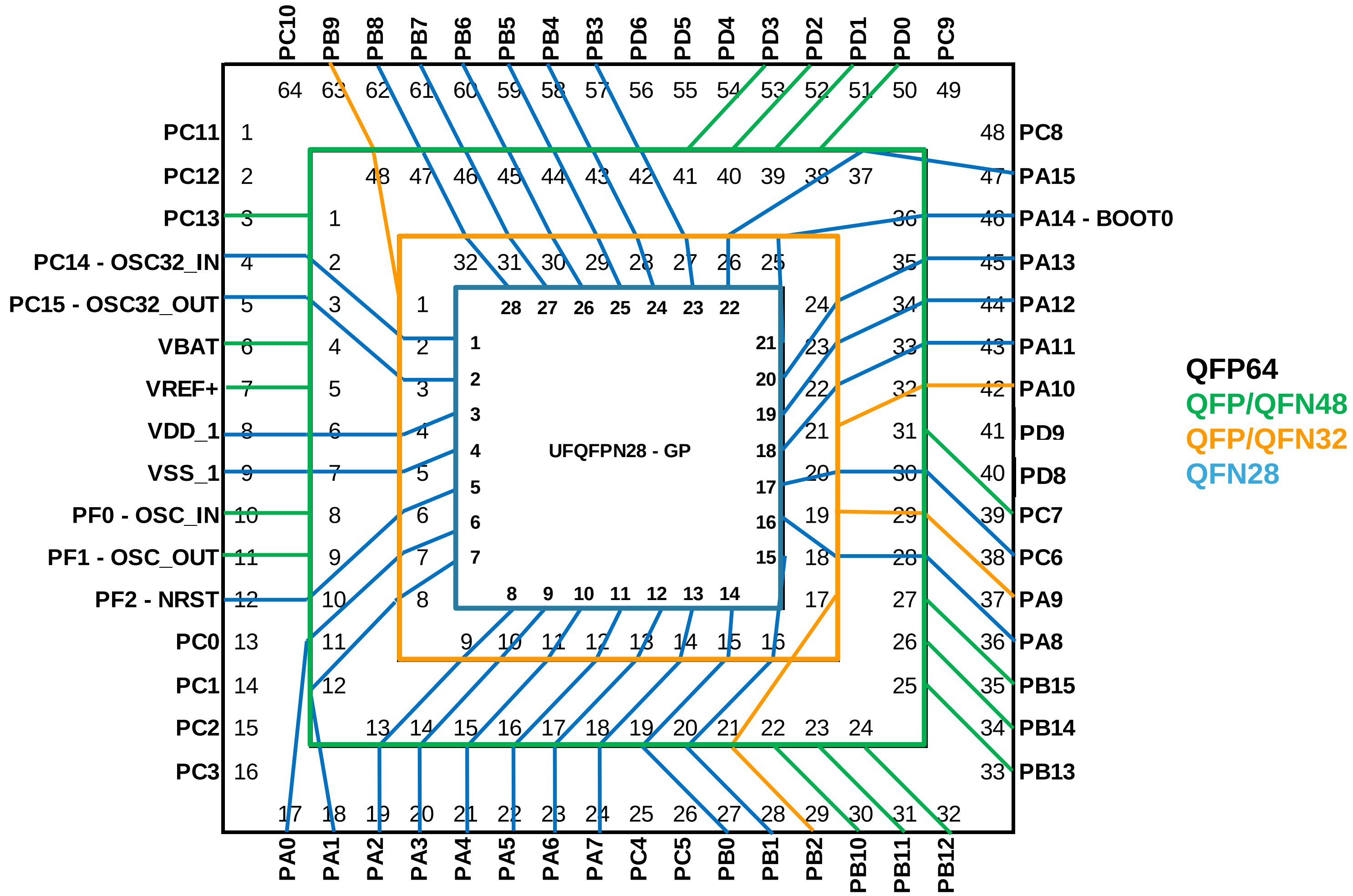
51 I/Os

Extra-port vs STM32F0 ☺



60 I/Os

Consistent and optimized pinout



STM32G0 ecosystem

Go fast, be first

HARDWARE TOOLS

STM32 Nucleo

Flexible prototyping

Discovery kit

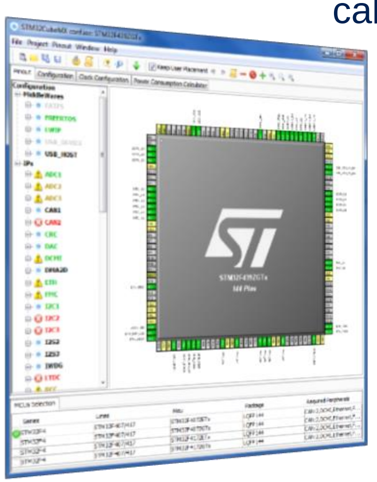
Key feature prototyping

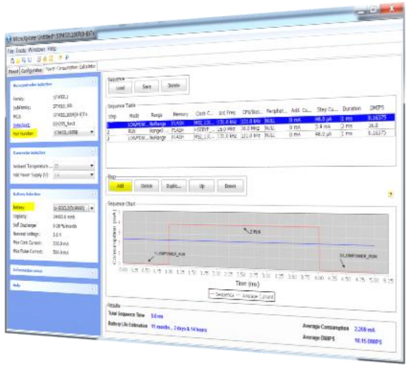
Evaluation board

Full feature evaluation

SOFTWARE TOOLS

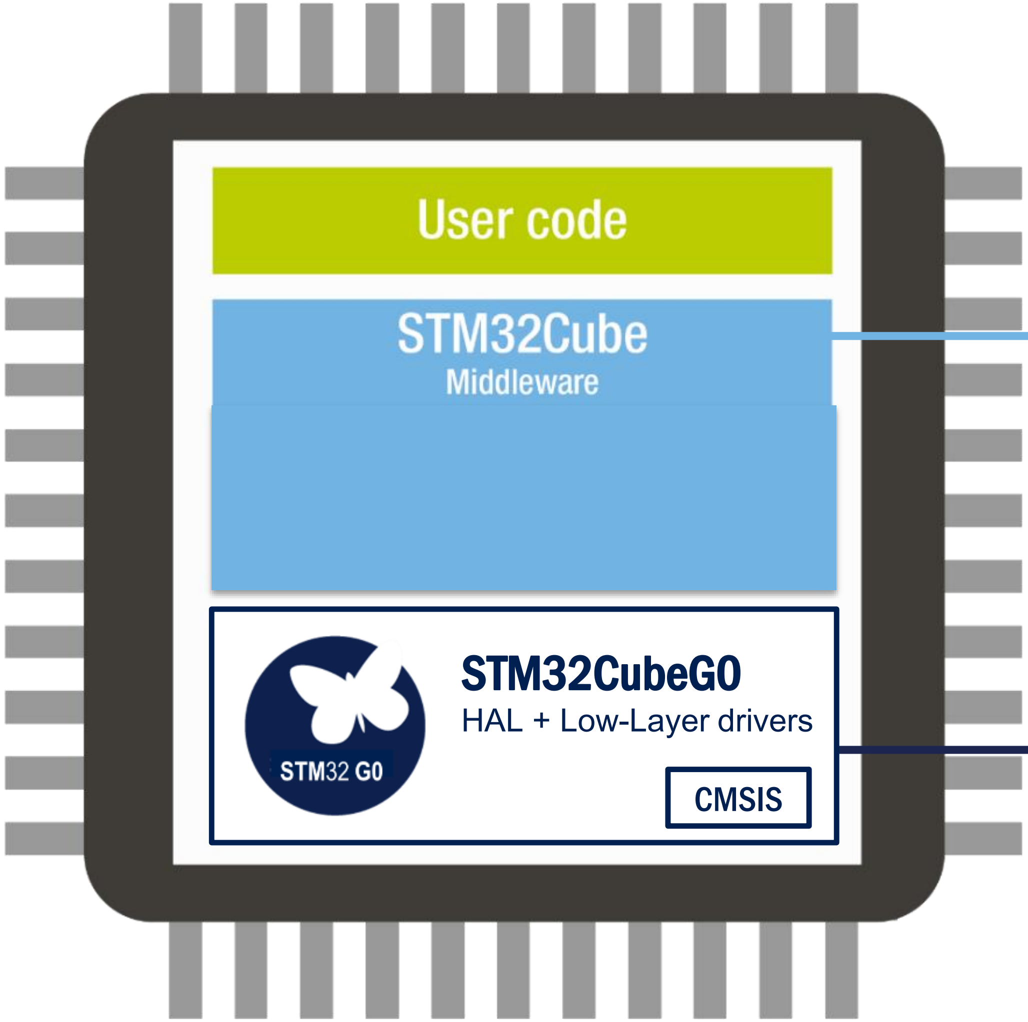
STM32CubeMX featuring intuitive pin selection, clock tree configuration, code generation and power consumption calculation





STM32G0 ecosystem

Platform approach or custom code: you choose



- Open-source FAT file system (FatFs)
- Open-source real-time OS (FreeRTOS)
- Dozens of examples

- STM32G0 Hardware Abstraction Layer (HAL) portable APIs
- **High-performance, light-weight low-layer (LL) APIs**
- High coverage for most STM32 peripherals
- Production-ready and fully qualified
- Dozens of usage examples
- Open-source BSD license

SUMMARY 3 Keys of STM32G0 series

1 **Efficient**

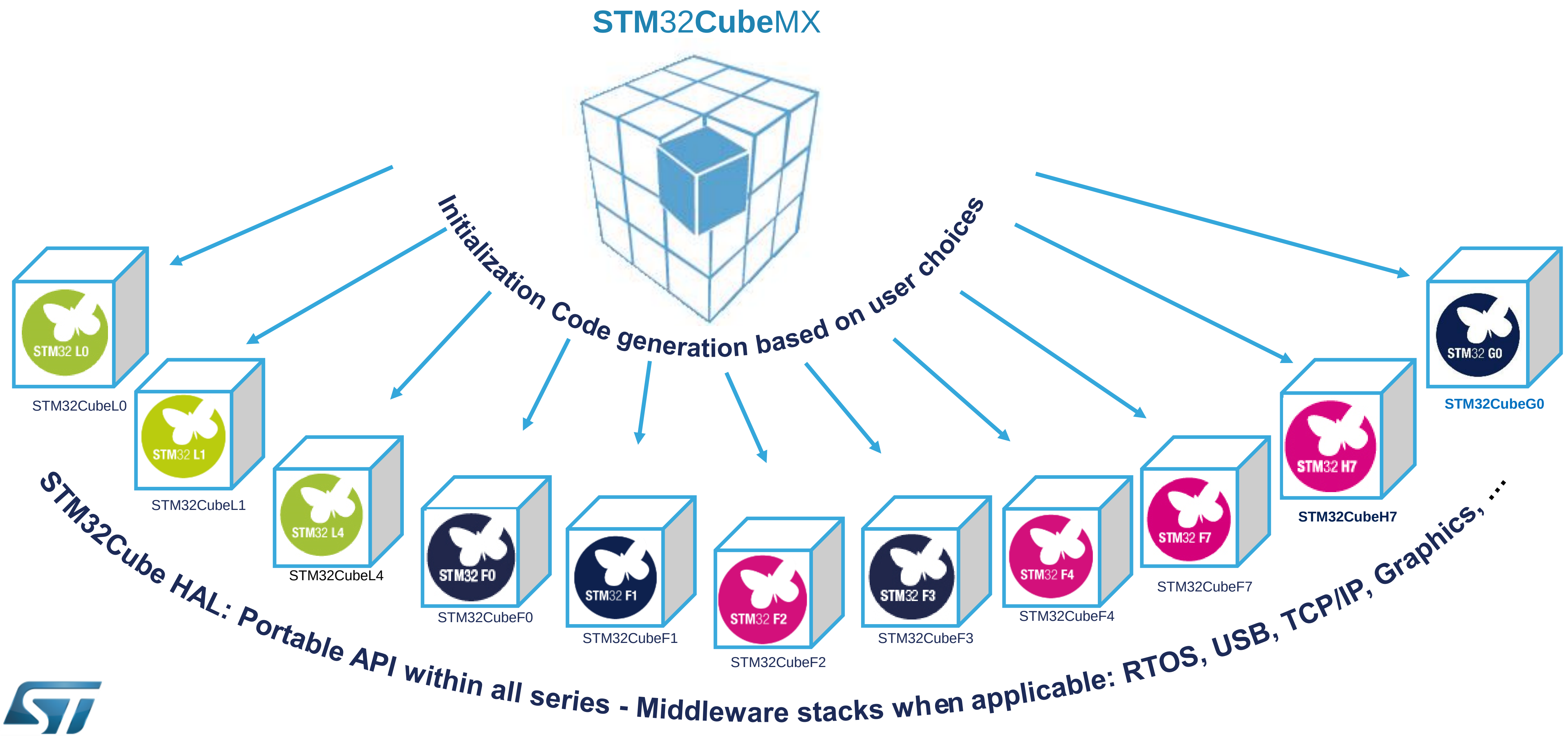
2 **Robust**

3 **Simple**

STM32CubeMX 5.0

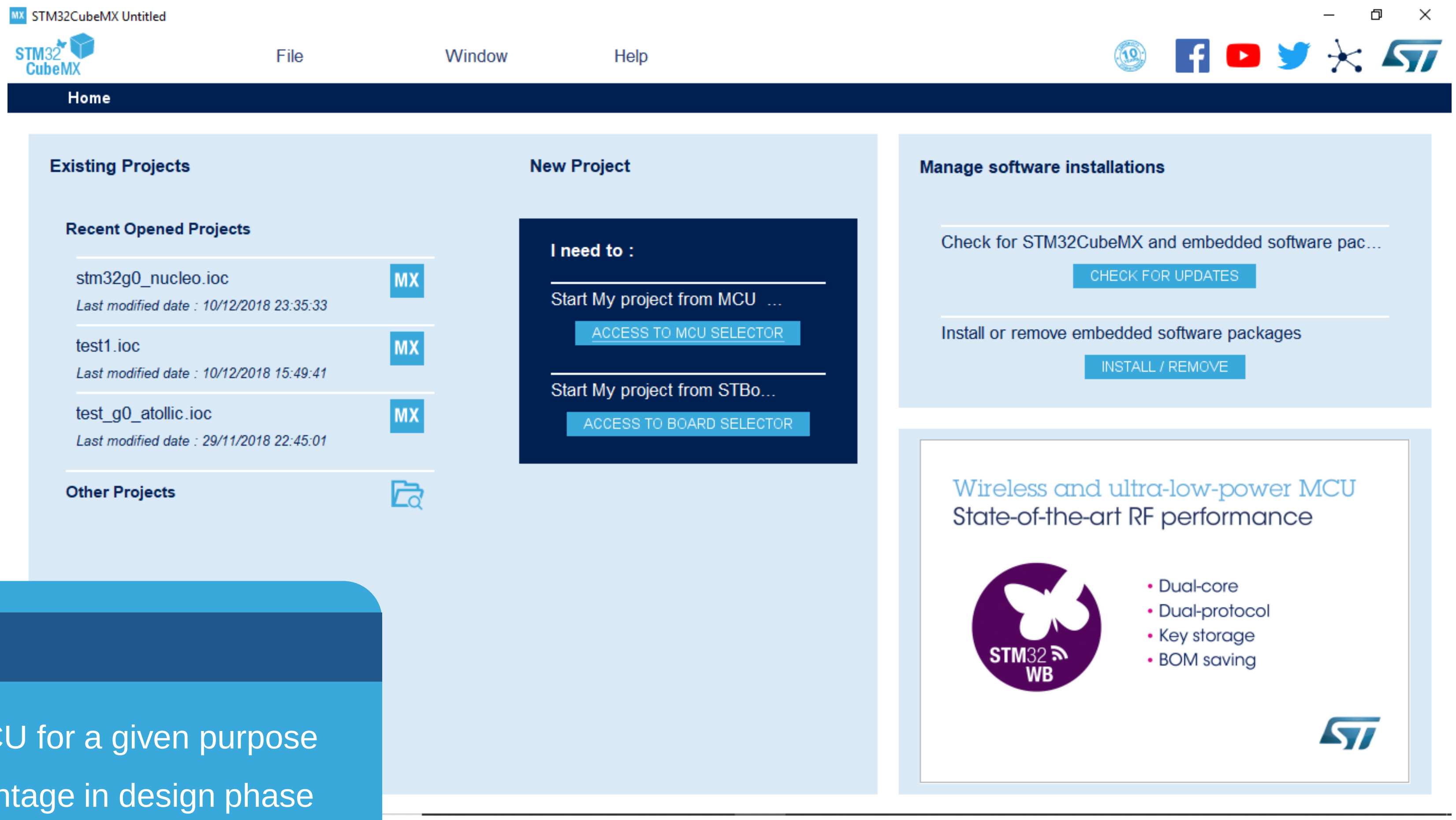
STM32CubeMX graphical software configuration tool

life.augmented


STM32CubeMX
Initialization Code generation based on user choices
STM32 L0
STM32CubeL0
STM32 L1
STM32CubeL1
STM32 L4
STM32CubeL4
STM32 F0
STM32CubeF0
STM32 F1
STM32CubeF1
STM32 F2
STM32CubeF2
STM32 F3
STM32CubeF3
STM32 F4
STM32CubeF4
STM32 F7
STM32CubeF7
STM32 H7
STM32CubeH7
STM32 G0
STM32CubeG0
STM32Cube HAL: Portable API within all series - Middleware stacks when applicable: RTOS, USB, TCP/IP, Graphics, …

Overview

- Choose ideal MCU and simply configure
  - Pinouts
  - Clocks and oscillators
  - Peripherals
  - Low-power modes
  - Middleware



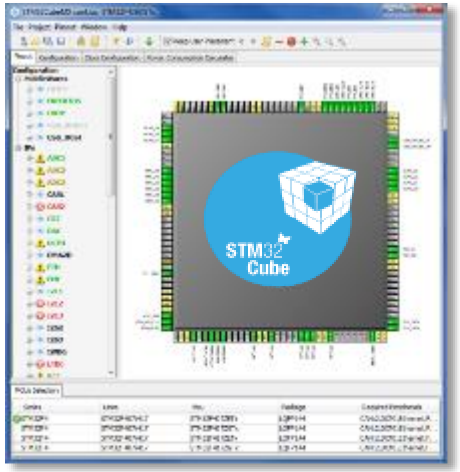
Application benefits

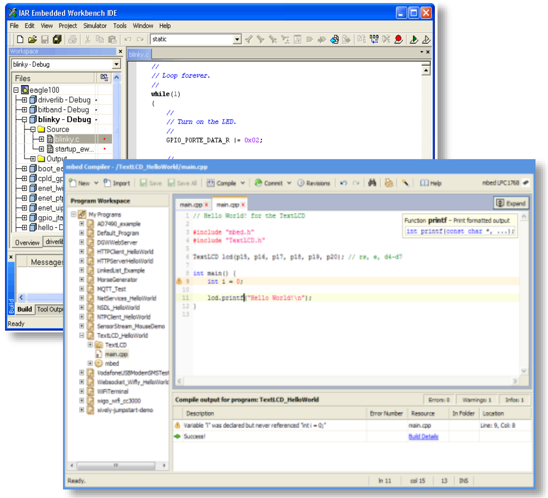
- Helps choose the correct MCU for a given purpose
- Simulation provides an advantage in design phase
- Boosts development speed with a headstart

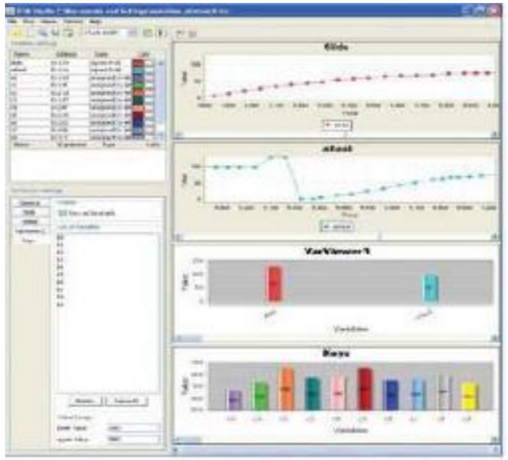
Key features

- Peripheral and middleware parameters
- Power consumption calculator
- Code generation
  - Possible to re-generate code while keeping user code intact.
- Option of command-line and batch operation
- Expandable by plugins
- MCU selector
  - Filter by family, package, peripherals or memory sizes.
  - Search for similar product.
- Pinout configuration
  - Choose peripherals to use and assign GPIO and alternate functions to pins.
- Configure NVIC and DMA
- Clock tree initialization
  - Choose oscillator and set PLL and clock dividers.

Comprehensive choice of IDEs







STM32CubeMX

Generate Code

Partners IDEs

Compile & Debug

STMStudio

Monitor

IAR SYSTEMS

iSYSTEM

EMPROG

TASKING

KEIL LABS

SEGGER Embedded Studio

ARM KEIL Microcontroller Tools

CooCox

atollic

mbed

ac6

Free IDE

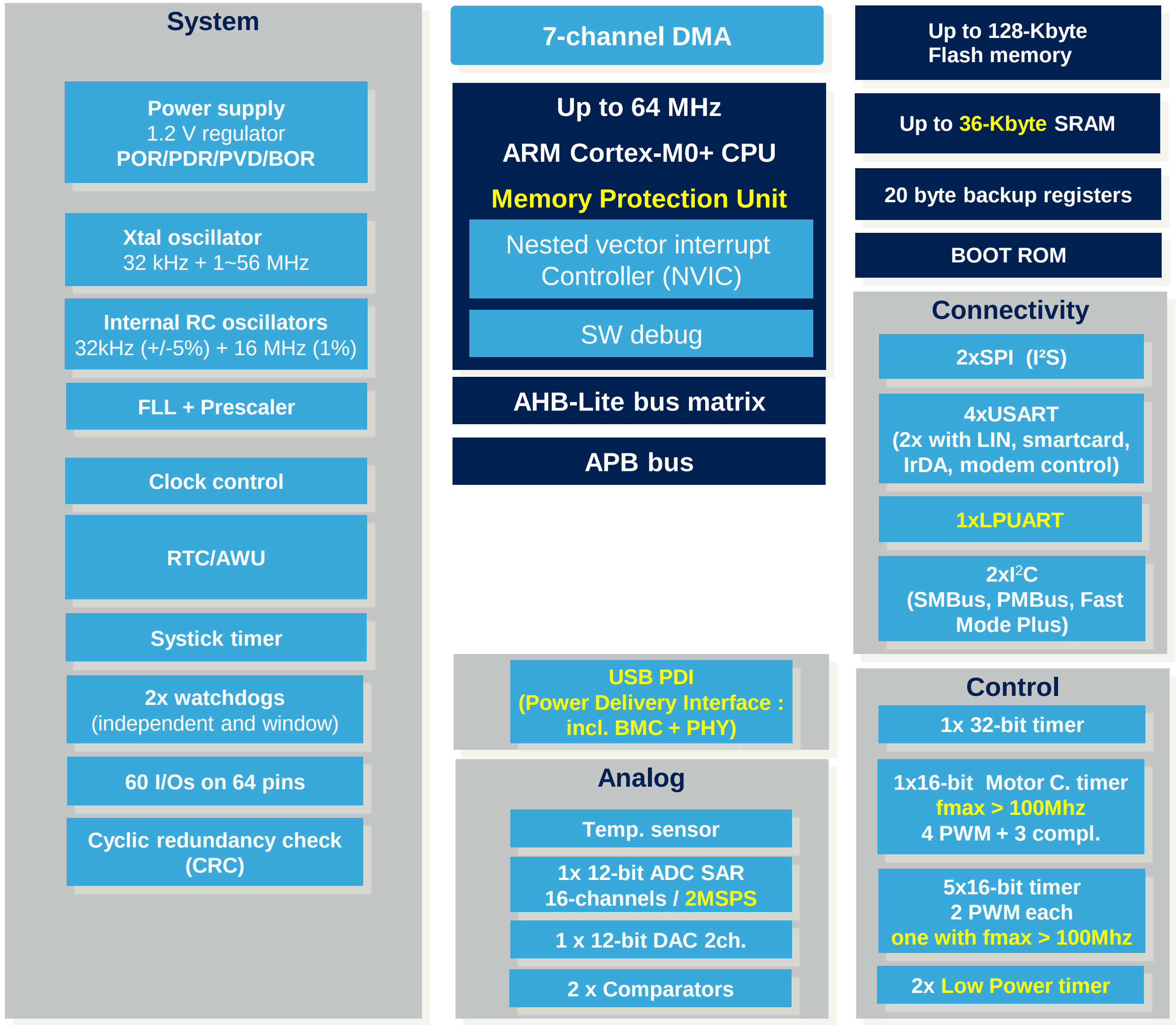
life.augmented

STM32G0 – System Architecture

STM32G071 Block diagram

Main specification

- 1.65V to 3.6V
- 0.93 DMIPS/MHz
- **Vbat** supply
- Vref+ pin
- **Max ambiant temp 125°C**
- One Supply pair
- Securable Memory Area
- High sink I/Os
- <100µA/MHz run mode
- **Stand-by** <1µA @ room temperature
- Stop 5 µA @ room temperature
- **Shutdown mode**
- Low EMI SAE (2.5@24MHz)
- Robust EMC/ESD/EMS
- 28/32/48 and 64 pins



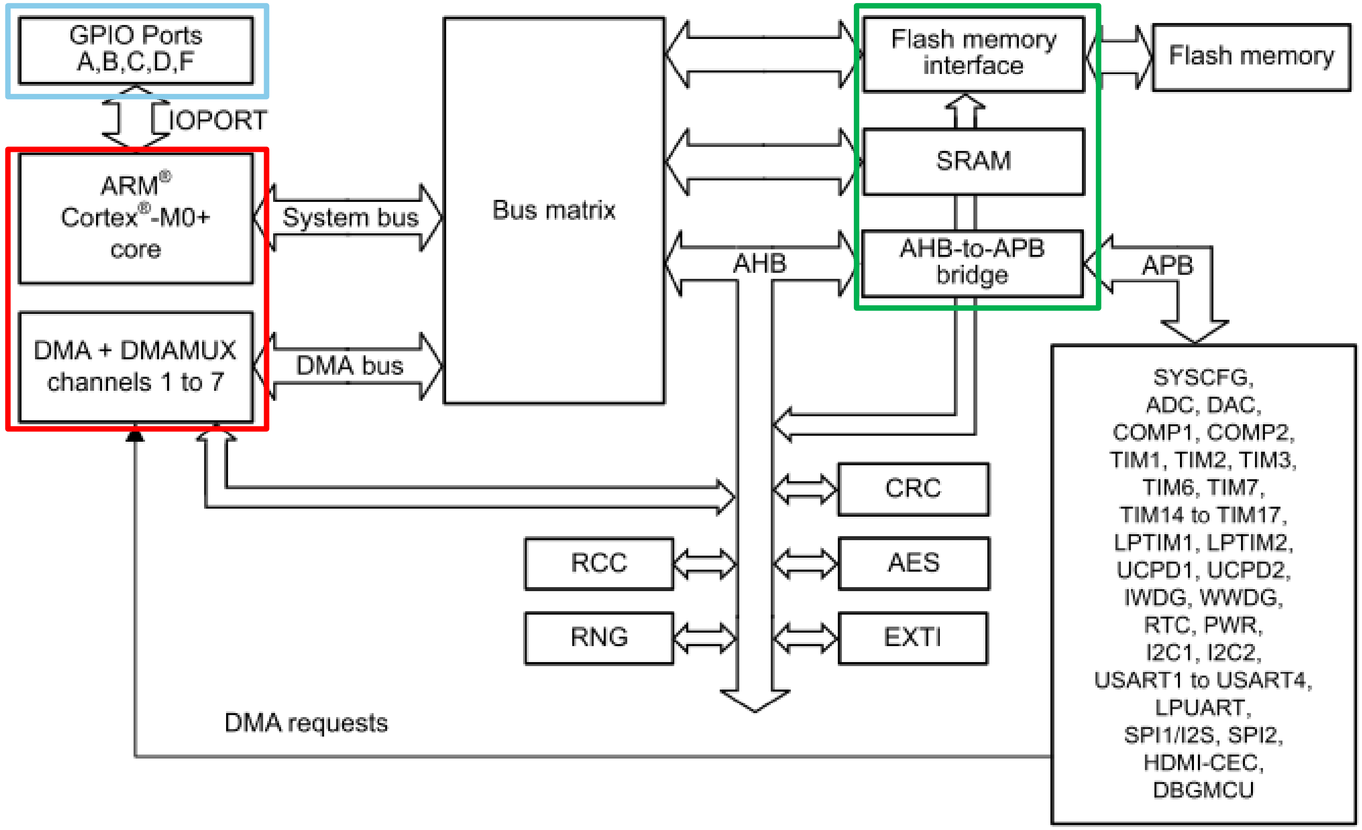
features highlight

Cortex M0+ vs Cortex M0

- STM32G0 has been upgraded with a Cortex M0+ core bringing more security and performances

	STM32F0	STM32G0
Relocatable vector table	No	Yes
Pipeline	3 Stages	2 Stages
Performance	2.20 Coremark/MHz (48MHz – 1WS)	2.23 Coremark/MHz (64MHz – 2WS)
MPU	No	Yes
Breakpoints	4	4
Single cycle 32bits x 32bits Multiplier	Yes	Yes

System architecture overview



- Two masters:
  - Cortex®-M0+ core
  - General-purpose DMA
- Three slaves:
  - Internal SRAM
  - Internal Flash memory
  - AHB with AHB-to-APB bridge that connects all the APB peripherals
- Dedicated IOPORT for accessing the GPIOs

STM32G0 - RCC

life.augmented

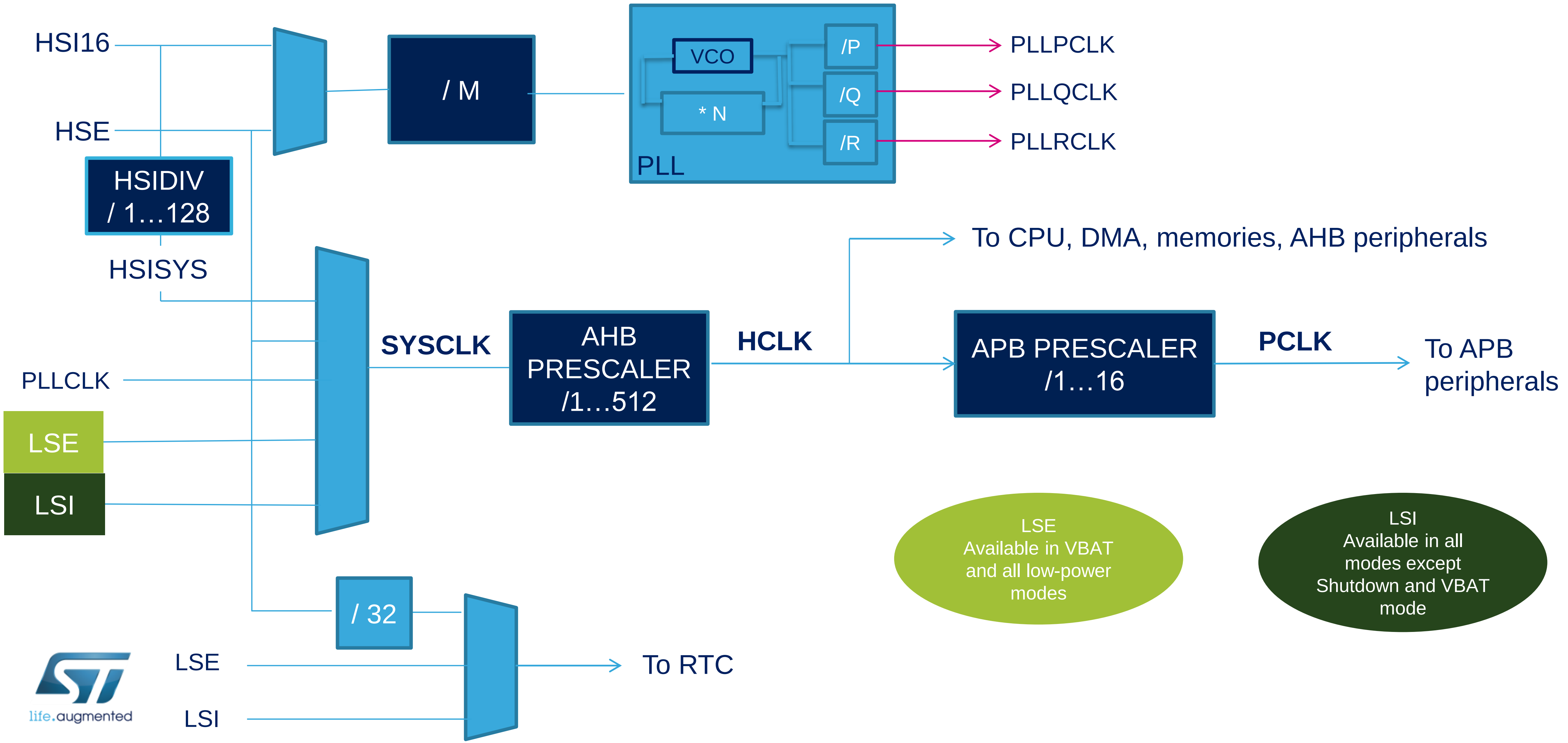
Main Differences with STM32F0

- The Reset and Clock Controller is similar to the one implemented in the STM32F0 family with some enhancements

	STM32F0	STM32G0
NRST	Input & output	GPIO, Input, Input & output
Reset Holder	No	Yes
PLL	One output	Three outputs
CSS on LSE + LSCO *	No	Yes
HSI divider to SYSCLK	No	Yes
Timer 1 & 15 running at 2xSYSCLK	No	Yes

* CSS on HSE was already available for both F0 and G0 products

Simplified clock tree

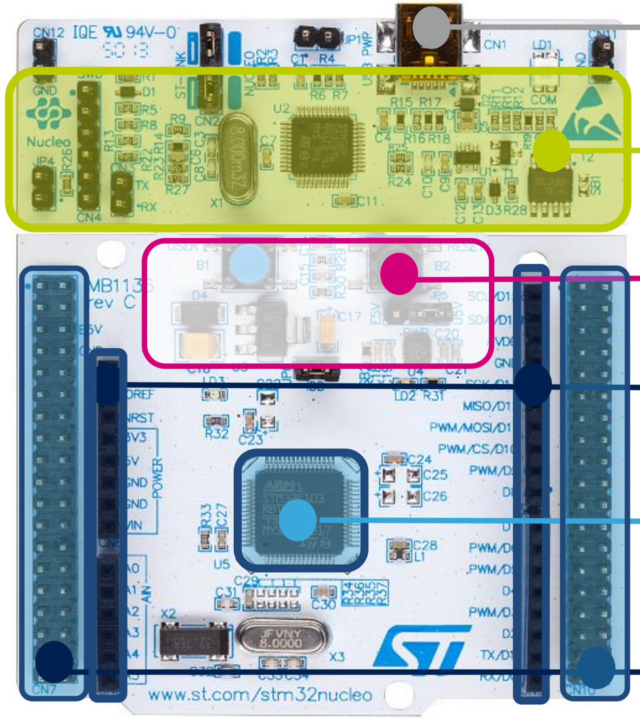


System clock

- Selected between HSI16, HSE, PLL, LSI, and LSE
- System clock, AHB and APB maximum frequency: 64 MHz

Voltage range	SYSCLK	HSI16	HSE	PLL
Range 1	64 MHz max.	16 MHz	48 MHz	VCO max = 344 MHz
Range 2	16 MHz max.	16 MHz	16 MHz	VCO max = 128 MHz
Low-power run/sleep	2 MHz max.	Allowed with divider	Not allowed	Not allowed

STM32 NUCLEO features



Flexible board power supply : through USB or external source

Integrated ST-Link/V2-1: mass storage device flash programming

2 push buttons, 2 color Leds

Arduino extension connectors : easy access to add-ons

One STM32 MCU flavor with 64 pins

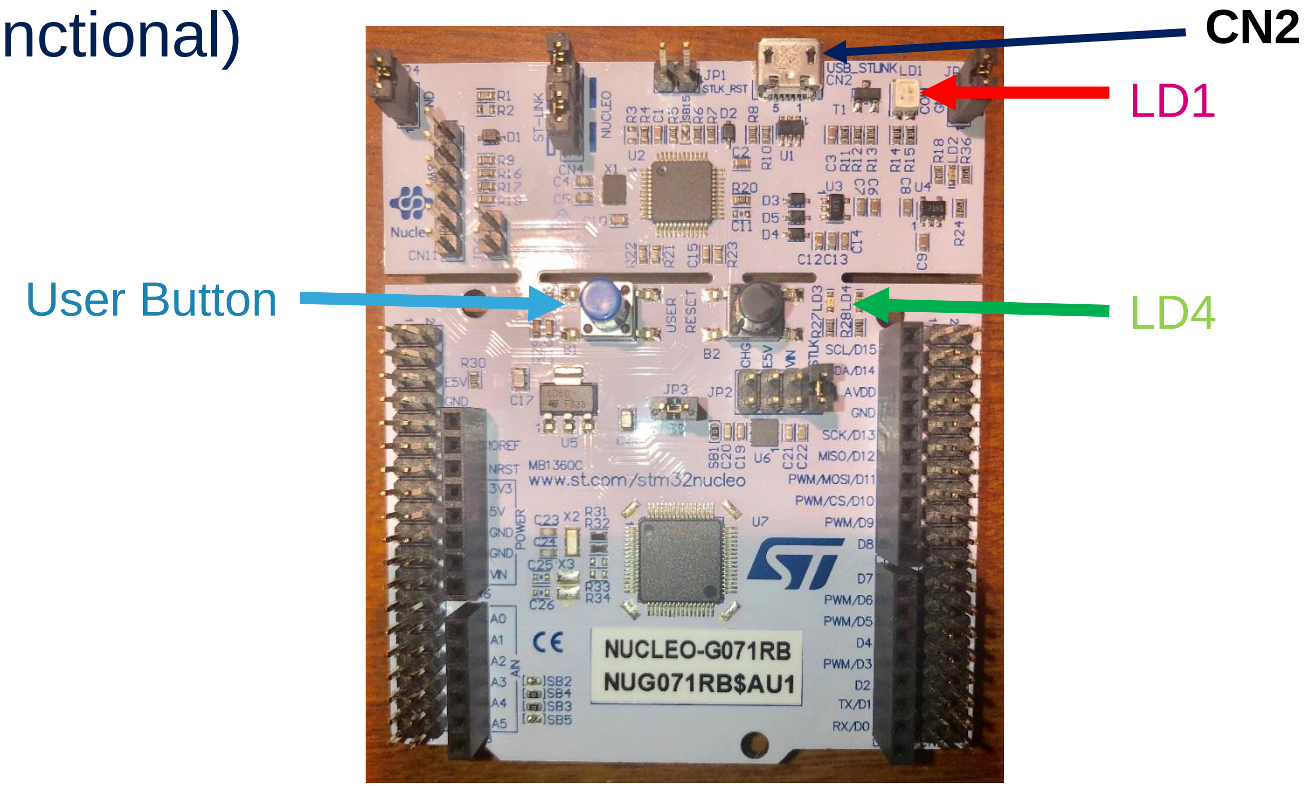
Morpho extension headers : direct access to all MCU I/Os

We are now going to provide you with the STM32G0 Nucleo board.

Board Check

- Connect USB ST-LINK (**CN2**) to your PC
  - ST-LINK driver may be installed if this is the first time the board is plugged in.

- LD1 should be ON and solid RED (indicating board power available and ST-Link is functional)



Note about Virtual Box or VMWare or Parallels users: you need to give USB resources (ST-LINK in our case) from Native OS to Windows environment.

Lab: Saving the original code

life.augmented

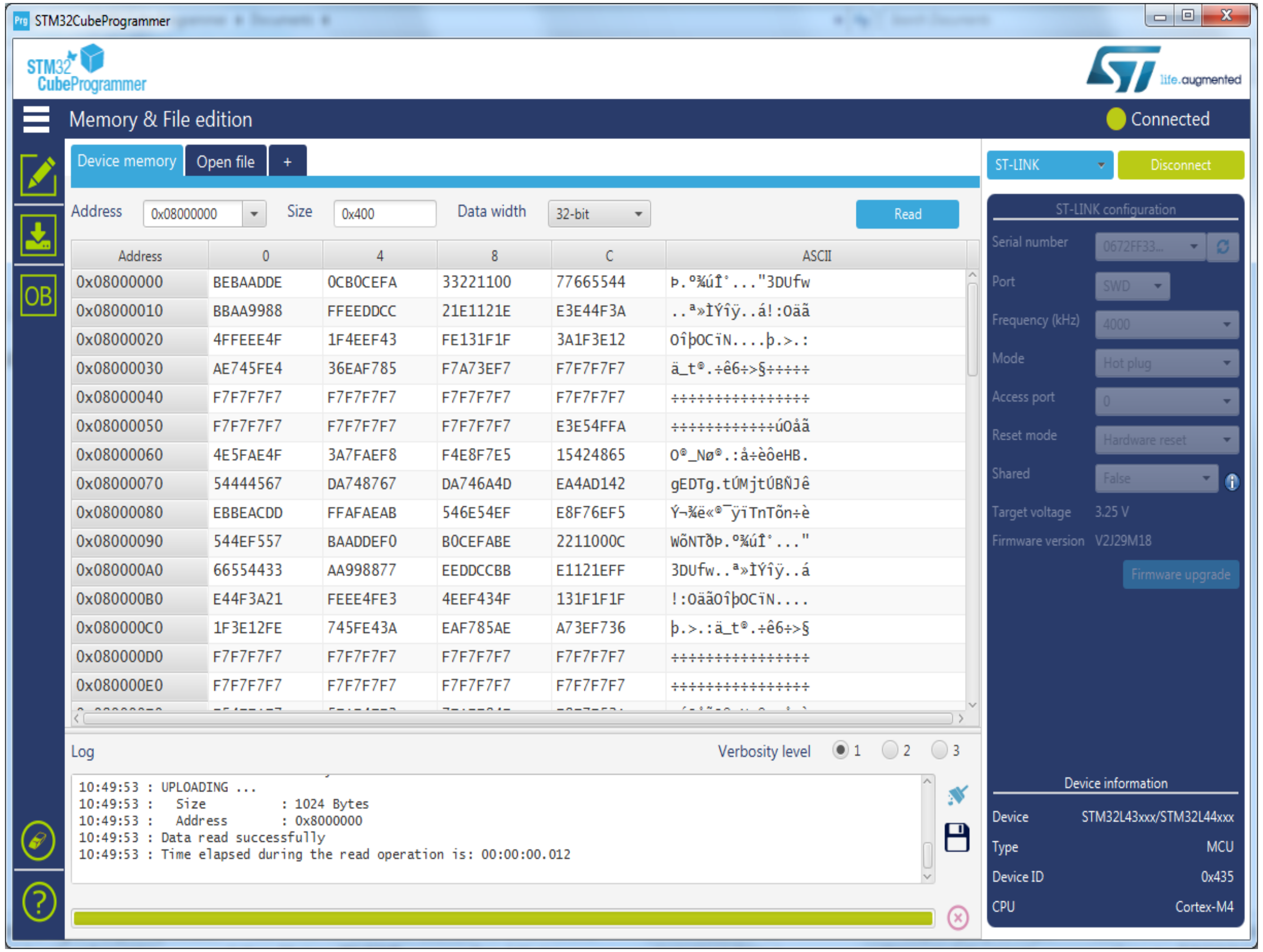
Lab : Saving the original code

Objective:

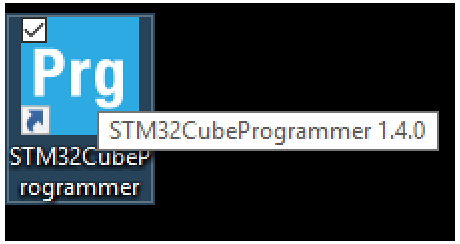
- The objective of this lab is to save the “out-of-the-box” firmware demonstration code using our stand-alone programmer: STM32CubeProgrammer

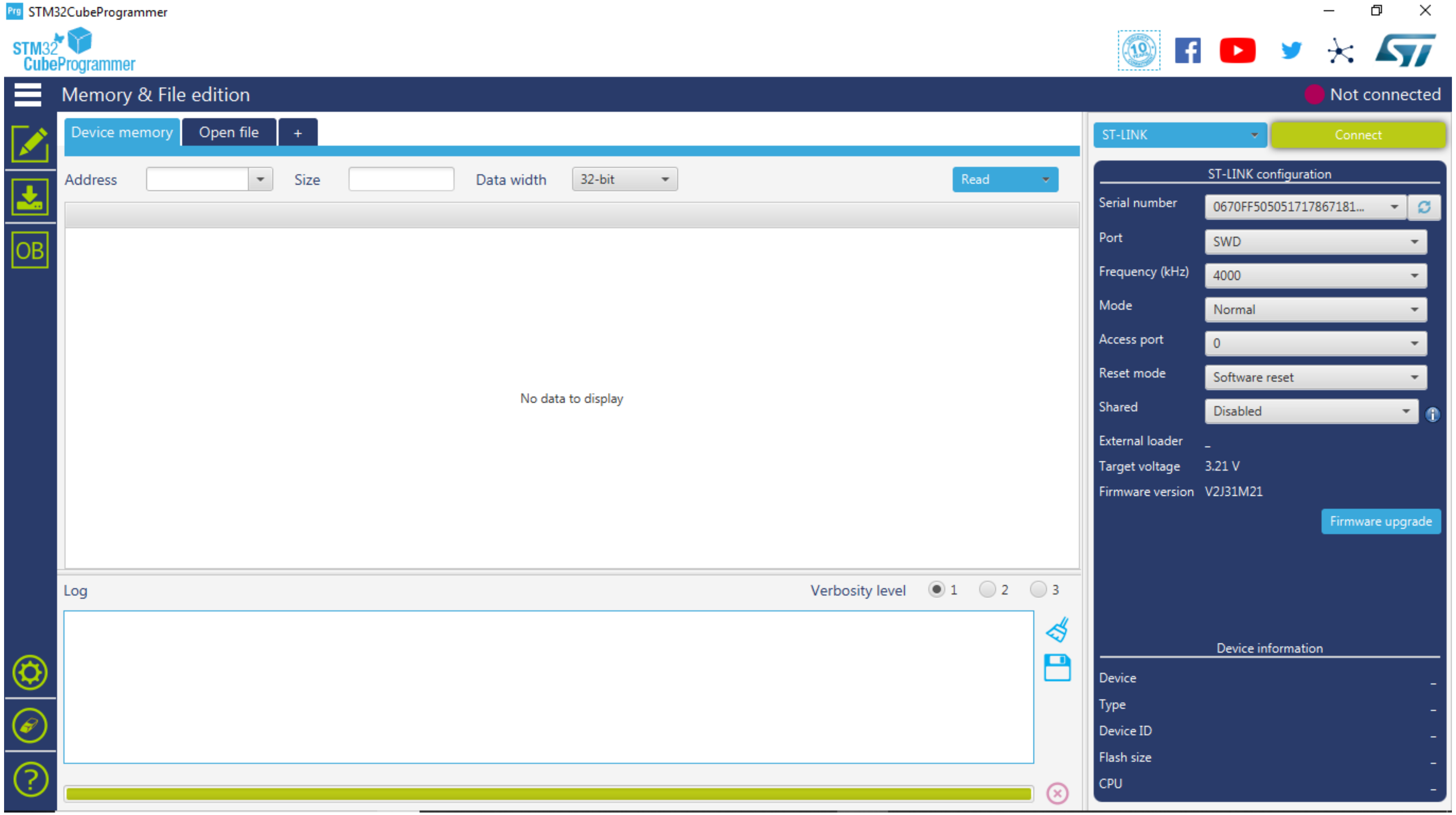
STM32CubeProgrammer features

- **Main Features:**
  - Unify existing programming software tools : Merge STVP, ST-Link Utility and Bootloader softwares tools in one solution.
  - Multiplatform (Windows, Linux and macOS)
  - Debug and bootloader interfaces support: ST-LINK debug probe (JTAG/SWD), Bootloader interfaces (UART, USB DFU, SPI, I²C and CAN)
  - Secure programming



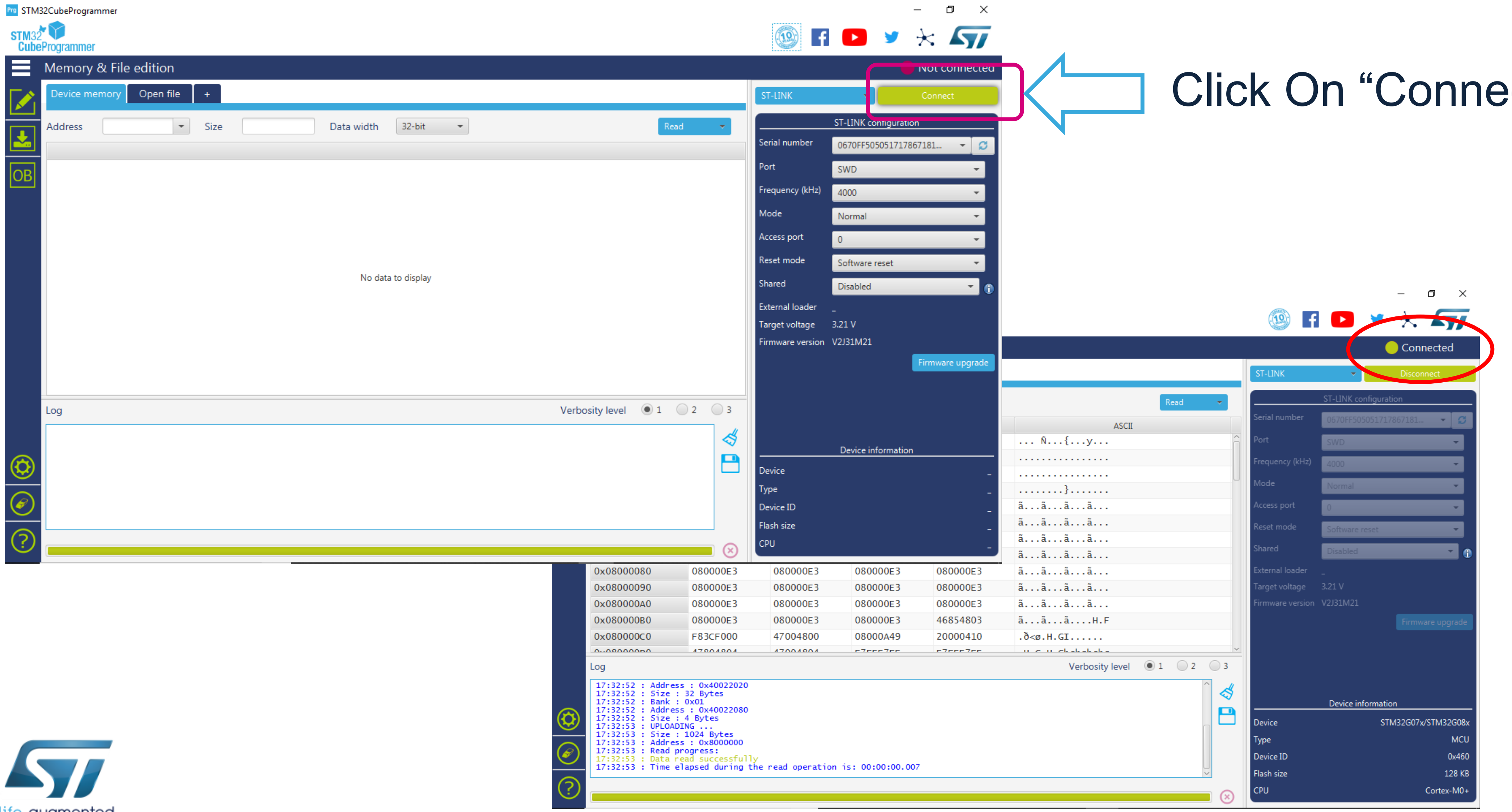
Run STM32CubeProgrammer





life.augmented

Connect to the ST-LINK

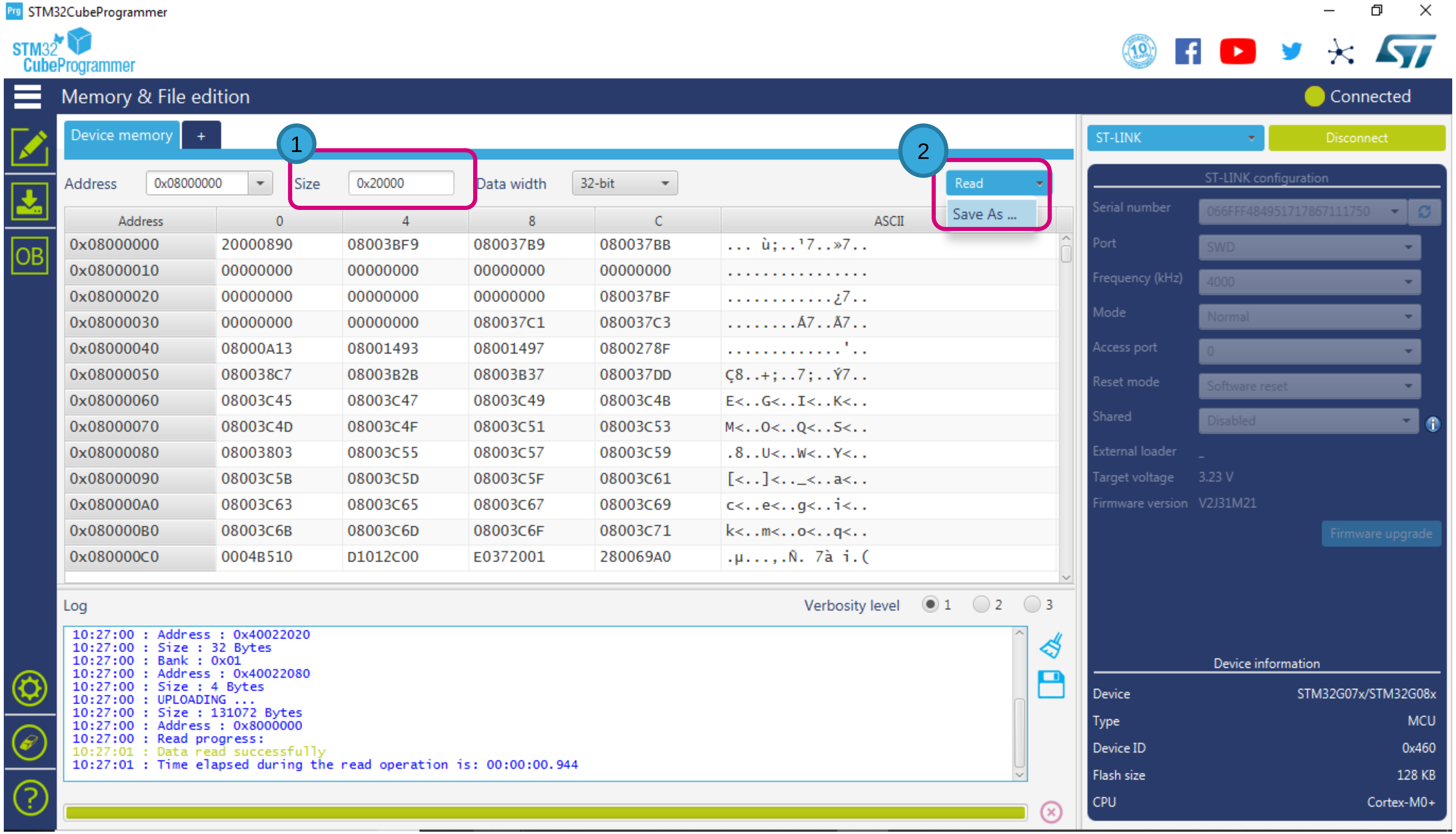


Click On “Connect”

Save the content of the flash

- Change the size to: 0x20000 1
  - Equivalent to 128K which is the size of the Flash of the STM32G0 on the Nucleo board

- Then click: 2
  Read -> Save As…

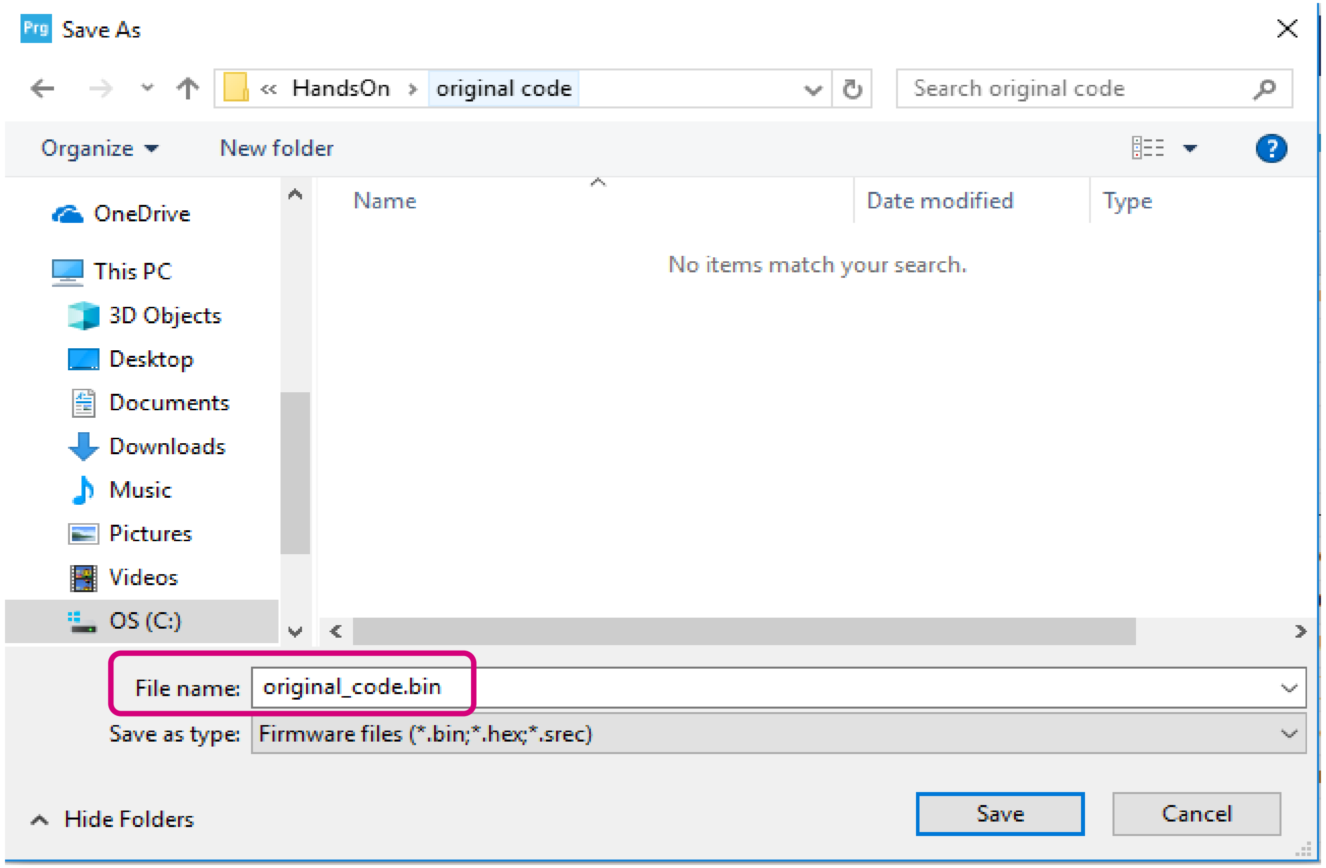


Save the content of the flash

- Save it to a location you will remember because we will restore it at the end of the workshop.

- Save as a binary file (*.bin)

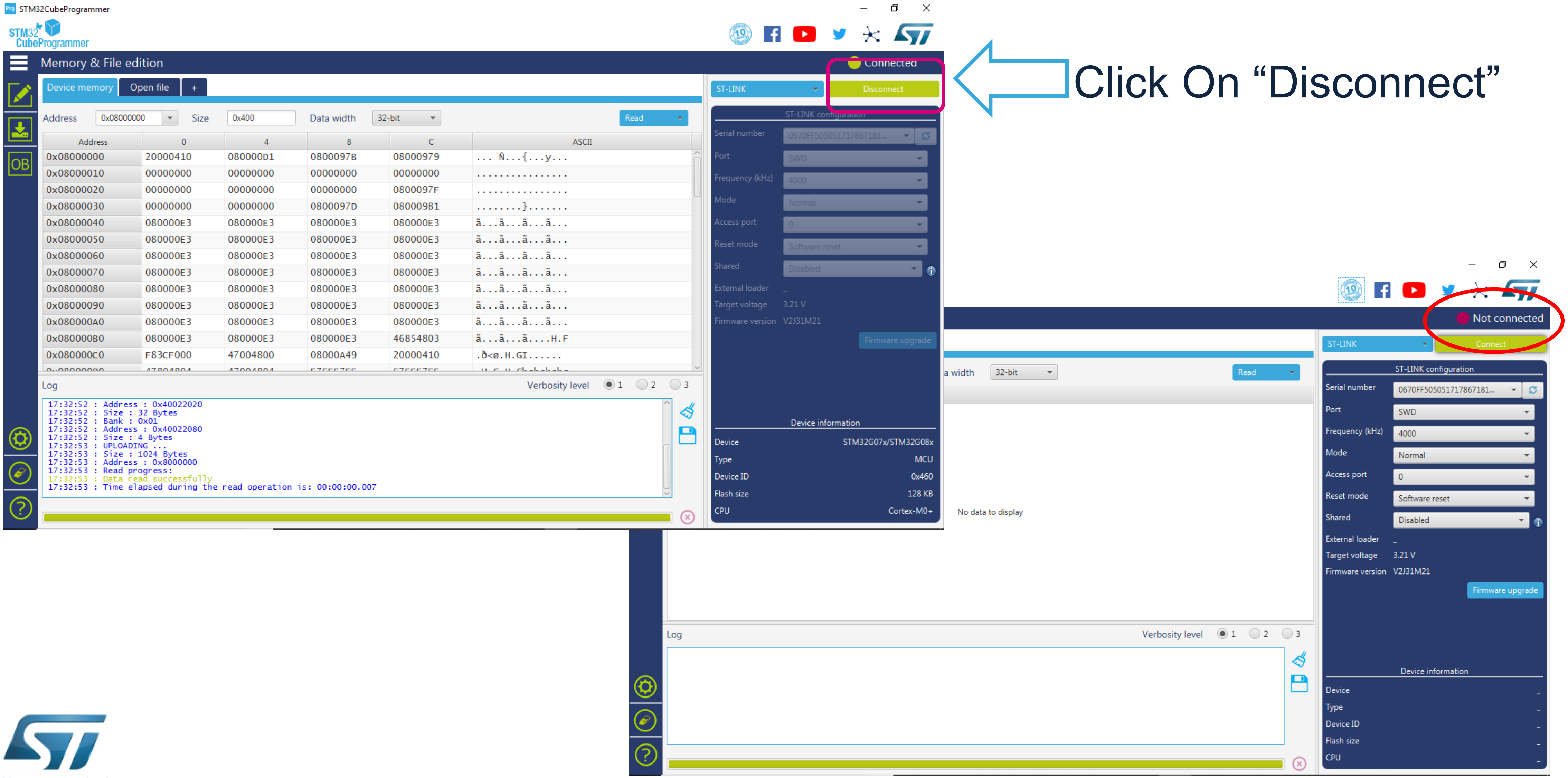
  For example: C:\STM32G0WorkShop\HandsOn\original code**orginal_code.bin**



life.augmented

Disconnect to the ST-LINK

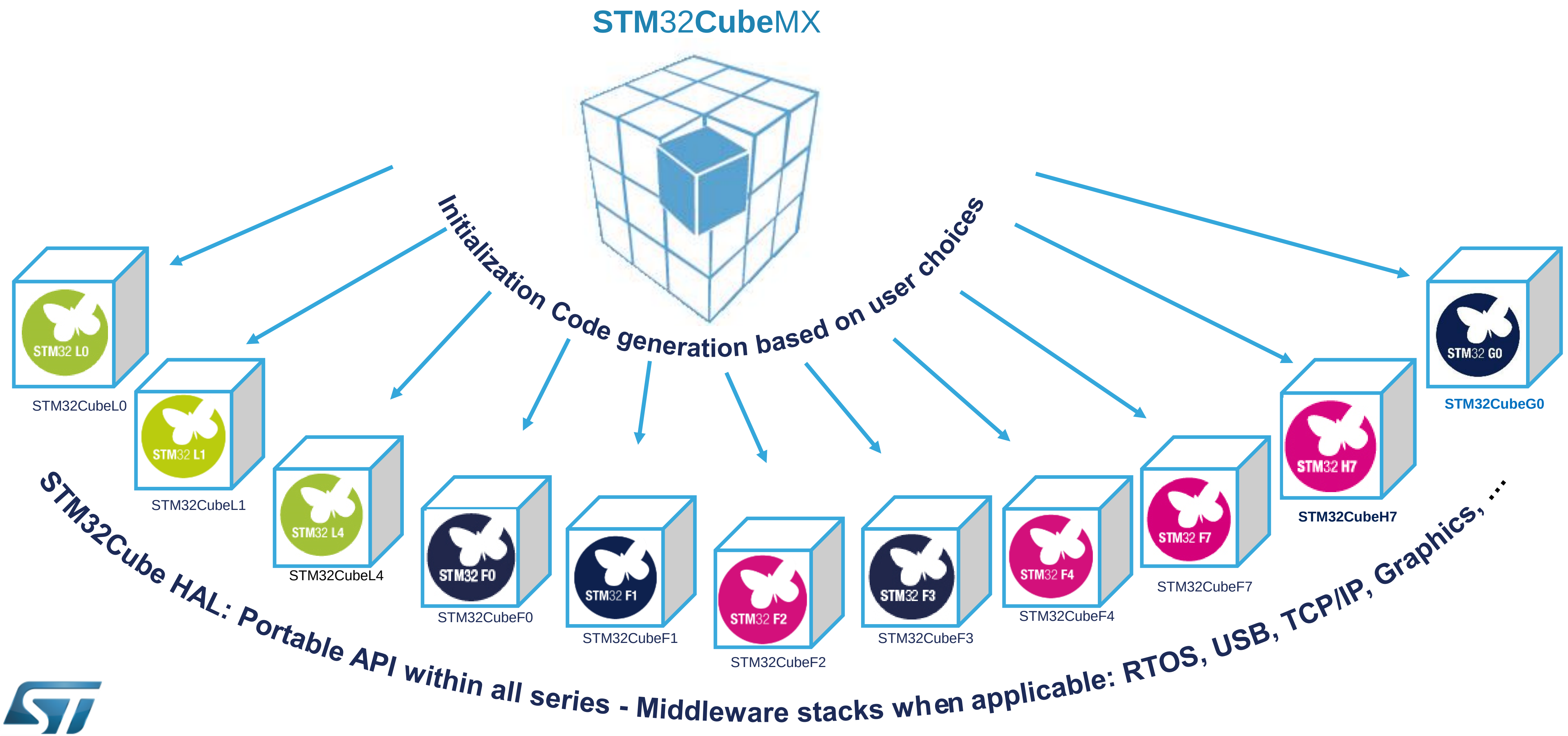
Click On “Disconnect”



life.augmented

STM32Cube and STM32CubeMX

life.augmented

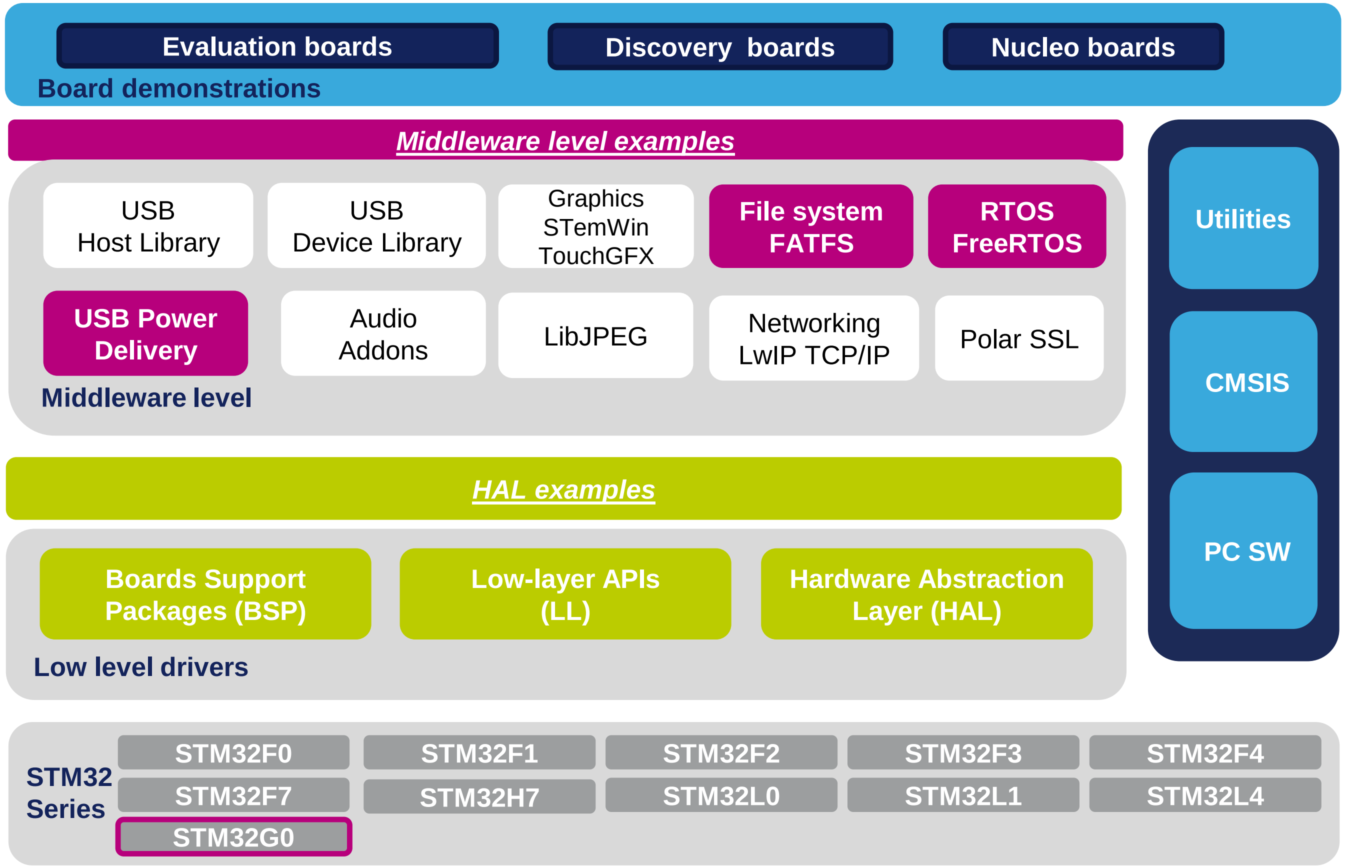


STM32CubeMX
Initialization Code generation based on user choices
STM32 L0
STM32CubeL0
STM32 L1
STM32CubeL1
STM32 L4
STM32CubeL4
STM32 F0
STM32CubeF0
STM32 F1
STM32CubeF1
STM32 F2
STM32CubeF2
STM32 F3
STM32CubeF3
STM32 F4
STM32CubeF4
STM32 F7
STM32CubeF7
STM32 H7
STM32CubeH7
STM32 G0
STM32CubeG0
STM32Cube HAL: Portable API within all series - Middleware stacks when applicable: RTOS, USB, TCP/IP, Graphics, ...

STM32Cube

life.augmented

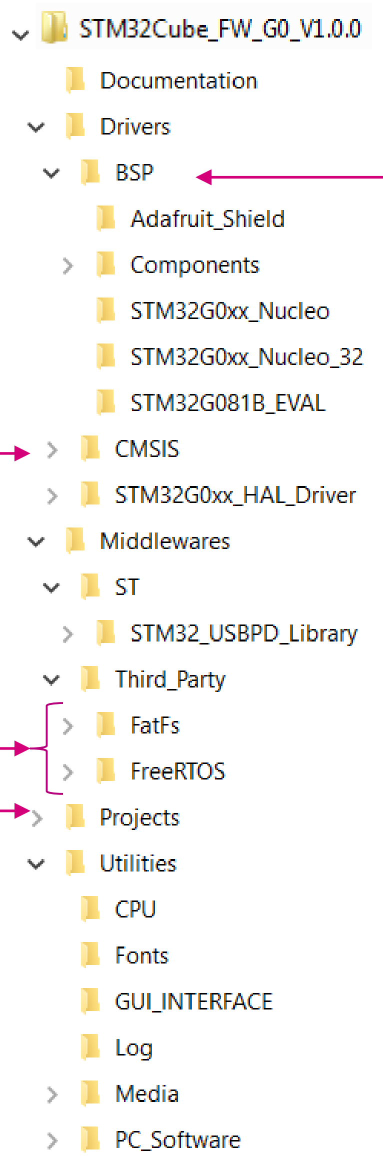
Overview



Application benefits

- Single package
- Compatible with all STM32 series
- Source code with open-source BSD license

Package organization



BSP drivers for the supported boards

Contains STM32G0xx CMSIS files that defines Peripheral's registers declarations, bits definition and the address mapping

STM32G0xx HAL drivers for all modules

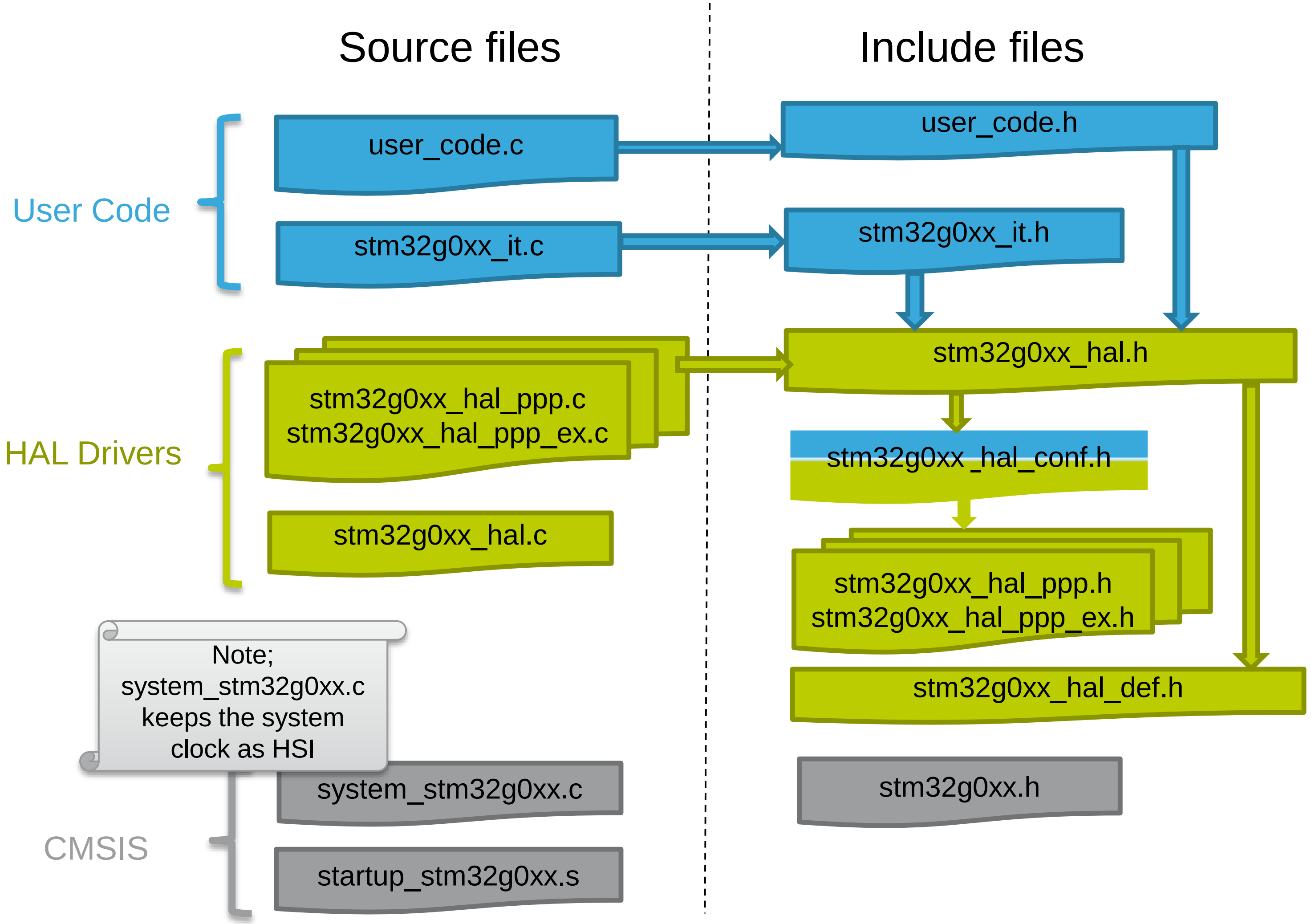
USB Power Delivery stack

Third party middleware:

- FAT File System and FreeRTOS

Set of examples/ applications organized by board provided with preconfigured projects.

HAL general concepts
HAL based project organization



HAL general concepts
HAL drivers file

File	Description
stm32g0xx_hal_ppp.c/.h	Peripheral driver with cross family portable APIs
stm32g0xx_hal_ppp_ex.c/.h	Extended peripheral features APIs
stm32g0xx_hal.c	HAL global APIs (HAL_Init, HAL_DeInit, HAL_Delay,…)
stm32g0xx_hal.h	HAL header file, it should be included in user code
stm32g0xx_hal_conf.h	Config file for HAL, should be customized by user to select the peripherals to be included
stm32g0xx_hal_def.h	Contains HAL common type definitions and macros

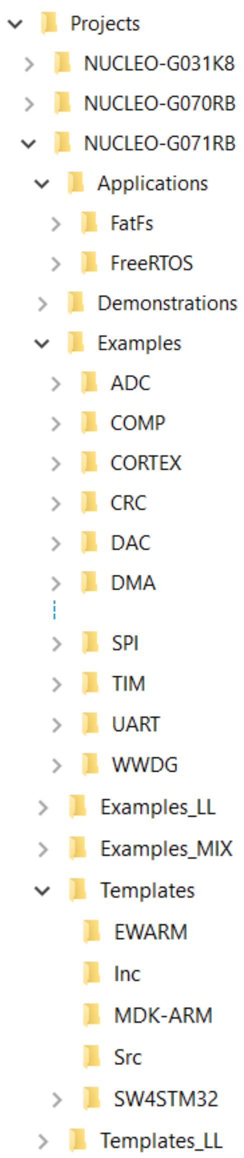
Key features

Layer	Category	Provided embedded software	Provided examples
HAL	Analog	Analog/Digital conversion, …	~66 examples on ST evaluation boards* !
	Timers	Timers, RTC, Watchdogs, …	
	Connectivity	I2C, USART, SPI, I2S, SDMMC, CEC, …	
Middleware	RTOS	FreeRTOS open source RTOS, with CMSIS-RTOS wrapper	~10 applications on ST evaluation boards* !
	USB Power Delivery	PD stack, device policy manager, policy engine, protocol layer	
	File System	FatFS open-source file system	
Application	Demonstration	Full demonstrations for ST boards	~1 demonstration project for ST boards!

*: STM32G0 Series

Examples overview (1/2)

- For each board, a set of examples is provided with preconfigured projects for EWARM, MDK-ARM and SW4STM32 toolchains
  - This figure shows the projects structure for the NUCLEO-G071RB board, which is identical for other boards
  - The examples are classified depending on the STM32Cube level they apply to, and are named as follows:
    - Examples in Level 0 are called **Examples**, and use HAL drivers without any middleware component
    - Examples in Level 1 are called **Applications**, and provide typical use cases of each middleware component
    - Examples in Level 2 are called **Demonstration**, and implement all the HAL, BSP and middleware components



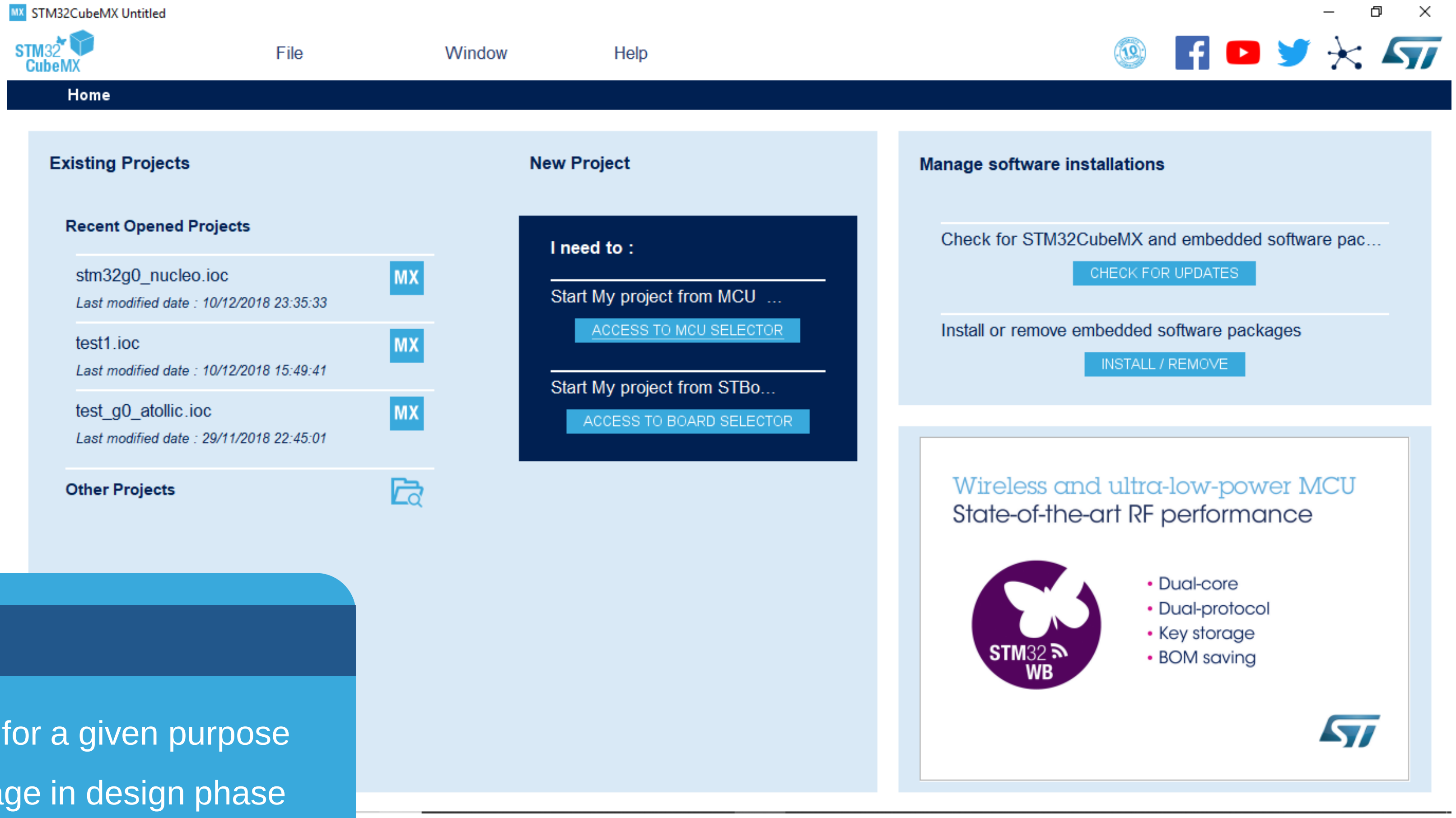
Examples overview (2/2)

- The Template project is provided to build quickly any firmware application for all supported boards
- All examples have the same structure,
  - \Inc folder contains all header files
  - \Src folder for the source code
  - \EWARM, \MDK-ARM and \SW4STM32 contain the preconfigured project for each toolchain
  - readme.txt describes example behavior and the environment needed to make it work
  - *.ioc file that allows user to open most of FW examples within STM32CubeMX (starting from STM32CubeMX 5.0)

STM32CubeMX

life.augmented

- Choose ideal MCU and simply configure
  - Pinouts
  - Clocks and oscillators
  - Peripherals
  - Low-power modes
  - Middleware

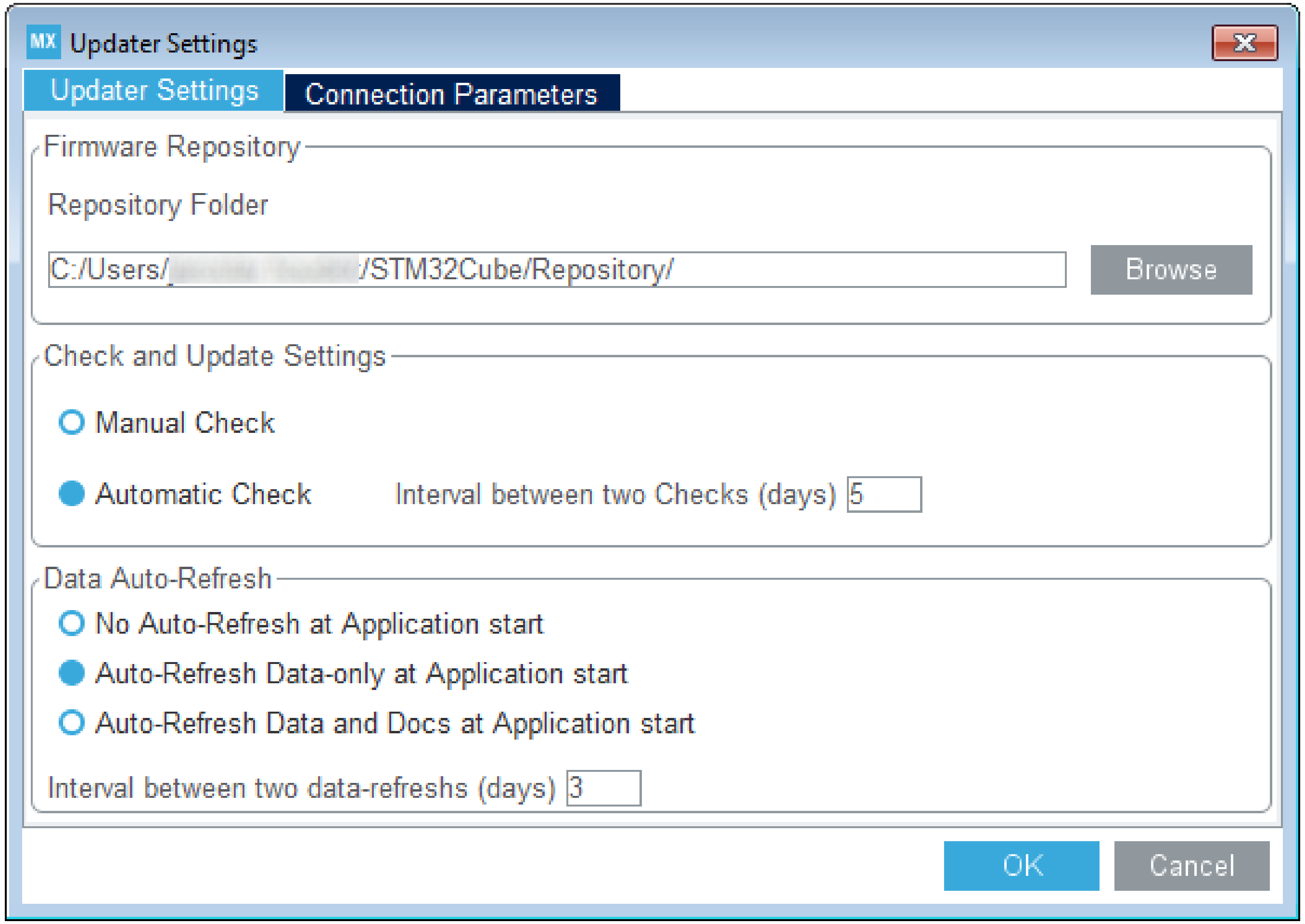


Application benefits

- Helps choose the correct MCU for a given purpose
- Simulation provides an advantage in design phase
- Boosts development speed with a headstart

Prerequisites and settings

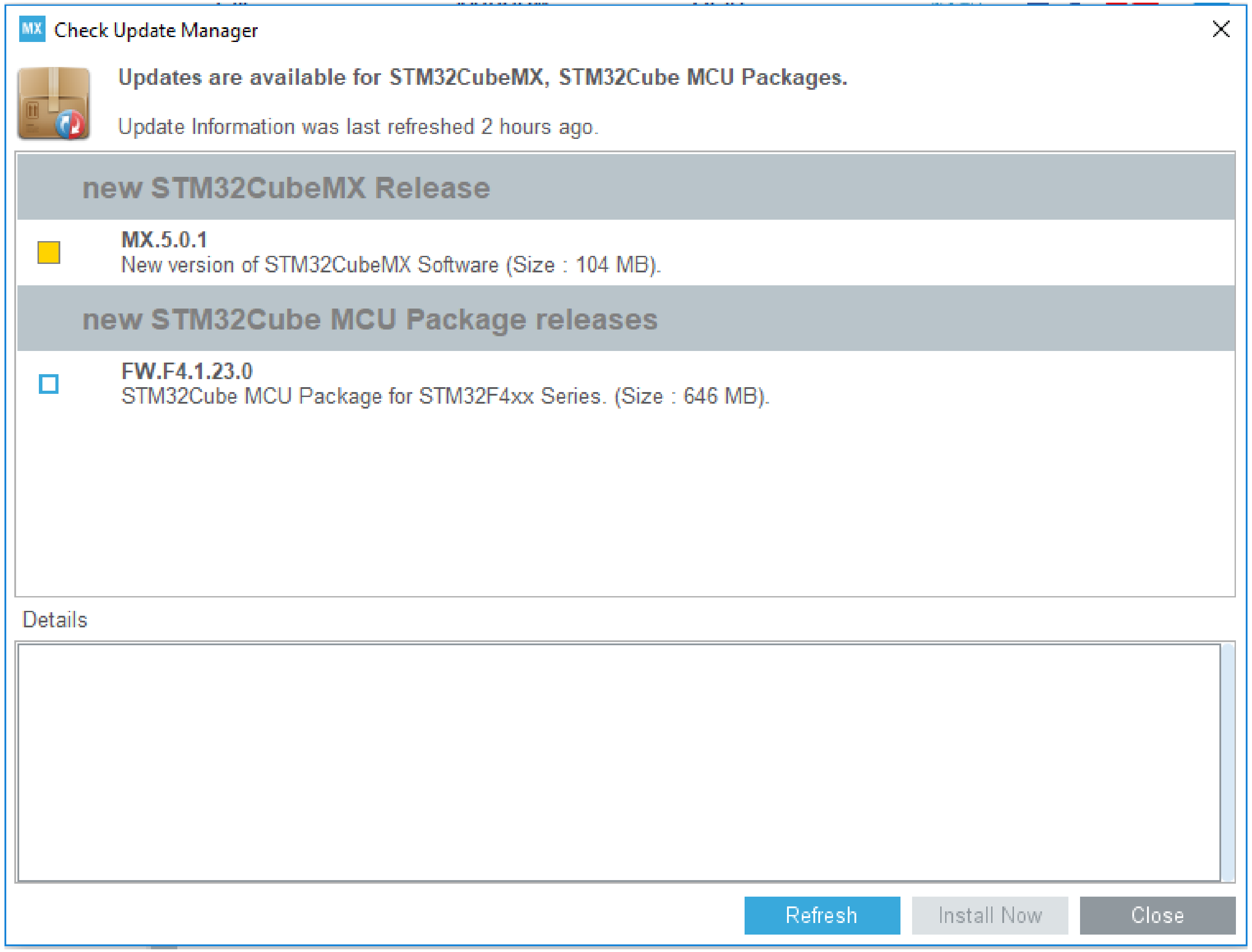
- STM32CubeMX needs Java RE
  - Check release notes of the particular version for additional requirements
  - Multiplatform tool runs on Windows, Linux and macOS
- After installation, hit Alt+S to configure the updater – not only for the GUI but also for Cube FW libraries
- Select SW library placement.



life.augmented

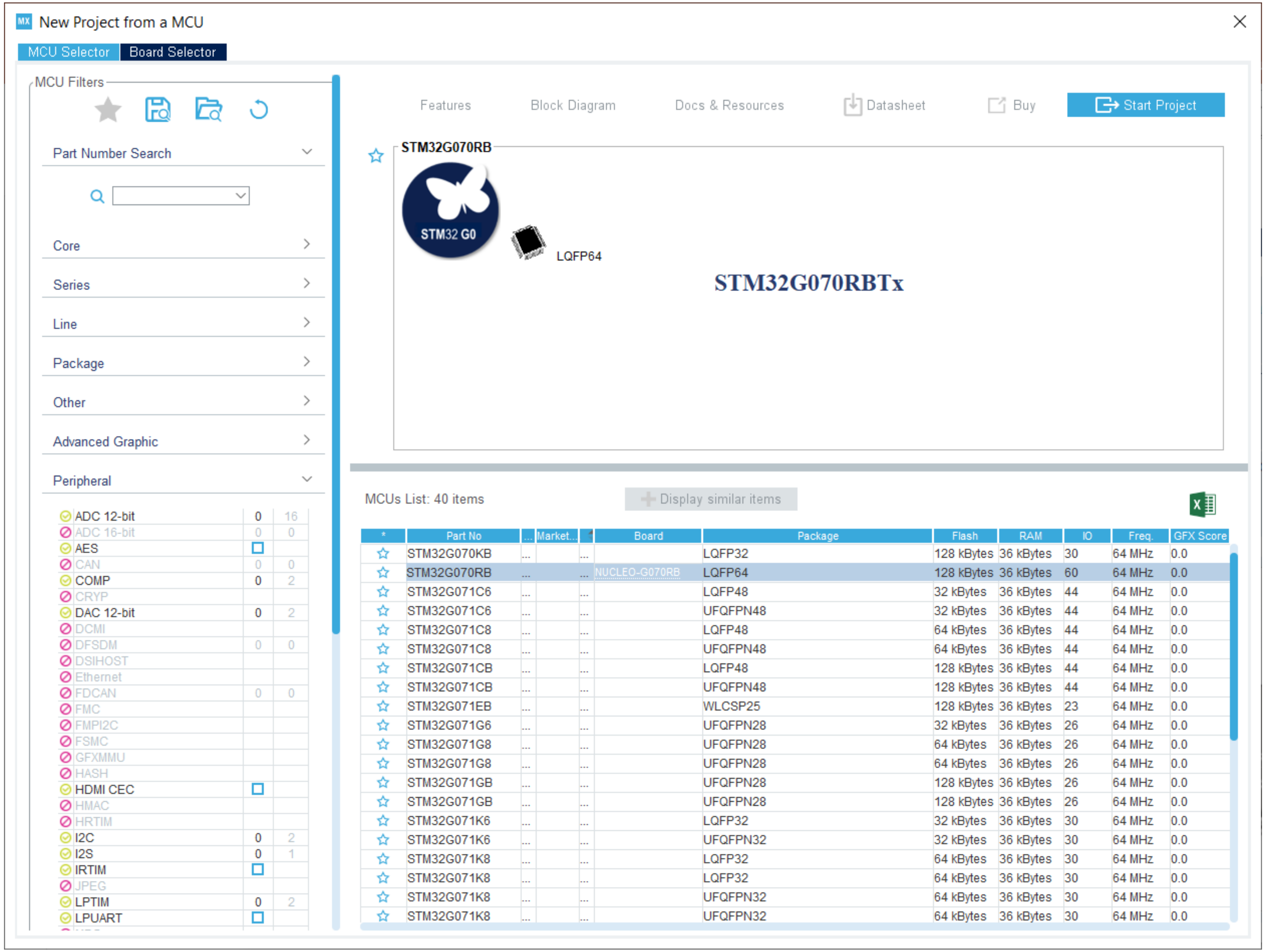
Updater

- Updates are accessible from the Help menu
- The tool updater can detect new releases of the tool and the associated Cube library
- Use the libraries manager to download new library packages



MCU selector

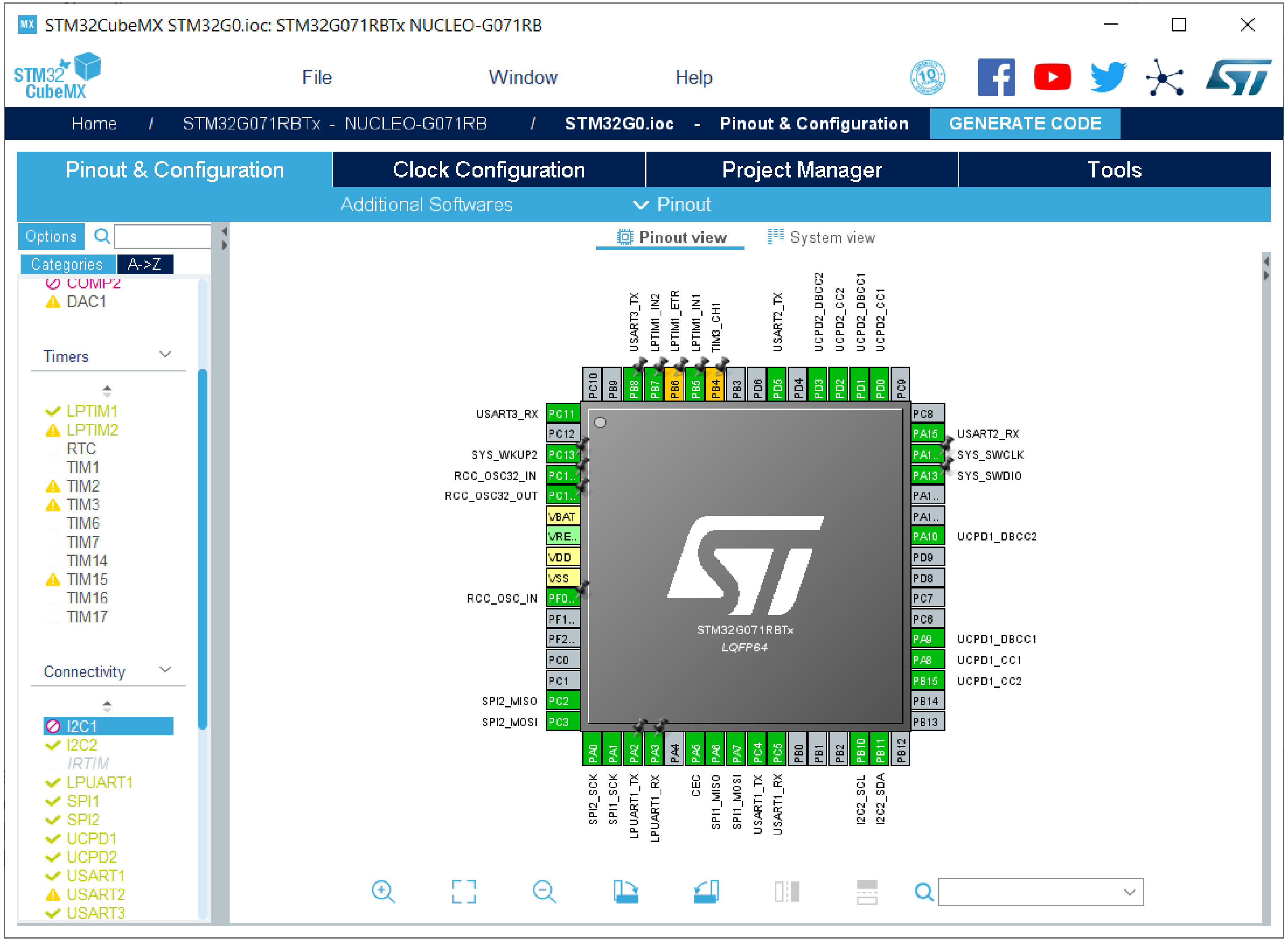
- Find MCU by name …
    - Quickly locate by Series and Lines
- … or application needs
    - Package (pin count)
    - RAM size
    - NV memory requirements
    - Embedded peripherals
    - Number and type of interfaces
    - Core and frequency
    - Price
- Convenient links to documentation
- Export table to Excel file



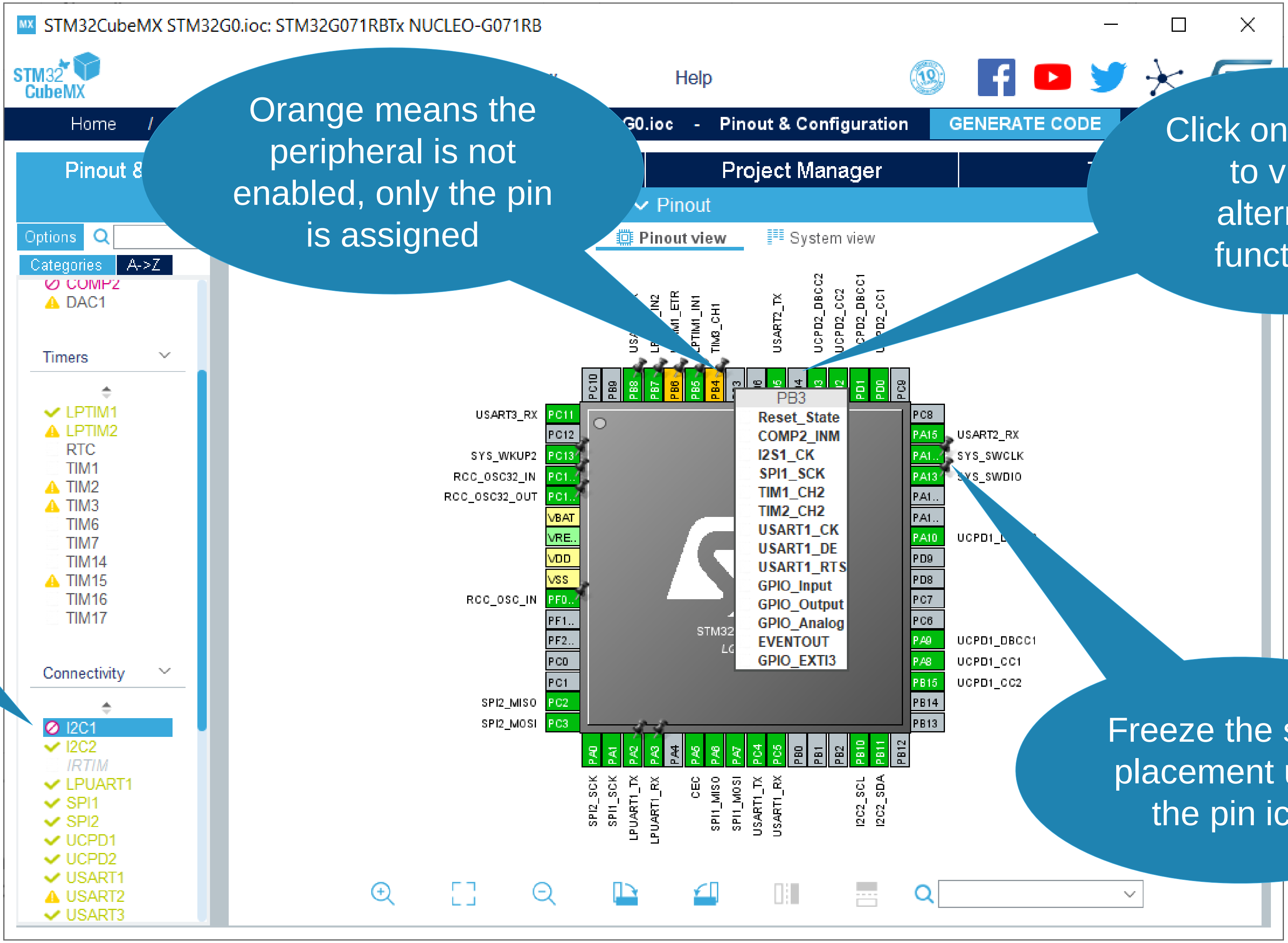
life.augmented

Pin assignment

- Pinout from:
  - Peripheral tree
  - Manually
- Automatic signal remapping
- Management of dependencies between peripherals and/or middleware (FatFS, USB …)



Pin assignment continued



Orange means the peripheral is not enabled, only the pin is assigned

Click on the pin to view alternate functions

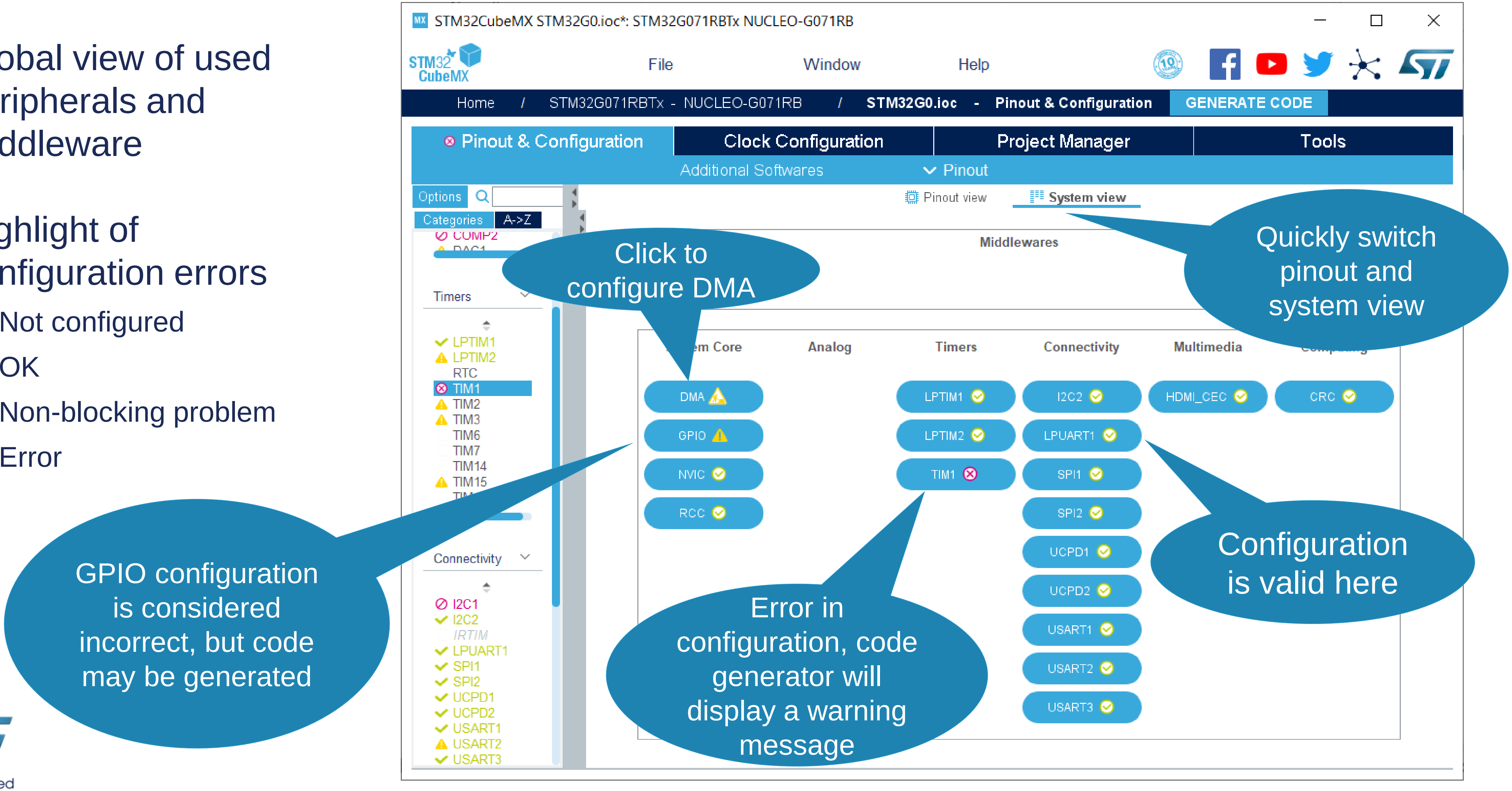
Peripheral is not available, all its alternate pins are assigned elsewhere

Freeze the signal placement using the pin icon

life.augmented

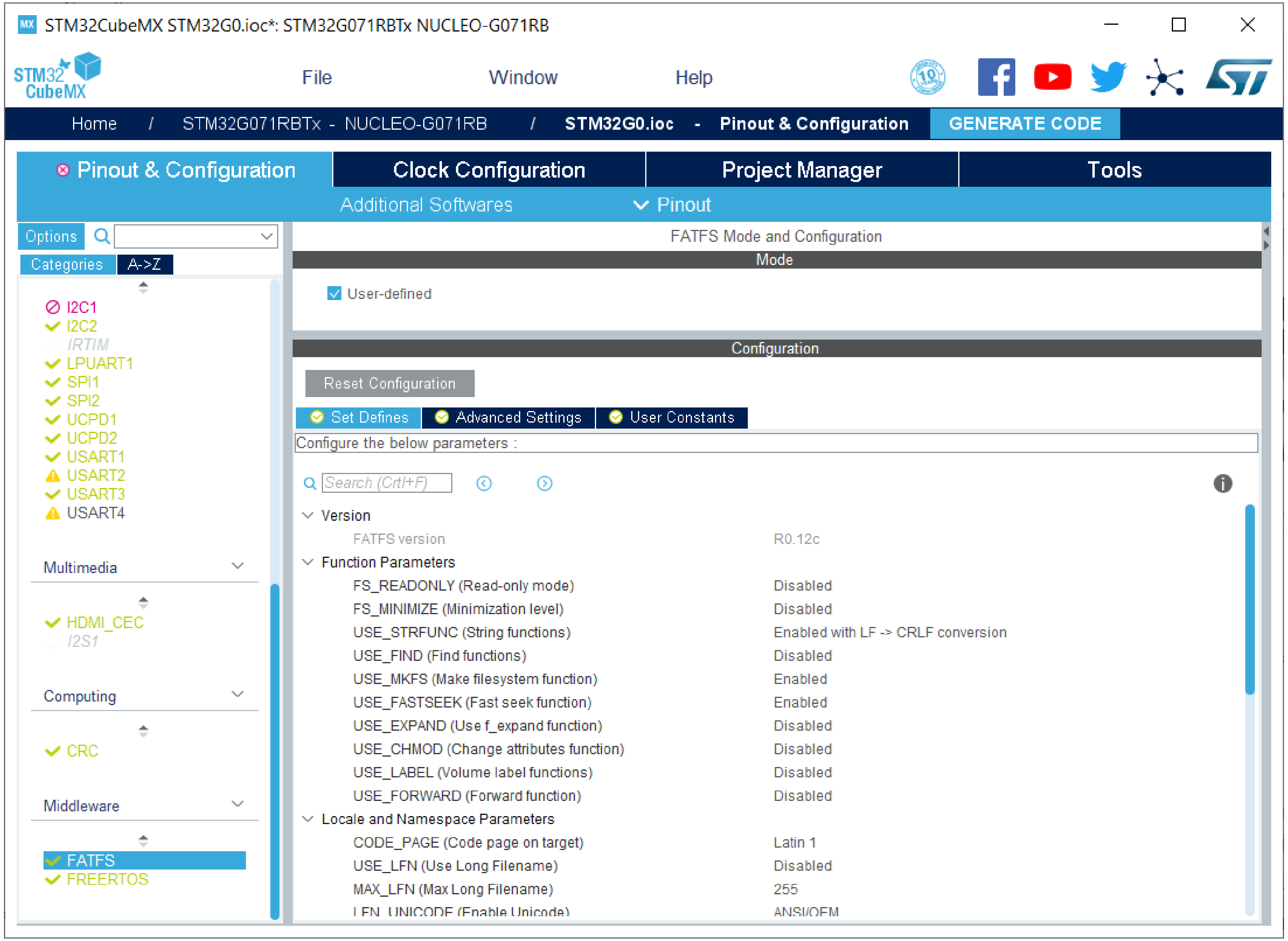
Peripheral and Middleware configuration

- Global view of used peripherals and middleware
- Highlight of configuration errors
  - + Not configured
  - √ OK
  - ⚠ Non-blocking problem
  - x Error



Middleware configuration

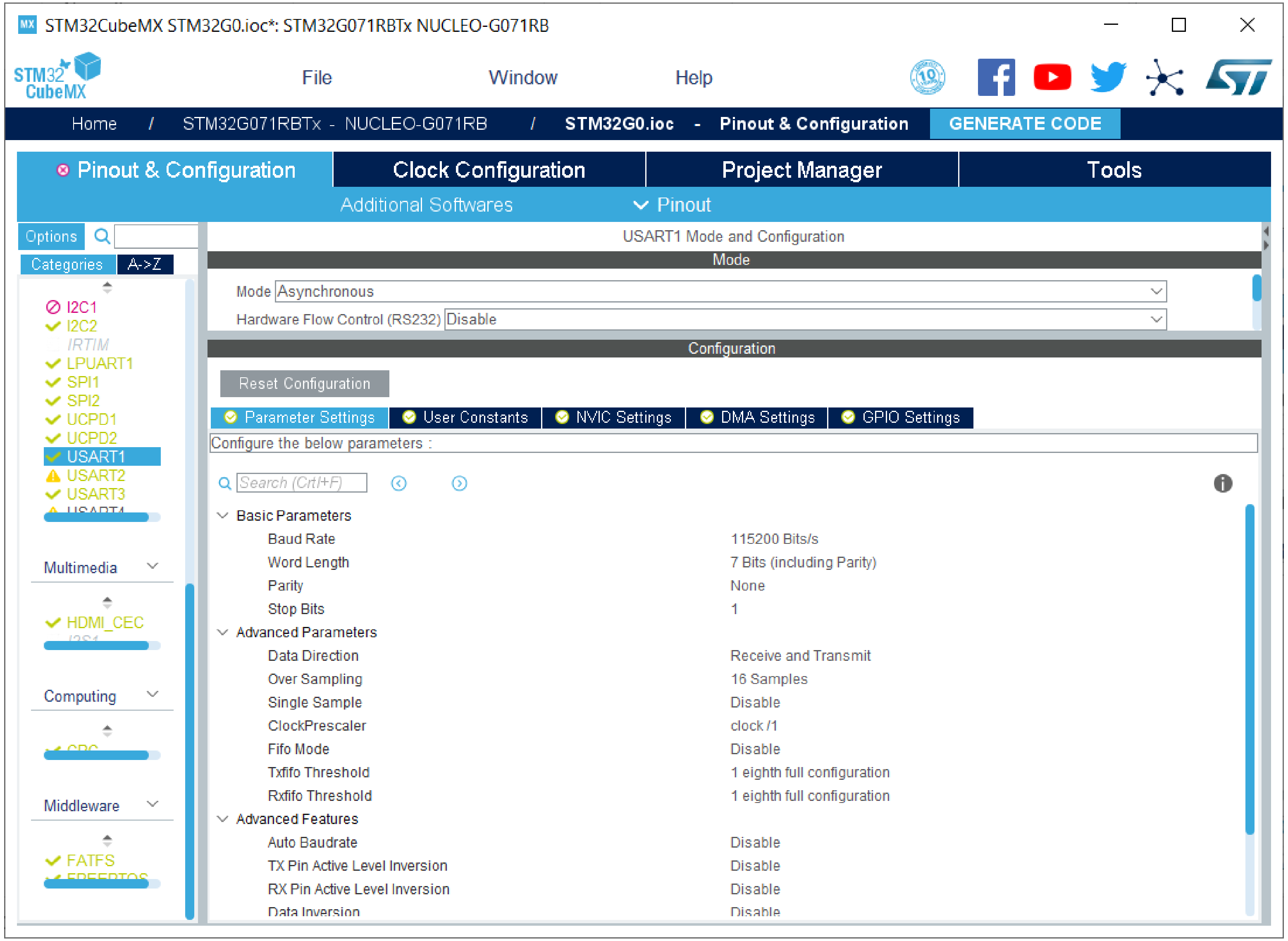
- Presents options specific to each supported software component
- All settings are organized in logical groups
- Description and constraints are available for quick reference



life.augmented

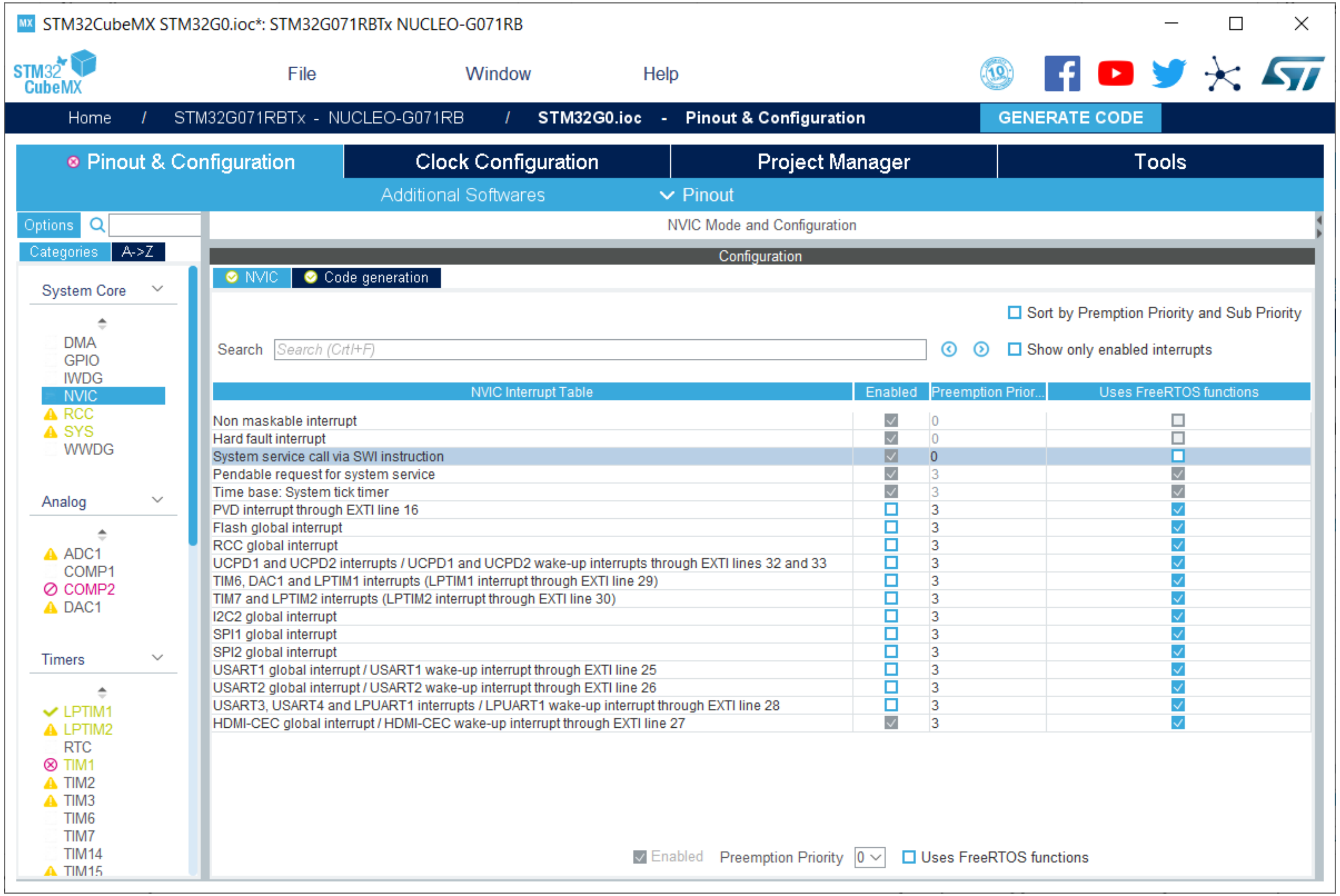
Peripheral configuration

- All available initialization parameters are presented with short description and options
- Interrupt may be assigned to peripherals
- DMA may be associated, where applicable
- GPIO settings for peripherals with input and/or output



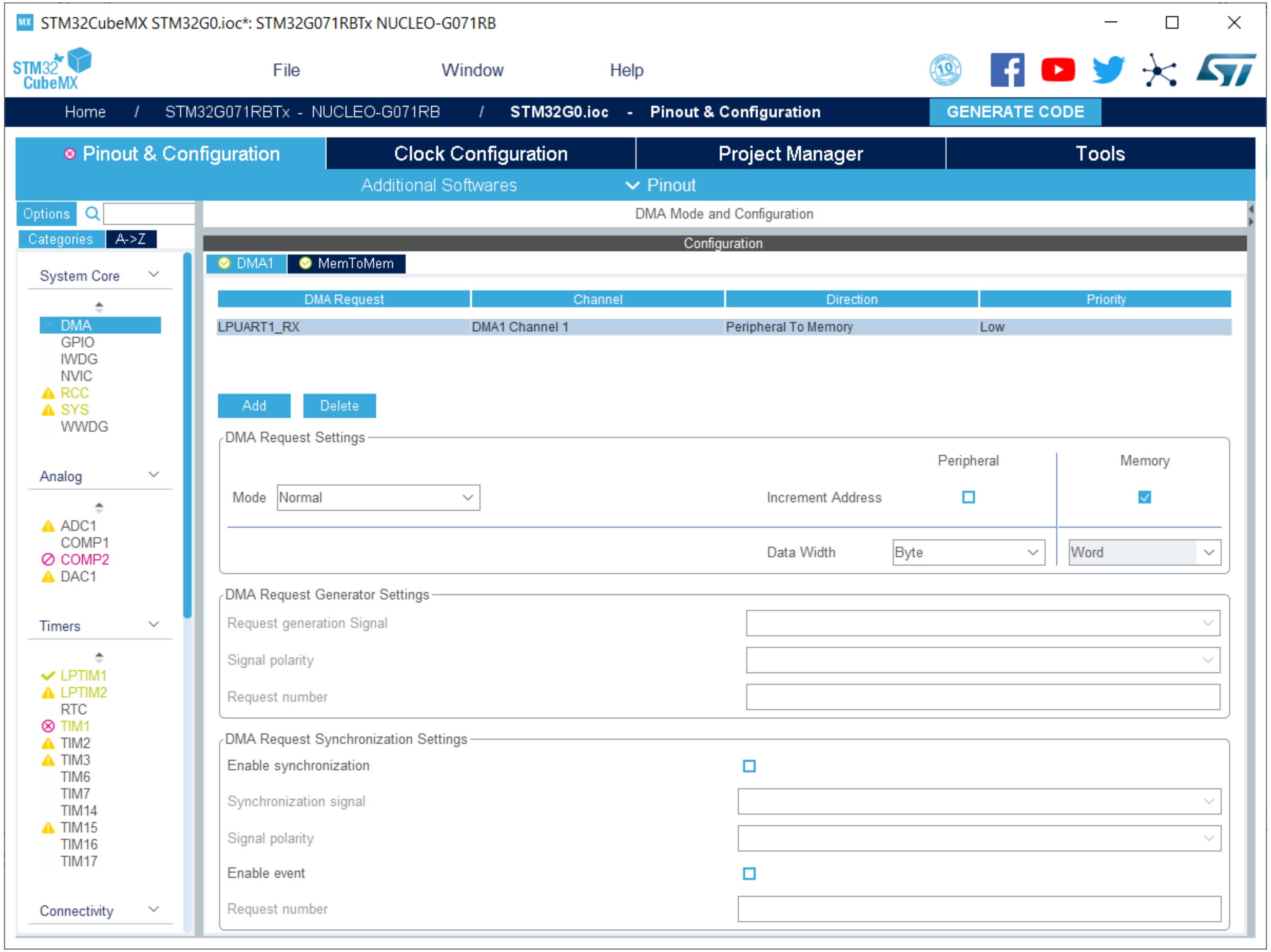
NVIC configuration panel

- Single control panel for all interrupts
- Manage priorities and sub-priorities
- Searching, filtering and sorting interrupts in the list
- Code generation tab allows to customize interrupt initialization



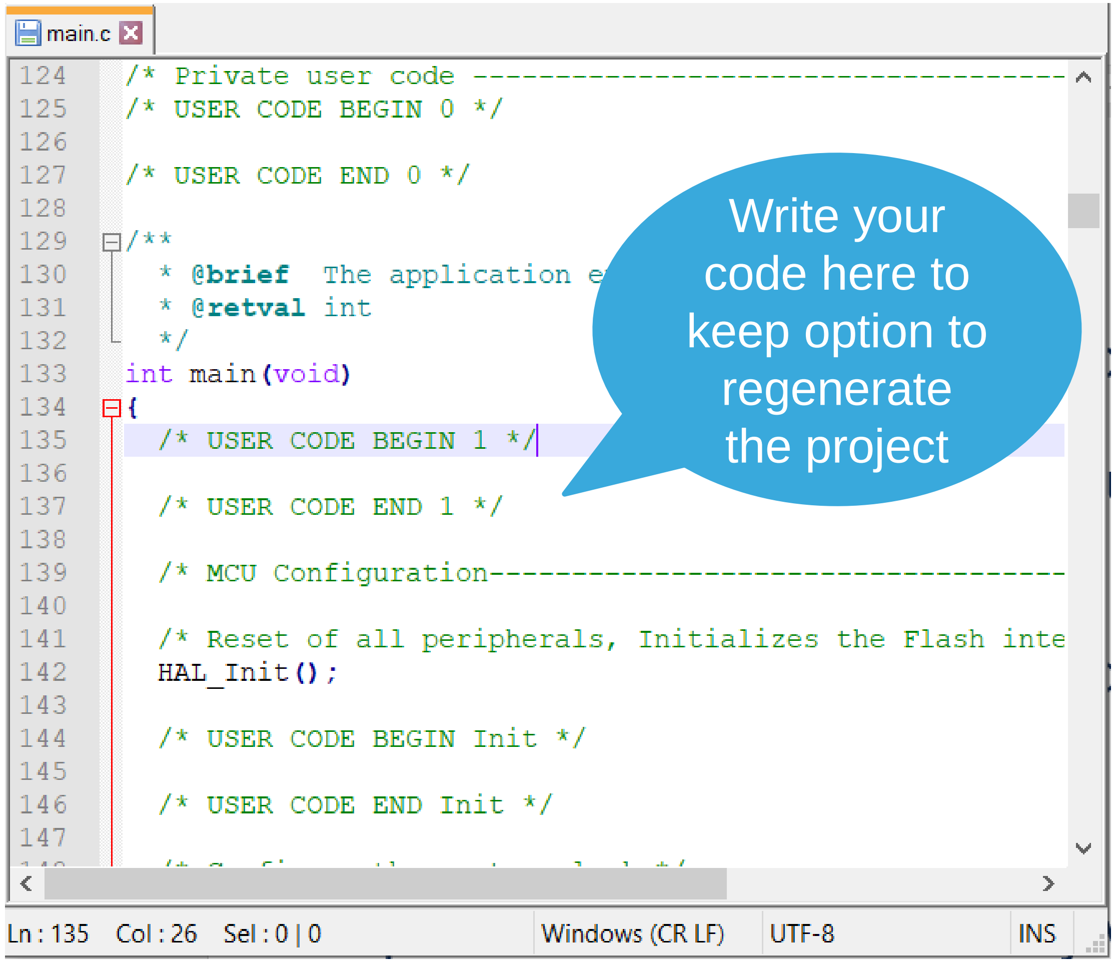
DMA configuration panel

- Manages all DMA requests including memory to memory
- Configure direction, priority and other settings



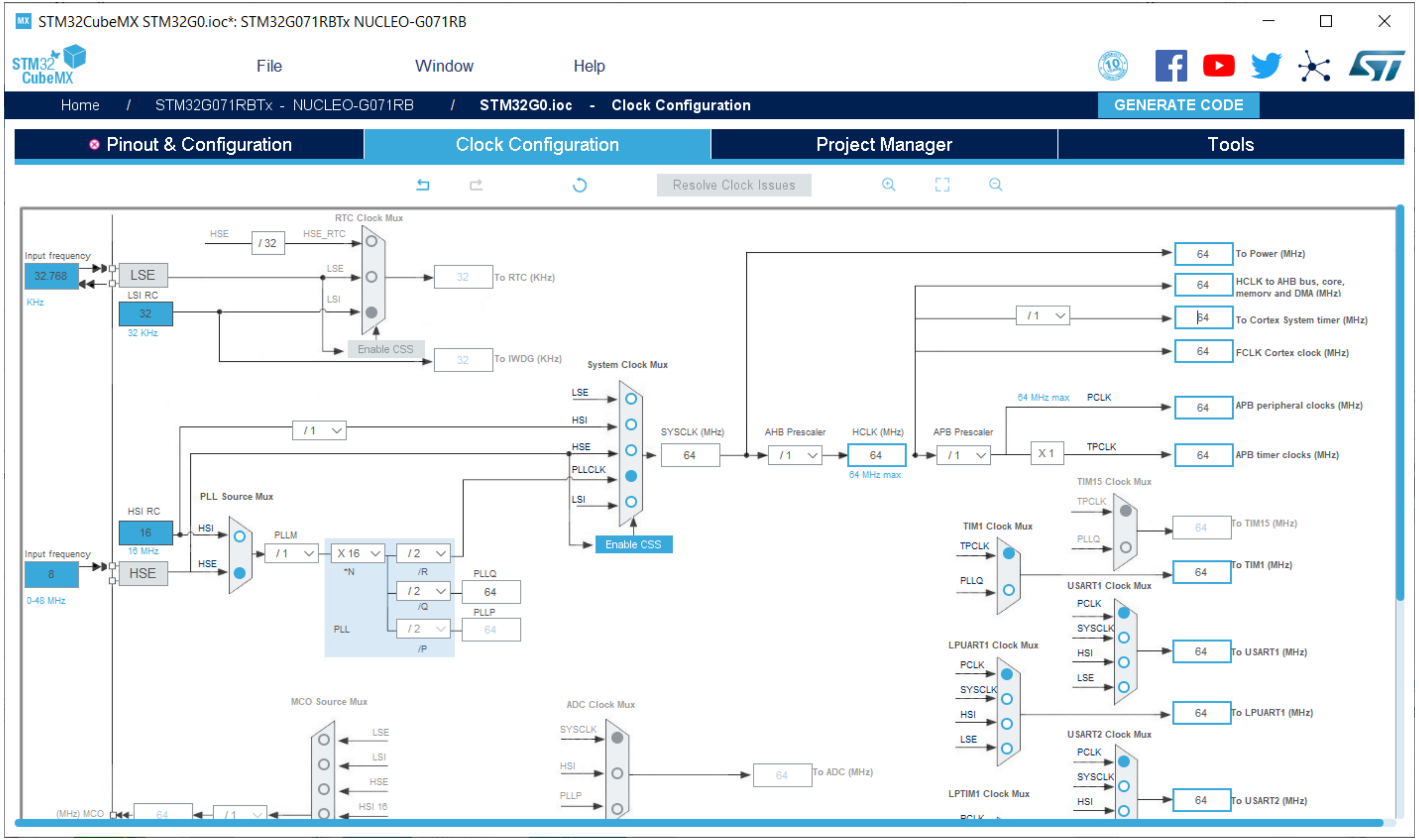
life.augmented

- Generate all the initialization code in C
- Generates project file for any supported development toolchain
- User code can be added in dedicated sections and will be kept upon regeneration
- Option to use the latest library version or keep the same even if regenerating



Clock configuration

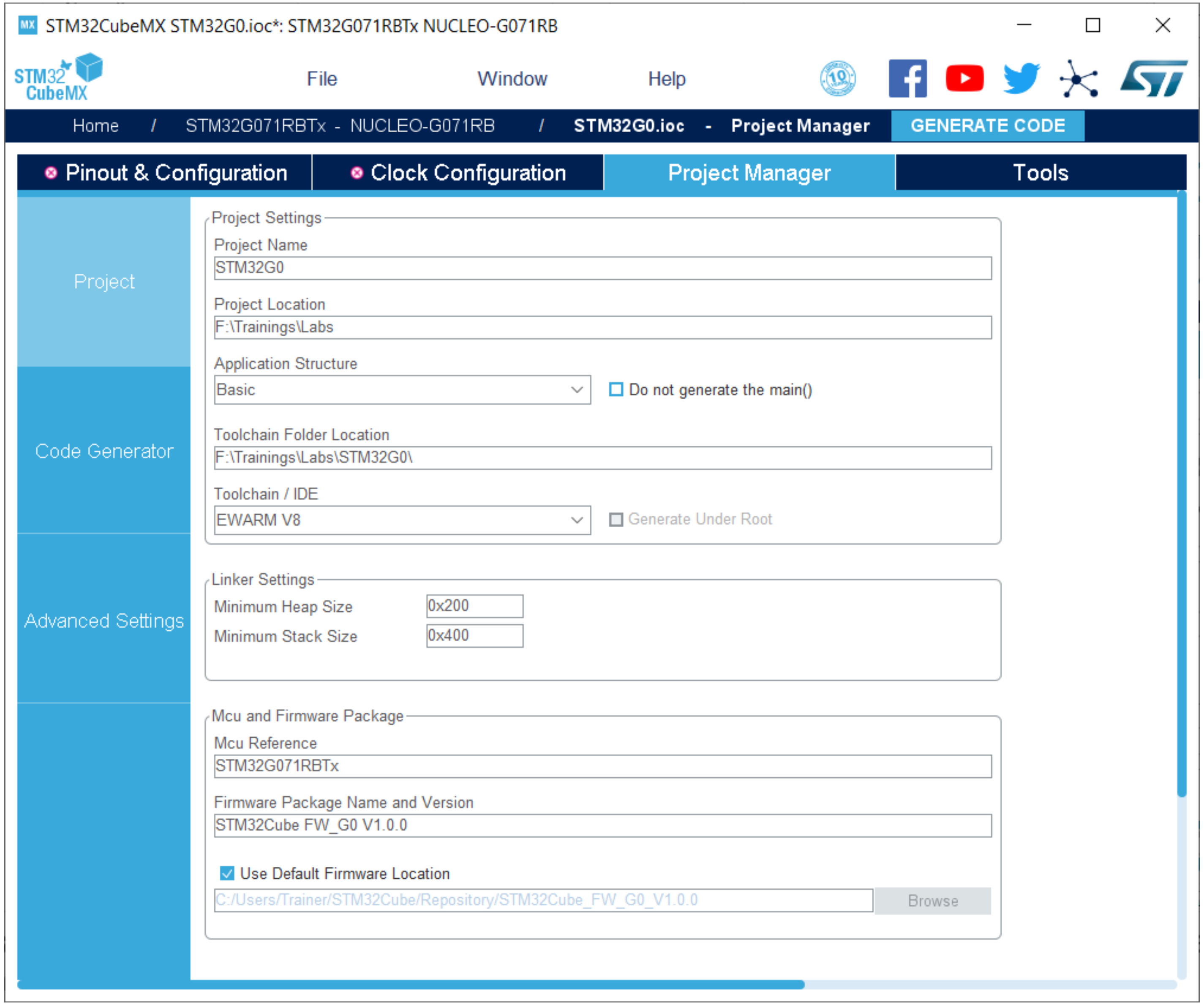
- Immediate display of all clock values
- Active and inactive clock paths are differentiated
- Management of clock constraints and features



life.augmented

Code generation project settings

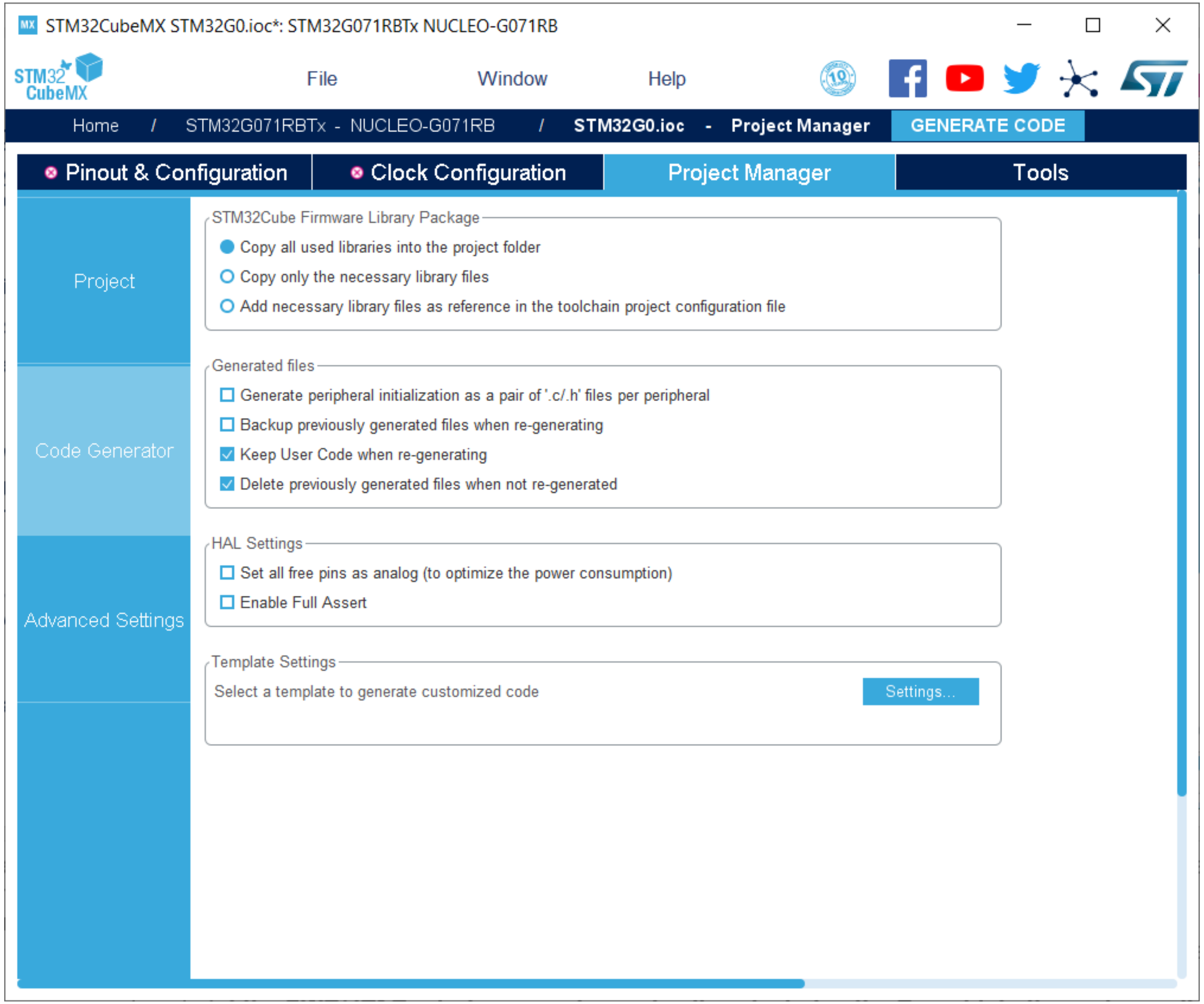
- Name your project when saving
- Browse for project location
- Pick the preferred toolchain
- Review the exact MCU type and library version



life.augmented

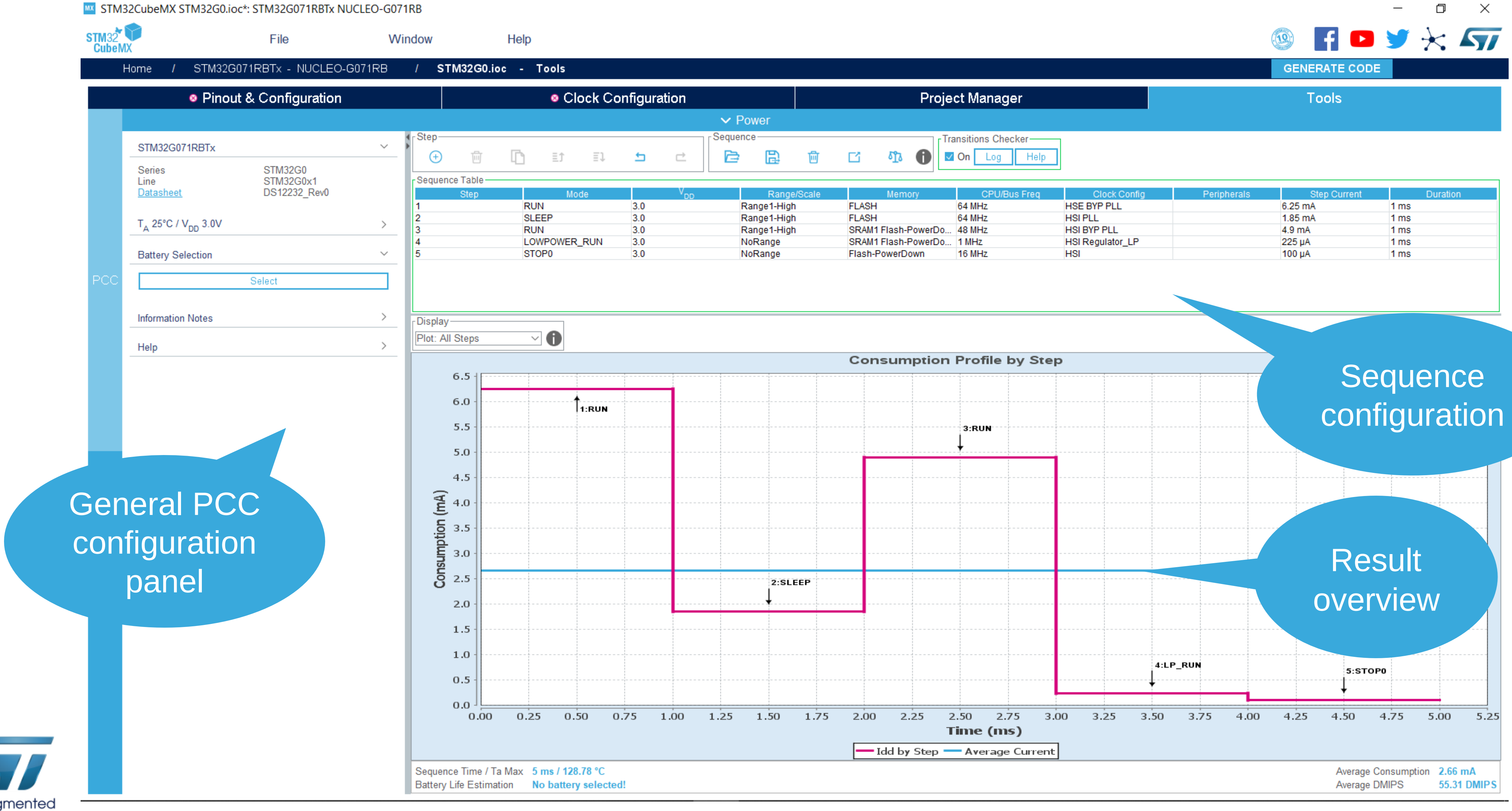
Code generation options

- Library package
  - Whole library or the necessary part may be copied to the generated project folder
  - Or keep the library in original place and refer to it from all projects
- Generated files
  - Each peripheral initialized in separate file or in common source file
  - Options for working with old files
  - **The option to keep user code intact is here**
- HAL settings
  - Setting available pins to analog reduces power consumption, but be careful to **explicitly select SWD/JTAG in pinout**
  - Full assert is useful for debugging



life.augmented

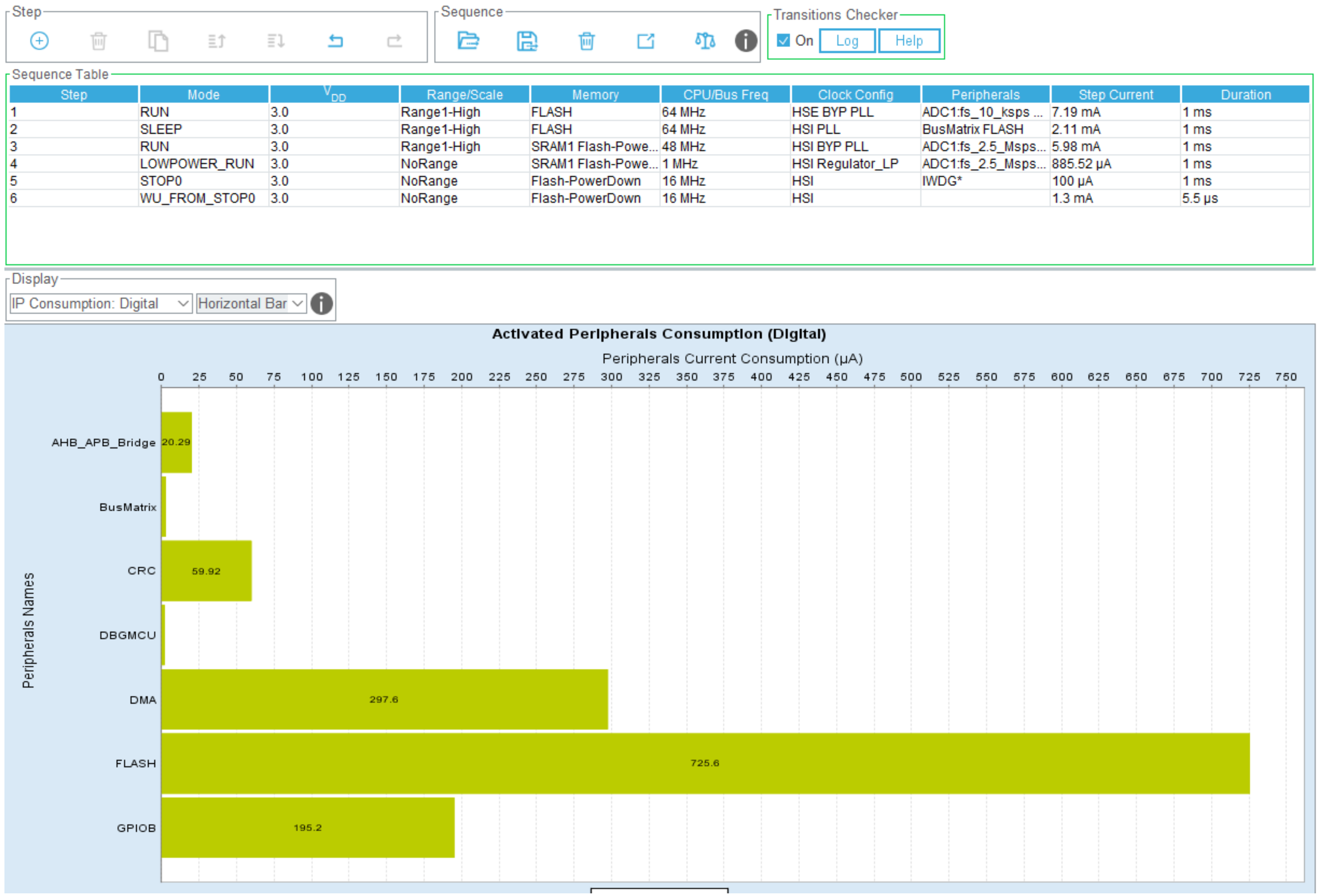
PCC (Power Consumption Calculator)



life.augmented

PCC - Sequence consumption profile display

- It’s possible to detach the charts to external display for presentation purposes
- Several different views selectable
  - Plot current vs time
  - Pie chart
  - Consumption of peripherals



Step	Mode	V_{DD}	Range/Scale	Memory	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	3.0	Range1-High	FLASH	64 MHz	HSE BYP PLL	ADC1:fs_10_ksps ...	7.19 mA	1 ms
2	SLEEP	3.0	Range1-High	FLASH	64 MHz	HSI PLL	BusMatrix FLASH	2.11 mA	1 ms
3	RUN	3.0	Range1-High	SRAM1 Flash-Powe...	48 MHz	HSI BYP PLL	ADC1:fs_2.5_Msps...	5.98 mA	1 ms
4	LOWPOWER_RUN	3.0	NoRange	SRAM1 Flash-Powe...	1 MHz	HSI Regulator_LP	ADC1:fs_2.5_Msps...	885.52 µA	1 ms
5	STOP0	3.0	NoRange	Flash-PowerDown	16 MHz	HSI	IWDG*	100 µA	1 ms
6	WU_FROM_STOP0	3.0	NoRange	Flash-PowerDown	16 MHz	HSI		1.3 mA	5.5 µs

life.augmented

Generating Project Report Files

- An optional step is to generate a PDF report
- The PDF report is also available without PCC
- Complete saved project work includes:
  - Project.ioc
  - Project.pcs
  - Project.pdf
  - Project.txt
  - Project.jpg
  - … and the generated project for a supported development environment

STM32G0 Project
Configuration Report

6. Power Consumption Calculator report

6.1. Microcontroller Selection

Series	STM32G0
Line	STM32G0x1
MCU	STM32G071RBTx
Datasheet	DS12232_Rev0

6.2. Parameter Selection

Temperature	25
Vdd	3.0

6.3. Battery Selection

Battery	Li-MnO2(CR1225)
Capacity	48.0 mAh
Self Discharge	0.12 %/month
Nominal Voltage	3.0 V
Max Cont Current	1.0 mA
Max Pulse Current	5.0 mA
Cells in series	1
Cells in parallel	1

life.augmented

Lab: Blinky

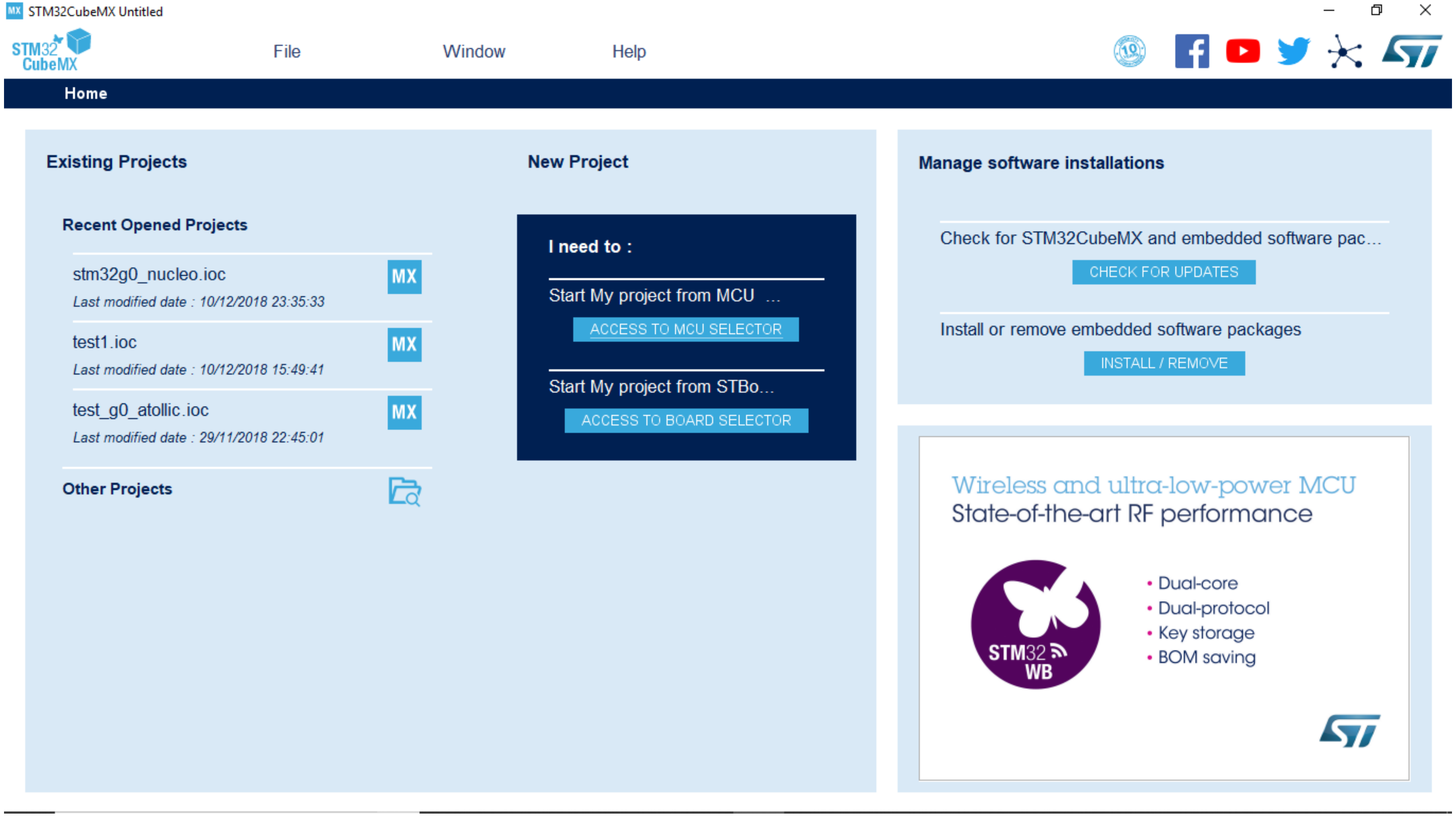
life.augmented

Lab: Blinky

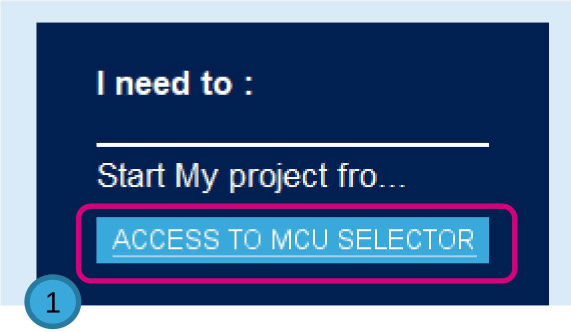
<u>Objective:</u>

- The objective of this lab is to generate a very simple project using STM32CubeMX Software.
- In this example we are going to blink one of the LEDs present on the STM32G0 Nucleo board, connected to PA5 of the STM32G0 MCU.

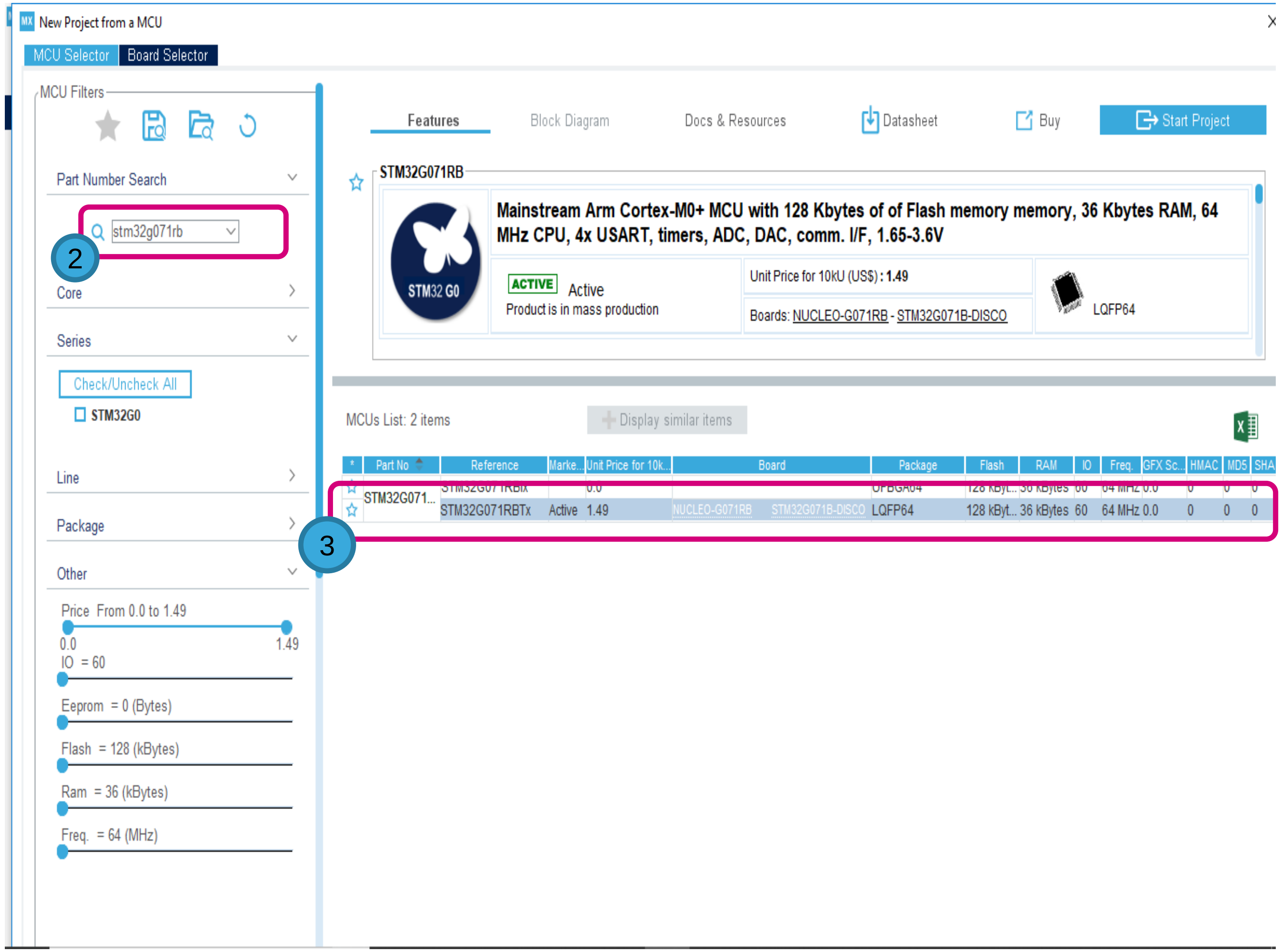
Run STM32CubeMX



Step 1: Create New Project



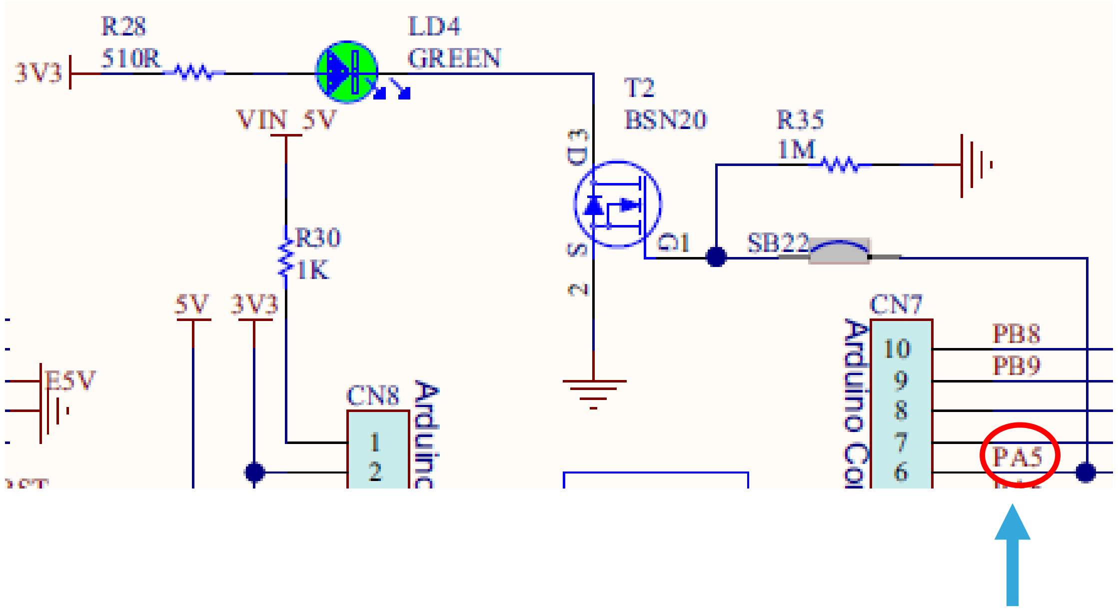
- Click **Access To MCU Selector** 1
- Type: “stm32g071rb” in the Part Number Search 2
- Then Select **STM32G071RBTx** 3
  - **LQFP64, 128kB Flash**
- Double Click

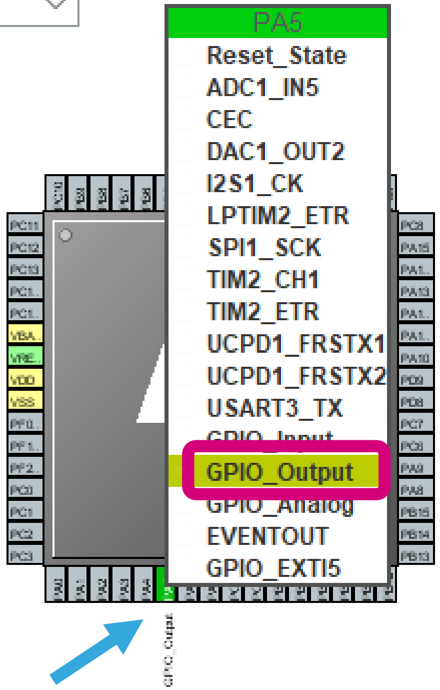


Step 2: Pin Configuration

- In this example we are going to use one of the LEDs present on the STM32G0 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



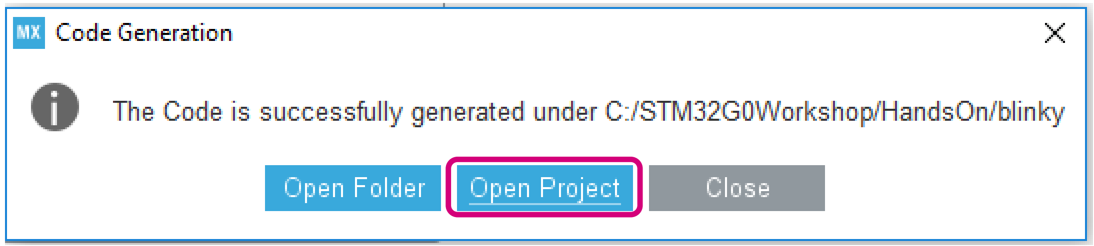


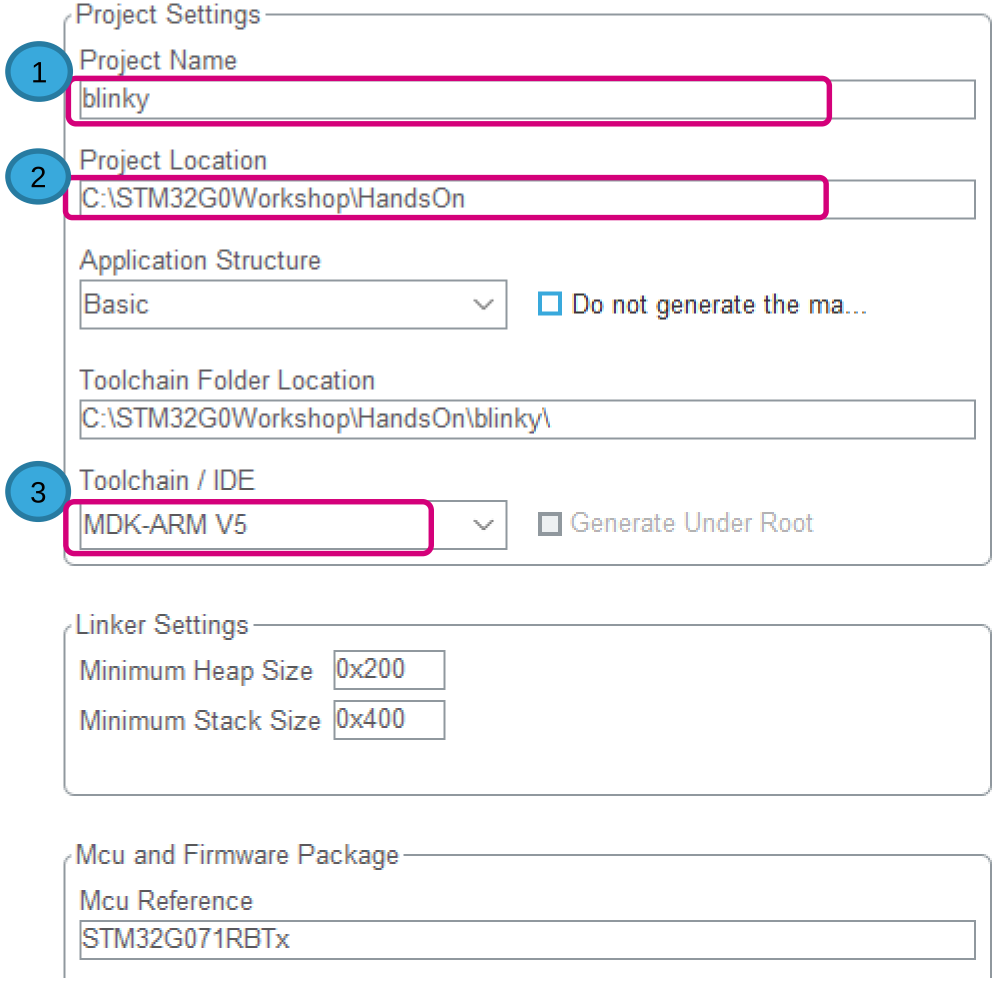
Step 3: Generate Source Code

- Open **Project Manager**

- Set the project name (**blinky**) and the project location **(C:\STM32G0Workshop\HandsOn)** 1 2
- Set the IDE Toolchain to **MDK-ARM V5** 3
- **Generate Code**

- Click **Open Project**





Note: STM32CubeMX projects have an *ioc* file extension

life.augmented

Free MDK-ARM for ST

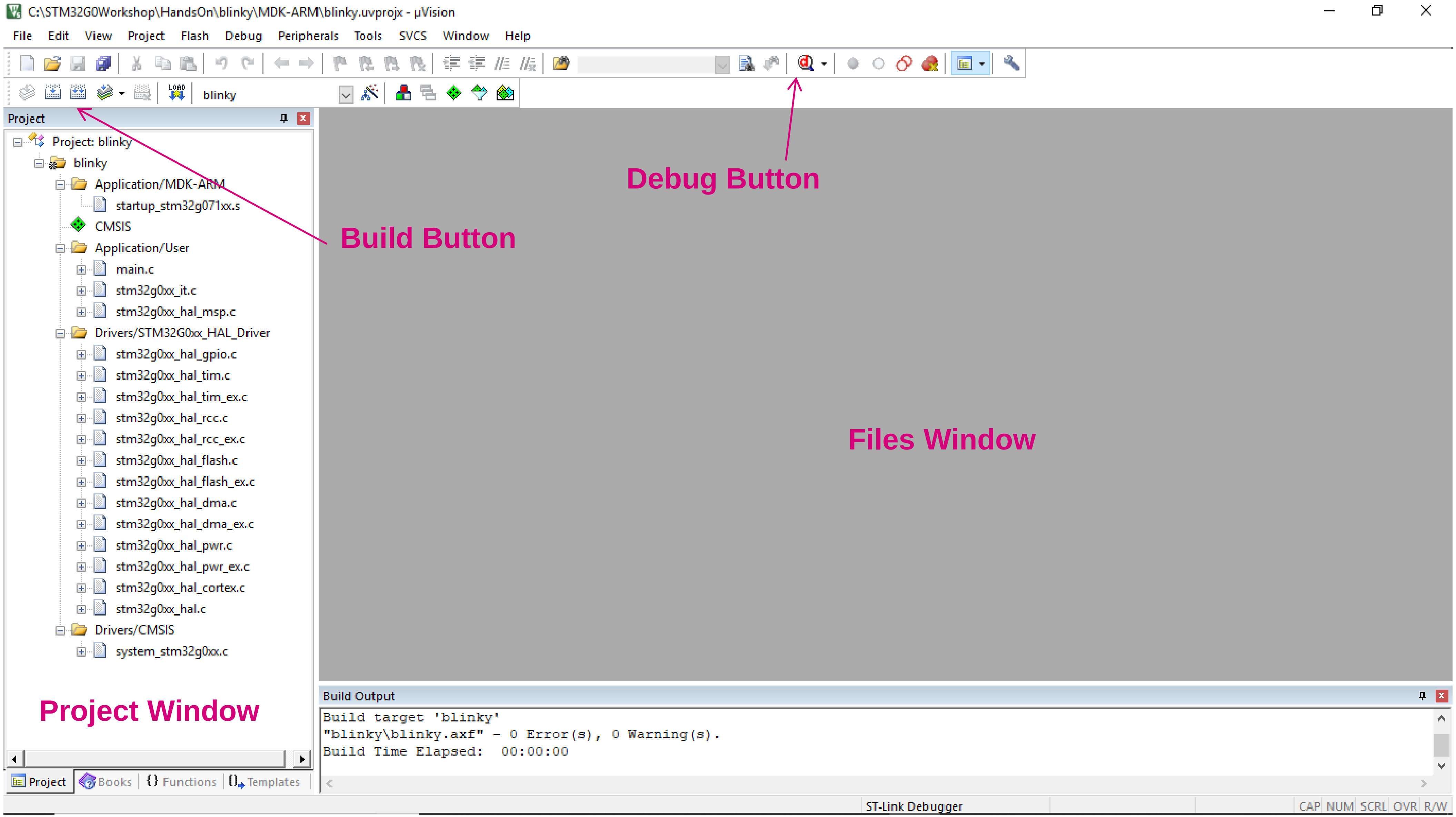
ARM KEIL Microcontroller Tools

- Free licenses for STM32 devices based on Cortex-M0/M0+ cores :
  - Applicable immediately to all **STM32G0**, STM32F0 and STM32L0 MCUs.
  - PC-locked multi-year licenses.
  - No code size limit.
  - Multiple language support.
  - Technical support included.
- Direct download from Keil website :
  - No limit of number of downloads by customer.
  - Direct access to configuration files for STM32 and associated boards.
  - Free access to MDK-ARM periodic updates.

- To get a free MDK-ARM license for STM32F0,STM32L0 and **STM32G0**:
  - Go to Keil website at : www.keil.com/mdk-st
  - Download MDK-ARM tool chain.
  - Activate the free license using this Product Serial Number (PSN) :
    4PPFW-QBEHZ-M0D5M

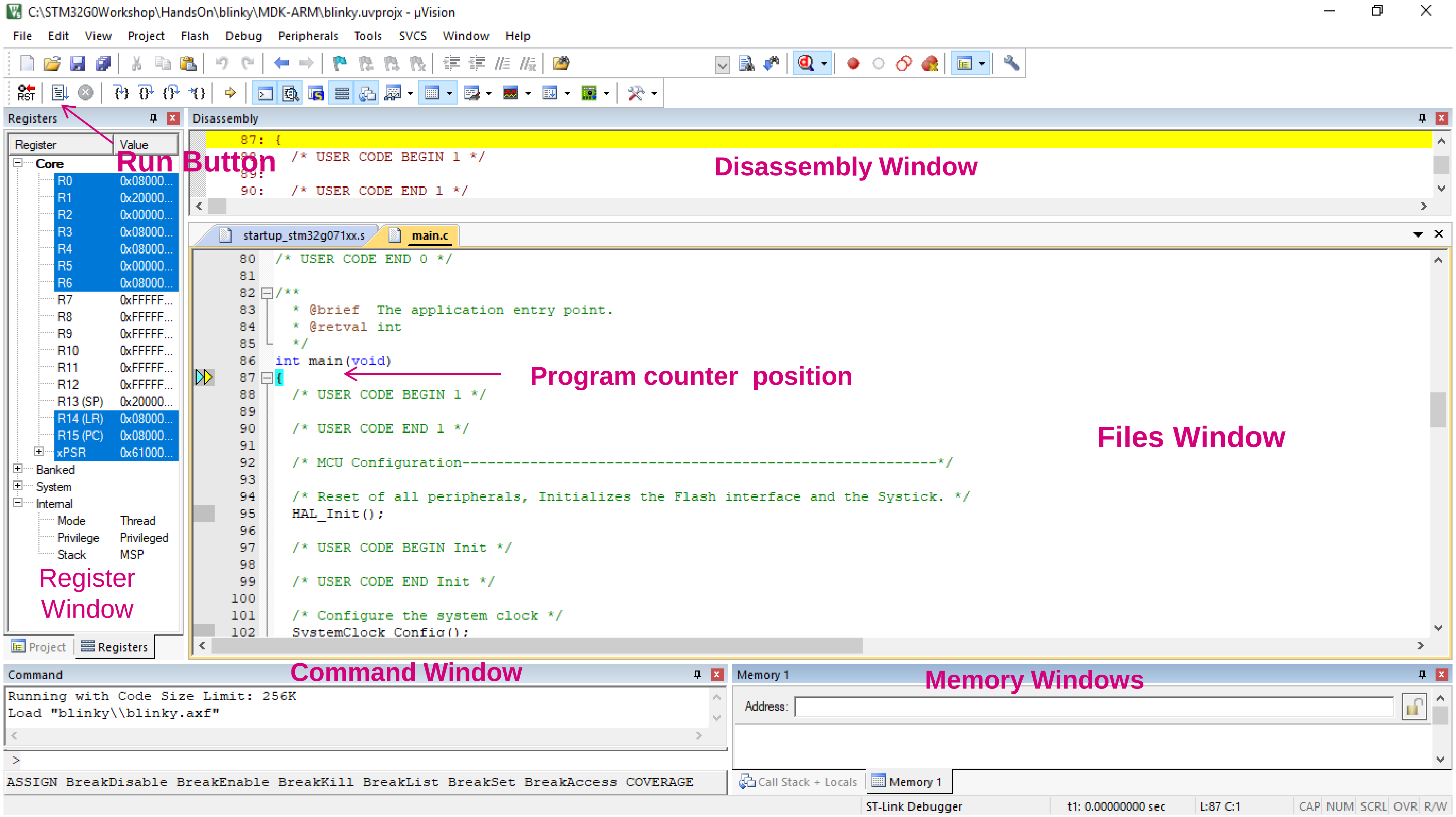
life.augmented

Inside Keil uVision (ARM-MDK)



life.augmented

The MDK-ARM IDE Debugger

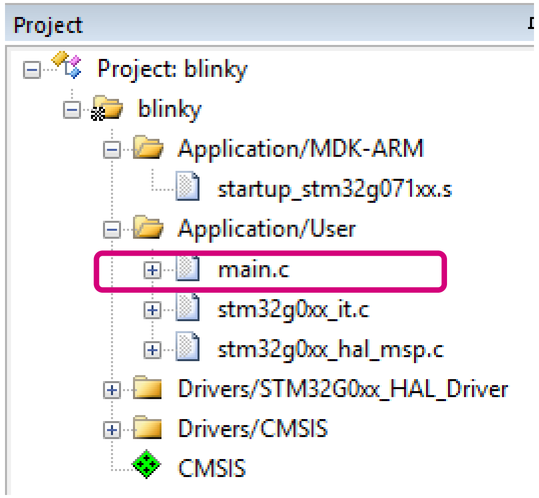


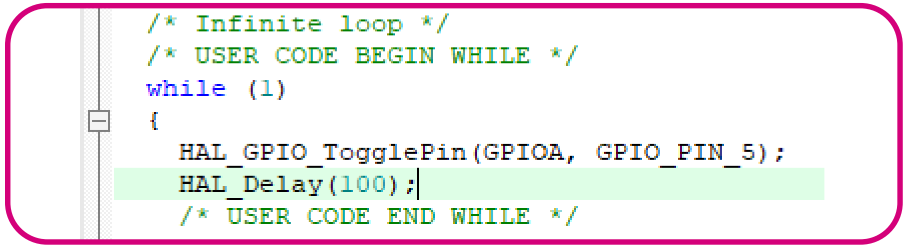
Step 4: Toggle The LED

In the Keil uVision5 IDE:

- 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(GPIOA, GPIO_PIN_5);
HAL_Delay(100);
```





Note: Code within the “USER CODE BEGIN WHILE” / “USER CODE END WHILE” section will be preserved after regeneration by STM32CubeMX.

IMPORTANT NOTE: The code to be added for the labs is located in a text file called: “**code_to_add_vx.x.txt**”, copying from the presentation may not work properly.

TIP: You can do a search of the USER CODE section where to add the code in Keil using shortcut: CTRL +F or Edit -> Find…

life.augmented

Step 5: Build the Project and Debug

- Click the “Build” button (F7)


```
Build target 'blinky'
"blinky\blinky.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:00
```

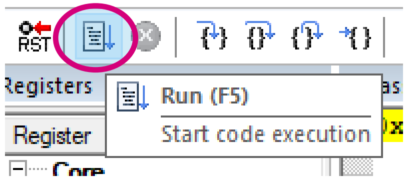
- Click the “Start/Stop Debug Session” button (Ctrl + F5)

Step 5: Build the Project

- If you see this warning message due to a minor Syntax error in the startup file, just press OK to continue.

Note: To correct the Syntax error in the startup_stm32g071x.s:
- Remove “<h2><center>©” from line 17.
- Remove “</center></h2>” from line 18.

- Click the “Run” button (F5)



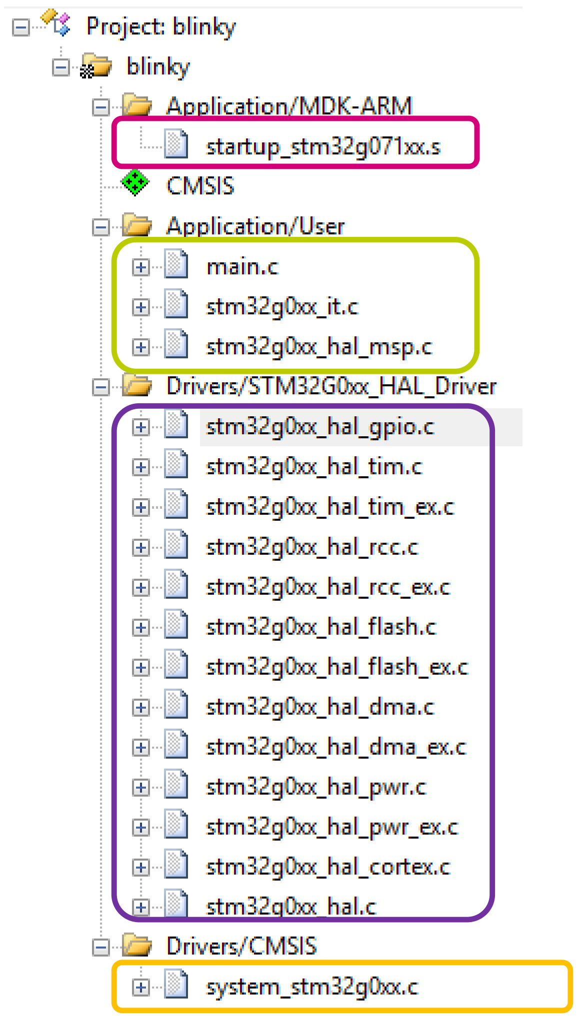
- Enjoy the flashing Green LED (LD4)!

Firmware Project Overview

life.augmented

Project Overview

- startup_stm32g071x.s
    - System initialization, vector table, reset and jump to main()
- User files
    - main.c (program entry point)
    - stm32g0xx_hal_msp.c (micro specific peripheral initialization and de-initialization functions)
    - stm32g0xx_it.c (Peripheral interrupt handlers)
- HAL driver files
    - Peripheral source files
- system_stm32g0xx.c
    - Contains SystemInit() function called at startup before branch to main()
    - Contains core clock variable (HCLK)
    - SystemCoreClockUpdate() function



startup_stm32g071xx.s

- Main Characteristics
  - Initializes stack pointer
  - Contains the device vector table
  - Contains Reset handler
    - Called after RESET
    - Calls SystemInit()
    - Branch to main()

```
;*******************************************************************************
;* File Name          : startup_stm32g071xx.s
;* Author             : MCD Application Team
;* Description        : STM32G071xx devices vector table for MDK-ARM toolchain.
;*                      This module performs:
;*                      - Set the initial SP
;*                      - Set the initial PC == Reset_Handler
;*                      - Set the vector table entries with the exceptions ISR address
;*                      - Branches to __main in the C library (which eventually
;*                        calls main()).
;*                      After Reset the CortexM0 processor is in Thread mode,
;*                      priority is Privileged, and the Stack is set to Main.
;* <<< Use Configuration Wizard in Context Menu >>>
;*******************************************************************************
```

```
; Vector Table Mapped to Address 0 at Reset
                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors
                EXPORT  __Vectors_End
                EXPORT  __Vectors_Size

__Vectors       DCD     __initial_sp                   ; Top of Stack
                DCD     Reset_Handler                  ; Reset Handler
                DCD     NMI_Handler                    ; NMI Handler
                DCD     HardFault_Handler              ; Hard Fault Handler
```

```
; Reset handler routine
Reset_Handler    PROC
                 EXPORT  Reset_Handler                 [WEAK]
        IMPORT  __main
        IMPORT  SystemInit
                 LDR     R0, =SystemInit
                 BLX     R0
                 LDR     R0, =__main
                 BX      R0
                 ENDP
```

- SystemInit():
  - This function is called at startup after reset and before branch to main.

SystemInit()

```
/**
  * @brief  Setup the microcontroller system.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* Configure the Vector Table location add offset address ------------------*/
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
#endif
}
```

main.c

- **Header files**
  - Main.h

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

- **main() function**
  - Configures system clock
  - Call peripheral config. functions
  - Infinite loop

```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();
```

- **Configuration functions**
  - SystemClock_Config()
  - MX_GPIO_Init()

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

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  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);

}
```

```c
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /**Configure the main internal regulator output voltage
  */
  HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);
  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSIDiv = RCC_HSI_DIV1;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

Example HAL driver file

- stm32g0xx_hal_gpio.c
  - GPIO Initialize and De-Initialize API functions

```c
void HAL_GPIO_Init(GPIO_TypeDef  *GPIOx, GPIO_InitTypeDef *GPIO_Init)
{
  uint32_t position = 0x00u;
```

  - GPIO Read and Write API function

```c
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
  GPIO_PinState bitstatus;
```

  - GPIO EXTI IRQ handler

```c
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  /* EXTI line interrupt detected */
  if (__HAL_GPIO_EXTI_GET_RISING_IT(GPIO_Pin) != 0x00u)
  {
    __HAL_GPIO_EXTI_CLEAR_RISING_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Rising_Callback(GPIO_Pin);
  }

  if (__HAL_GPIO_EXTI_GET_FALLING_IT(GPIO_Pin) != 0x00u)
  {
    __HAL_GPIO_EXTI_CLEAR_FALLING_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Falling_Callback(GPIO_Pin);
  }
}
```

  - GPIO Callback function

```c
__weak void HAL_GPIO_EXTI_Rising_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
   */
}
```

Lab: PWM (Pulse Width Modulation) Timer

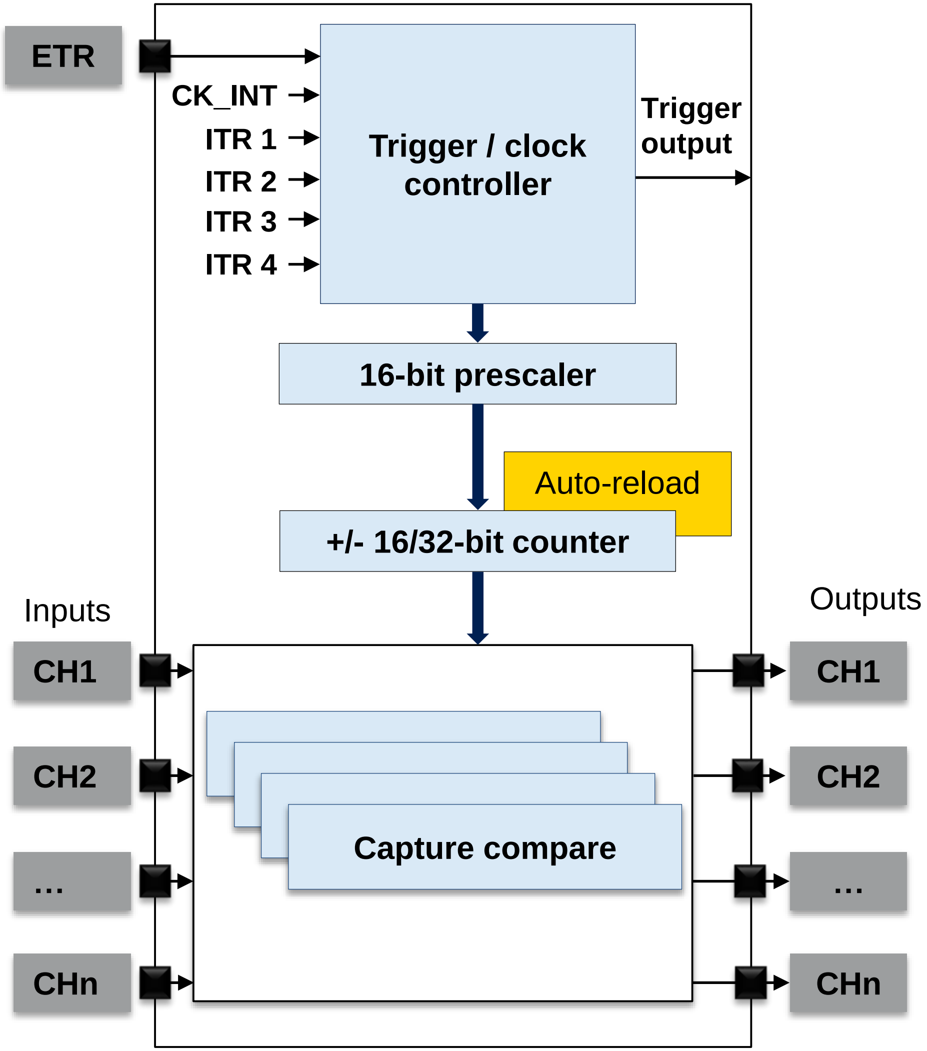
life.augmented

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 mode to blink the LED that we previously controlled with a GPIO.
- PA5 has an alternate Timer channel alternate function which is Timer 2 Channel 1: TIM2_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

STM32G0 timer instance features

Feature		TIM1 (Advanced Control)	TIM2	TIM3	TIM6	TIM7	TIM14	TIM15	TIM16	TIM17
			(General-Purpose)		(Basic)		(General-Purpose)			
Clock source		CK_INT External input pin External trigger input ETR	CK_INT External input pin External trigger input ETR Internal trigger inputs		CK_INT		CK_INT	CK_INT External input pin Internal trigger inputs	CK_INT External input pin	
Resolution		16-bit	**32-bit**	16-bit	16-bit		16-bit	16-bit		
Prescaler		16-bit								
Counter direction		Up, Down, Up&Down	Up, Down, Up&Down		Up		Up	Up		
Repetition counter		✓	-		-		-	✓		
Synchronization	Master	✓	✓		✓		-	✓		
	Slave	✓	✓		-		-	✓	-	
Number of channels		6: ➢ CH1/CH1N ➢ CH2/CH2N ➢ CH3/CH3N ➢ CH4 ➢ CH5 and CH6 output only, not available externally	4: ➢ CH1 ➢ CH2 ➢ CH3 ➢ CH4		0		1: ➢ CH1	2: ➢ CH1/CH1N ➢ CH2	1: ➢ CH1/CH1N	
Trigger input		✓	✓							

life.augmented

STM32G0 timer instance features

<table>
<tr><th>Feature</th><th>TIM1 (Advanced Control)</th><th>TIM2</th><th>TIM3</th><th>TIM6</th><th>TIM7</th><th>TIM14</th><th>TIM15</th><th>TIM16</th><th>TIM17</th></tr>
<tr><td>Input capture mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>✓</td><td colspan="3">✓</td></tr>
<tr><td>PWM input mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td></td><td>✓</td><td colspan="2">-</td></tr>
<tr><td>Forced output mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>✓</td><td colspan="3">✓</td></tr>
<tr><td>Output compare mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>✓</td><td colspan="3">✓</td></tr>
<tr><td>PWM</td><td>Standard
Asymmetric
Combined
Combined 3-phase
6-step PWM</td><td colspan="2">Standard
Asymmetric
Combined</td><td colspan="2">-</td><td>Standard</td><td>Standard
Asymmetric
Combined</td><td colspan="2">Standard</td></tr>
<tr><td>Programmable dead-time</td><td>✓ (CH1-3)</td><td colspan="2">-</td><td colspan="2">-</td><td>-</td><td>✓ (CH1)</td><td colspan="2">-</td></tr>
<tr><td>Break inputs</td><td>2 bidirectional</td><td colspan="2">0</td><td colspan="2">0</td><td>0</td><td colspan="3">1 bidirectional</td></tr>
<tr><td>One-Pulse Mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>✓</td><td colspan="3">✓</td></tr>
<tr><td>Retriggerable one pulse mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>-</td><td>✓</td><td colspan="2">-</td></tr>
<tr><td>Encoder interface mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>-</td><td colspan="3">-</td></tr>
<tr><td>Timer input XOR function</td><td>✓</td><td colspan="2">-</td><td colspan="2">-</td><td>-</td><td>✓</td><td colspan="2">-</td></tr>
<tr><td>DMA</td><td>✓</td><td colspan="2">✓</td><td colspan="2">✓</td><td>-</td><td colspan="3">✓</td></tr>
</table>

Timer clocking schemes

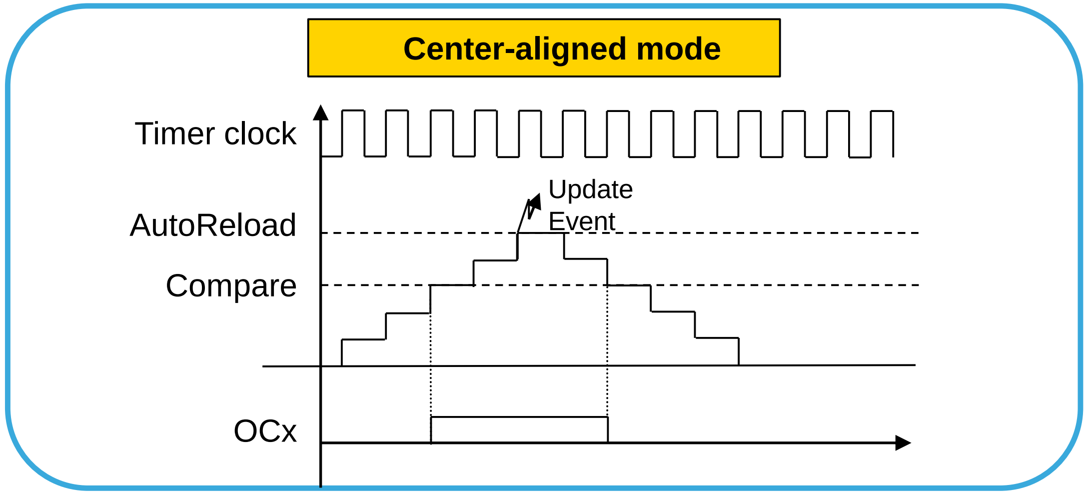
Multiple internal or external clocking options

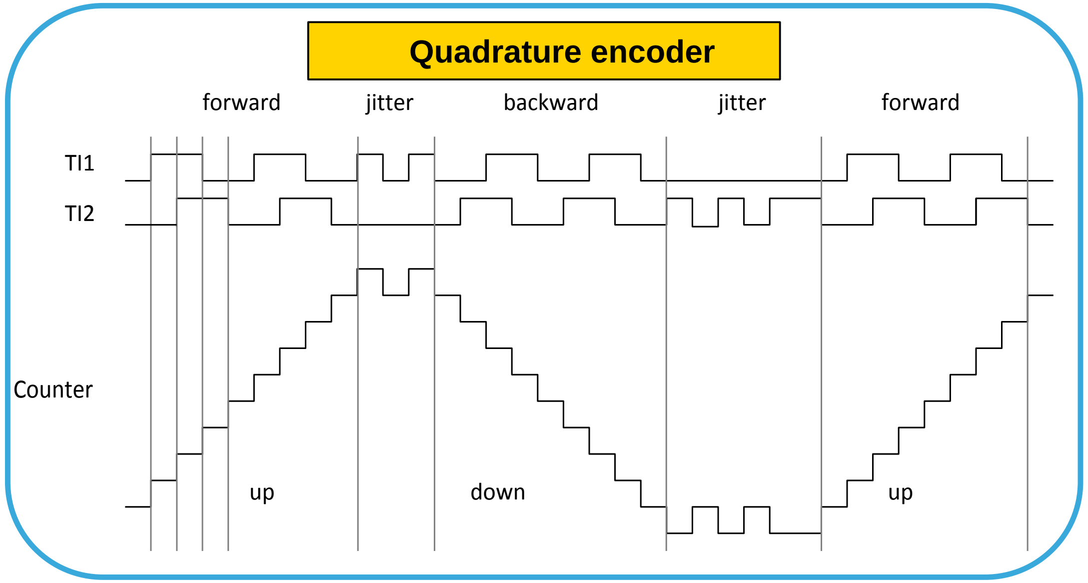
- Timers 1 and 15 are clocked up to 128 MHz to bring additional resolution below 8 ns
  - Finer resolution for buck converters (10-bit accuracy @ 100 kHz PWM)
  - Lower frequency steps for variable frequency resonant converters (e.g. LLC), e.g. 0.4kHz max. frequency step at 200 kHz switching (0.2%)
- Uses cases
  - Timer 1 has 3 complementary pairs: LLC primary and secondary sides (synchronous rectification), boundary conduction mode PFC, buck
  - Timer 15 has one pair only (buck, LLC primary side)

Counting mode

Support of incremental / quadrature encoders and motor drive applications

- Up- and down-counting modes supported
  - On TIM1, TIM2 and TIM3
- Center-aligned PWM generation
  - Direction changes on overflow and underflow
  - Reduces acoustic noise in electric motors
- Built-in support of quadrature encoders
  - Rotary encoder / digital potentiometer
  - Position sensor
  - Allows direct angle reading in timer



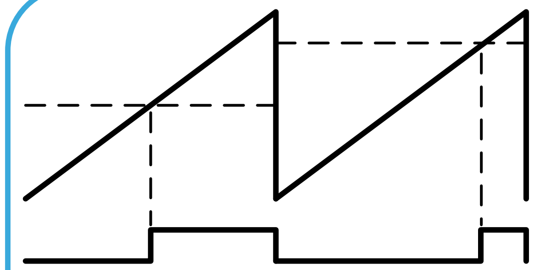


life.augmented

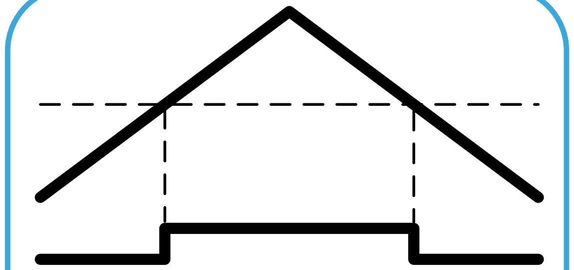
A few PWM modes

A variety of PWM modes to address multiple applications

- Basic PWM, edge- or center-aligned

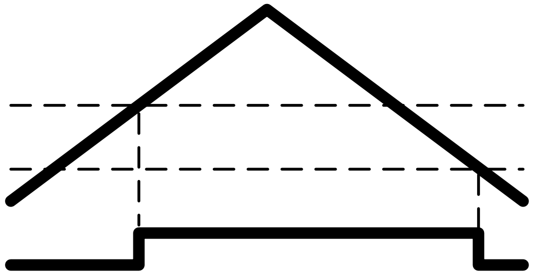


Edge-aligned PWM

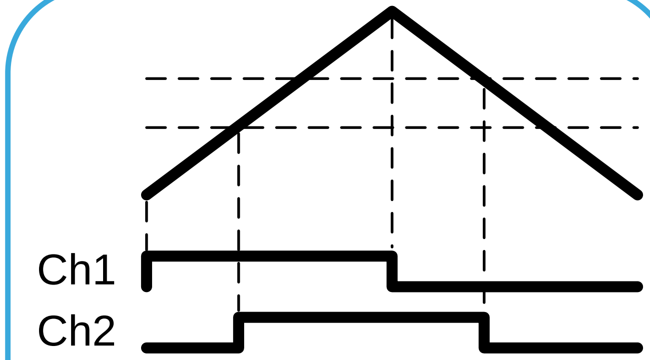


Center-aligned PWM

- Asymmetric center-aligned PWM



Asymmetric PWM



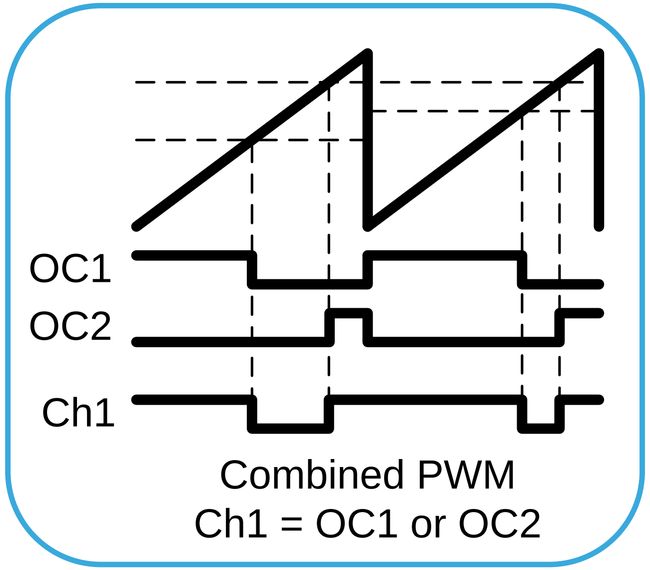
Phase-shifted PWM
(asymmetric mode)

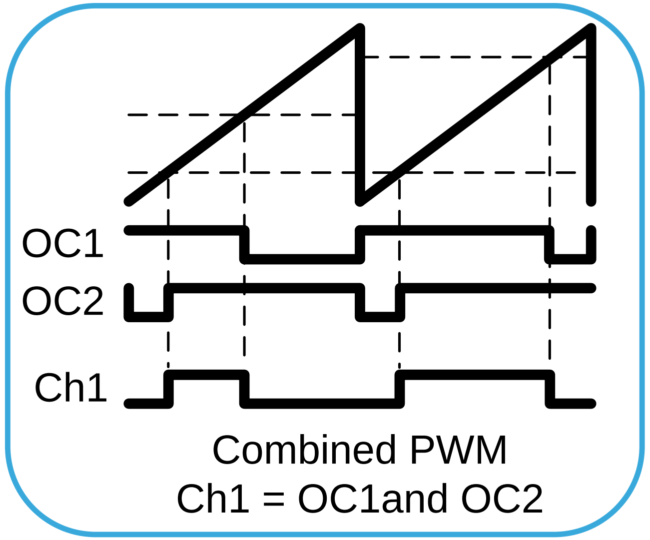
life.augmented

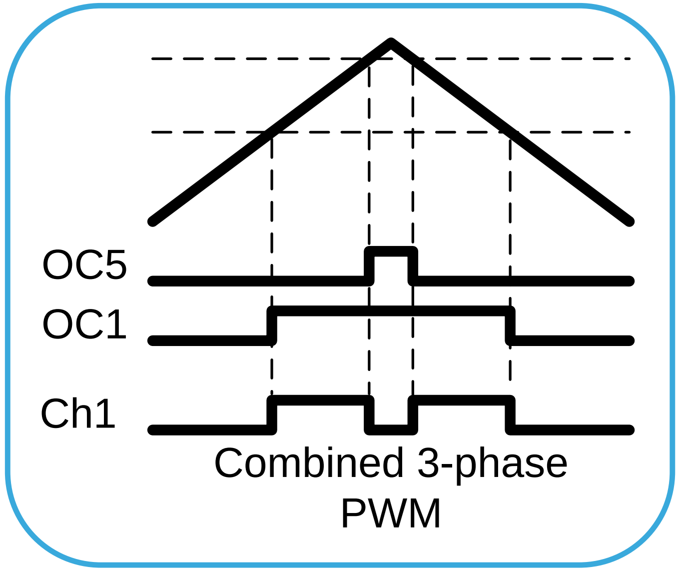
Some more PWM modes

Extends the PWM capabilities and avoids external glue logic

- Combined PWM mode
  - Combines two channels with OR or AND function for more complex waveforms
- Combined 3-phase mode
  - Allows a 4th PWM to be combined with a regular 3-phase PWM for zero vector insertion



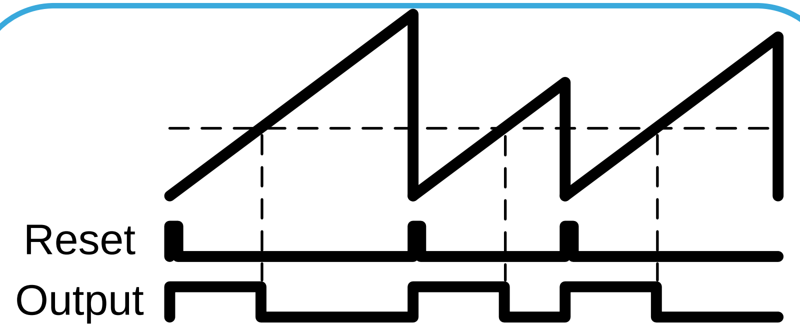




Advanced PWM modes

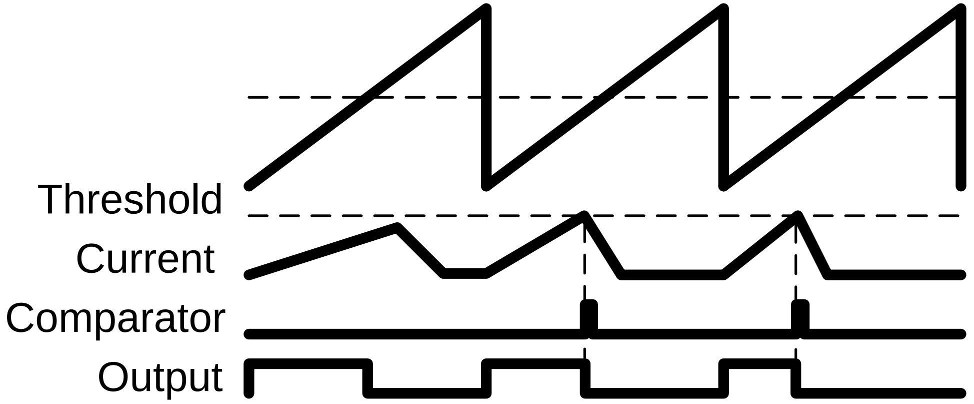
For PWM signals requiring external control

- Variable-frequency PWM
  - Driven by an external signal



Variable frequency
Fixed ON time PWM

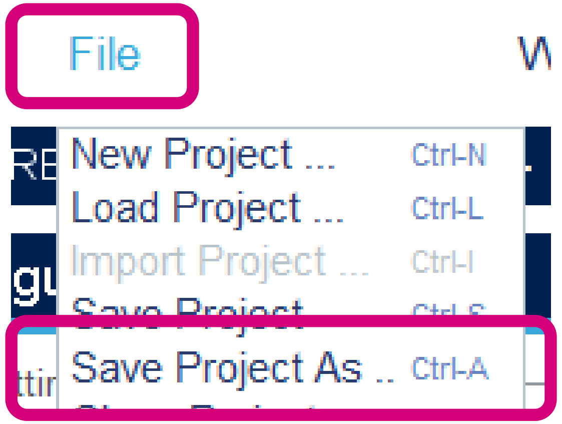
- Cycle-by-cycle controlled duty cycle
  - For current loops, driven by comparator or external pin

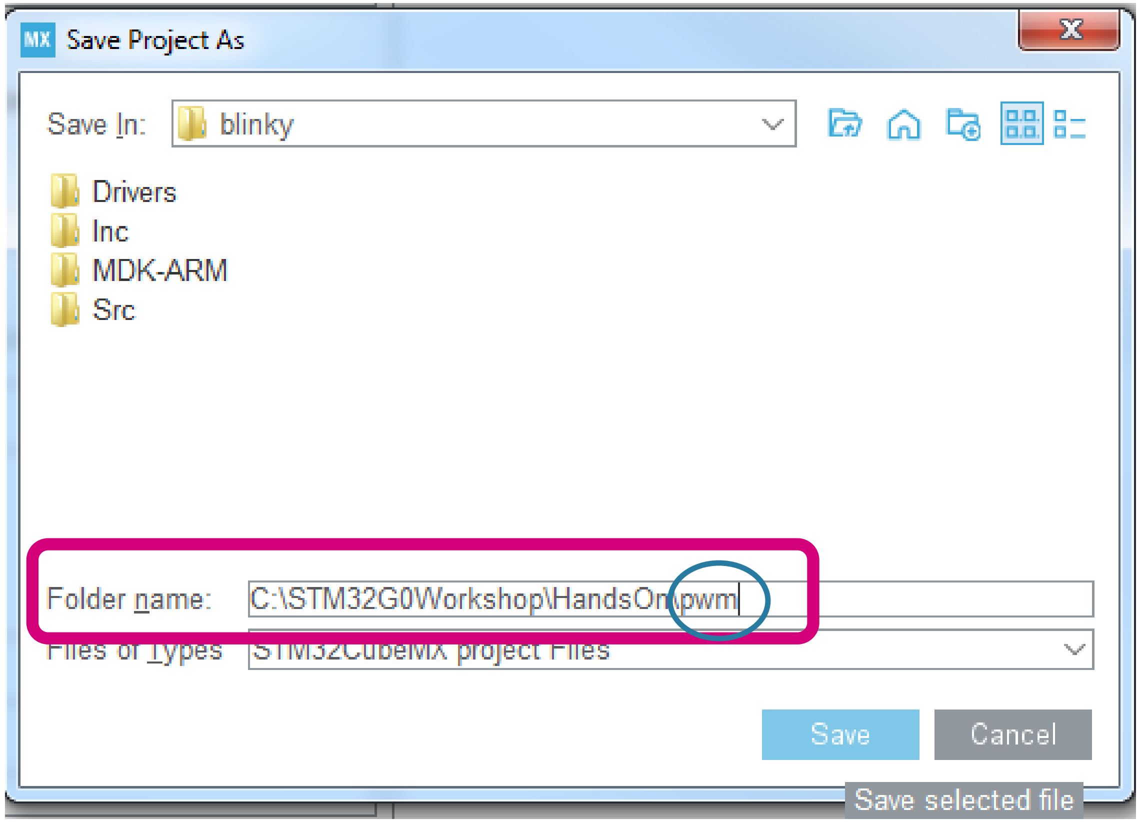


Cycle-by-cycle controlled PWM

Lab: Rename the project

- Close Keil uVision5 IDE if it is open
- Open the last STM32CubeMX project ("**blinky**") (using File->Recent Projects) and save it as a **new project** name "**pwm**" (using File -> Save Project As) as seen below:

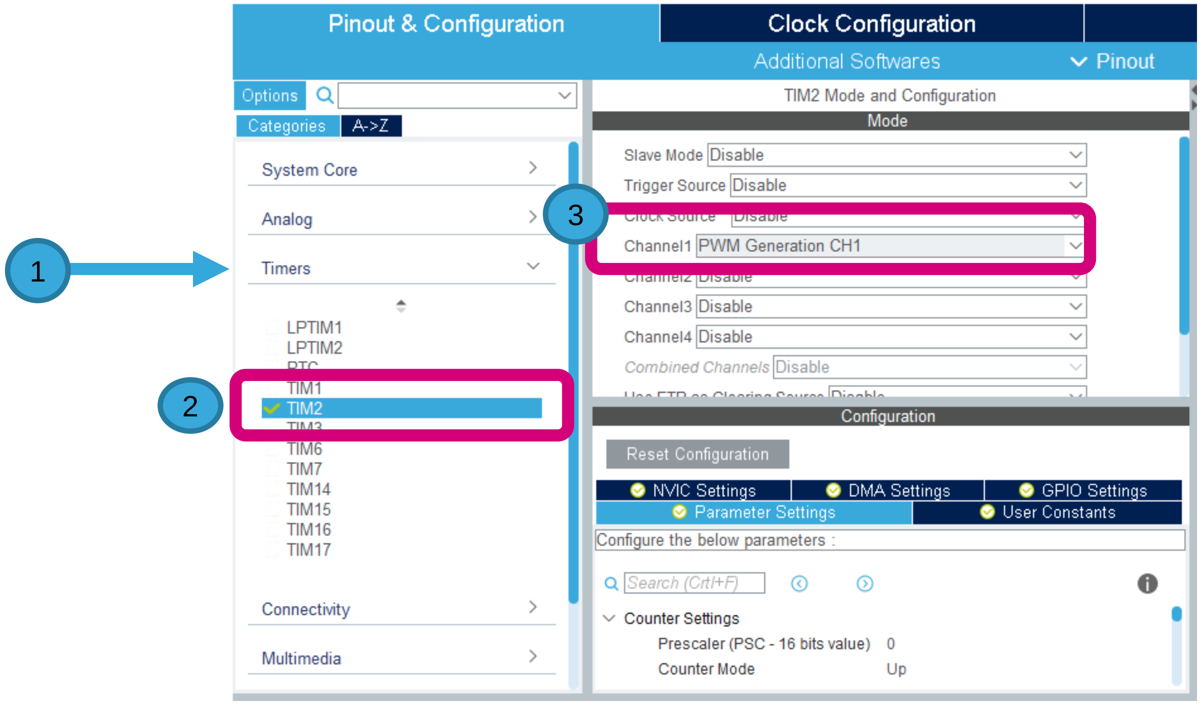


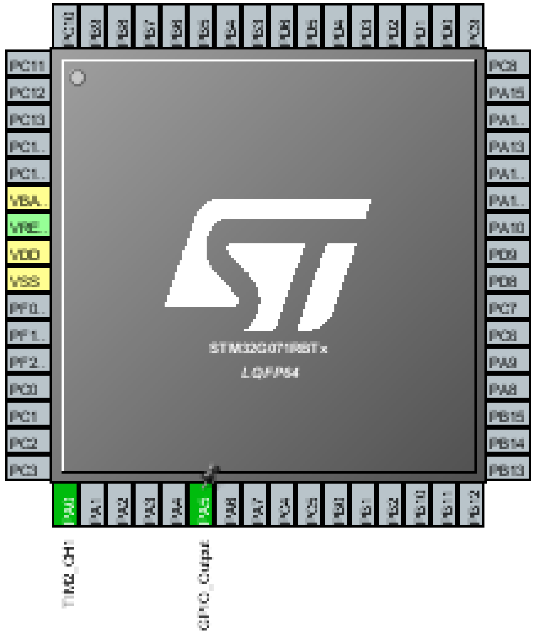


Lab: Timer 2 CH1 Configuration

- In this STM32CubeMX project we are going to add Timer 2 Channel 1 to blink LD4 (PA5) on the Nucleo board.

- In the **Pinout & Configuration** tab, Expand **Timers** Categories, then click on **TIM2** peripheral and set Channel1 to “**PWM Generation CH1**”.

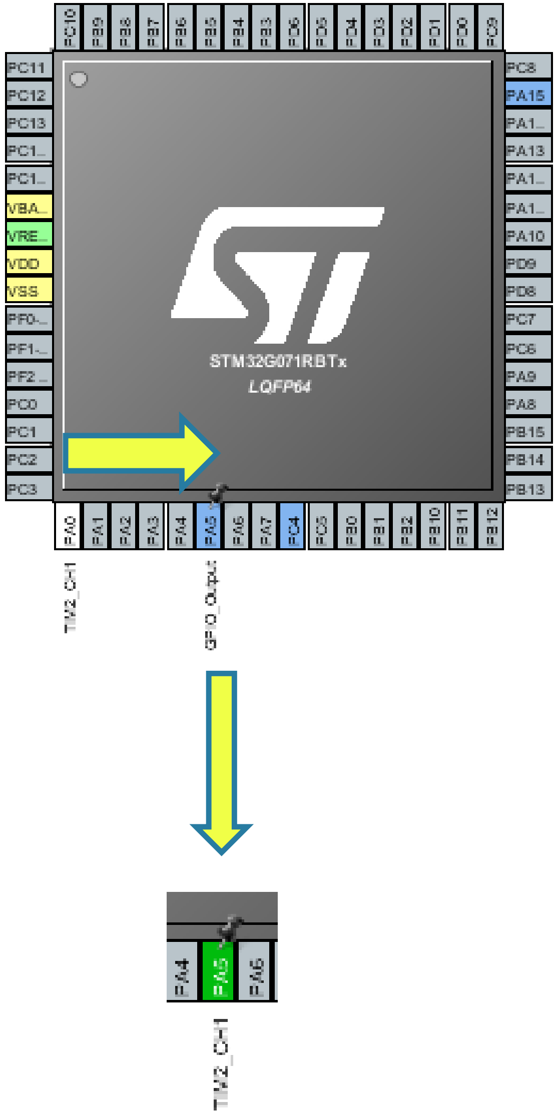




life.augmented

Remapping Timer 2 CH1 output to PA5

- By default the tool will configure Timer 2 CH1 to **PA0**
- We want to remap it to **PA5**
  - NOTE: PA5 is connected to LD4
- Hold “**Ctrl**” button and left mouse click on **PA0**
- Then drag the mouse pointer to **PA5** and then release

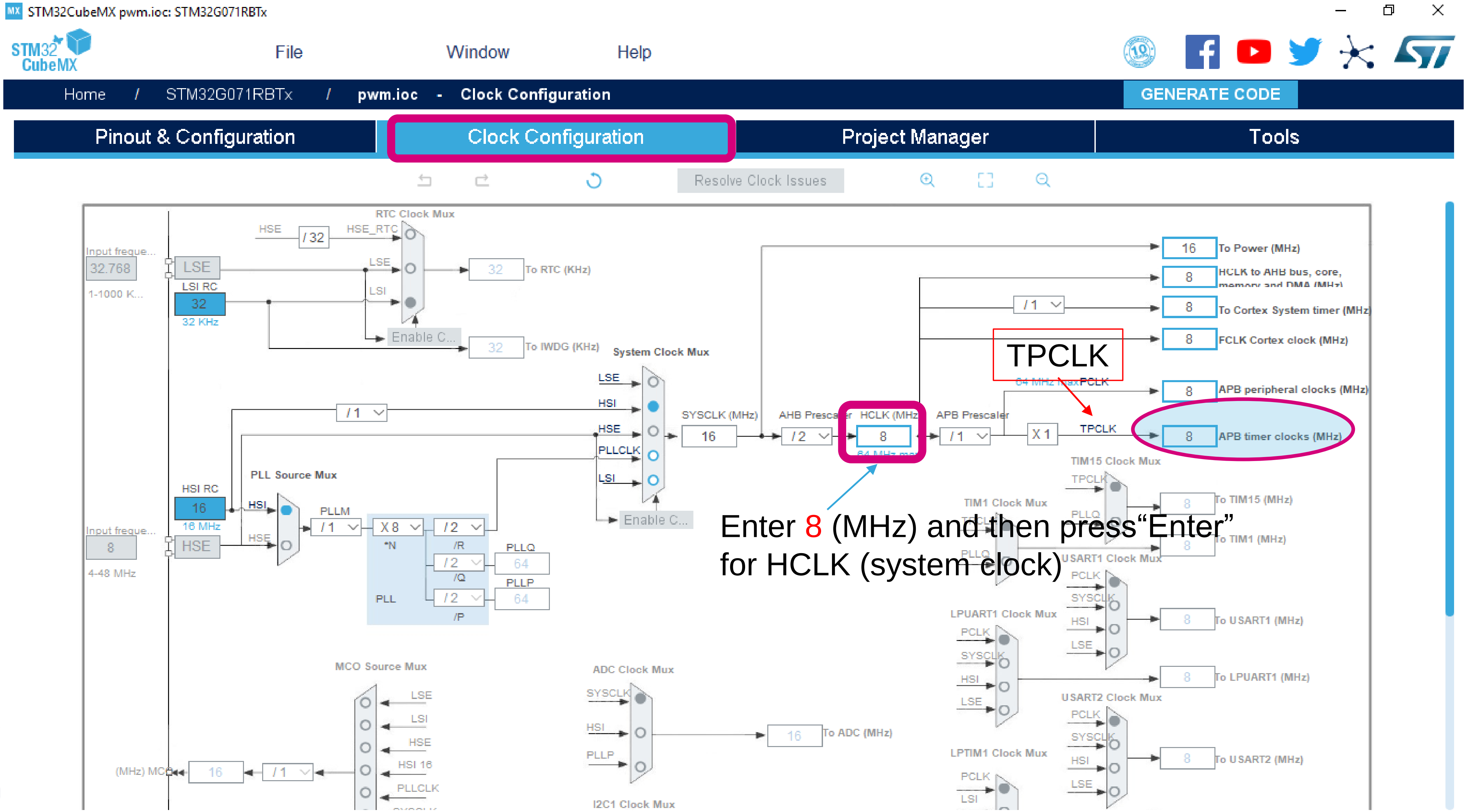


Timer Parameters Calculation

- 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 8 MHz.
- **Prescaler** for the Timer is set to **128**. The resulting timer counter clock is:

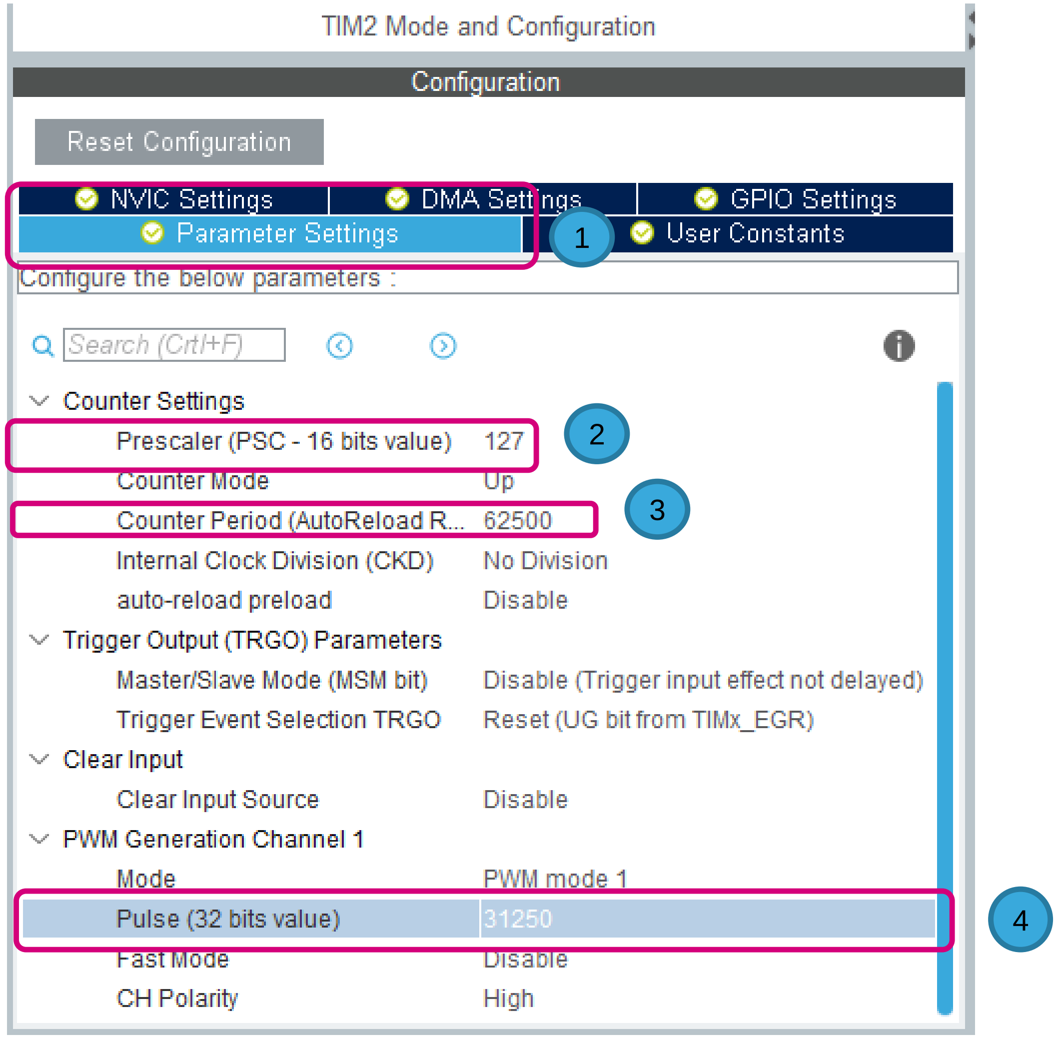
  CK_CNT = **TPCLK / Prescaler =** 8MHz / 128 = **62500** Hz
- To get T=1 Hz (or 1 sec period) the **Counter Period** needs to be set to: **62500**
  - **Counter Period** = CK_CNT / T = 62500 / 1 = **62500**
- To get D=50% duty cycle the **Pulse** needs to be set to: **31250**
  - Pulse = Counter Period / 2 = 62500 / 2 = **31250**

Clock Tree Configuration



TIM2 Configuration – 4 steps

- Select the **Pinout & Configuration**
- In **Parameters Settings** of the TIM2 1
- Configure 1 Hz timer
  - PSC Prescaler -1 = 127 2
  - Counter Period = 62500 3
- Set CH1 PWM
  - Pulse = 31250 4



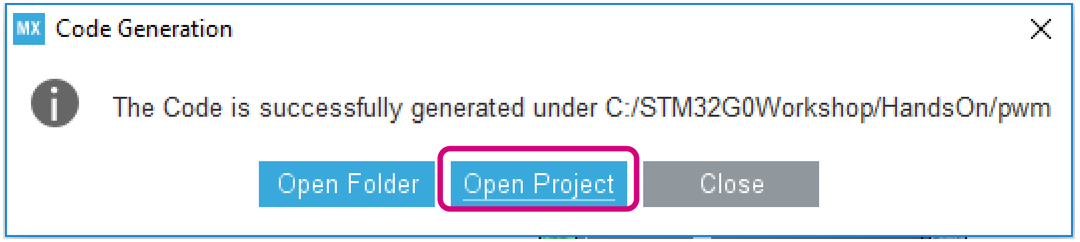
<u>Note about Prescaler</u>: prescaler = PSC +1

life.augmented

Generate Source Code

- **Generate Code**

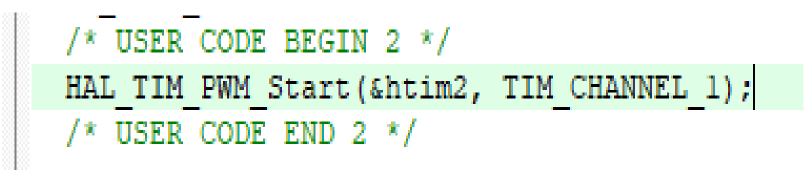
- Click **Open Project**



- 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(&htim2, TIM_CHANNEL_1);
```

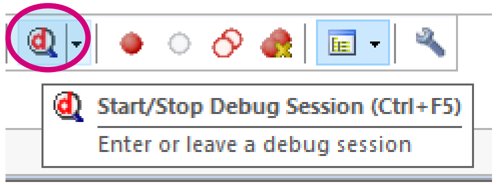


life.augmented

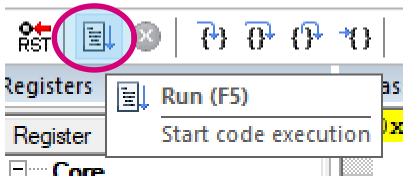
Build the Project

- Click the “Build” button

- Click the “Start/Stop Debug Session” button



- Click “Run” button



- Enjoy the flashing LED (LD4)!
  - LD4 is flashing using the PWM Timer

life.augmented

Lab: NVIC + External Interrupts

life.augmented

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.

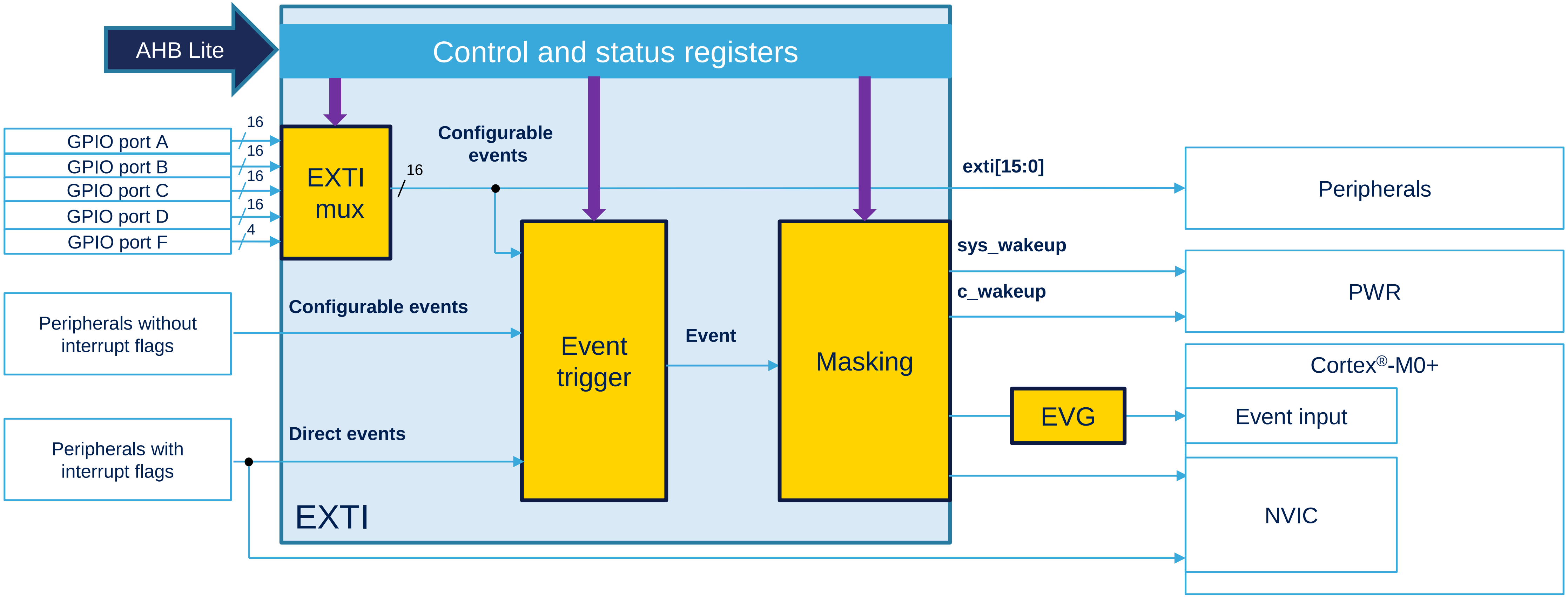
EXTI - Key features

- 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, PVD, and COMPx

- Direct events
  - Status flag provided by related peripheral
  - Linked to:
    - RTC, TAMP, I2C1, USARTx, CEC, LPUART1, LPTIMx, LSE_CSS and UCPDx

EXTI - block diagram



EVG: EVent Generator

EXTI - lines mapping

EXTI line	Line source	Line type
0-15	**GPIO**	**Configurable**
16	**PVD output**	**Configurable**
17	**COMP1 output**	**Configurable**
18	**COMP2 output**	**Configurable**
19	**RTC**	**Direct**
20	Reserved	Direct
21	**TAMP**	**Direct**
22	Reserved	Direct
23	**I2C1 wakeup**	**Direct**
24	Reserved	Direct
25	**USART1 wakeup**	**Direct**
26	**USART2 wakeup**	**Direct**
27	**CEC wakeup**	**Direct**
28	**LPUART1 wakeup**	**Direct**
29	**LPTIM1**	**Direct**
30	**LPTIM2**	**Direct**
31	**LSE_CSS**	**Direct**
32	**UCPD1 wakeup**	**Direct**
33	**UCPD2 wakeup**	**Direct**

NVIC - Overview

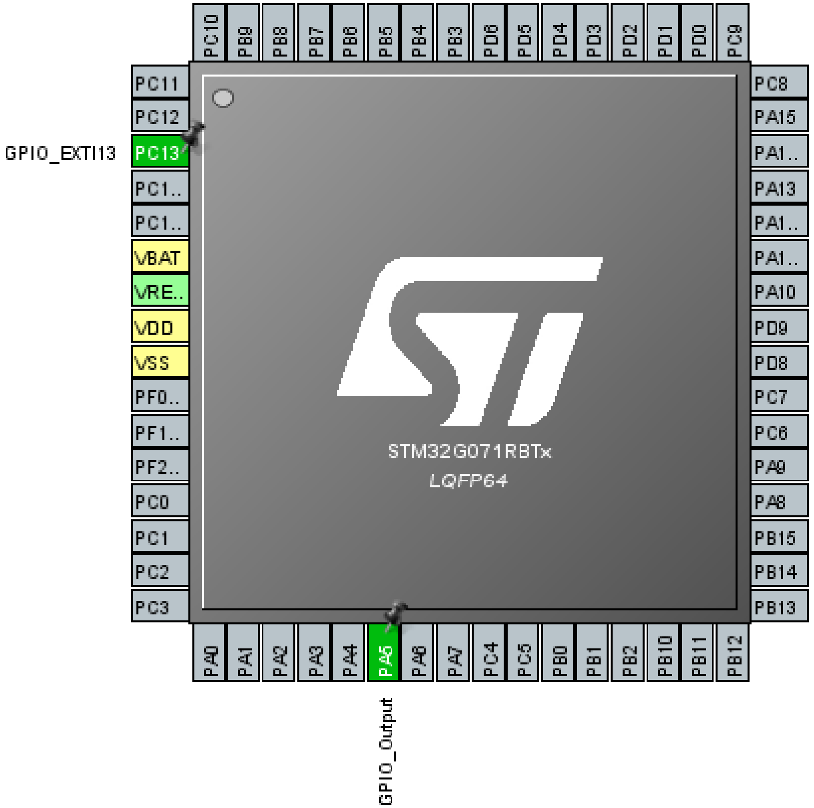
- The NVIC (Nested vector Interrupt Controller) is integrated in the Cortex®-M0+ CPU:
  - 32 maskable interrupt channels
  - 4 programmable priority levels
  - Low-latency exception and interrupt handling
  - Power management control

Application benefits

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

Lab: Pinout Configuration

- Close Keil uVision5 IDE if it is open; Open the “**blinky**” STM32CubeMX project (using File->Recent Projects) and save it as a new project named “**exti**”.
- Add configuration of the IO that is connected to the User Button (connected to **PC13**) to toggle the LED LD4 (connected to **PA5**) on the STM32G0 Nucleo board.
- **PA5** is already configured as **GPIO** output push-pull.
- Left-click on **PC13** and set it to **GPIO_EXTI13** mode.



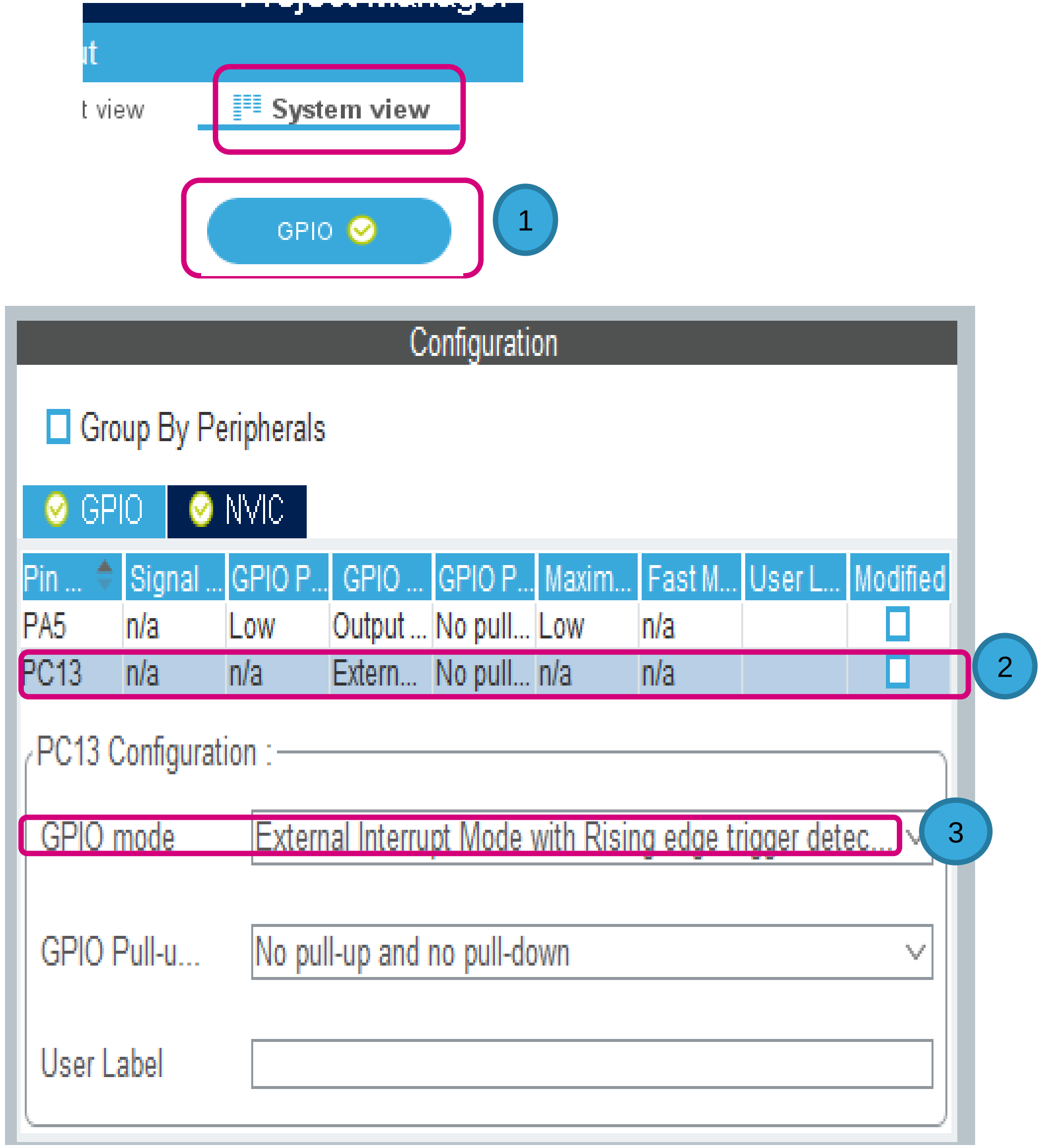
life.augmented

GPIO Configuration

- Select **GPIO** under **System View** 1

- Click on Pin Name **PC13** 2

- Make sure GPIO mode is “External **Interrupt Mode** with Rising edge trigger detection” 3

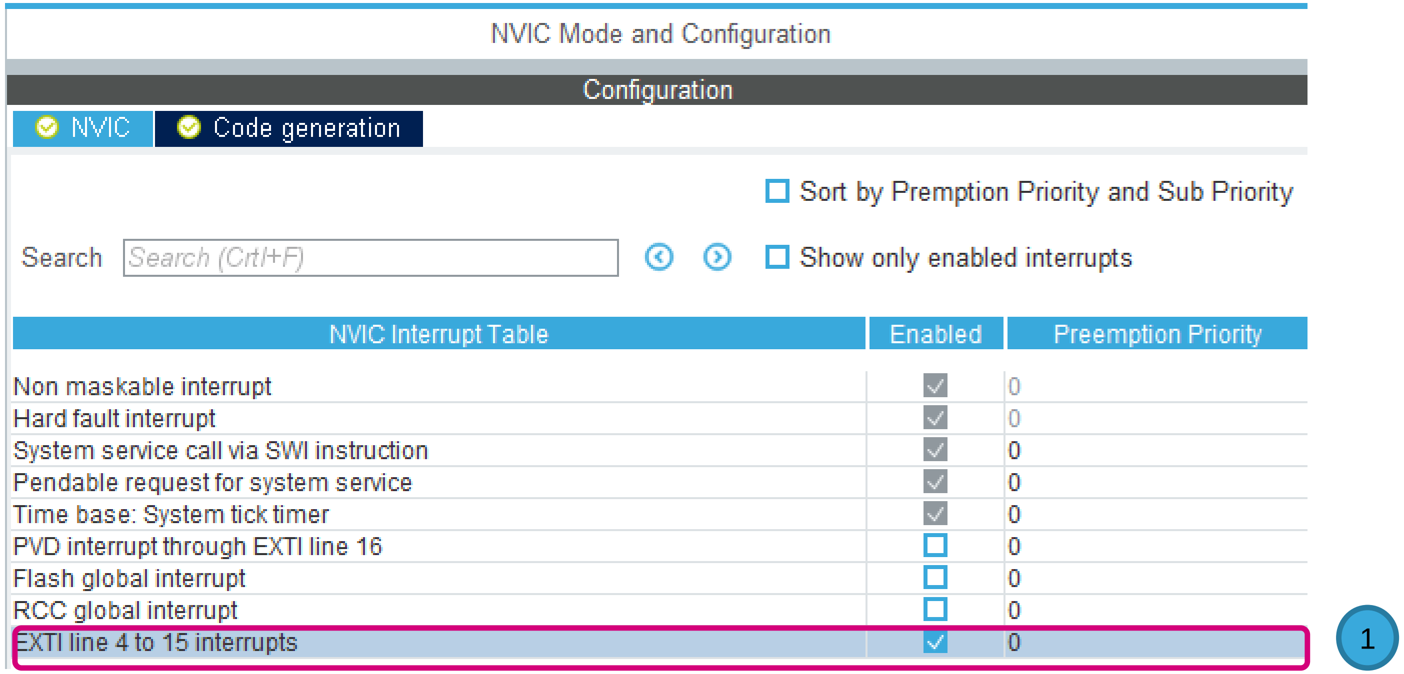


life.augmented

NVIC Configuration

- Select **NVIC** under **System View**

- Enable "**EXTI line 4 to 15 interrupts**" (by checking the box) 1

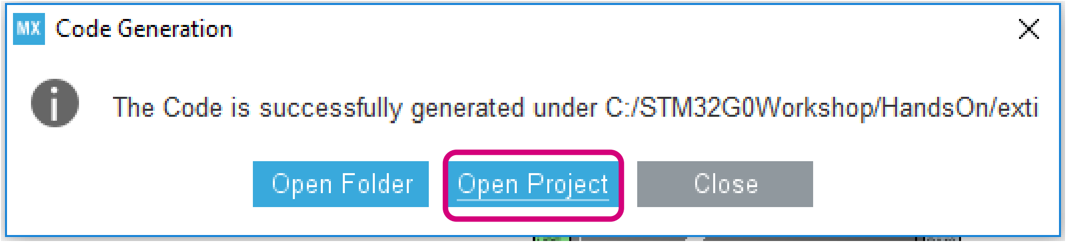


life.augmented

Generate Source Code

- **Generate Code**

- Click **Open Project**



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

```
uint8_t PC13_flag = 0;
```

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

Add EXTI Rising Edge Callback Function

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

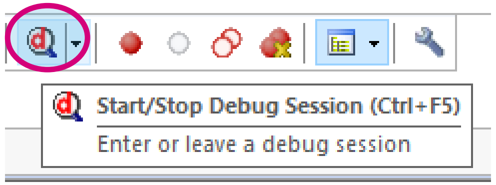
```
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  PC13_flag++;
  if ( ( PC13_flag & 0x01 ) == 0x01 )
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  }
}
```

```
/* 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 */
```

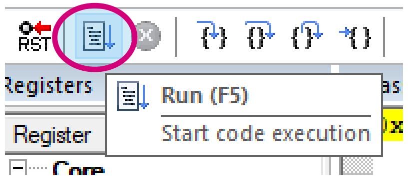
Build the Project

- Click the “Build” button

- Click the “Start/Stop Debug Session” button



- Click “Run” button



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

Lab: Low Power

life.augmented

Objective:

- In this lab we are going use the STOP 1 mode and wakeup from RTC which is configured to wakeup the STM32 every 5 seconds.
- When the STM32 wakes up it will turn on the LED (LD4) for one second and then go back to STOP mode.
- The MCU can also wake-up using the user button which is configured as EXTI.

Low Power Modes

Wakeup time	Mode	Consumption
	RUN (Range1) at 64 MHz	100 µA / MHz
	RUN (Range2) at 16 MHz	93 µA / MHz
	LPRUN at 2 MHz	90 µA / MHz
6 cycles	SLEEP at 16 MHz	42 µA / MHz
0.7 µs	LPSLEEP at 2 MHz	32 µA / MHz
4 µs	STOP 0	100 µA
5 µs	STOP 1	4.1 µA*
14 µs	STANDBY + SRAM	320 nA/670 nA*
14 µs	STANDBY	130 nA/480 nA*
258 µs	SHUTDOWN	40 nA/380 nA*
	VBAT	340 nA*

Typ @ VDD =3 V @ 25 °C

*** : with RTC***

FlexPowerControl

- Efficient running
- 7 low-power modes, several sub-modes
- High flexibility

Application benefits

- High performance
  - ➔ CoreMark score = 142.88
- Outstanding power efficiency

life.augmented

Stop modes

Lowest power modes with full retention, 5 µs wakeup time to 16 MHz

- SRAM and all peripheral registers retention
    - All high-speed clocks are stopped
    - Flash can be switched OFF
- LSE (32.768 kHz external oscillator) and LSI (32 kHz internal oscillator) can be enabled
- Several peripherals can be active and wake up from Stop modes
- System clock at wakeup is HSI16 (2 µs wakeup time on RAM, 5.5µs on FLASH not powered)
- Stop 1 is equivalent to Stop 0 with Main Regulator off, resulting in a smaller current consumption but longer wake up time

life.augmented

Stop 0 mode

97 µA @ 3.0 V

Wakeup time to 16 MHz:
- In SRAM: 2 µs
- In Flash ON: 2 µs
- In Flash OFF: 5.5 µs

Available peripherals

GPIO
DMA
BOR
PVD
USART
LP UART
I2C 1
I2C 2
SPI
ADC
DAC
COMP
Temp Sensor
Timers
LPTIM 1
LPTIM 2
IWDG
WWDG
Systick Timer
UCPD
RNG
AES
CRC
CEC

I/Os kept, and configurable

Zzz

Cortex M0+

Flash memory

SRAM (36 Kbytes)

Main regulator (MR)

Range 1 (up to 64 MHz)

Range 2 (up to 16 MHz)

Low Power regulator (LPR) up to 2 MHz

Backup domain

Backup Register (5x32 bits)

RTC & TAMPER

Wake-up event

NRST
BOR
PVD
RTC + Tamper
USART
LP UART
I2C 1
CEC
COMP
LPTIM 1
LPTIM 2
IWDG
GPIOs

Available clocks

HSI16
HSE
LSI
LSE

Active cell

Clocked-off cell

Cell in power-down

Available Periph and clock

life.augmented

Stop 1 mode

Flash memory not powered:
- w/o RTC: 1.3 µA @ 3.0 V
- w/ RTC: 4.1 µA @ 3.0 V

Flash memory powered:
- w/o RTC: 7.0 µA @ 3.0 V

Wakeup time to 16 MHz:
- In SRAM: 5 µs
- In Flash ON: 5 µs
- In Flash OFF: 9 µs

Available peripherals

GPIO
DMA
BOR
PVD
USART
LP UART
I2C 1
I2C 2
SPI
ADC
DAC
COMP
Temp Sensor
Timers
LPTIM 1
LPTIM 2
IWDG
WWDG
Systick Timer
UCPD
RNG
AES
CRC
CEC

I/Os kept, and configurable

Zzz

Cortex M0+

Flash memory

SRAM (36 Kbytes)

Main regulator (MR)

Low Power regulator (LPR) up to 2 MHz

Backup domain

Backup Register (5x32 bits)

RTC & TAMPER

Wake-up event

NRST
BOR
PVD
RTC + Tamper
USART
LP UART
I2C 1
CEC
COMP
LPTIM 1
LPTIM 2
IWDG
GPIOs

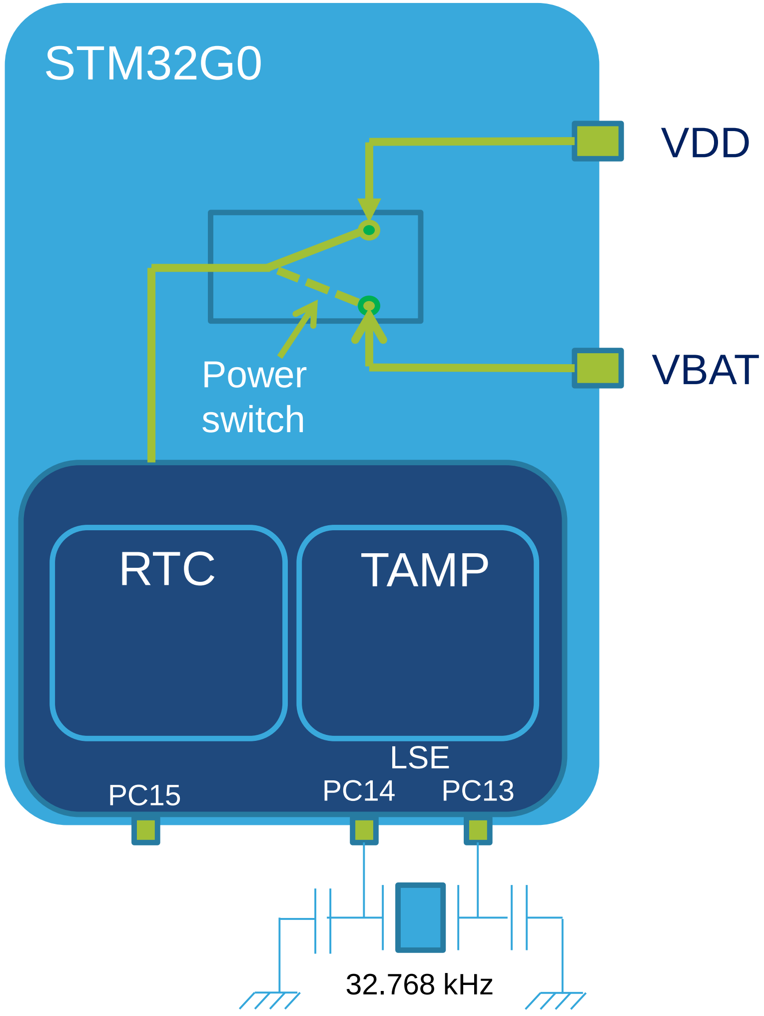
Available clocks

HSI16
HSE
LSI
LSE

Active cell | Clocked-off cell | Cell in power-down | **Available Periph and clock**

life.augmented

RTC - Overview



- The RTC provides an ultra-low-power hardware calendar with alarms, in all low-power modes
- It belongs to the Battery Backup Domain, so it is kept functional when the main supply is off and VBAT is present
- The TAMP peripheral features the backup registers and tamper detection

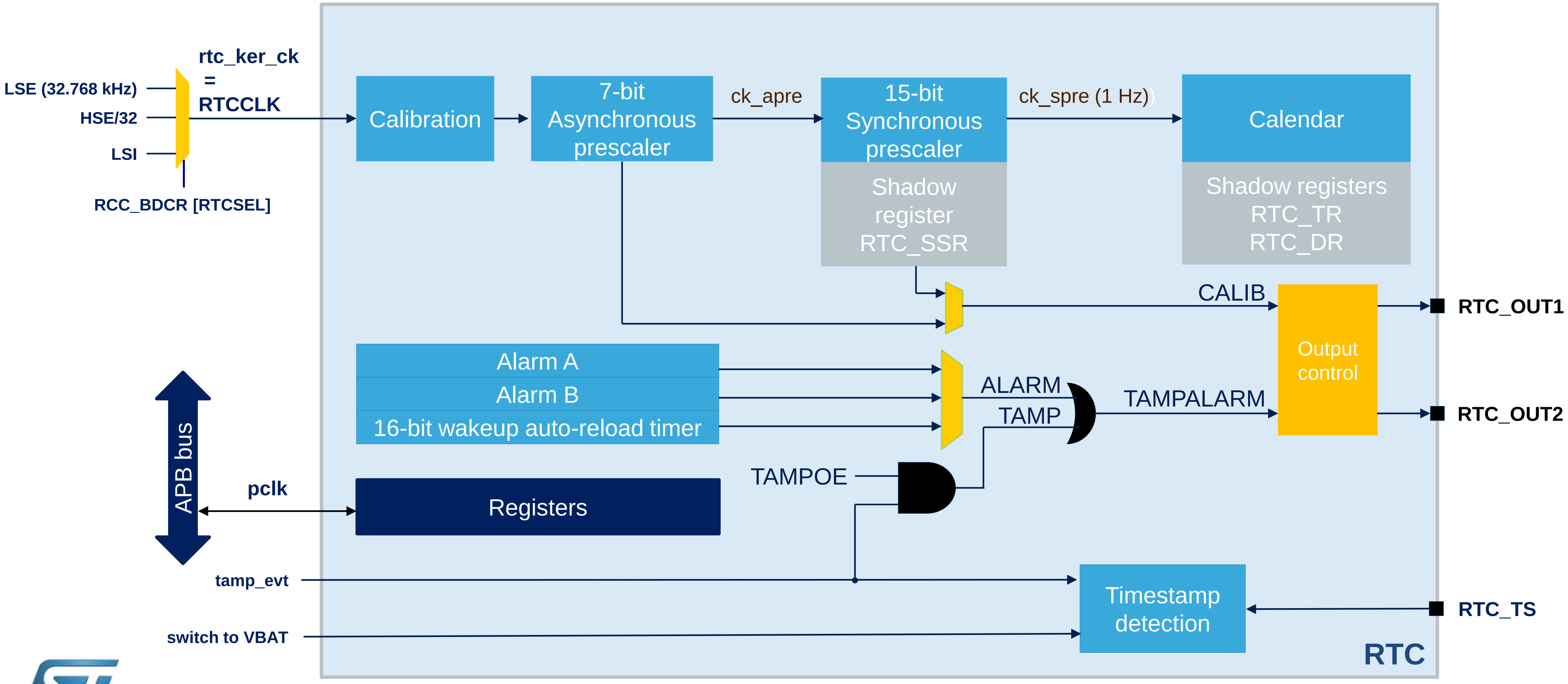
Application benefits

- Ultra-low power: 300 nA at 1.8 V
- Hardware BCD calendar to reduce software load

RTC - Key features

- Sub-seconds, seconds, minutes, hours, week day, date, month, year in BCD format
- “On the fly” programmable daylight savings compensation
- Two programmable alarms with wakeup interrupt function
- A periodic event with programmable resolution, triggering wakeup interrupt
- A reference clock source (50 or 60 Hz) can be used to enhance the calendar precision
- Digital calibration circuit to achieve 0.95 ppm accuracy
- Timestamp feature which can be used to save the calendar content with sub-second precision (one event)

RTC - Block diagram



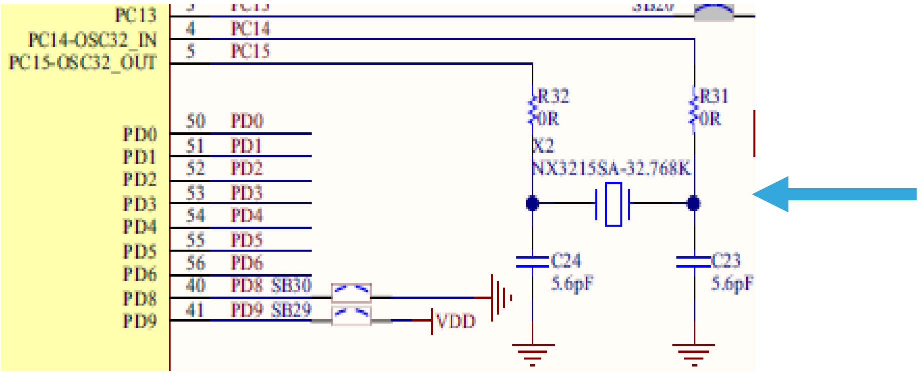
- RTC not affected by system reset when clocked by LSE

Lab: Low Power

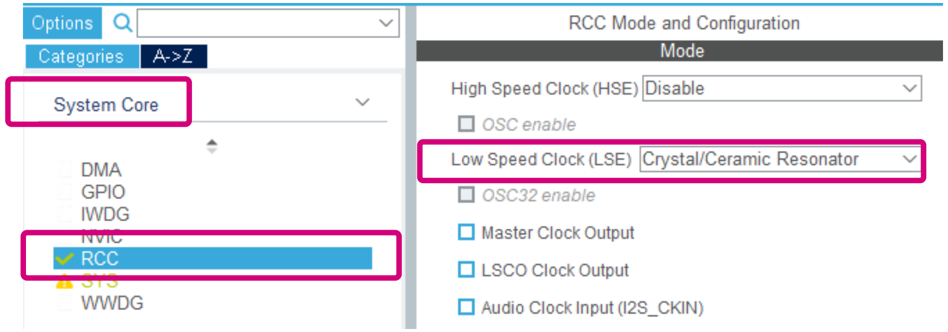
- Close Keil uVision5 IDE if it is open; In STM32CubeMX open the "**exti**" STM32CubeMX project save it as a new project like "**lowpower**".

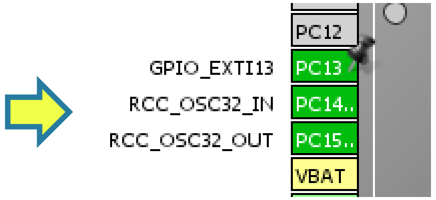
Enable LSE (Low Speed External) Clock

- We are going to use the 32 KHz Crystal that is on the Nucleo board (see schematic below) to clock the RTC:



- In the **Pinout & Configuration** tab, expand **RCC** (in **System Core**) and choose Crystal/Ceramic Resonator for Low Speed Clock (LSE) clock:

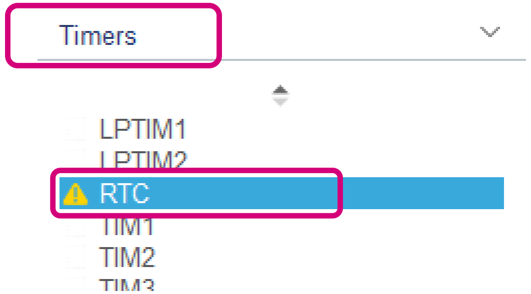




life.augmented

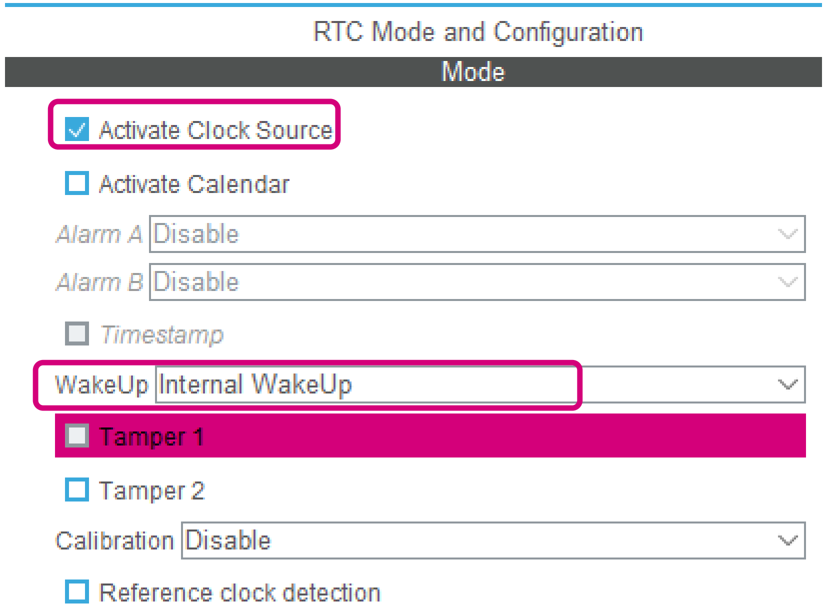
Enable and Configure the RTC

- In the **Pinout & Configuration** tab, under **Timers**, expand **RTC**



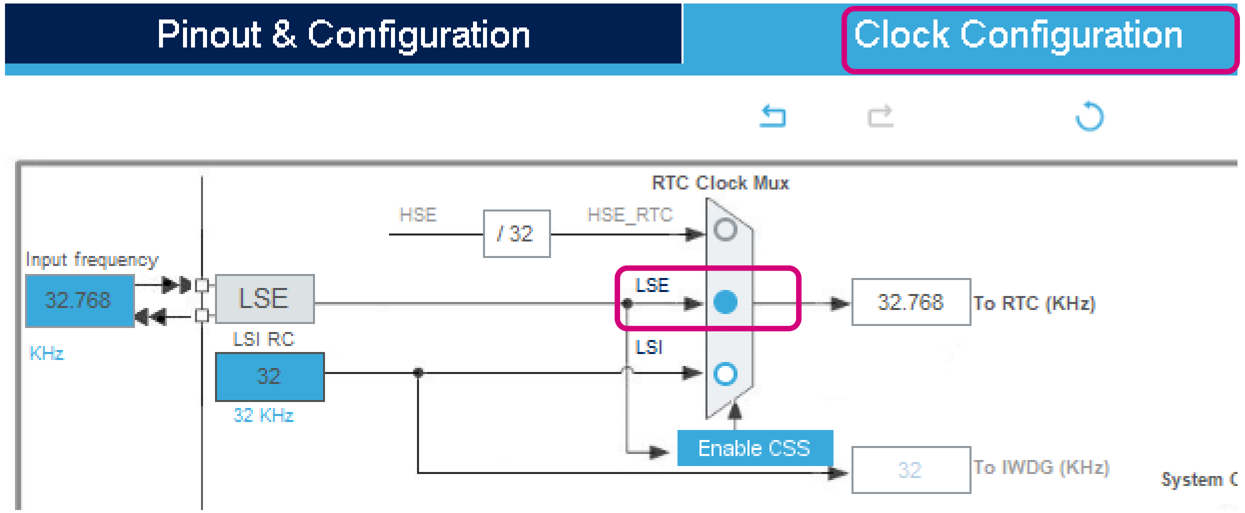
- Check the **Activate Clock Source**

- Select **Internal Wakeup** for the **Wakeup** mode



Choose RTC clock source

- In the **Clock Configuration** tab, select **LSE** as input clock for **RTC**



Note: For applications that do not require precise RTC timings the LSI (Low Speed Internal RC) can be used to clock the RTC

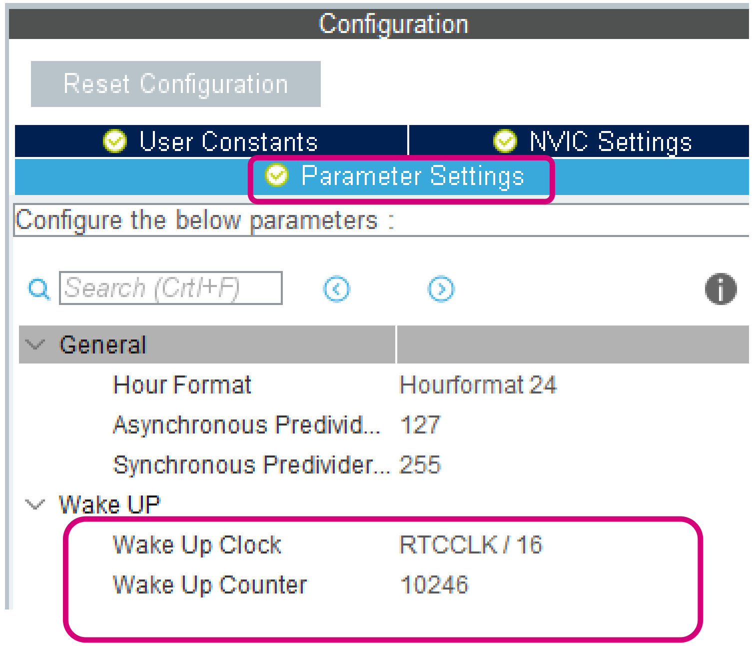
Wakeup Counter Calculation:

- To configure the wake up timer for 5s, the WakeUpCounter should be set to 10246 as calculated below:
- With RTC Clock set to RTCCLK /16
- Wakeup Time Base = RTC_PRESCALER / LSE = 16 /(32.768KHz) = 0.488 ms
- Wakeup Time = Wakeup Time Base * WakeUpCounter = 0.488ms * WakeUpCounter

==> WakeUpCounter = 5s /0.488ms = 10246

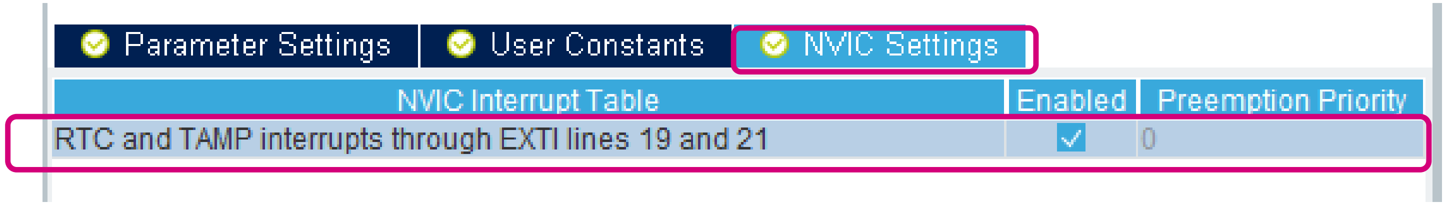
RTC configuration

- Based on previous calculation we will configure the RTC
- In the **Pinout & Configuration** tab, click on **RTC** (under Timers category)
- Enter the following configuration:

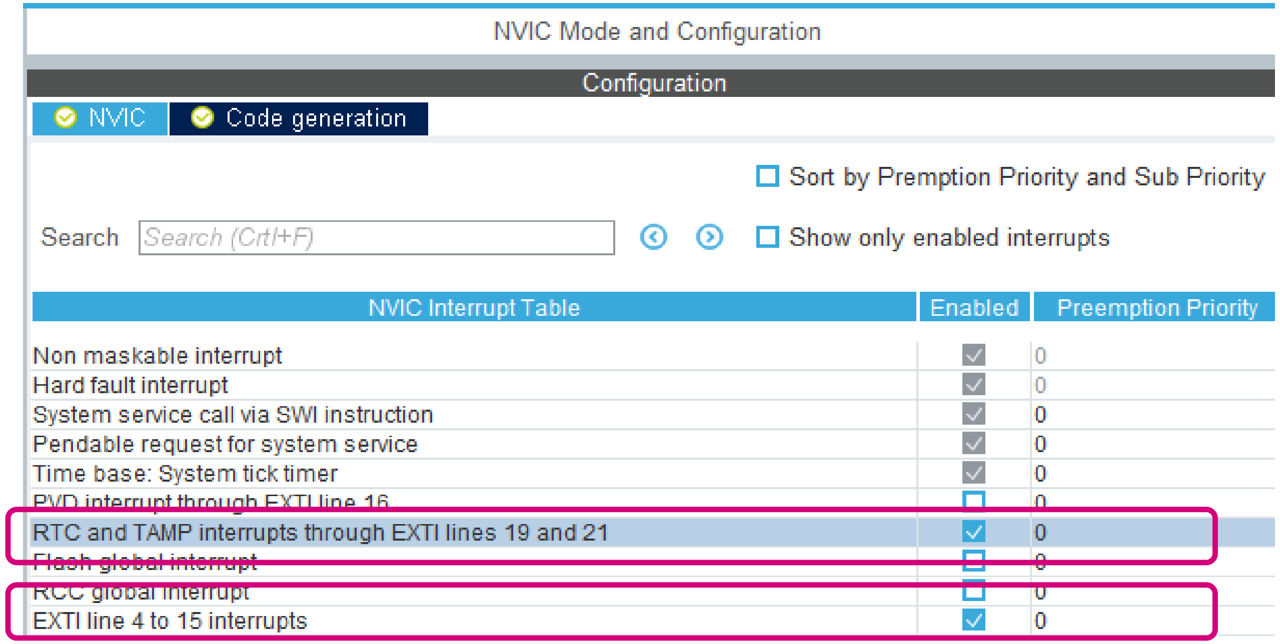


Enable Interrupts

- In the **Configuration** tab, go to **NVIC settings** and then enable the interrupt for **RTC**:



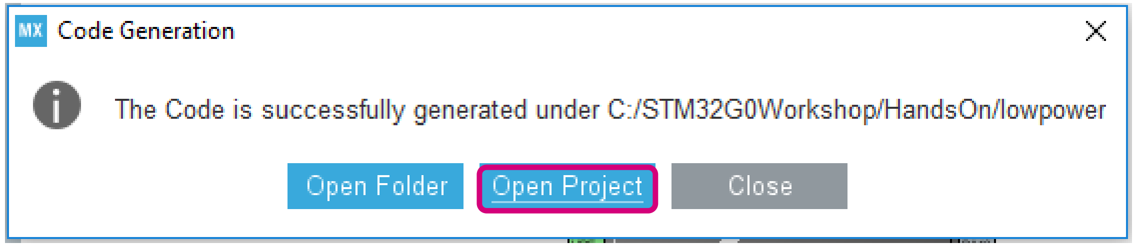
- In the “System View” in the NVIC, check that both RTC and EXTI[4…15] are enabled, if not re-enable them both:



life.augmented

- **Generate Code**

- Click **Open Project**



Add code – to main function

- Open the **main.c**, add the following code in the **while(1)** loop of the main function in the USER CODE WHILE section:

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

   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
   HAL_Delay(1000);
   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

   // enter STOP mode
   HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON, PWR_STOPENTRY_WFI);

   // reconfigure system clock
   SystemClock_Config();
   /* USER CODE END WHILE */

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

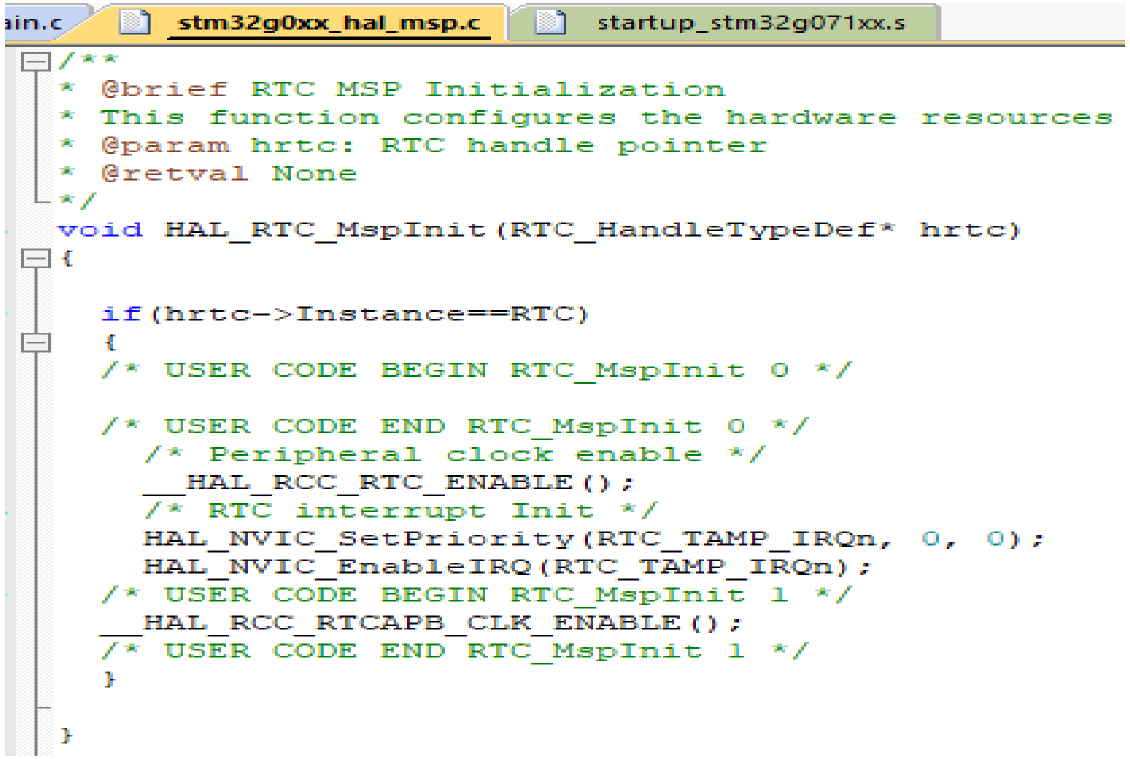
```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

    // enter STOP mode
    HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON, PWR_STOPENTRY_WFI);

    // reconfigure system clock
    SystemClock_Config();
    /* USER CODE END WHILE */

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

life.augmented

Add code – to init function

- Open the **stm32g0xx_hal_msp.c (under Application/User)**, Add the following line of code (marked in red below) to the msp init function HAL_RTC_MspInit():

```
void HAL_RTC_MspInit(RTC_HandleTypeDef* hrtc)
{

  if(hrtc->Instance==RTC)
  {
  /* USER CODE BEGIN RTC_MspInit 0 */

/* USER CODE END RTC_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_RTC_ENABLE();
/* RTC interrupt Init */
    HAL_NVIC_SetPriority(RTC_TAMP_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(RTC_TAMP_IRQn);
  /* USER CODE BEGIN RTC_MspInit 1 */
  __HAL_RCC_RTCAPB_CLK_ENABLE();
  /* USER CODE END RTC_MspInit 1 */
  }

}
```

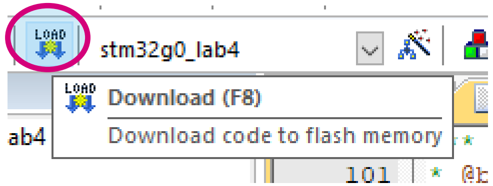


life.augmented

Build the Project

- Click the “Build” button; or use menu Project > Build target.

- Click the “**Load**” button (F8) to flash the code into the STM32 (not using the debug session because we are using low power modes)



- Press Reset on your board (black button) once the code is loaded and the application will work as follows:
  - RUN mode for 1 second (LD4 LED on)
  - STOP mode for 5 seconds (LD4 LED off) with wakeup by RTC
  - If during the STOP mode (LD4 LED off) you press the user button: the interrupt (EXTI) will wakeup from STOP mode

life.augmented

Lab: Estimation of power consumption

life.augmented

Lab: Estimation of power consumption

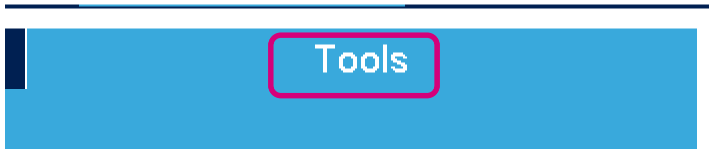
Objective:

- Use the Power tool inside the STM32CubeMX to estimate the average power consumption of the low power lab we just finished.

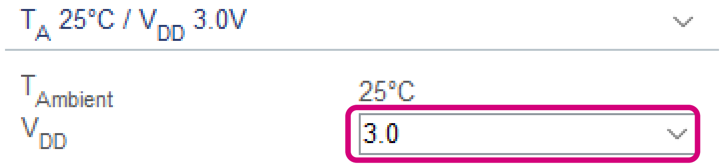
Power Supply and Power Source Selection

- Using the “**lowpower**” project in STM32CubeMX

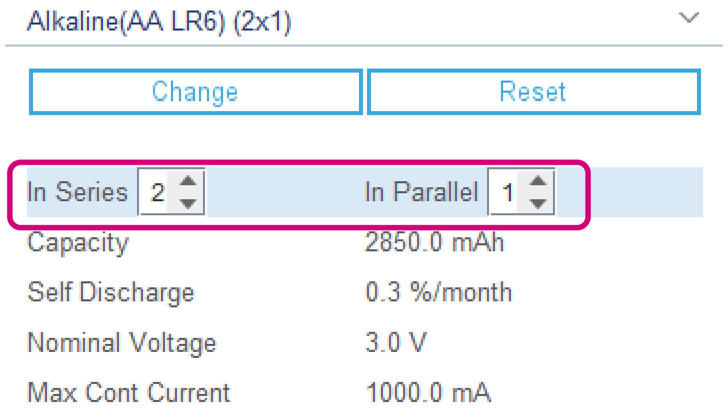
- Click on the **Tools** tab in STM32CubeMX



- Select **3V** for **VDD**



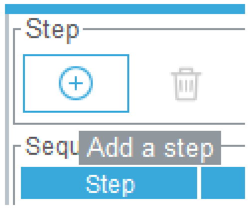
- In the **Battery Selection** section, select **AA** Alkaline batteries (2 in series, 1 in parallel) as the power source for the application



life.augmented

Adding a RUN mode step

- **Add a step to our power sequence:**
  - **Click: Step.. Add**



- **Configure a first step: RUN mode**

1
- Mode: Run
- Power Range: Range2 -Medium
- Memory Fetch Type: Flash
- VDD: 3.0
- Voltage Source: Battery

2
- CPU Frequency: 16 MHz
- Clock Configuration: HSI

3
- Enable IPs from Pinout function

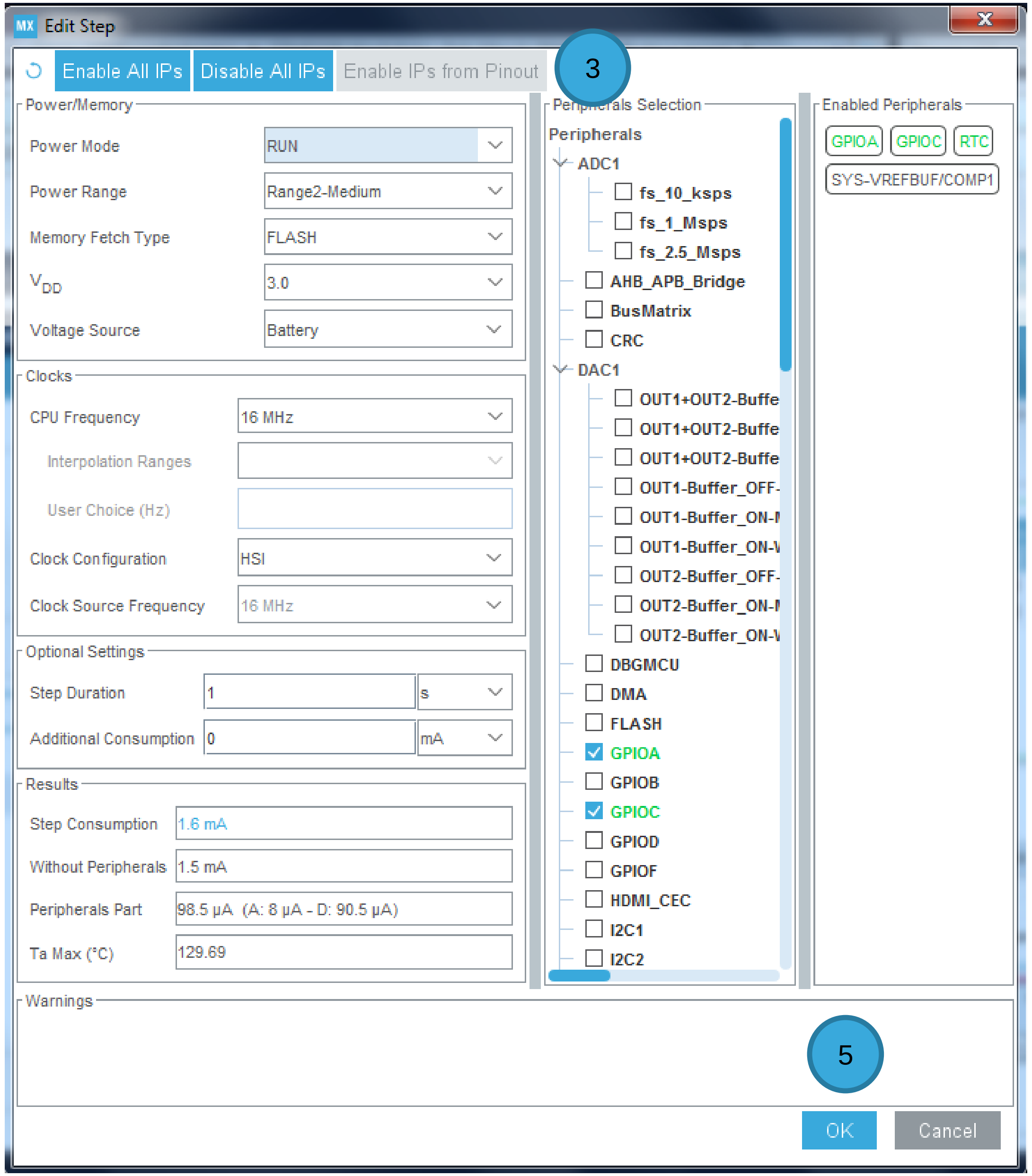
4
- Duration: 1 second

5
- Click “Add”

1

2

4

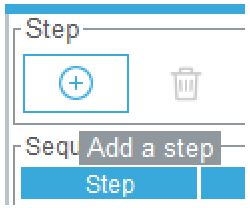


Resulting step consumption should be 1.6mA

life.augmented

Add a STOP1 mode step

- **Add a step to our power sequence:**
  - **Click: Step.. Add**



- **Add a second step: STOP1 mode**

1
- Power Mode: STOP1
- Fetch Type: Flash -PowerDown
- VDD: 3V

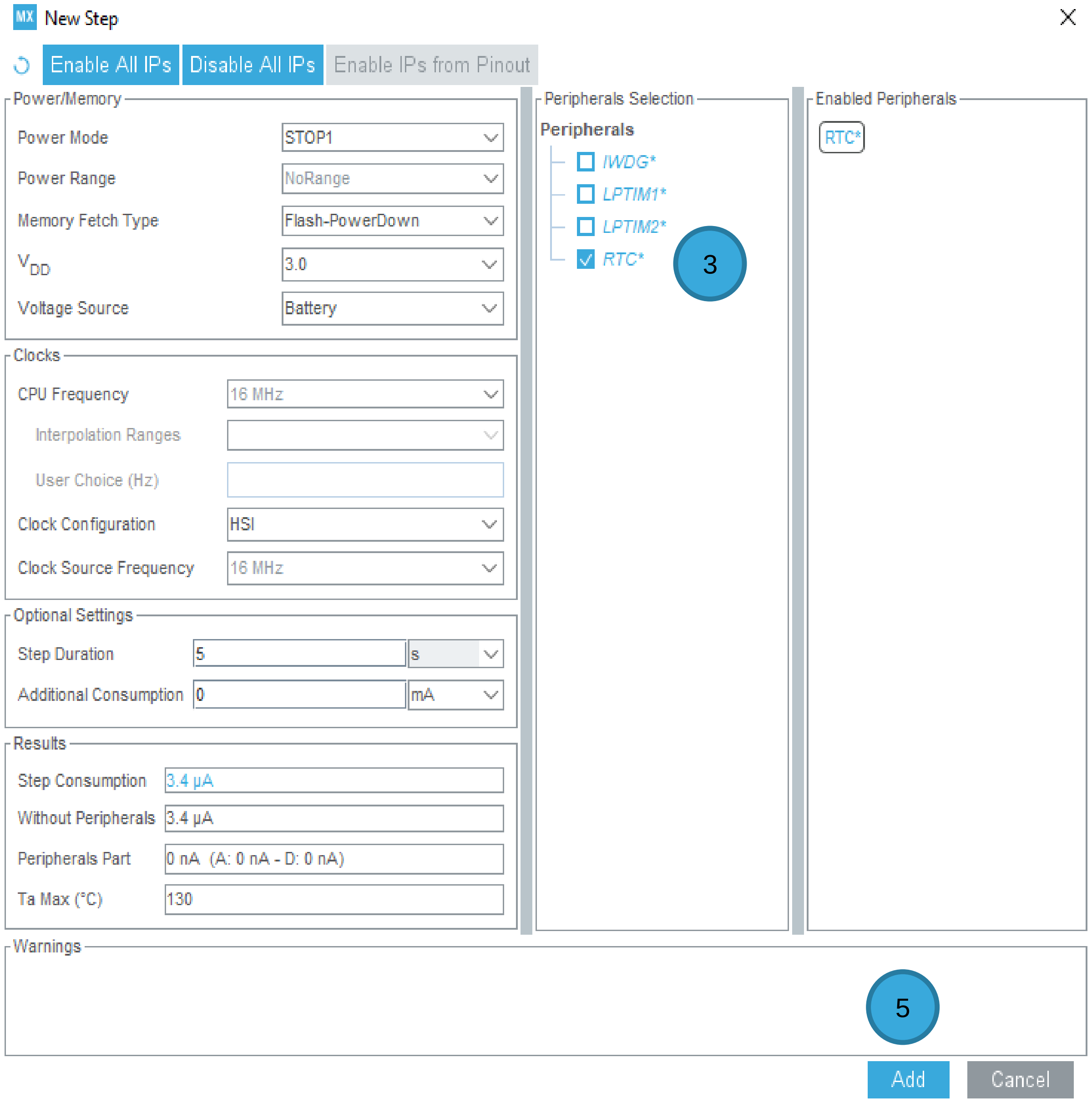
2
- Clocks HSI 16 MHz

3
- RTC enabled (to wakeup the system)

4
- Step Duration: 5 seconds

5
- Click “Add”

- Resulting step consumption should be 3.4 uA



life.augmented

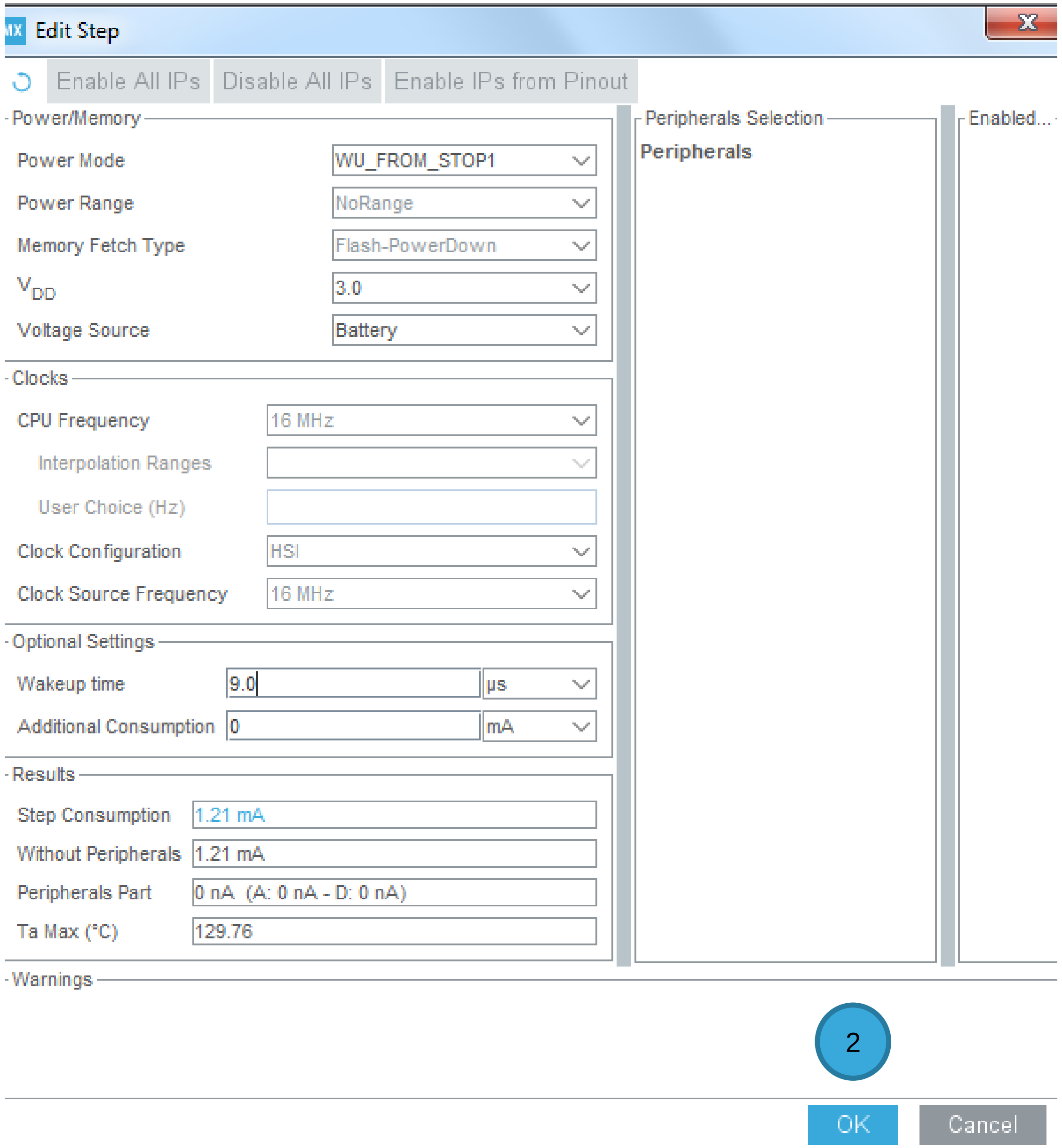
Add a Wakeup from STOP1 mode step

- **Add a last step: Wakeup from STOP1 mode**

1
- Power Mode: WU_FROM_STOP1
- VDD = 3V
- Voltage source: Battery

2
- Click “Add”

- Resulting step consumption should be 1.21 mA



Average Current Consumption Result

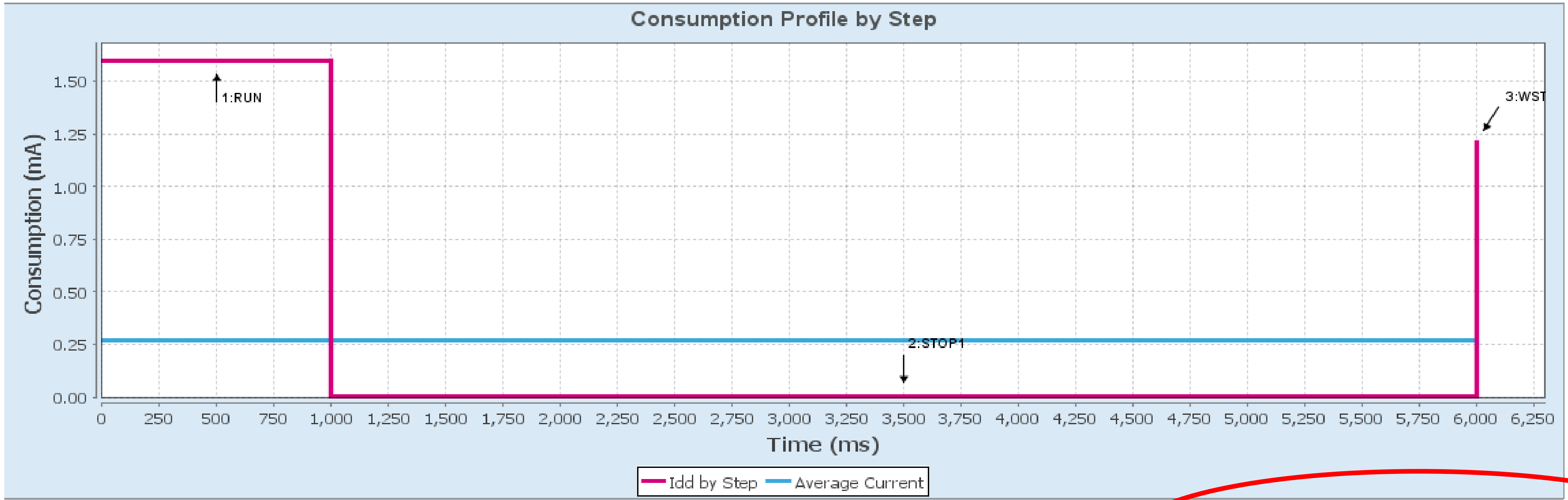
- **Note: the Current consumption numbers are for the MCU only.**

Sequence Table

Step	Mode	Vdd	Range/Scale	Memory	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	3.0	Range2-Medium	FLASH	16 MHz	HSI	GPIOA GPIOC ...	1.6 mA	1 s
2	STOP1	3.0	NoRange	Flash-PowerDown	16 MHz	HSI	RTC*	3.4 µA	5 s
3	WU_FROM_ST...	3.0	NoRange	Flash-PowerDown	16 MHz	HSI		1.21 mA	9.0 µs

Display

Plot: All Steps



Sequence Time / Ta Max 6 s / 129.69 °C

Battery Life Estimation 1 year, 1 month, 27 days

Average Consumption 269.5 µA

Average DMIPS 20.0 DMIPS

life.augmented

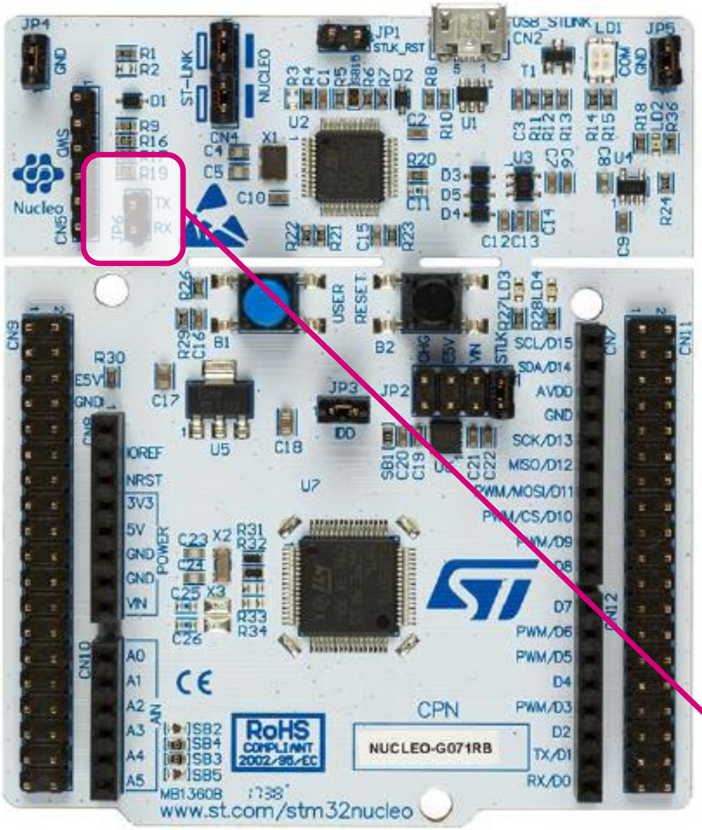
Optional Lab: printf() debugging using UART

Lab: printf() debugging using UART

Objective:

- Redirect Printf output to LPUART1 which is connected to the ST-LINK Virtual COM port on the Nucleo board
- Using a Terminal like Teraterm we can view the printf output.

printf() debugging settings overview



LPUART1 debug will be used via the ST-LINK Virtual-COM port

Set up additional GPIO / Clocks:
PA2 – LPUART1, "LPUART1-TX"
PA3 – LPUART1, "LPUART1-RX"
LPUART1 Clock = PCLK1 (64MHz)

LPUART1 settings:
Asynchronous Mode - 115200 N/8/1, No HW Flow control Tx/Rx,No advanced features

Teraterm Terminal will be used to display the printf output

LPUART1 is routed to the ST-LINK's USART, and brought via the USB Virtual-COM port class (SB16/18 located on the back on the board have been soldered)

life.augmented

STM32G0 USART/LPUART features

USART features	USART1/2	USART3/4	LPUART1
Hardware flow control for modem	x	x	x
Multiprocessor communication	x	x	x
Synchronous mode (Slave/Master)	x	x	-
Smartcard mode	x	-	-
Single wire half duplex communication	x	x	x
IrDA SIR ENDEC	x	-	-
LIN mode	x	-	-
Dual clock domain and wakeup from Stop mode	x	-	x
Receiver timeout	x	-	-
Auto baudrate detection	x	-	-
Driver enable	x	x	x
Data length	7, 8 and 9 bits		
TX/RX FIFO	x	-	x
TX/RX FIFO size (data word)	8	-	8

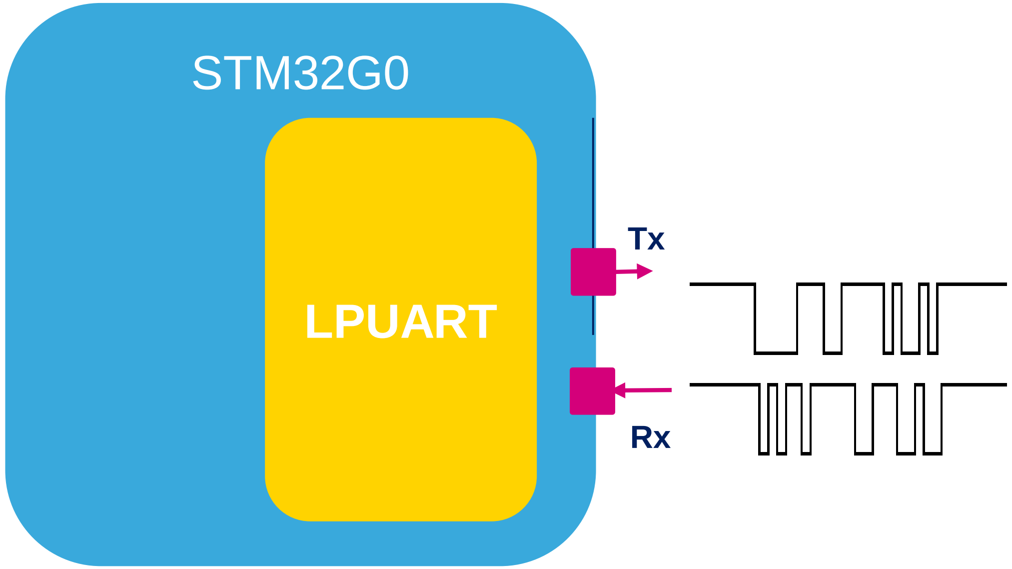
life.augmented

USART- FIFO

- Transmit FIFO (TXFIFO) and Receive FIFO (RXFIFO)
- Transmission/Reception even during stop modes
- FIFO mode is enabled/disabled by software
- TXFIFO and RXFIFO are each 8 data words in length
- Adjustable TXFIFO and RXFIFO interrupt request thresholds

LPUART - Overview

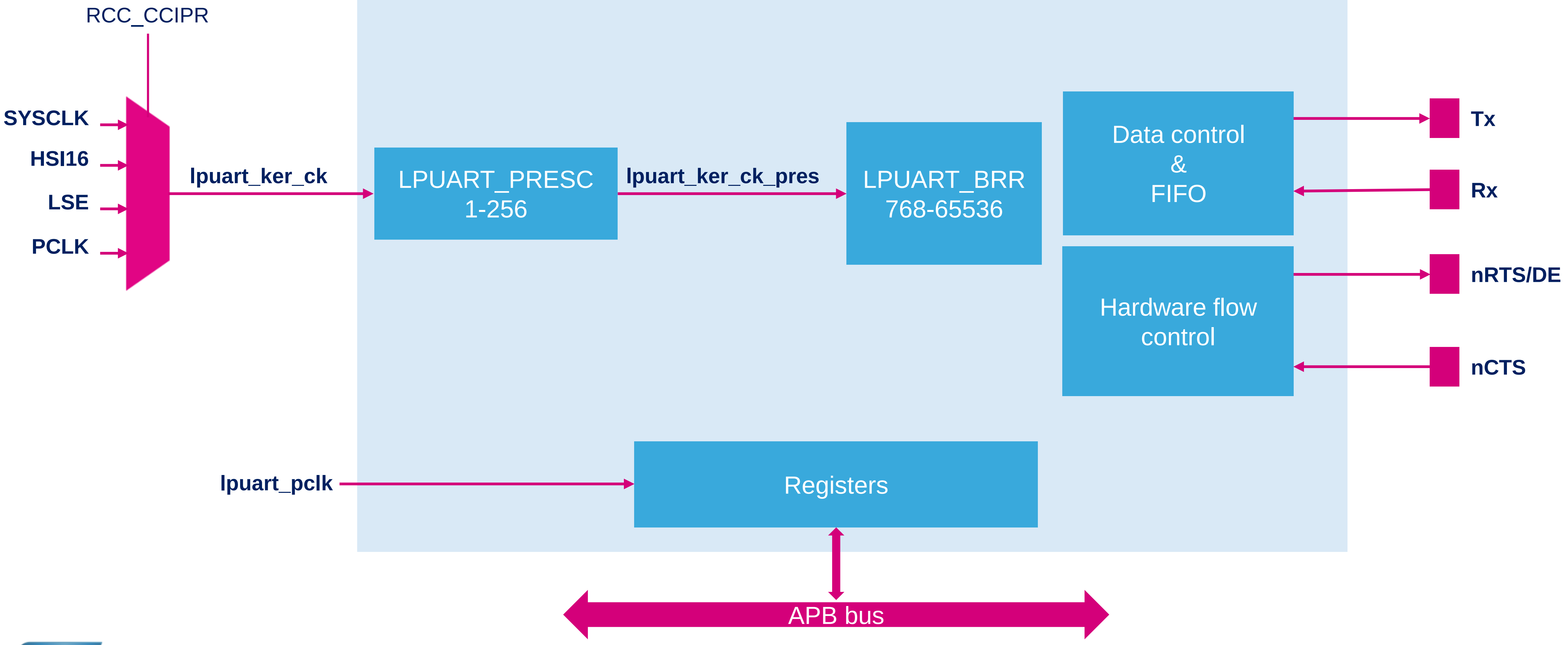
- LPUART (Low Power Universal Asynchronous Receiver/Transmitter)
  - Full UART communication at 9600 baud with wakeup from stop modes capability when using the low-speed 32.768 kHz external oscillator (LSE).
- Higher baud rates are available with other clock sources.



Application benefits

- Inexpensive communication link between devices
- Simple hardware, only a few pins needed
- Wakes from low-power STOP modes
- Transmit and Receive FIFOs, with capability to transmit and receive in stop modes.

LPUART Block diagram

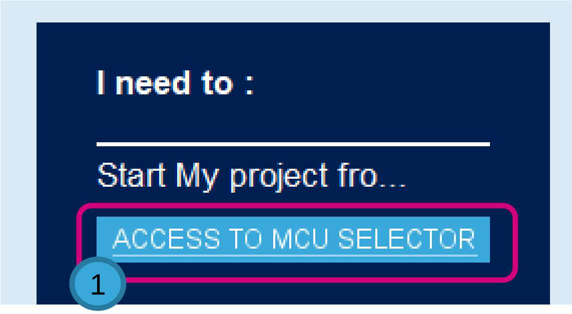


life.augmented

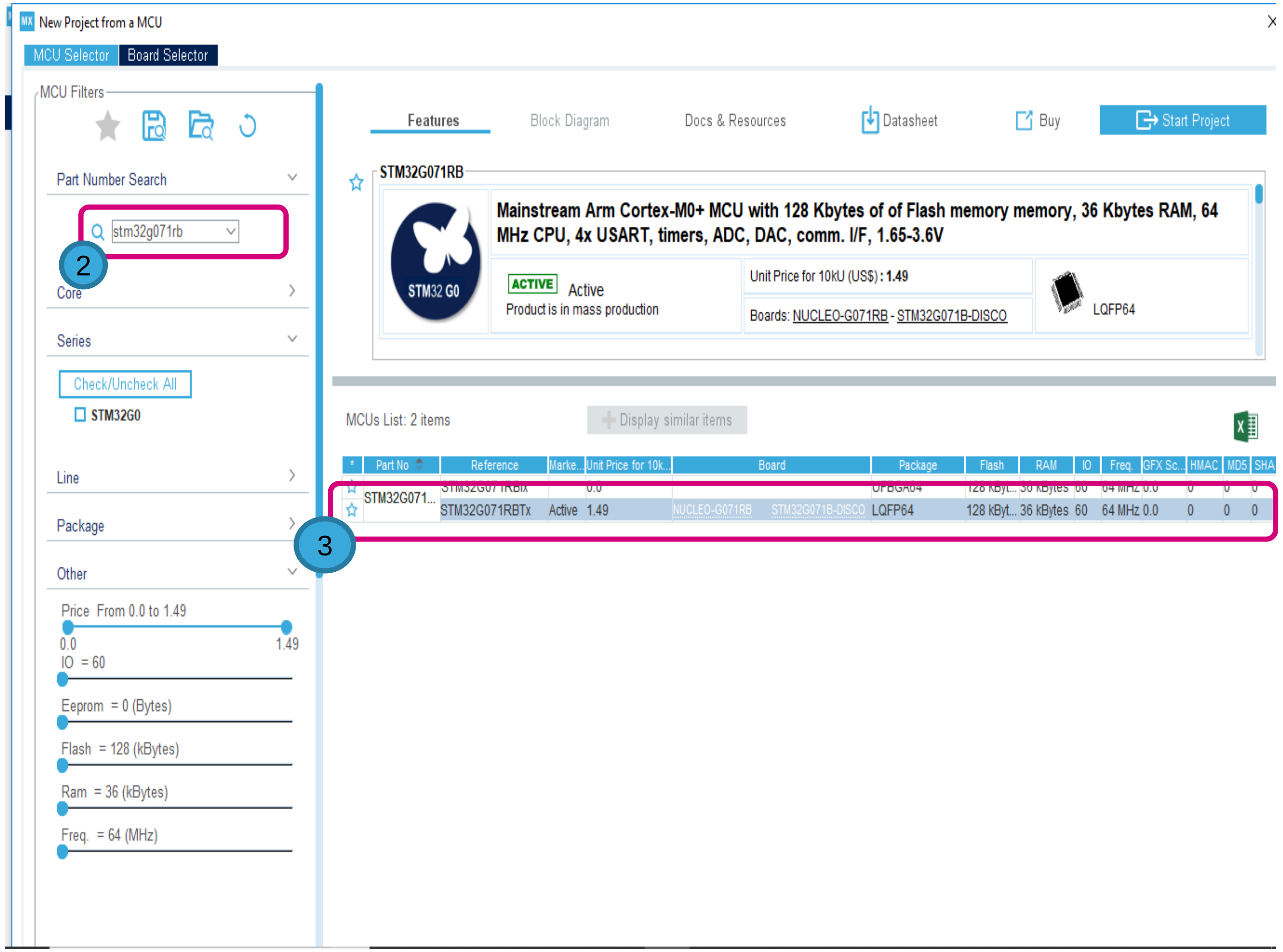
Create New Project

- In STM32CubeMX, click “Home”

- Click **Access To MCU Selector** 1



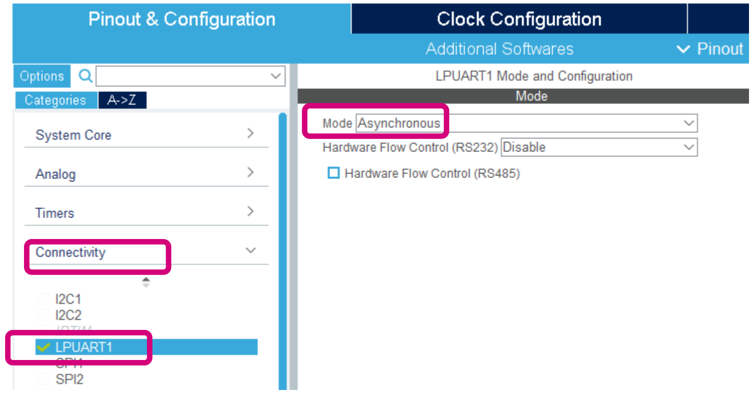
- Select **STM32G071RBTx** 2
  - **LQFP64, 128KB Flash**
- Double Click 3



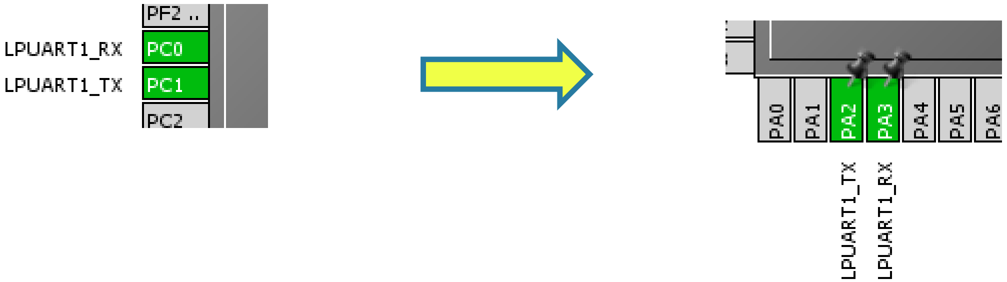
life.augmented

GPIO Configuration additions

- Click on **LPUART1** dialog (under Connectivity), and select **Asynchronous** mode:



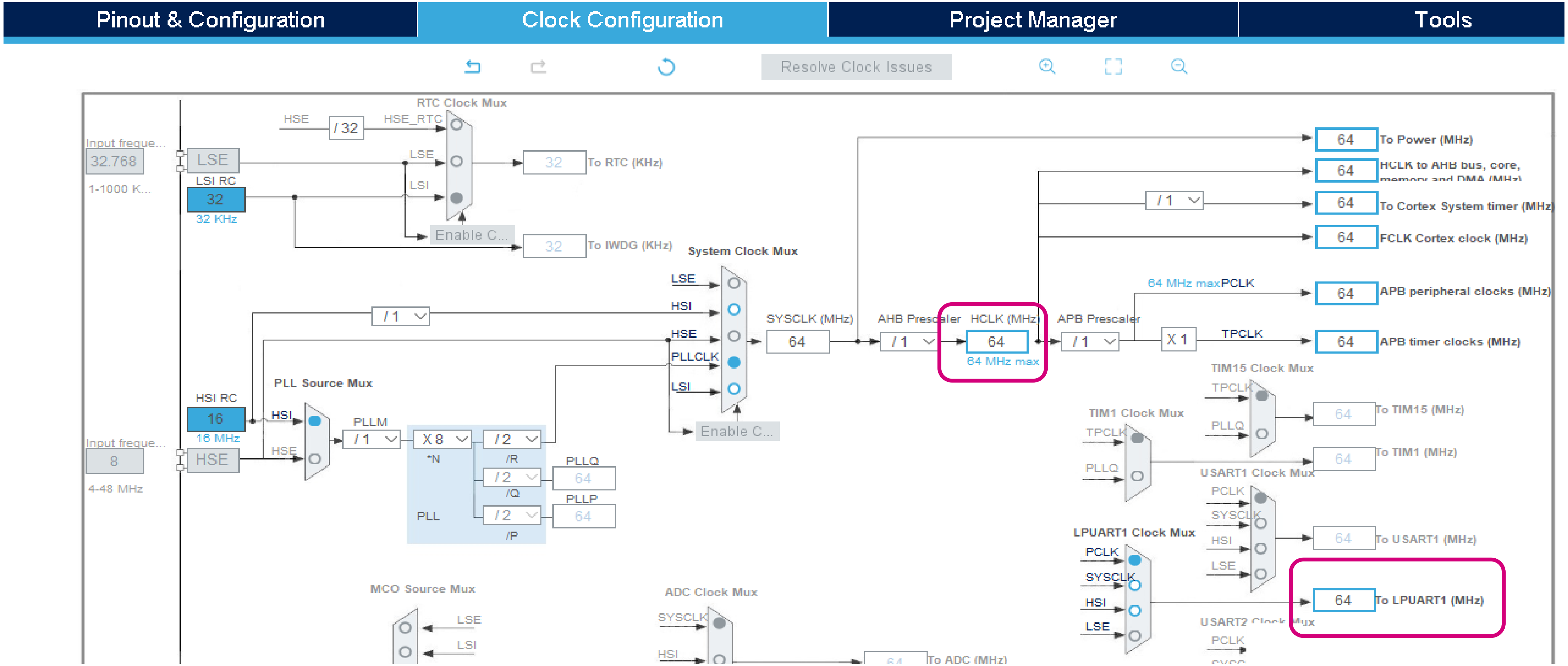
- Use PA2 & PA3 for Tx / Rx pins:
  - These are the alternate mapping pins (PC0/PC1 are default)
  - So need to remap



life.augmented

Clock Configuration

- Run the STM32G0 at 64 MHz for this lab, the LPUART1 clock also at 64 MHz.

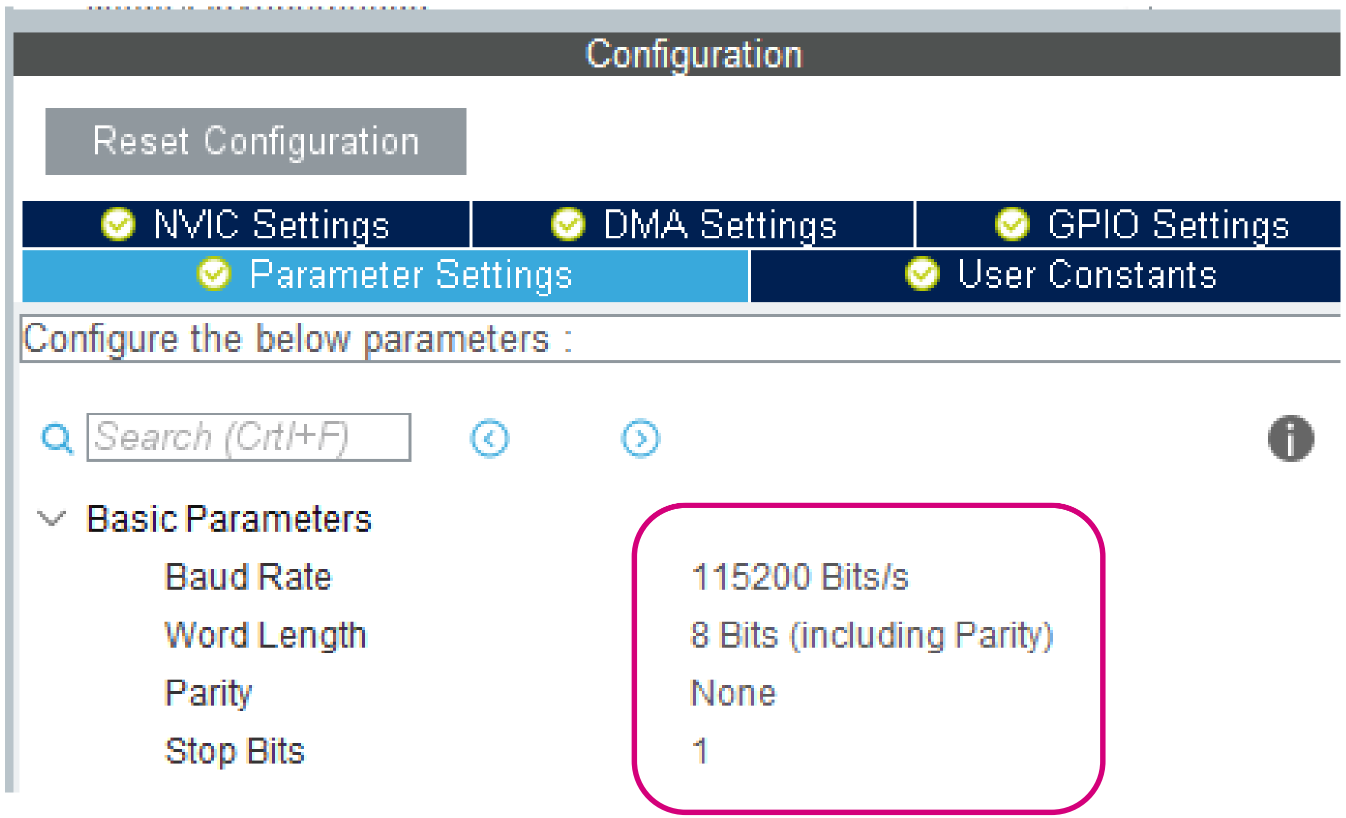


life.augmented

LPUART1 Configuration

- Click on the Configuration tab and select LPUART1

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



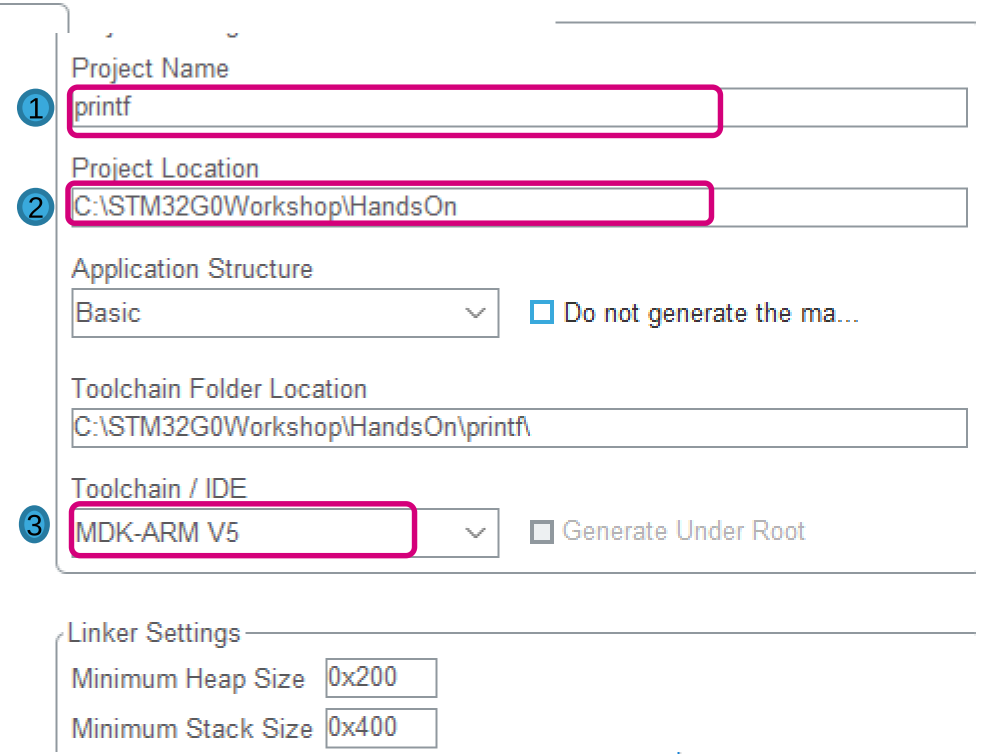
life.augmented

Generate Source Code

- Open **Project Manager**

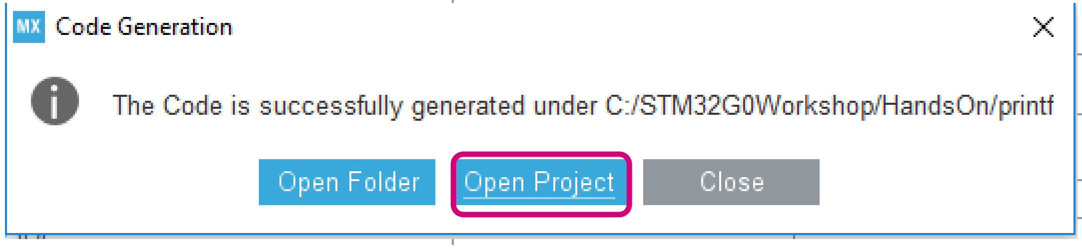
- Set the project name (**printf**) and the project location **(C:\STM32G0Workshop\STM32G0\HandsOn)**

- Set the IDE Toolchain to **MDK-ARM V5**



- **Generate Code**

- Click **Open Project**



life.augmented

Adding printf redirecting code in main.c

1- Add the stdio include:

```
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* USER CODE END Includes */
```

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

2- Add following code in the section below:

```
/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------*/
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
/* USER CODE END PFP */
```

```
/* USER CODE BEGIN PFP */
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
/* USER CODE END PFP */
```

3- Add following function in the section below:

```
/* USER CODE BEGIN 4 */
PUTCHAR_PROTOTYPE
{
  HAL_UART_Transmit(&hlpuart1, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
/* USER CODE END 4 */
```

```
/* USER CODE BEGIN 4 */
PUTCHAR_PROTOTYPE
{
  HAL_UART_Transmit(&hlpuart1, (uint8_t*)&ch, 1, 0xFFFF);
  return ch;
}
/* USER CODE END 4 */
```

life.augmented

Adding application code in main.c

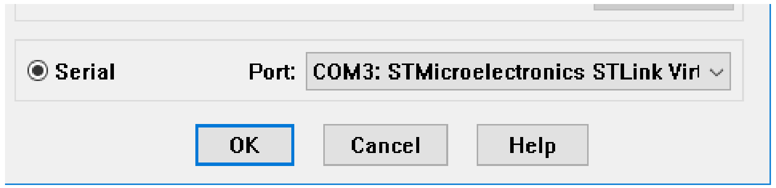
Add application code in main loop:

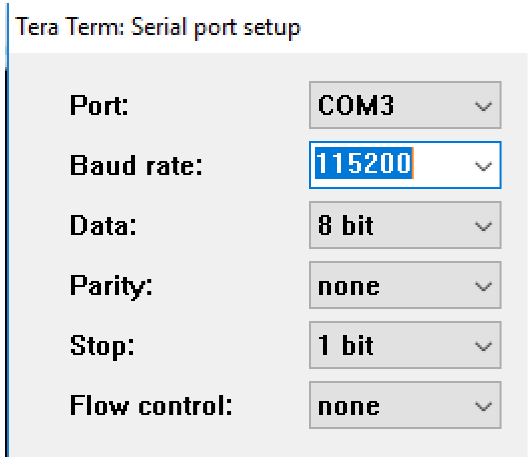
```
/* USER CODE BEGIN WHILE */
 while (1)
 {
     printf("** Hello World ** \n\r");
     HAL_Delay(1000);
 /* USER CODE END WHILE */
```

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

  /* USER CODE END WHILE */

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

Build the Project and run the application

- Click the “Build” button; or use menu Project > Build target.
- Click the “Start/Stop Debug Session” button
- Click “Run” button
- Open a Terminal emulator like Teraterm, using LPUART1 settings, connect ST-LINK Virtual COM port xx





- You should see the printf message being displayed.

STM32Cube Low Layer Drivers

life.augmented

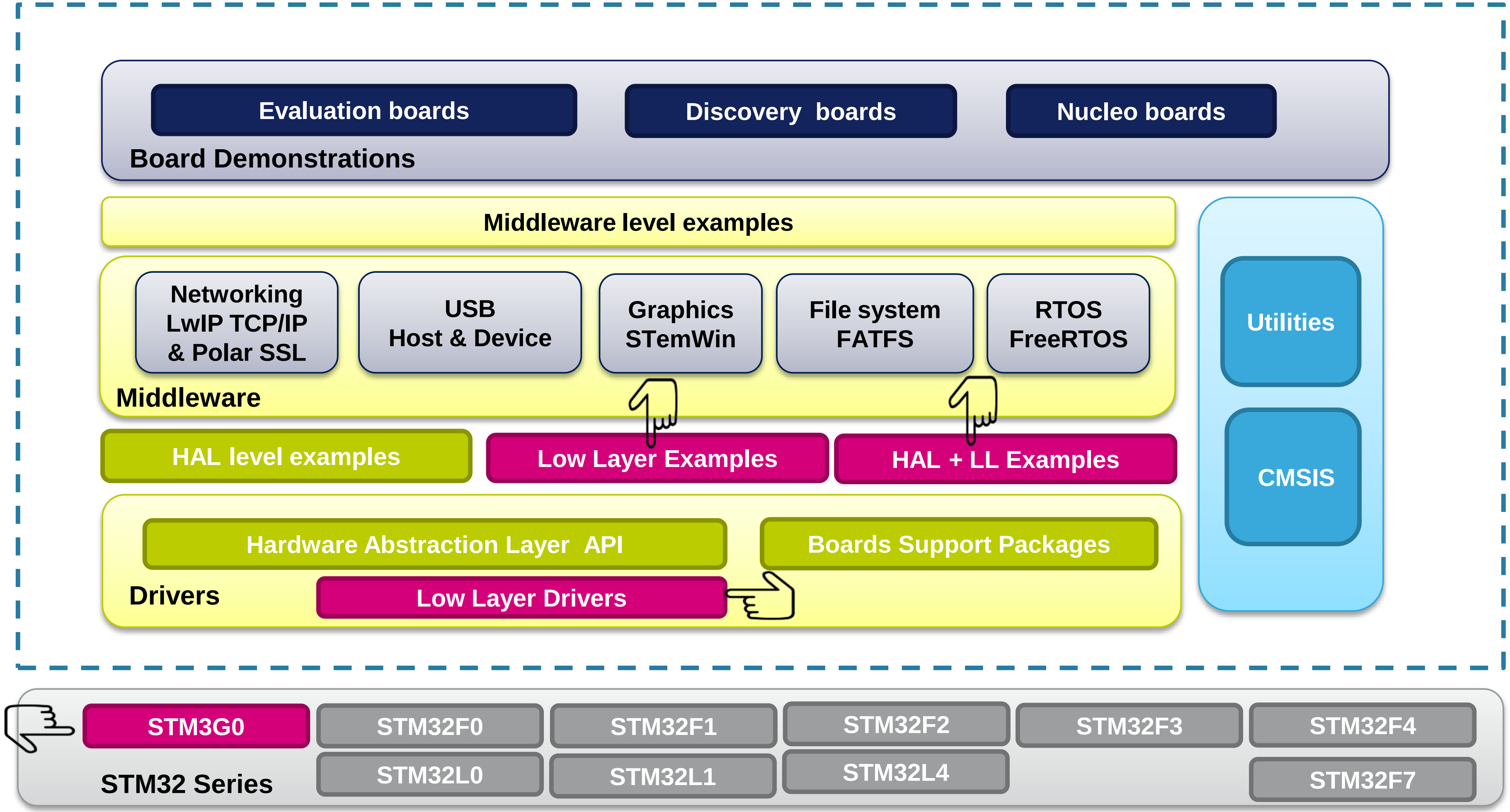
Overview

- 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 but less portability and require deep knowledge of the product/IPs specification

Low Layer (LL) Library Features

- The Low Layer (LL) Library offers the following services:
  - Set of static inline function 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
- The LL APIs are Not Fully Portable across the STM32 families; the availability of some macros depends on the physical availability of the relative feature on the product
- Most STM32 Peripherals covered
- Same standard compliancy as HAL (MISRA-C, ANSIC...)
- Low Layer (LL) is available in STM32CubeMX i.e. user can choose between HAL and LL by Peripheral

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

Optional Lab: Using the Low Layer (LL) Drivers

life.augmented

Lab: Using the Low Layer Drivers

Objective:

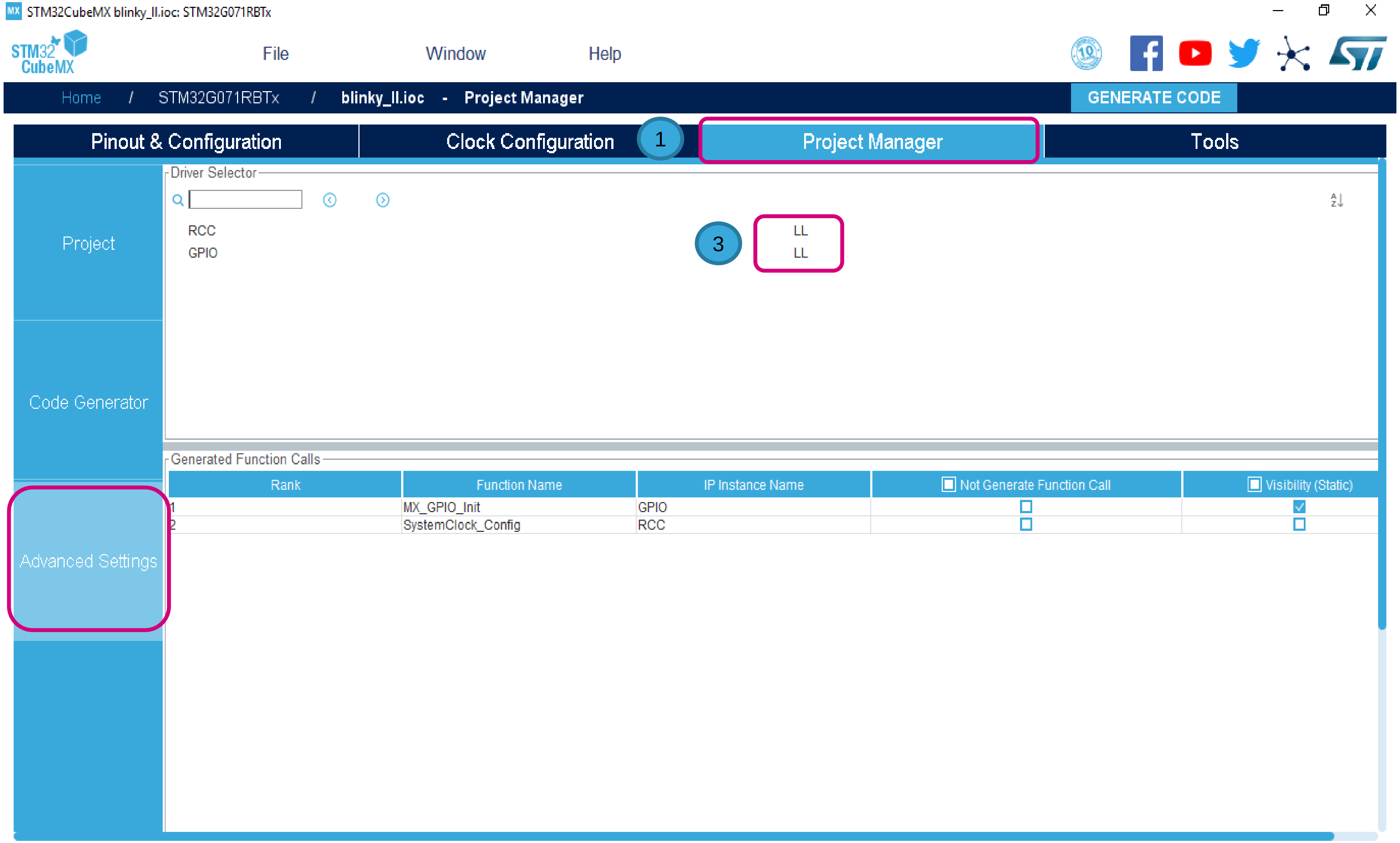
- Generate a project with STM32CubeMX using the Low Layer Drivers and check how much improvement we get compared to a HAL project in term of Flash and RAM usage

Project

- Close Keil uVision5 IDE if it is open; In STM32CubeMX Open the project ("**blinky**") and save it as a new project name like "**blinky_ll**" through File -> Save Project As

Configuration

- In Project Manager Tab 1
- In the Advanced Settings Tab 2
- Instead of HAL drivers select LL (Low Level) for both RCC and GPIO 3

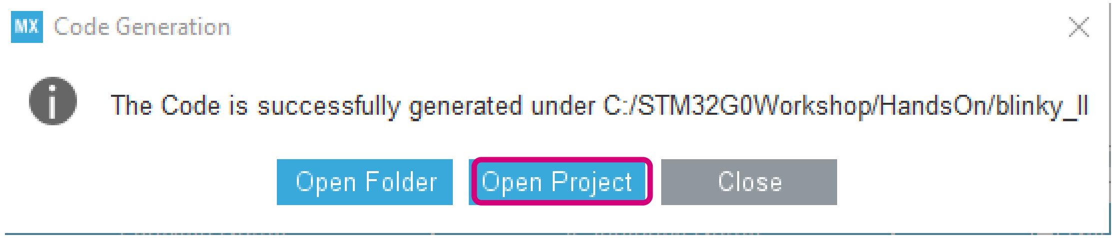


life.augmented

Generate Source Code

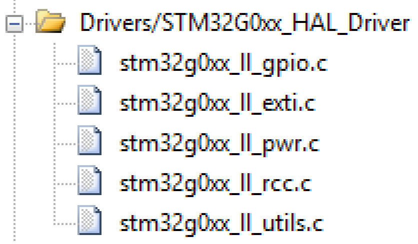
- **Generate Code**

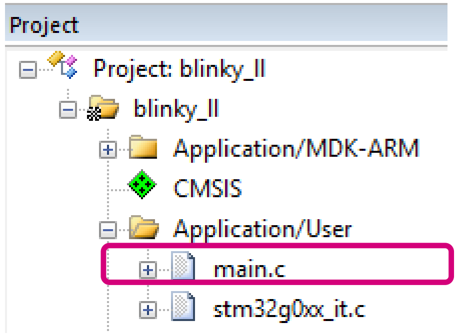
- Click **Open Project**



Step 4: Toggle The LED

- In Keil uVision5 IDE
- Expand the “Drivers” and notice that now the drivers are low layer (_ll):





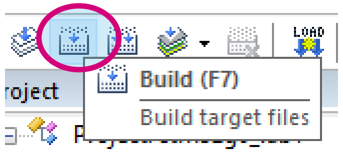
- Expand the file tree and open the **main.c** file
- Add the following code **inside the while(1) loop**
  - Add within “USER CODE BEGIN WHILE” / “USER CODE END WHILE” section (this will preserve your code after code regeneration)

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

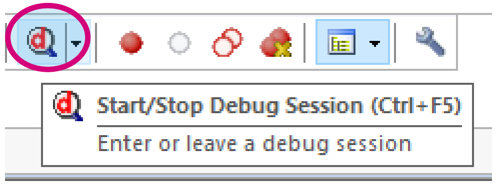
```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  LL_GPIO_TogglePin(GPIOA, LL_GPIO_PIN_5);
  // Delay 100 ms
  LL_mDelay(100);
  /* USER CODE END WHILE */
```

Build the Project

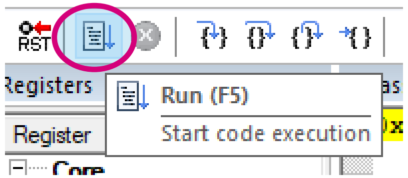
- Click the “Build” button



- Click the “Start/Stop Debug Session” button



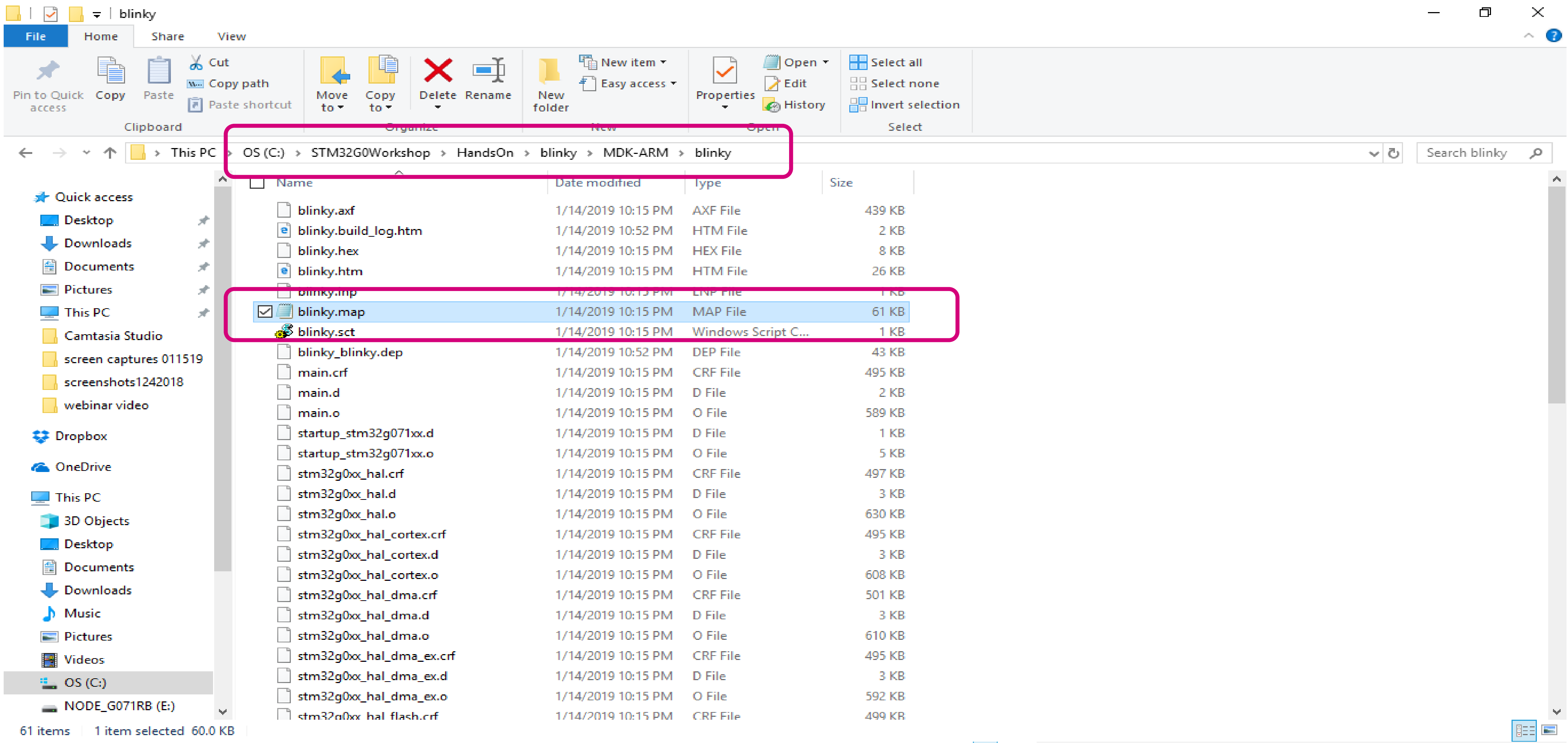
- Click “Run” button



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

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

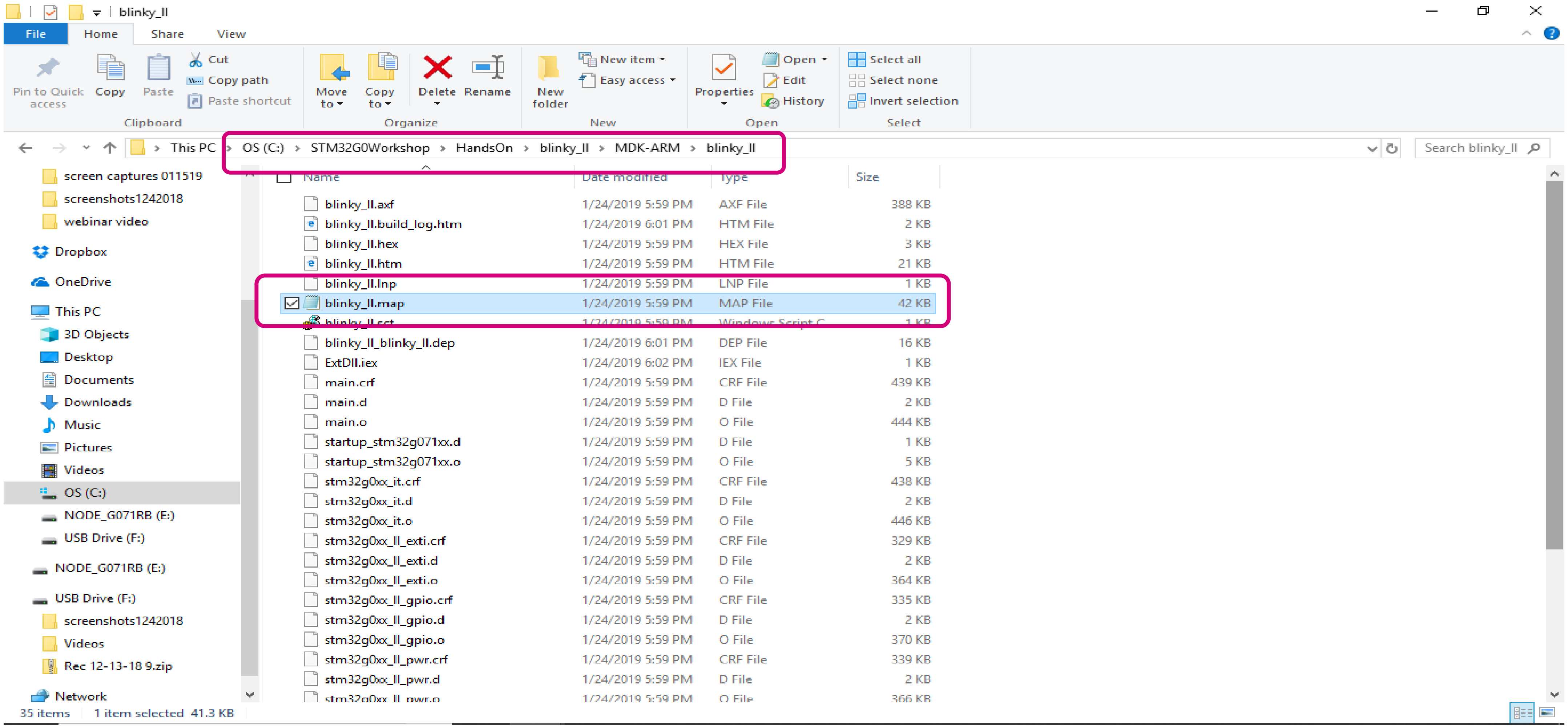
Use a text editor to open the map file for the “blink” project that is using HAL:”blinky.map” located here:



life.augmented

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

Open the map file for the “blinky_ll” project that is using LL: “blinky_ll.map” located here:



life.augmented

Let’s compare the Flash and RAM size first

Scroll to the bottom of the files

Using HAL:

blinky.map - Notepad

File Edit Format View Help

```
         44          0          0          0          0         72   uidiv.o

    ----------------------------------------------------------------------
        168         16          0          0          0        240   Library
Totals
          2          0          0          0          0          0   (incl.
Padding)

    ----------------------------------------------------------------------

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug   Library Name

        166         16          0          0          0        240   mc_p.l

    ----------------------------------------------------------------------
        168         16          0          0          0        240   Library
Totals

    ----------------------------------------------------------------------

==============================================================================


      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

      2532        174        284         16       1024     415174   Grand Totals
      2532        174        284         16       1024     415174   ELF Image
Totals
      2532        174        284         16          0          0   ROM Totals

==============================================================================

    Total RO  Size (Code + RO Data)                 2816 (   2.75kB)
    Total RW  Size (RW Data + ZI Data)              1040 (   1.02kB)
    Total ROM Size (Code + RO Data + RW Data)       2832 (   2.77kB)

==============================================================================
```

Using LL:

blinky_ll.map - Notepad

File Edit Format View Help

```
         30          0          0          0          0          0   handlers.o
         36          8          0          0          0         68   init.o
         36          0          0          0          0        100   memseta.o
         44          0          0          0          0         72   uidiv.o

    ----------------------------------------------------------------------
        168         16          0          0          0        240   Library Totals
          2          0          0          0          0          0   (incl. Padding)

    ----------------------------------------------------------------------

      Code (inc. data)   RO Data    RW Data    ZI Data      Debug   Library Name

        166         16          0          0          0        240   mc_p.l

    ----------------------------------------------------------------------
        168         16          0          0          0        240   Library Totals

    ----------------------------------------------------------------------

==============================================================================


      Code (inc. data)   RO Data    RW Data    ZI Data      Debug

       680         66        220          4       1028     376886   Grand Totals
       680         66        220          4       1028     376886   ELF Image
Totals
       680         66        220          4          0          0   ROM Totals

==============================================================================

    Total RO  Size (Code + RO Data)                  900 (   0.88kB)
    Total RW  Size (RW Data + ZI Data)              1032 (   1.01kB)
    Total ROM Size (Code + RO Data + RW Data)        904 (   0.88kB)

==============================================================================
```

life.augmented

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

- Using below configuration:
  - Heap Size = 512 Bytes / Stack Size = 512 Bytes
  - STM32Cube G0 1.0.0
  - uVision 5.26
  - CubeMX 5.0.1

	HAL Drivers	Low layer Drivers
read-only code memory (Bytes)	2816	900
read write data memory (Bytes) (*)	1040-1024(**)=16	1032-1024(**)=8

ROM Size divided by ~3

RAM Size reduction divided by 2

- **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

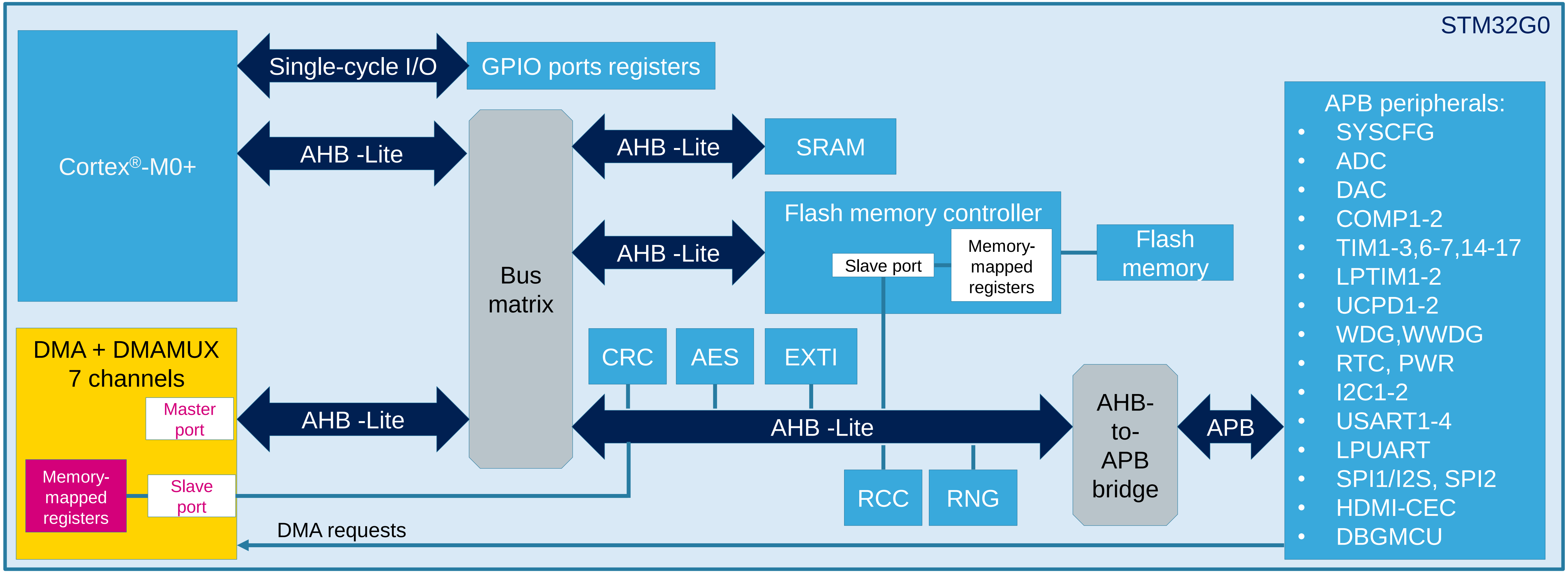
(**) 1024 = 512 bytes Heap size + 512 bytes Stack size

life.augmented

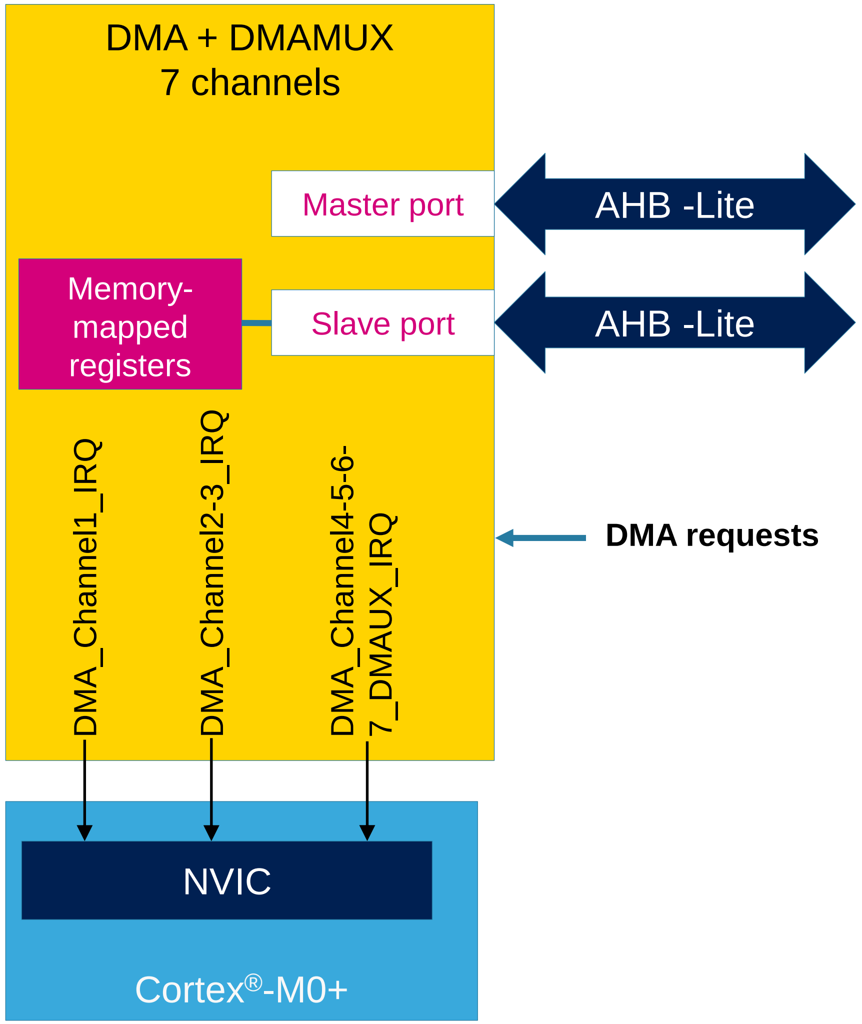
STM32G0 – DMA

life.augmented

DMA - Overview



DMA - Overview



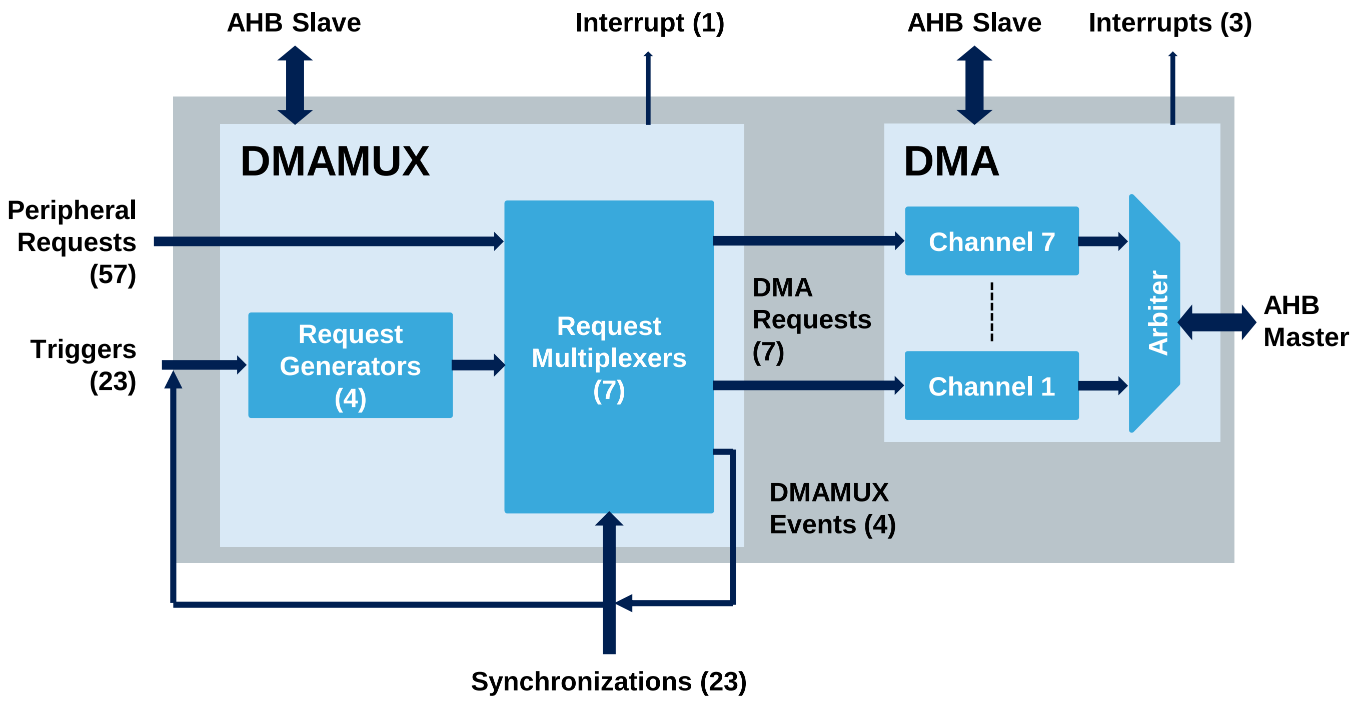
- DMA features
  - AHB master bus
  - Flexible configuration
  - Hardware and software priority management
  - Configurable data transfer modes
    - Peripheral-to-Peripheral, Peripheral-to-Memory, Memory-to-Peripheral, and Memory-to-Memory modes

Application benefits

- DMA support for timers, ADC, and communication peripherals
- Offloads CPU from data transfer management
- Simple integration

DMA - Features

- STM32G0 DMA features
  - 1x DMA controller
    - Programmable block transfers with 7 concurrent channels, independently configurable
    - Programmable channel-based priority
    - Data transfers via the AHB master port (connected to the bus matrix)
  - 1x new DMA request multiplexer (DMAMUX)
    - Programmable mapping of a DMA request e.g. from any peripheral
    - Event-triggered & synchronized DMA request generation



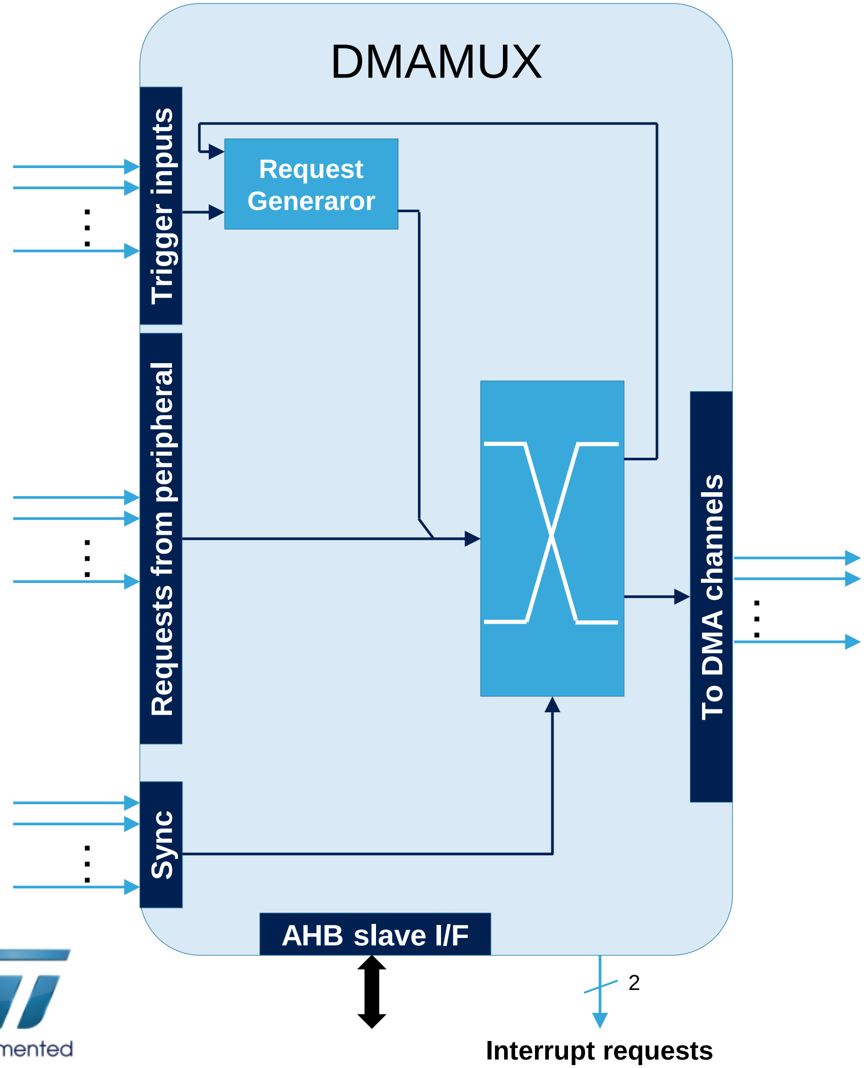
life.augmented

DMA - Main differences with STM32F0

- The DMA Controller is similar to the one implemented in STM32F0 microcontrollers, but with the additional DMA request multiplexer (DMAMUX)

	STM32F0	STM32G0
DMA	2 DMAs	1 DMA
DMA Features	Same	
DMAMUX	No	Yes

DMAMUX - Overview



- The DMA request router (DMAMUX) manages:
  - The assignment of DMA request lines to peripherals
  - The request forwarding synchronization with events on synchronization inputs
  - The request chaining using the DMA request counter and Event generator for DMA

Application benefits

- High flexibility in choice of DMA request mapping
- External and internal DMA request management
- Request synchronization
- Request chaining capability

DMAMUX features

- DMAMUX is a DMA request multiplexer/router
  - DMAMUX provides a programmable routing of any of the 7 DMA (hardware) requests from any peripheral request
- Additionally, there are 4 request generator channels
  - Software can configure a DMA request to be generated by the DMAMUX itself, upon a trigger input
  - Are programmable:
    - The trigger selection: EXTI0..15, LPTIM1/2OUT, TIM14_OC, or any of the 4 generated DMAMUX events
    - The trigger event: rising edge, falling edge or either edge
    - The number of generated DMA requests upon the trigger event
  - There is a trigger overrun flag & interrupt in order to alert the software when the number of generated DMA requests (as paced by the DMA) have not been completed before a next trigger event

STM32G0 DMA & DMAMUX instance

DMAMUX features	DMAMUX
Number of peripheral requests	57
Number of request generator channels	4
Number of trigger inputs	23
Number of synchronization inputs	23
Number of output DMA requests	7

DMA features	DMA
Number of channels	7

Optional Lab: DMA

life.augmented

Objective:

- Open a provided STM32CubeMX project (*.ioc) from our STM32CubeG0 Library examples and review the configuration.
- Run the Keil uVision5 (ARM-MDK) project example for the DMA configured as a memory to memory to transfer a buffer from Flash to internal SRAM.

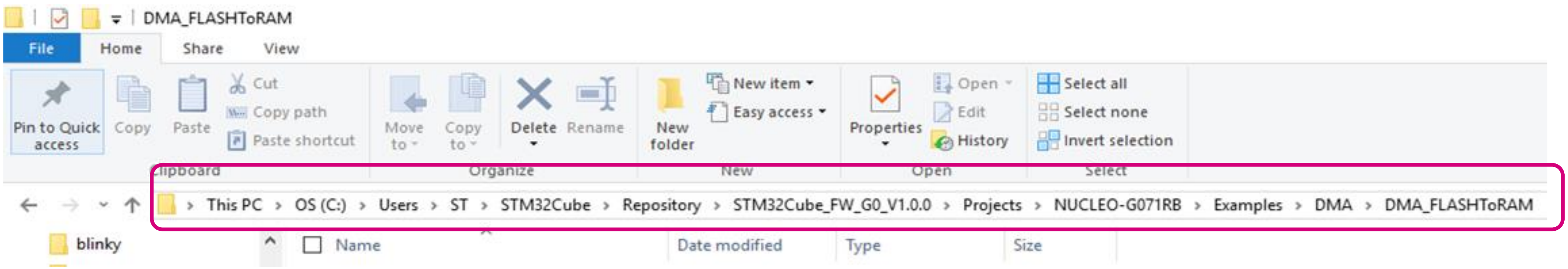
STM32CubeMX Project

- Close Keil uVision5 IDE if it is open and Close STM32CubeMX if it is open

- In a Windows Explorer window open the following location:

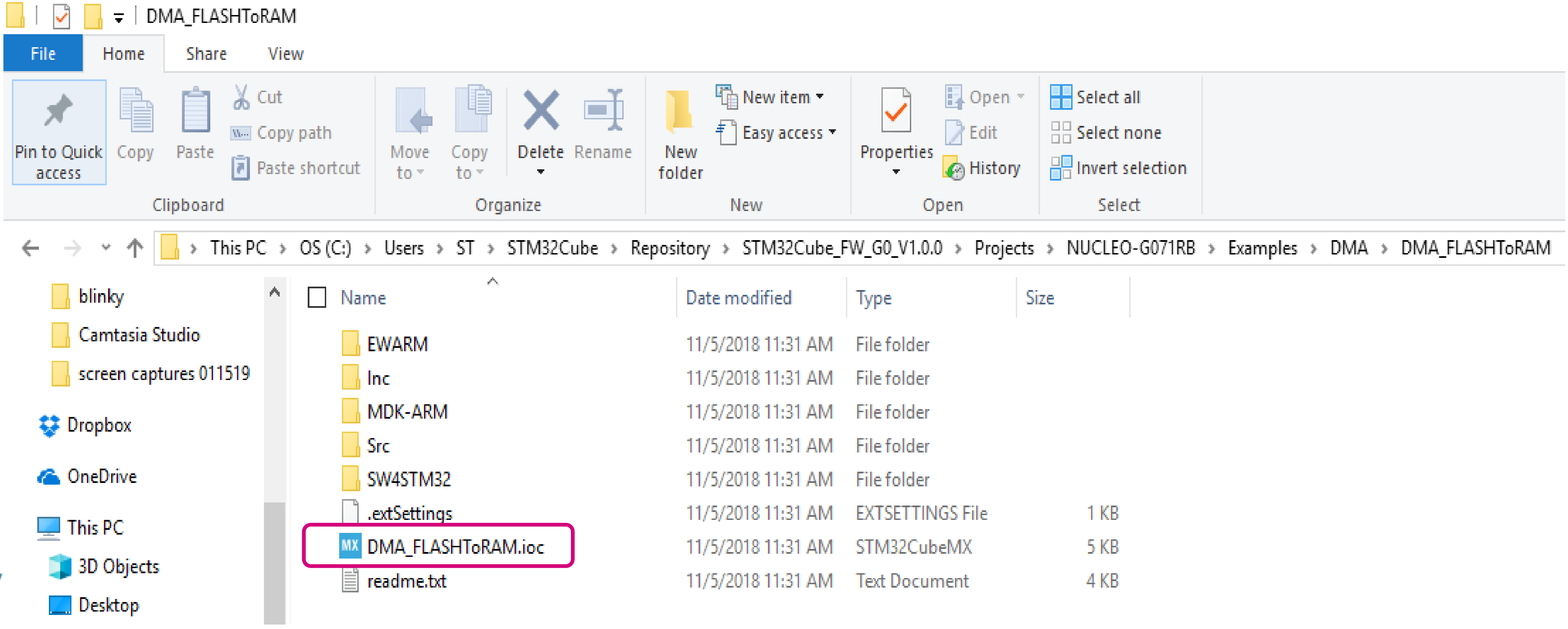
C:\Users\ST\STM32Cube\Repository\STM32Cube_FW_G0_V1.x.0\Projects\NUCLEO-G071RB\Examples\DMA\DMA_FLASHToRAM

Where "ST" is your Windows username and where 1.x .0 is the version of the STM32CubeG) Library that you have installed



Open the ioc file

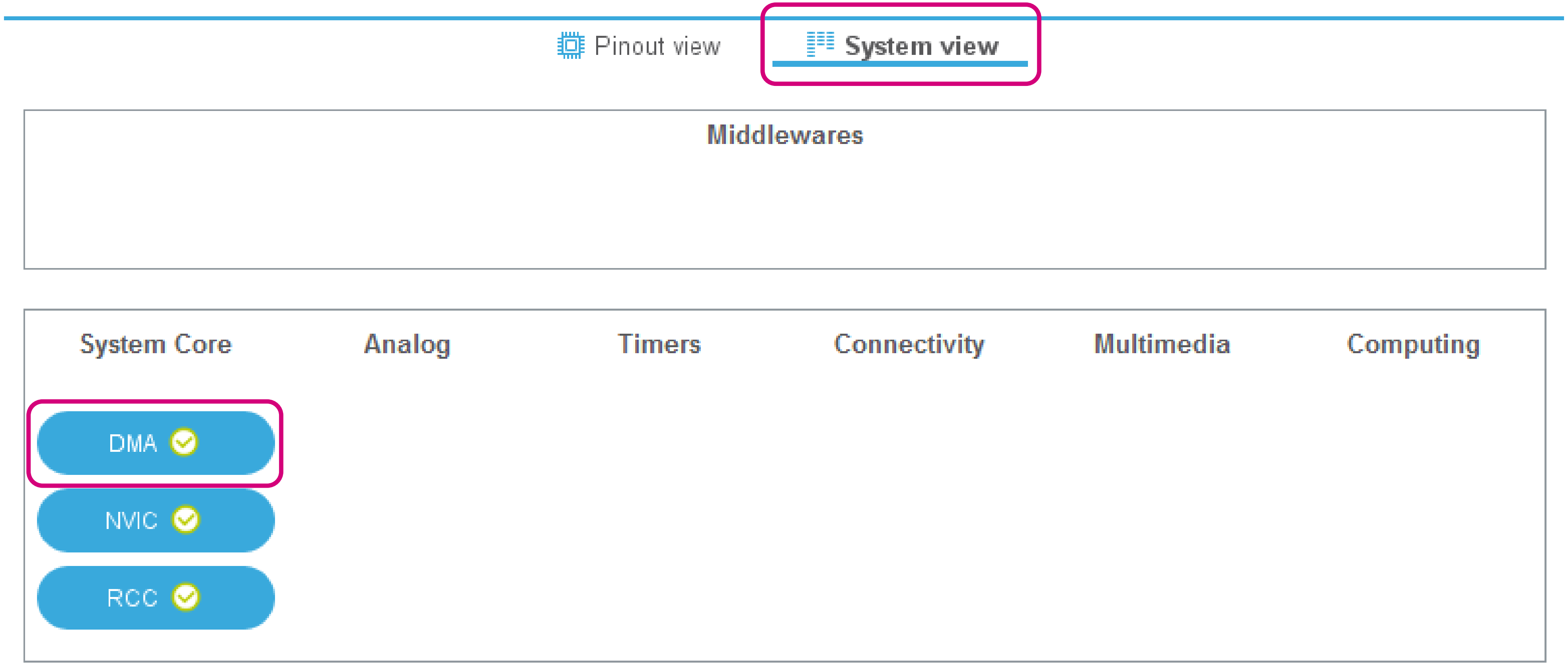
- Double click on the ioc file (DMA_FLASHToRAM.ioc) to open the configuration file with STM32CubeMX:



life.augmented

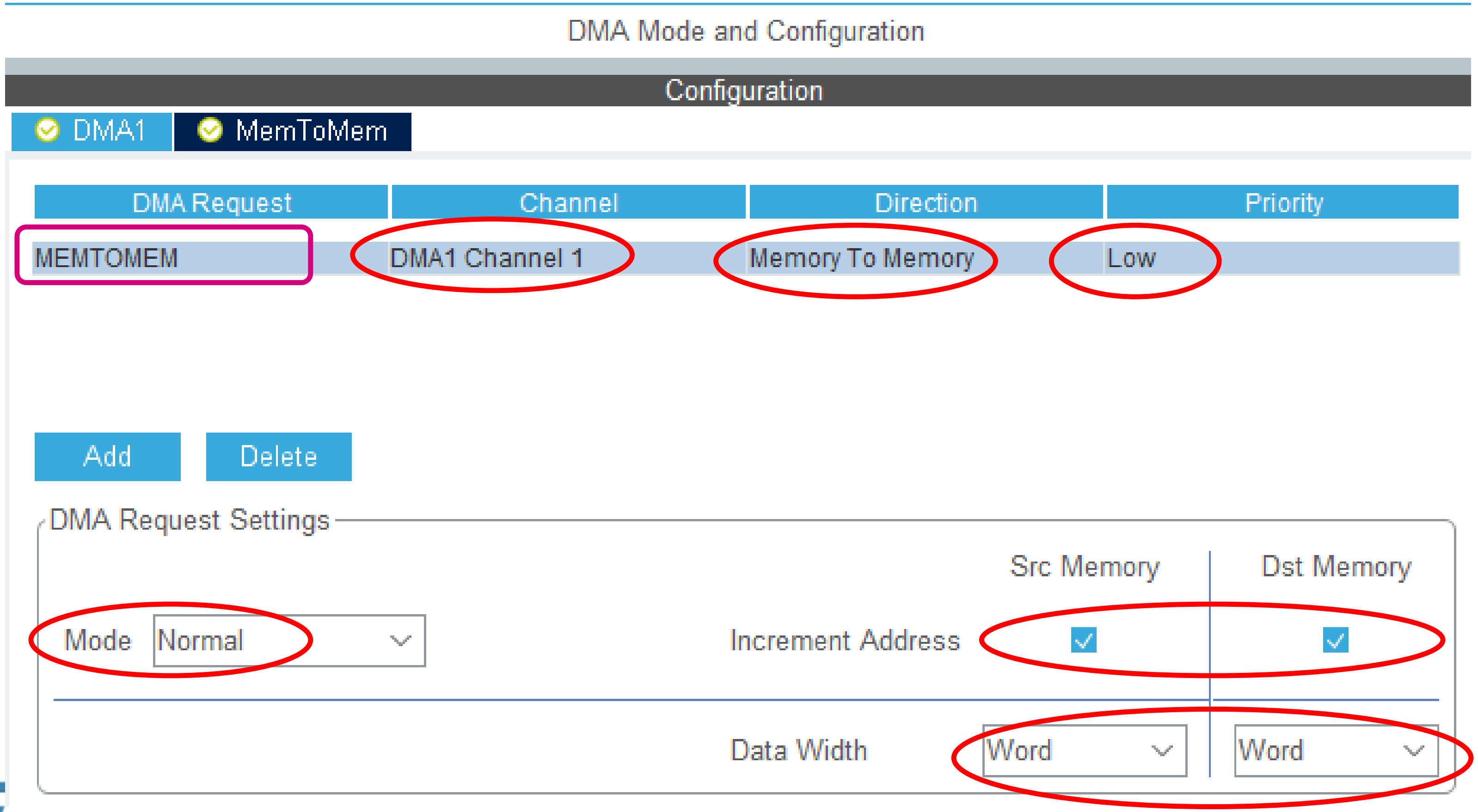
Check the DMA configuration in STM32CubeMX

- In STM32CubeMX click on “**System View**” and then “**DMA**”



Check the DMA configuration in STM32CubeMX

- Review the DMA configuration by clicking on the “**MEM2MEM**” under “DMA Request” as shown below:



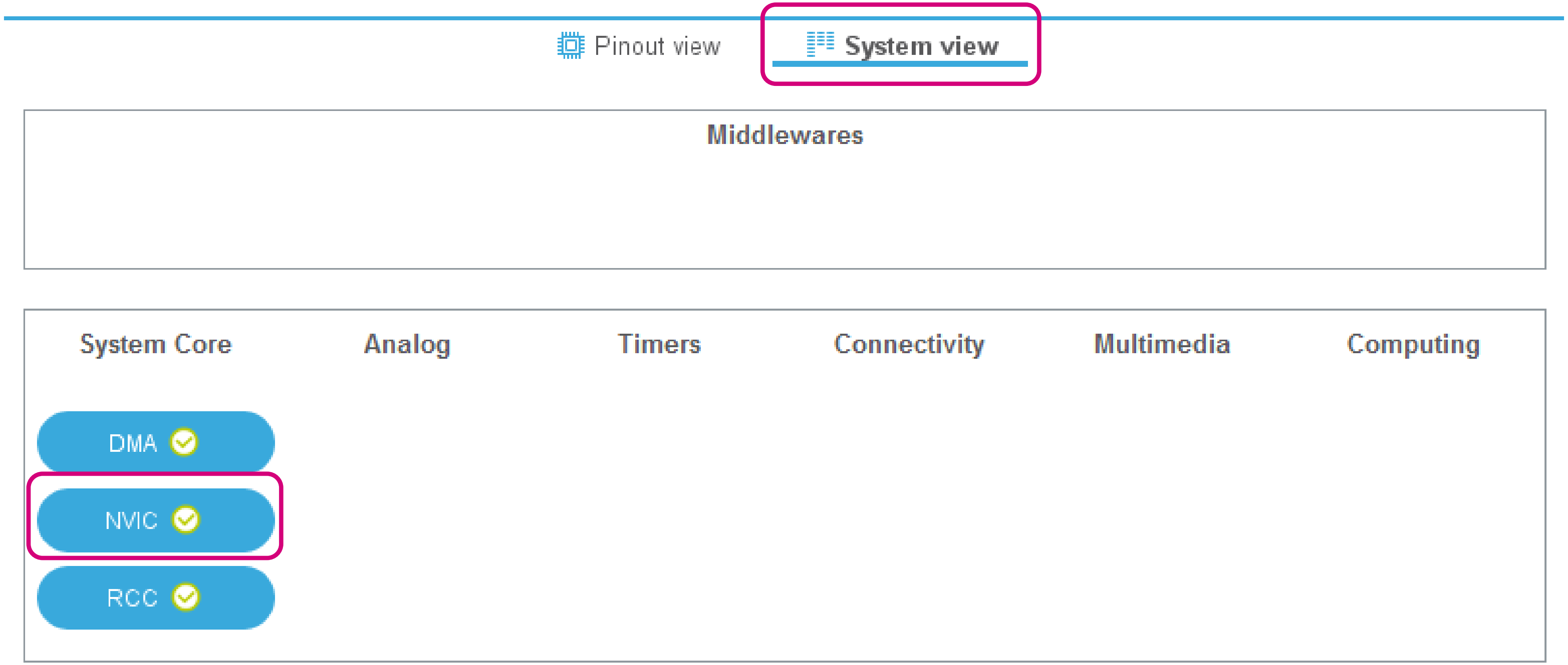
DMA Configuration:

- DMA1 Channel 1
- Memory to Memory
- Priority: Low
- Normal mode
- Incremental Address for source et destination
- Data Width: Word

life.augmented

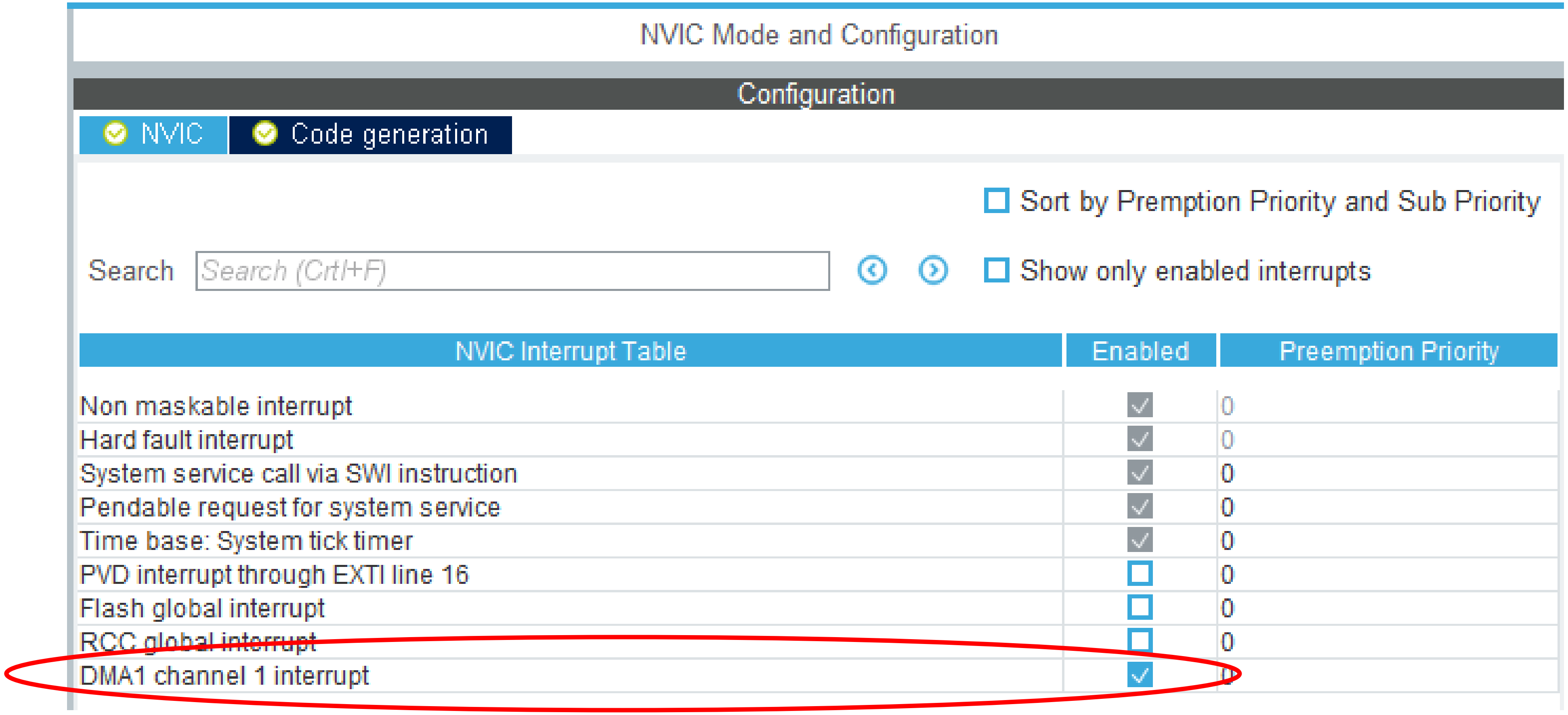
Check the DMA configuration in STM32CubeMX

- In STM32CubeMX click on “**System View**” and then “**NVIC**”



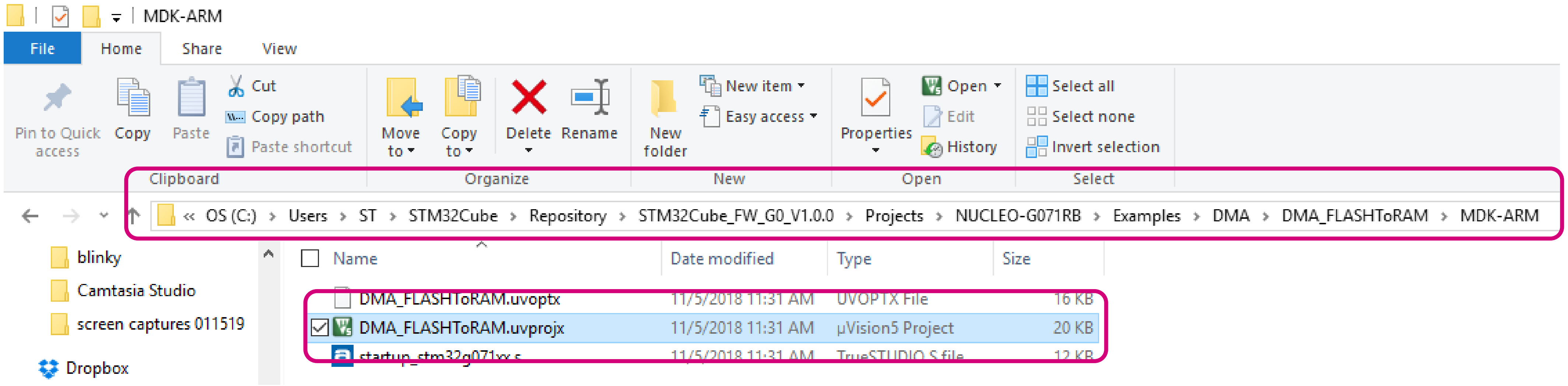
Check the DMA configuration in STM32CubeMX

- Notice that the Interrupt is enabled for the DMA1 Channel1:



- In a Windows Explorer window open the following location where the ARM-MDK (Keil uVision5) project is located and double click on the “DMA_FLASHToRam.uvprojx” file:

C:\Users\ST\STM32Cube\Repository\STM32Cube_FW_G0_V1.x.0\Projects\NUCLEO-G071RB\Examples\DMA\DMA_FLASHToRAM\MDK-ARM



life.augmented

Check the DMA configuration in the ARM-MDK project

- Open main.c and look for the function MX_DMA_Init() and notice that the configuration is the same as in STM32CubeMX:

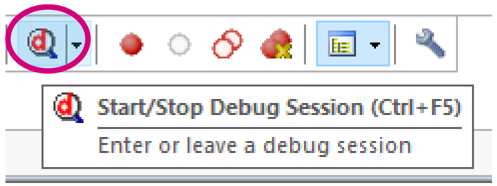
```
static void MX_DMA_Init(void)
{
  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* Configure DMA request hdma_memtomem_dma1_channel1 on DMA1_Channel1 */
  hdma_memtomem_dma1_channel1.Instance = DMA1_Channel1;
  hdma_memtomem_dma1_channel1.Init.Request = DMA_REQUEST_MEM2MEM;
  hdma_memtomem_dma1_channel1.Init.Direction = DMA_MEMORY_TO_MEMORY;
  hdma_memtomem_dma1_channel1.Init.PeriphInc = DMA_PINC_ENABLE;
  hdma_memtomem_dma1_channel1.Init.MemInc = DMA_MINC_ENABLE;
  hdma_memtomem_dma1_channel1.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
  hdma_memtomem_dma1_channel1.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
  hdma_memtomem_dma1_channel1.Init.Mode = DMA_NORMAL;
  hdma_memtomem_dma1_channel1.Init.Priority = DMA_PRIORITY_LOW;
  if (HAL_DMA_Init(&hdma_memtomem_dma1_channel1) != HAL_OK)
  {
    Error_Handler();
  }

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);}

/* USER CODE BEGIN 4 */
```

life.augmented

Build the Project

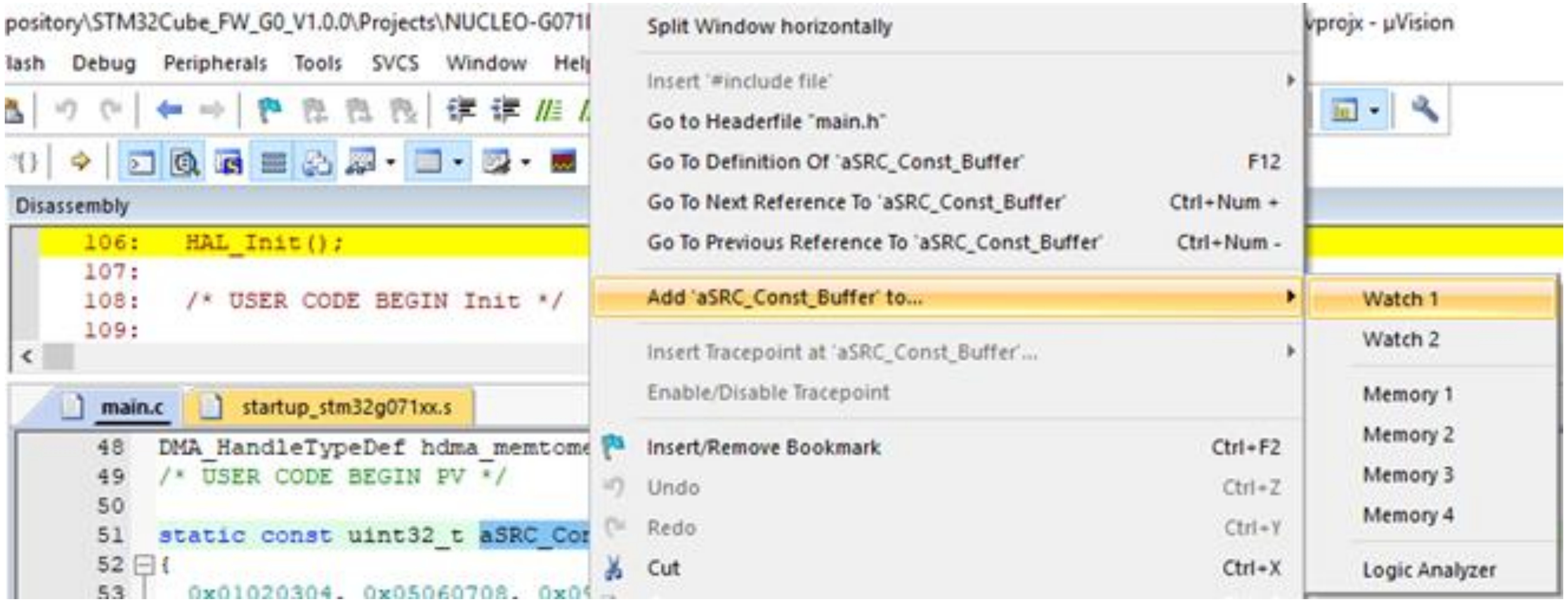
- Click the “Build” button

- Click the “Start/Stop Debug Session” button



Add 2 variables to the watch window

- Add “aSRC_Const_Buffer” (Source buffer located in Flash) to Watch 1 as shown below:



- Do the same thing for “aDST_Buffer” (Destination Buffer located in internal SRAM)

Add 2 breakpoints in main.c

- Add the first breakpoint at this location in main.c just before the execution of he code that will start the DMA transfer:

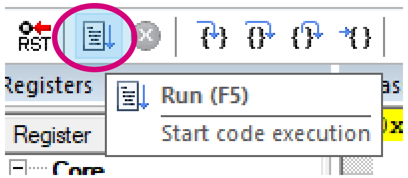
```
/* Configure the source, destination and buffer size DMA fields and Start DMA Channel/Stream transfer */
/* Enable All the DMA interrupts */
if (HAL_DMA_Start_IT(&hdma_memtomem_dma1_channel1, (uint32_t)&aSRC_Const_Buffer, (uint32_t)&aDST_Buffer, BUFFER_SIZE) != HAL_OK)
{
  /* Transfer Error */
  Error_Handler();
}
```

- Add another breakpoint at this code location which will be hit when the DMA transfer has been completed successfully:

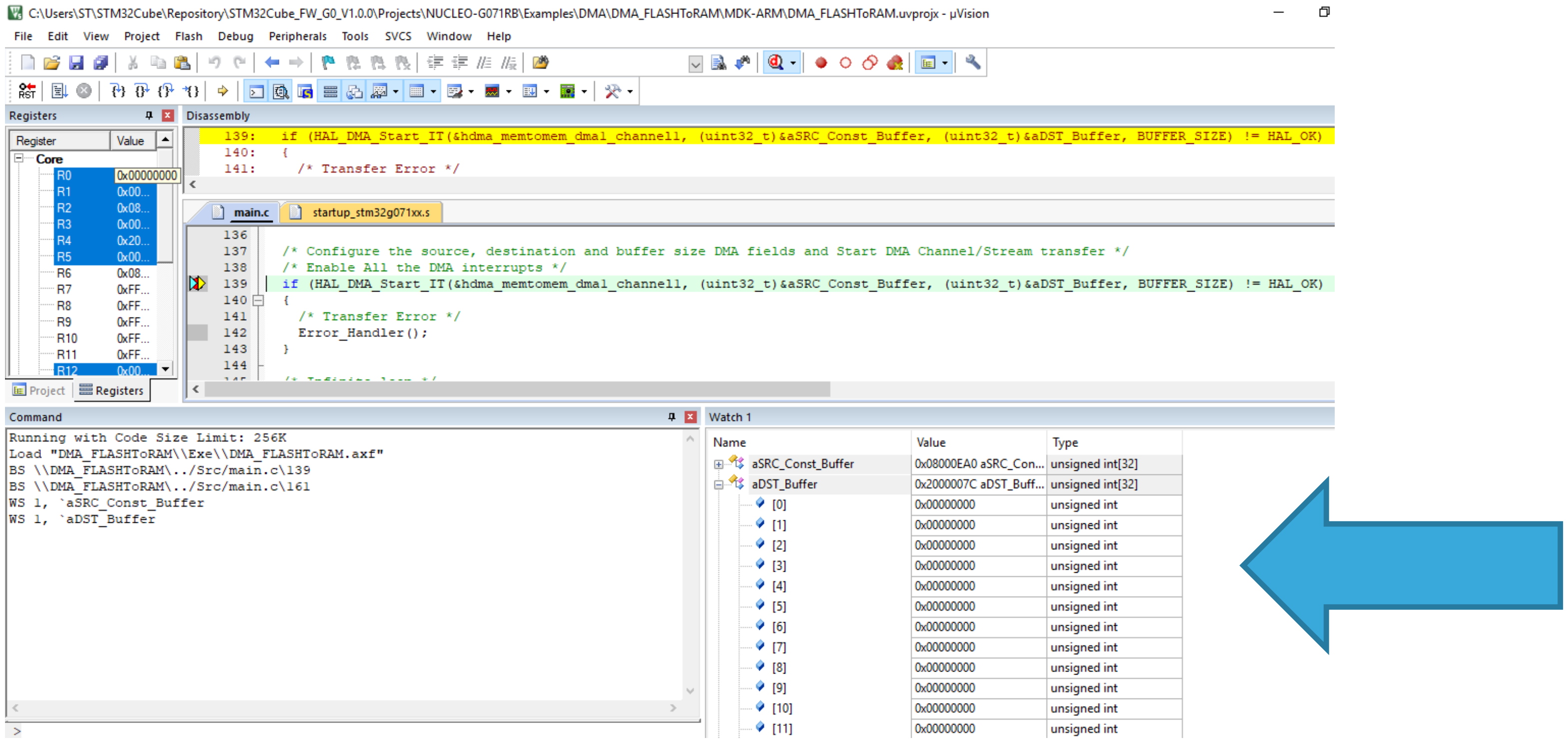
```
    if (transferCompleteDetected == 1)
    {
      /* Turn LED4 on*/
      BSP_LED_On(LED4);
      transferCompleteDetected = 0;
    }
  }
  /* USER CODE END 3 */
```

Execute the code until it reaches the first breakpoint

- Run the code:

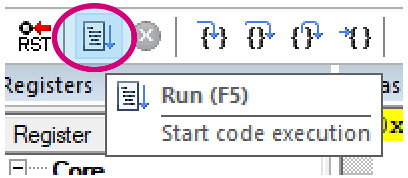


- Once the first breakpoint is hit, notice in the Watch 1 window that the Destination Buffer (aDST_Buffer) is filled with '0'

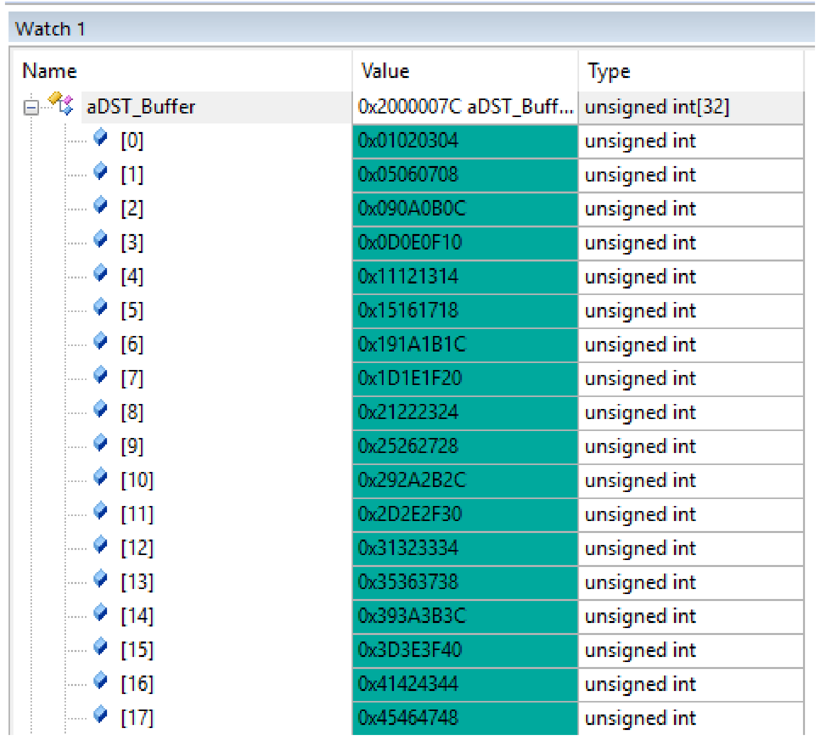


Execute the code until it reaches the first breakpoint

- Now Continue executing the code until it reaches the second breakpoint, press “Run” again:



- Now the destination buffer is filled with values that are the same as the source buffer:


Watch 1

Name	Value	Type
aDST_Buffer	0x2000007C aDST_Buff...	unsigned int[32]
[0]	0x01020304	unsigned int
[1]	0x05060708	unsigned int
[2]	0x090A0B0C	unsigned int
[3]	0x0D0E0F10	unsigned int
[4]	0x11121314	unsigned int
[5]	0x15161718	unsigned int
[6]	0x191A1B1C	unsigned int
[7]	0x1D1E1F20	unsigned int
[8]	0x21222324	unsigned int
[9]	0x25262728	unsigned int
[10]	0x292A2B2C	unsigned int
[11]	0x2D2E2F30	unsigned int
[12]	0x31323334	unsigned int
[13]	0x35363738	unsigned int
[14]	0x393A3B3C	unsigned int
[15]	0x3D3E3F40	unsigned int
[16]	0x41424344	unsigned int
[17]	0x45464748	unsigned int

Flashing code into the STM32

life.augmented

Flash - Key features

- Up to 128 Kbytes of single-bank Flash memory
- 2-Kbyte page granularity
- Fast erase (22 ms) and fast programming time (82 µs for double-words)
- Prefetch & Instruction Cache
- Error Code Correction (ECC): 8 bits for 64-bit double-words
  - Single-bit error detection and correction, notification through a maskable interrupt
  - Double-bit error detection and notification through assertion of the NMI

Flash memory organization

Flash area	Flash memory address	Size	Name
Main memory	0x0800 0000 – 0x0800 07FF	2 Kbytes	Page 0
	…	…	…
	0x0801 F800 – 0x0801 FFFF	2 Kbytes	Page 63
Information block	0x1FFF 0000 – 0x1FFF 6FFF	28 Kbytes	System memory
	0x1FFF 7000 – 0x1FFF 73FF	1 Kbyte	OTP area
	0x1FFF 7800 – 0x1FFF 787F	128 bytes	Option bytes

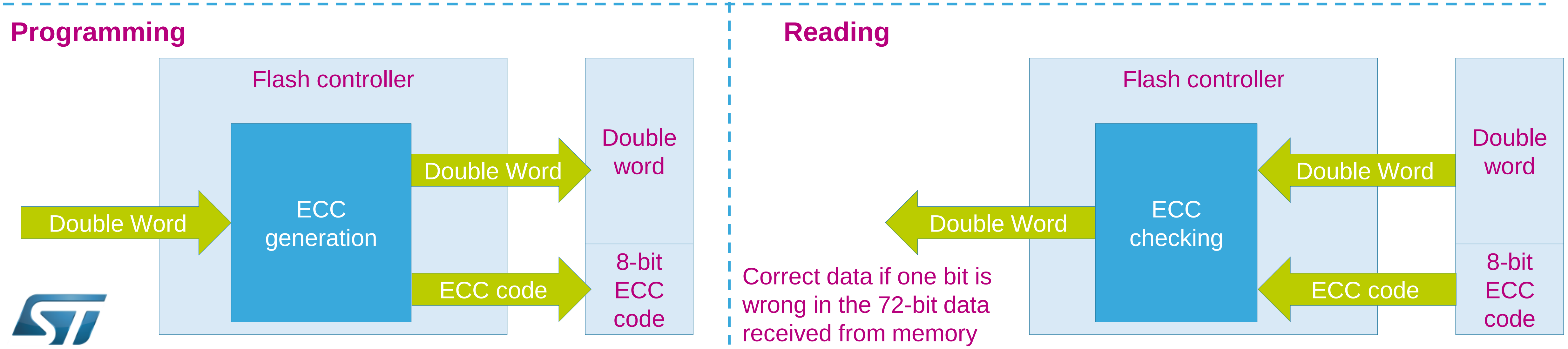
Operation	Granularity
Programming	8-Byte
Fast-programming	Row of 256 Bytes
Erase	2-Kbyte page
Securable memory	
Write protection	
Read protection	Global
Proprietary Code Readout Protection	512 Bytes

life.augmented

Flash memory features (1/2)

Robust memory integrity and safety

- **ECC** (Error Code Correction): 8 bits long for a 64-bit word
  - Single error correction: ECCC bit set in FLASH_ECCR, optional interrupt generation
  - Double error detection: ECCD bit set in FLASH_ECCR => NMI
  - **Failure address saved in FLASH_ECCR register**



Flash memory features (2/2)

Robust memory integrity and safety

- Programming granularity is 64 bits (really 72 bits incl. 8-bit ECC)
- 2 programming modes :
  - Standard (for main memory and OTP)
  - Fast (main memory only)
    - Programs 64 double-words without verifying the Flash memory location

Boot configuration

Boot mode configuration					Selected boot area
BOOT_LOCK bit	nBOOT1 bit	BOOT0 pin	nBOOT_SEL bit	nBOOT0 bit	
0	x	0	0	x	**Main Flash memory**
0	1	1	0	x	**System memory**
0	0	1	0	x	**Embedded SRAM**
0	x	x	1	1	**Main Flash memory**
0	1	x	1	0	**System memory**
0	0	x	1	0	**Embedded SRAM**
1	x	x	x	x	**Main Flash memory forced**

- BOOT_LOCK Forcing boot from Flash memory
  - It is possible to force booting from Main Flash memory regardless the other boot options

- The Empty bit is added in Flash memory register to check if programmed

life.augmented

Lab: Flashing code into the STM32

<u>Objective:</u>

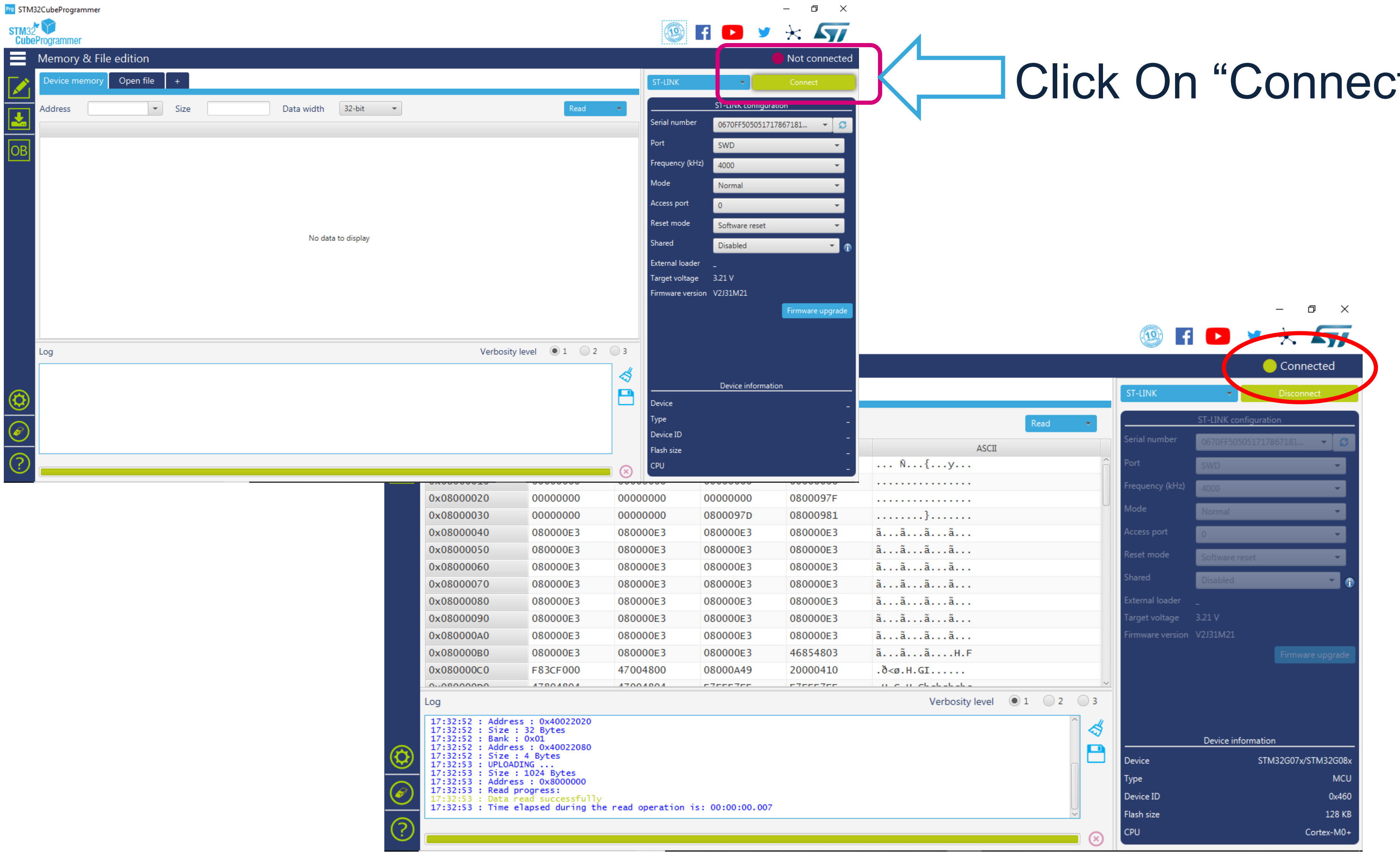
- The objective of this lab is to show you how to flash code into the STM32 by restoring the "out-of-the-box" firmware demonstration code that we saved at the beginning of the workshop using 3 different methods:
  - 1st method: using SWD (Serial Wire Debug)
  - 2nd method: using ST-LINK Mass Storage Feature
  - 3rd method: using the STM32G0 System Memory Bootloader

1st method: using SWD (Serial Wire Debug)

life.augmented

In STM32CubeProgrammer, Connect to the ST-LINK

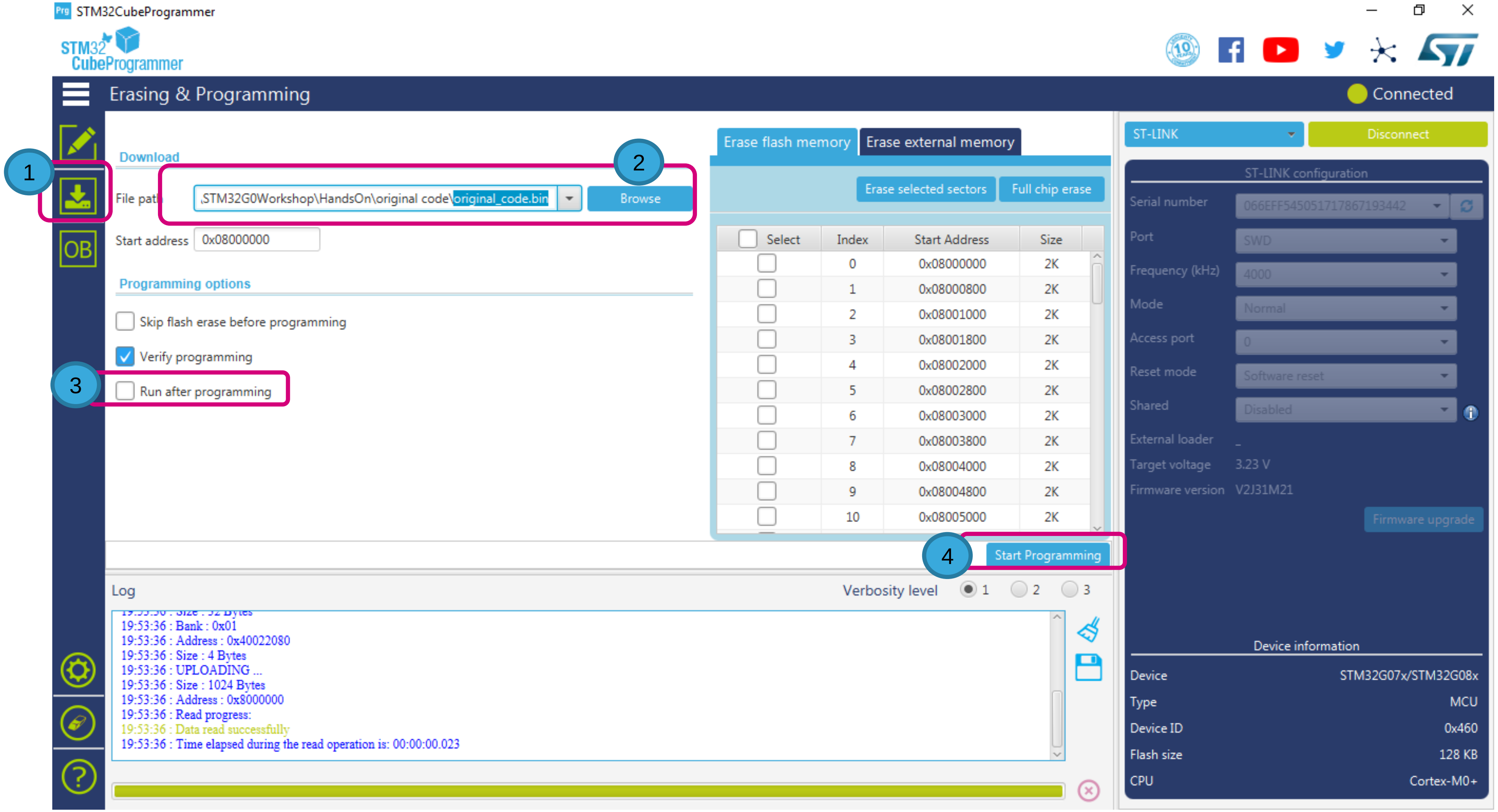
Click On "Connect"



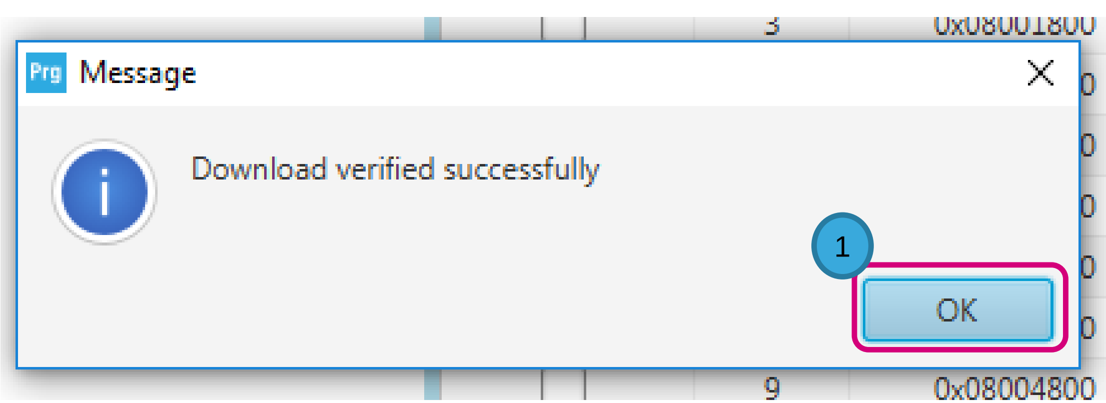
life.augmented

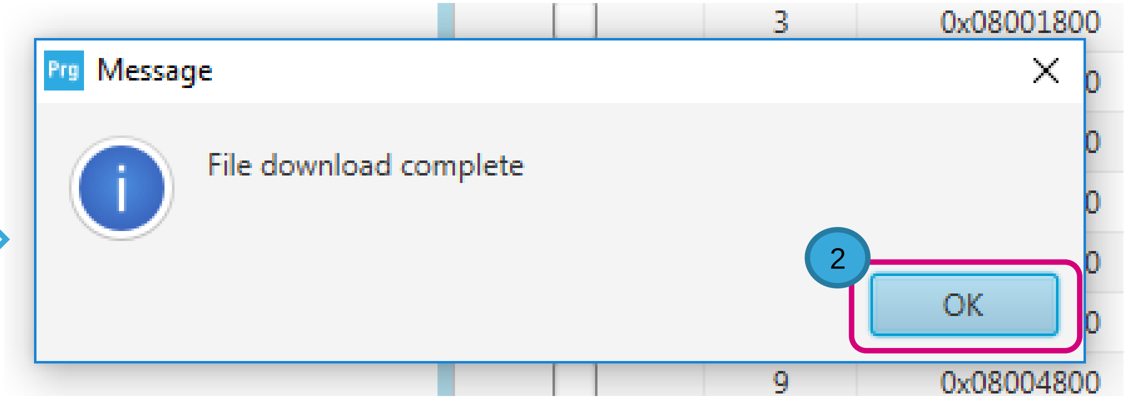
Program the flash

- Click on the icon 1
- Select the path of the original code saved at the beginning of the workshop 2
- Uncheck this box so that the code won't run after programming 3
- Press **Start Programming** 4

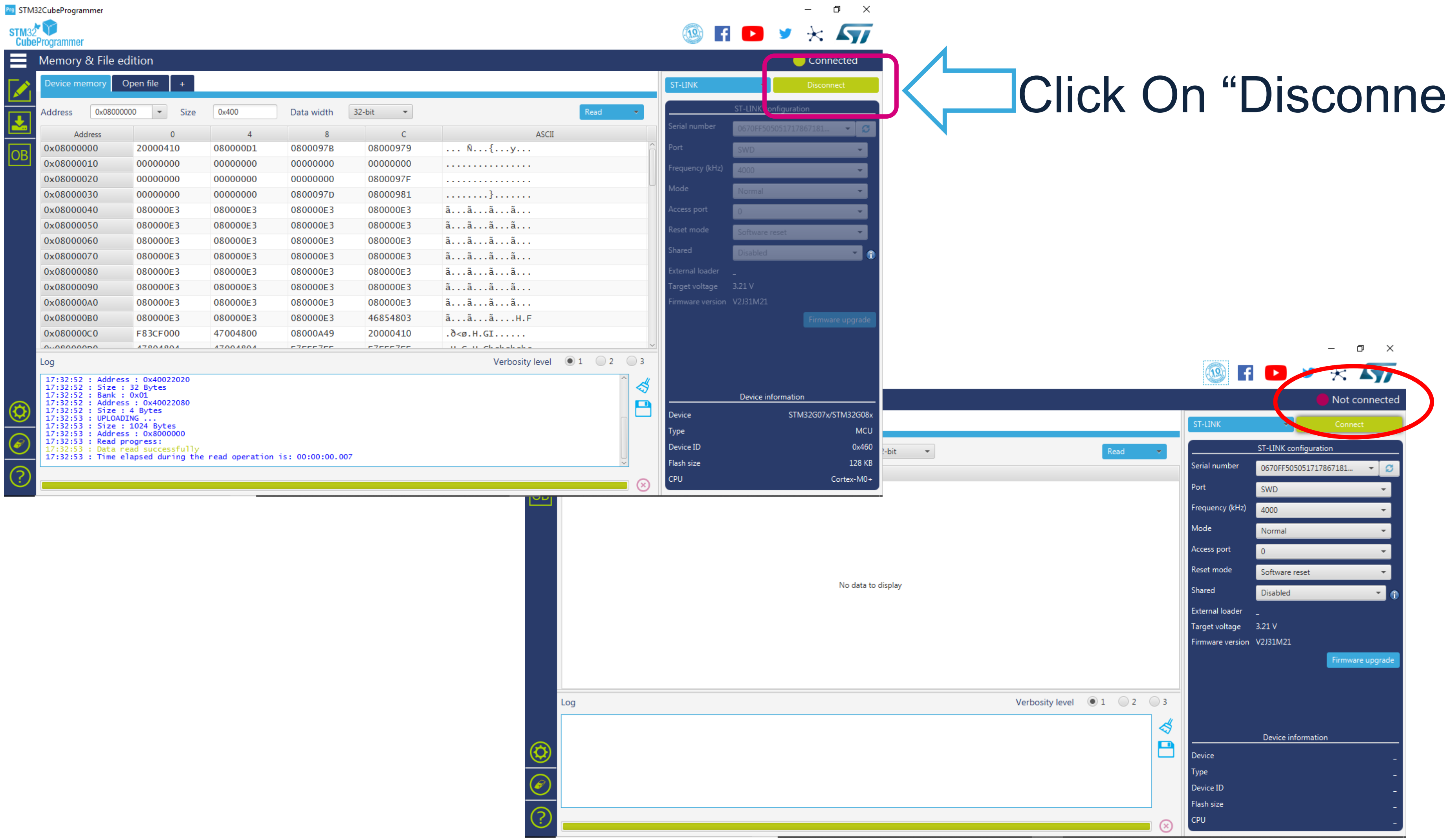


Flash Programmed





Disconnect to the ST-LINK



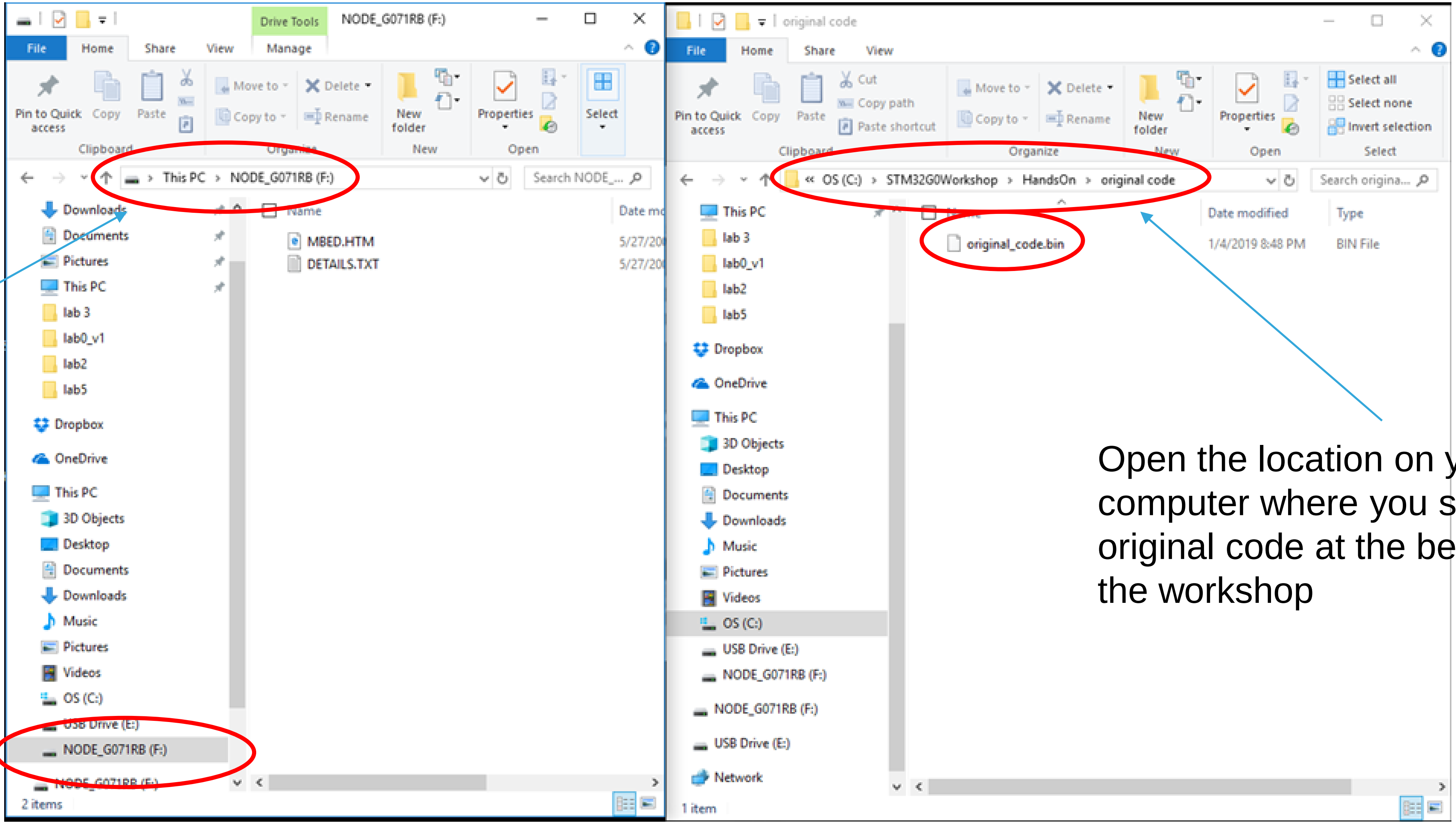
Click On “Disconnect”

Now press reset on the board to run the original code

2nd method: using ST-LINK Mass Storage Feature of the Nucleo board

life.augmented

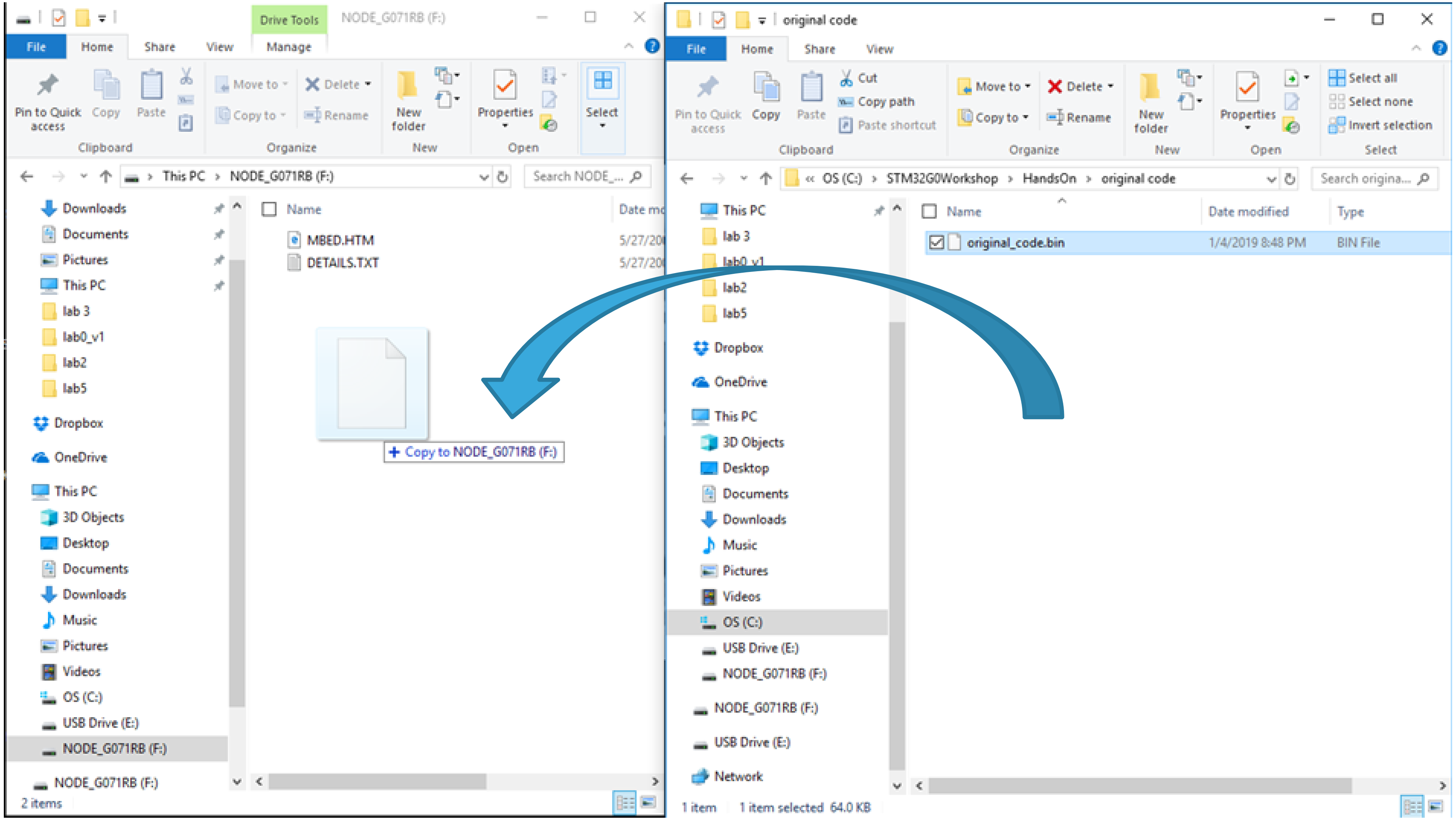
Prepare the copy of binary code to the NUCLEO drive



Open the Nucleo Drive on your computer

Open the location on your computer where you saved the original code at the beginning of the workshop

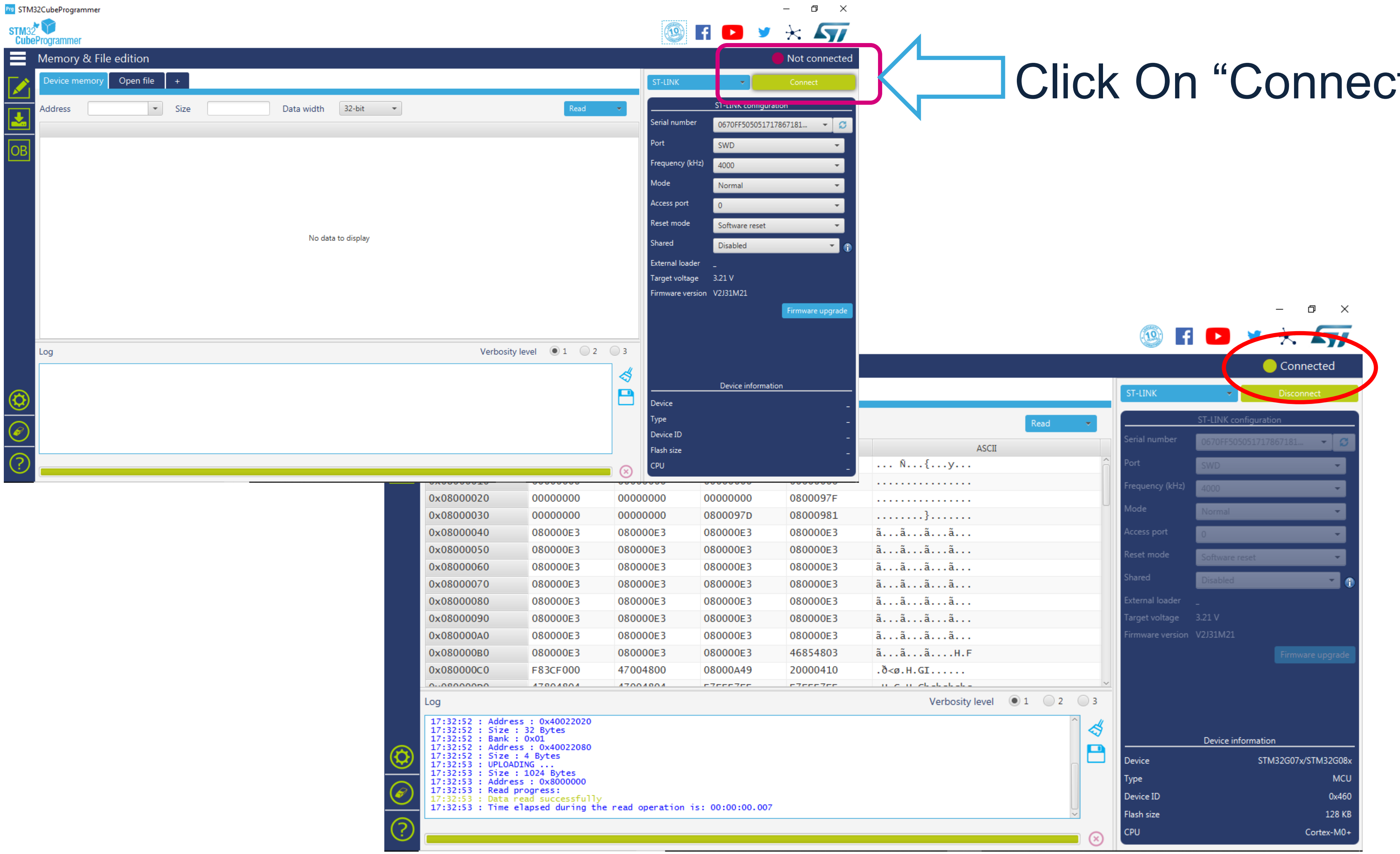
Drag and drop the binary code to the NUCLEO drive



The original code will run after the copy has been done

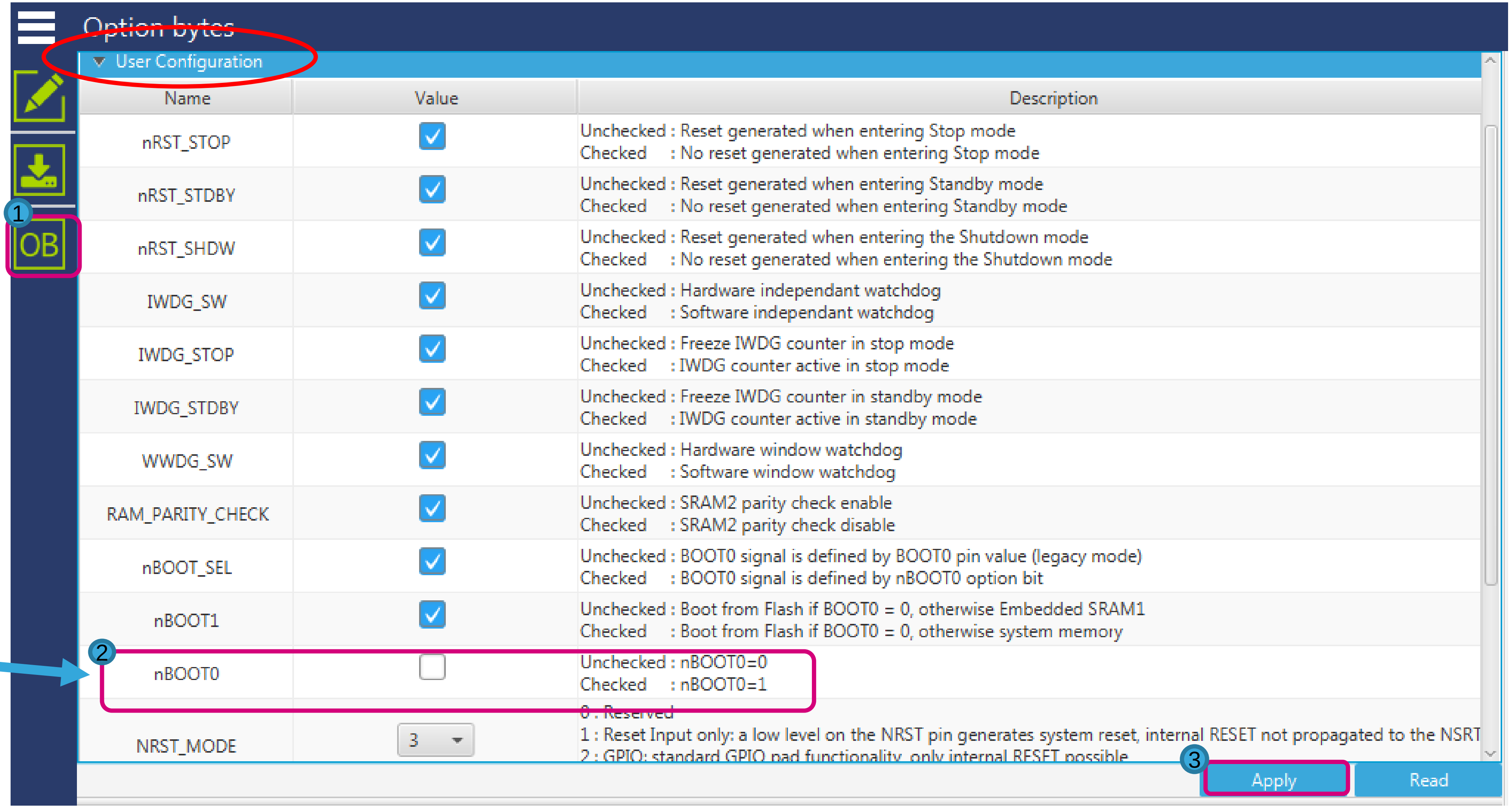
3rd method: using the STM32G0 System Memory Bootloader

Connect to the ST-LINK



Click On “Connect”

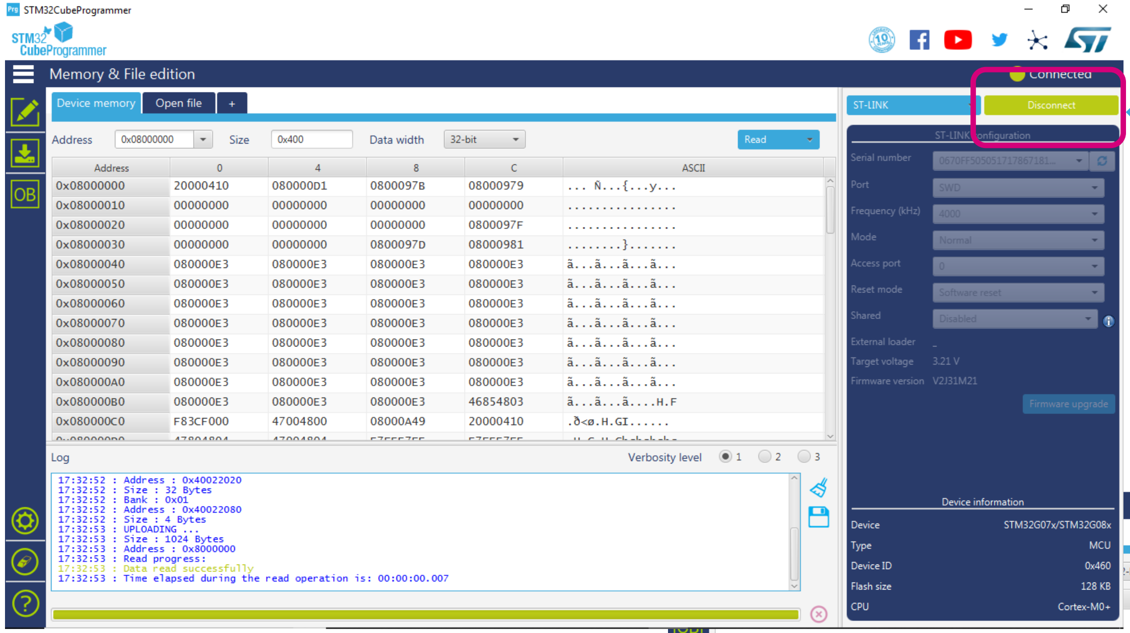
Change the option byte to boot from System Memory



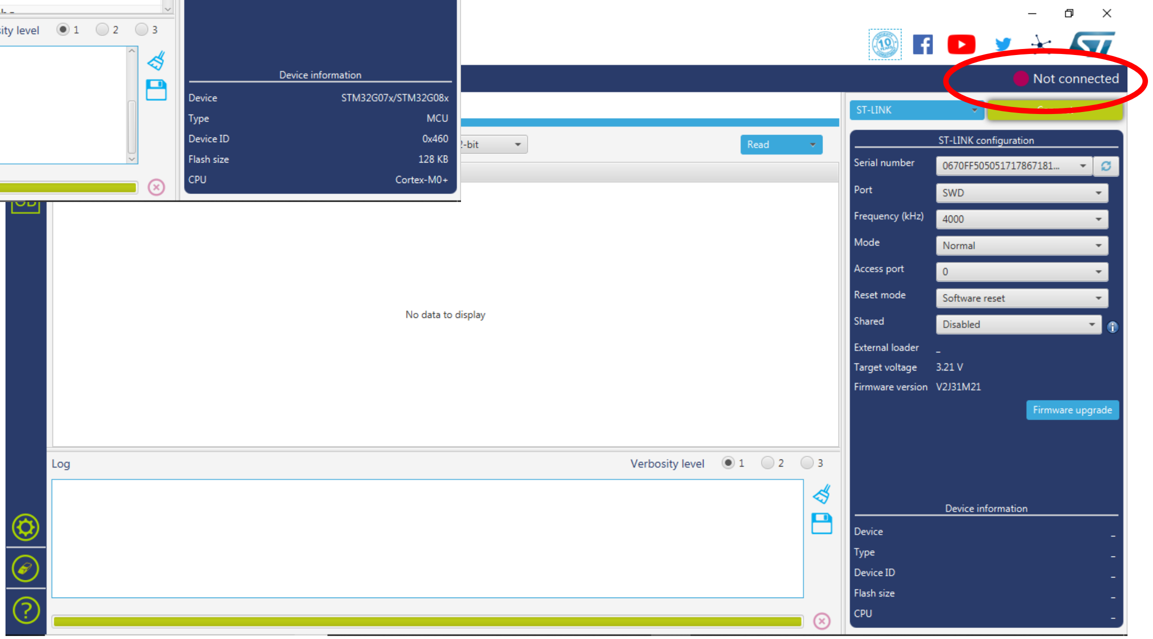
Uncheck **nBoot0** to boot from System Memory Bootloader

life.augmented

Disconnect to the ST-LINK and reset the board



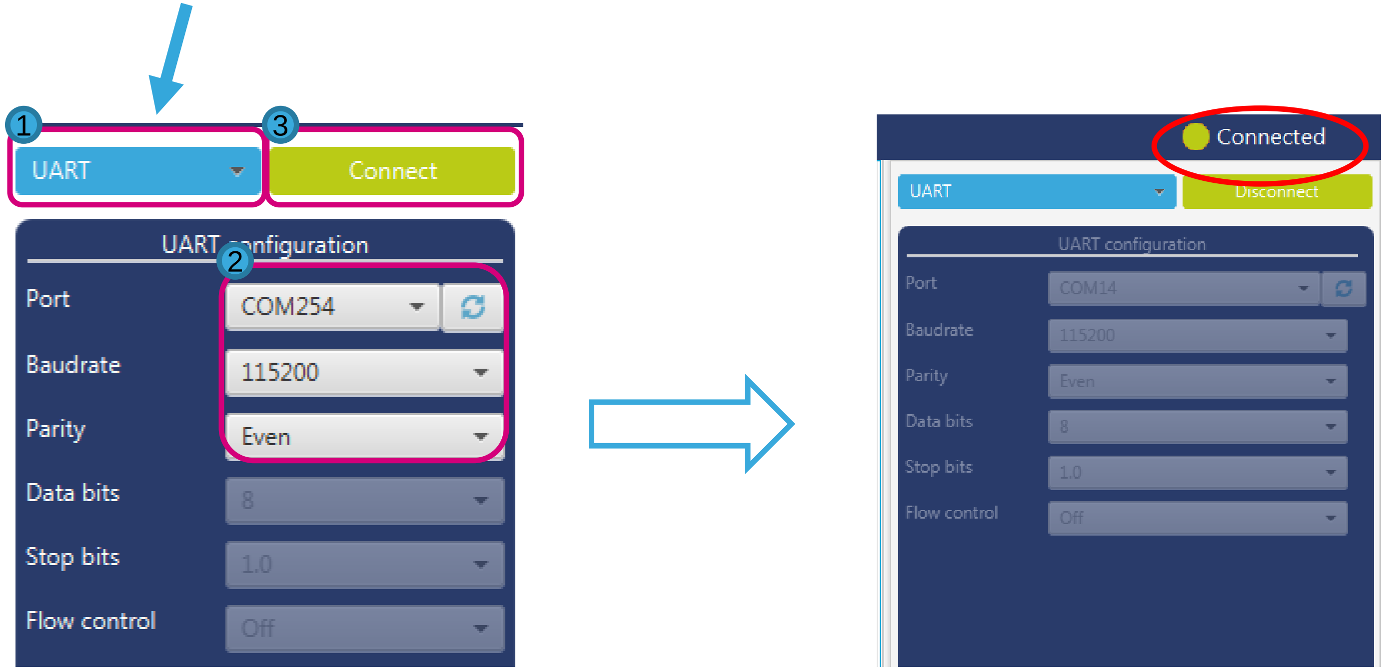
Click On “Disconnect”



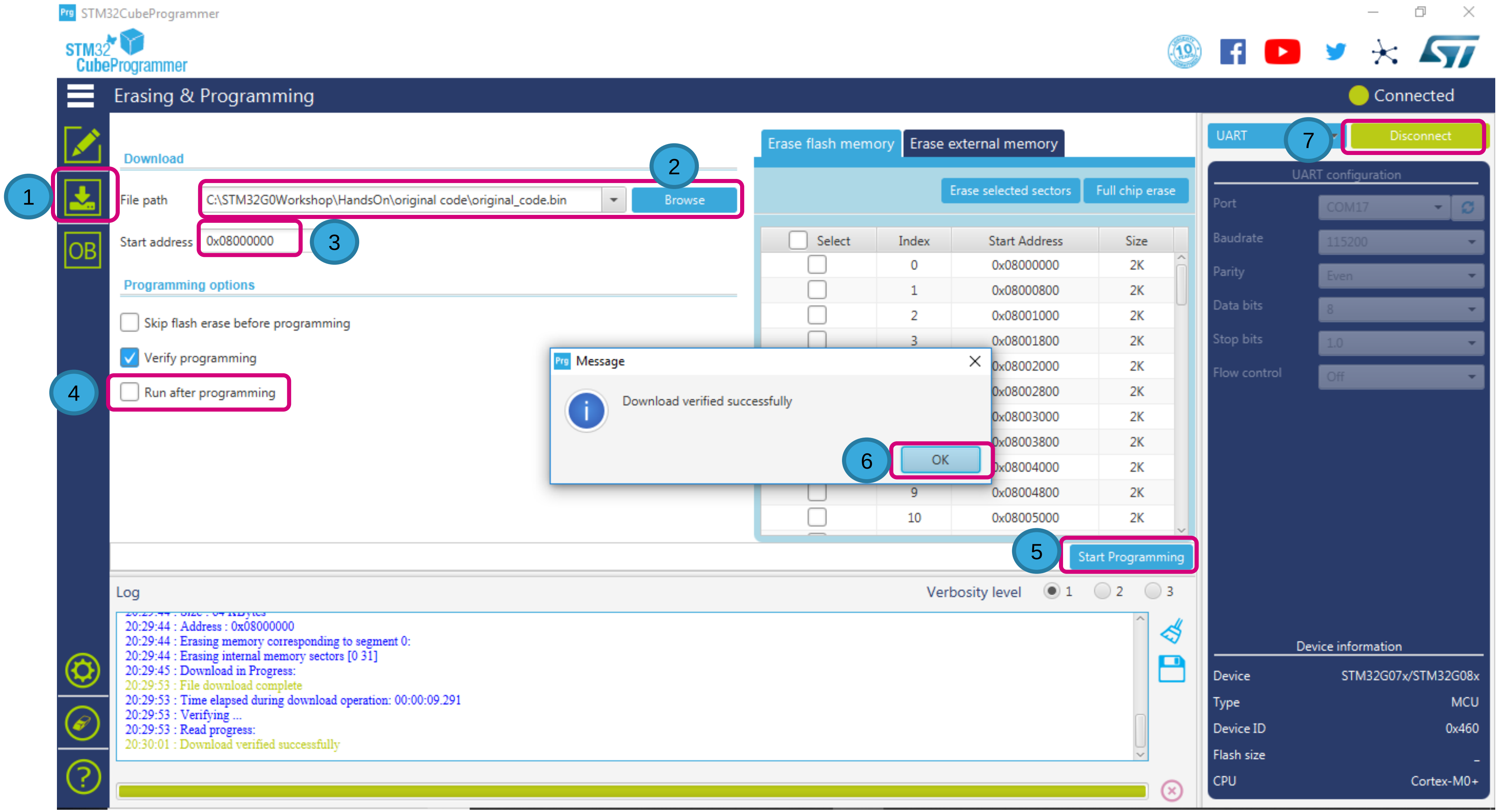
Once disconnected is clicked, unplug and replug the Nucleo board’s USB cable

With STM32 Cube Programmer connect to the System Memory trough UART

Instead of “ST-LINK” use “UART” to connect to the System Memory Bootloader through UART mode

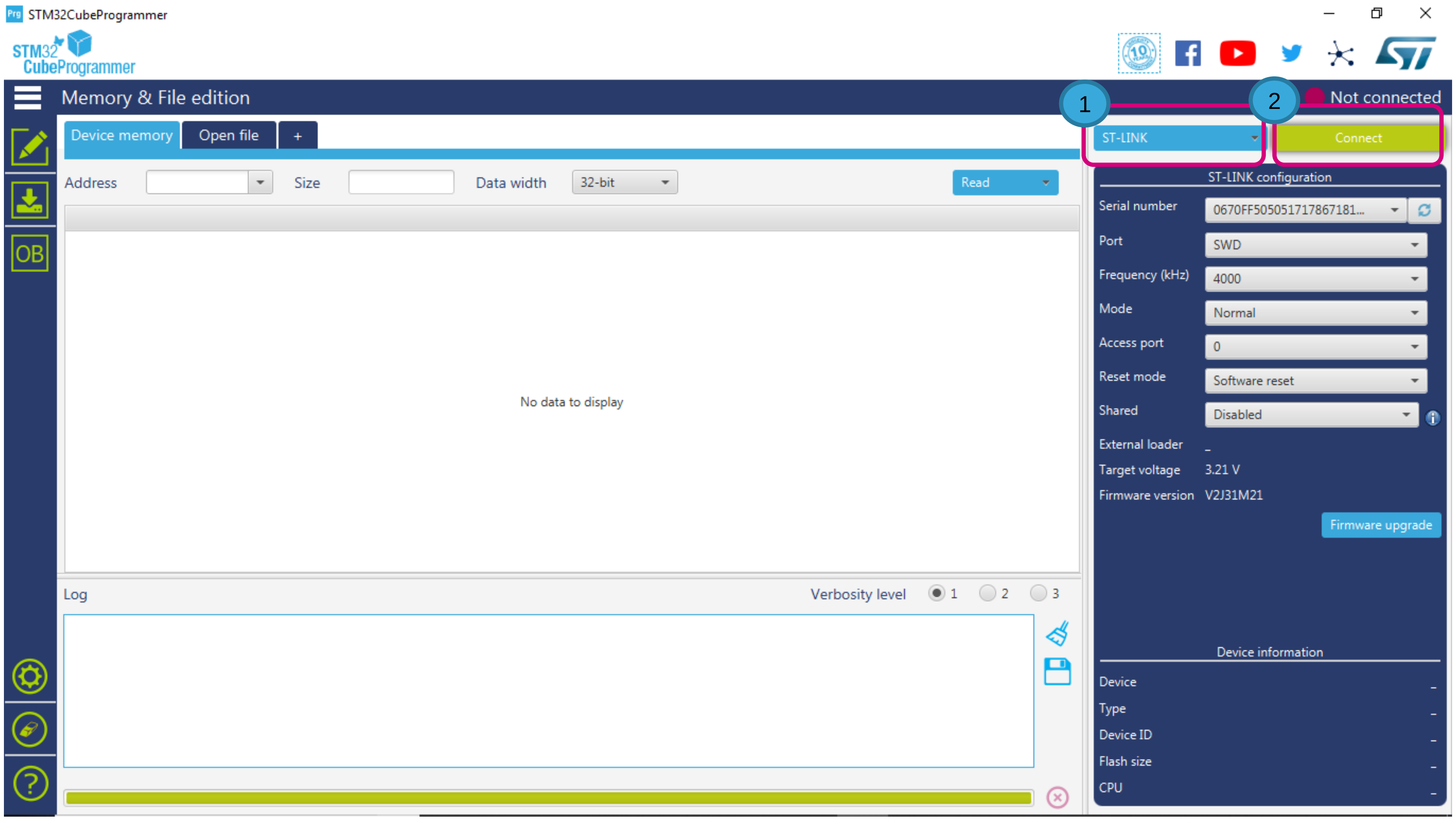


Program the binary code

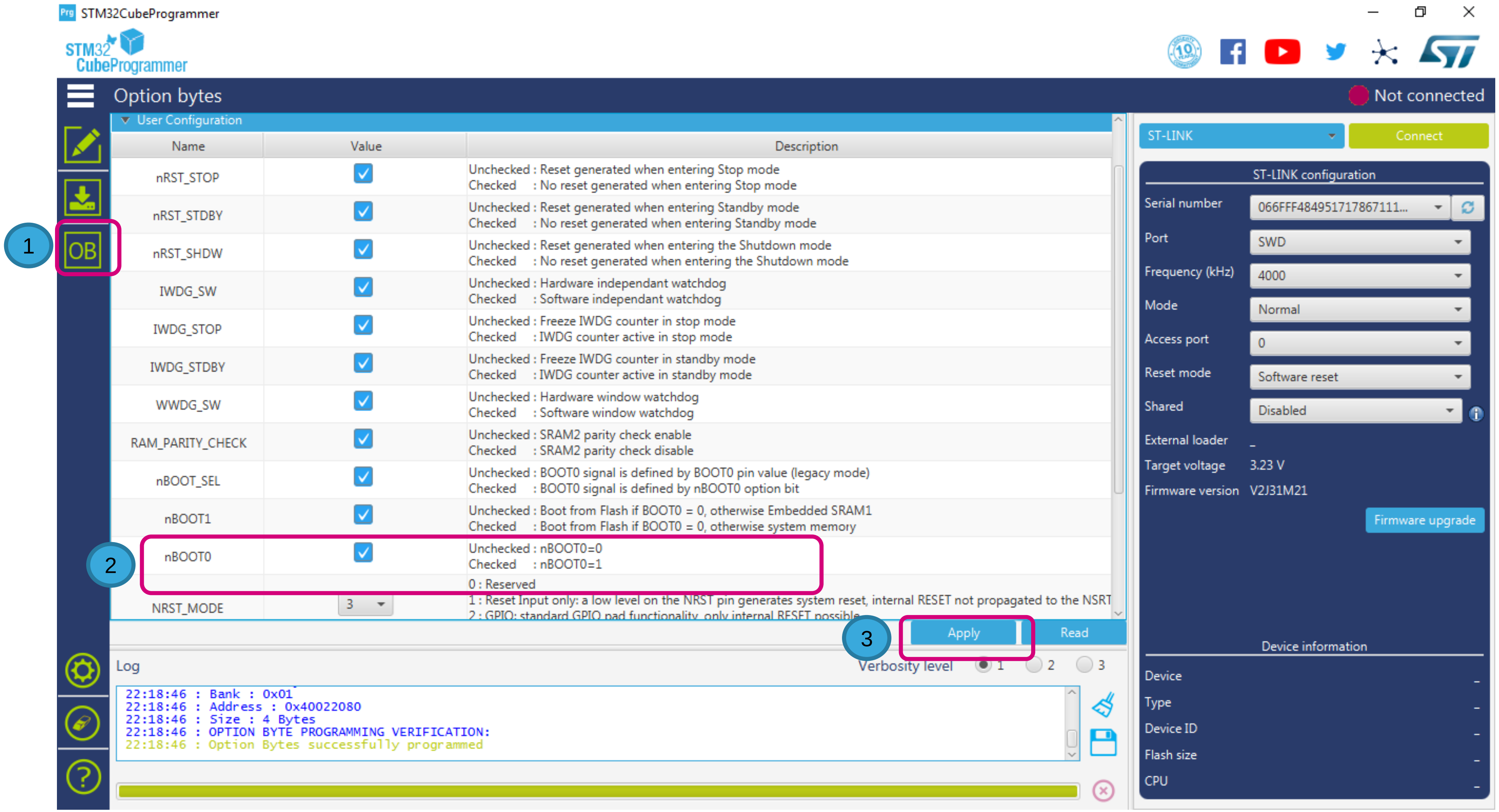


Restore the option bytes to boot from internal flash

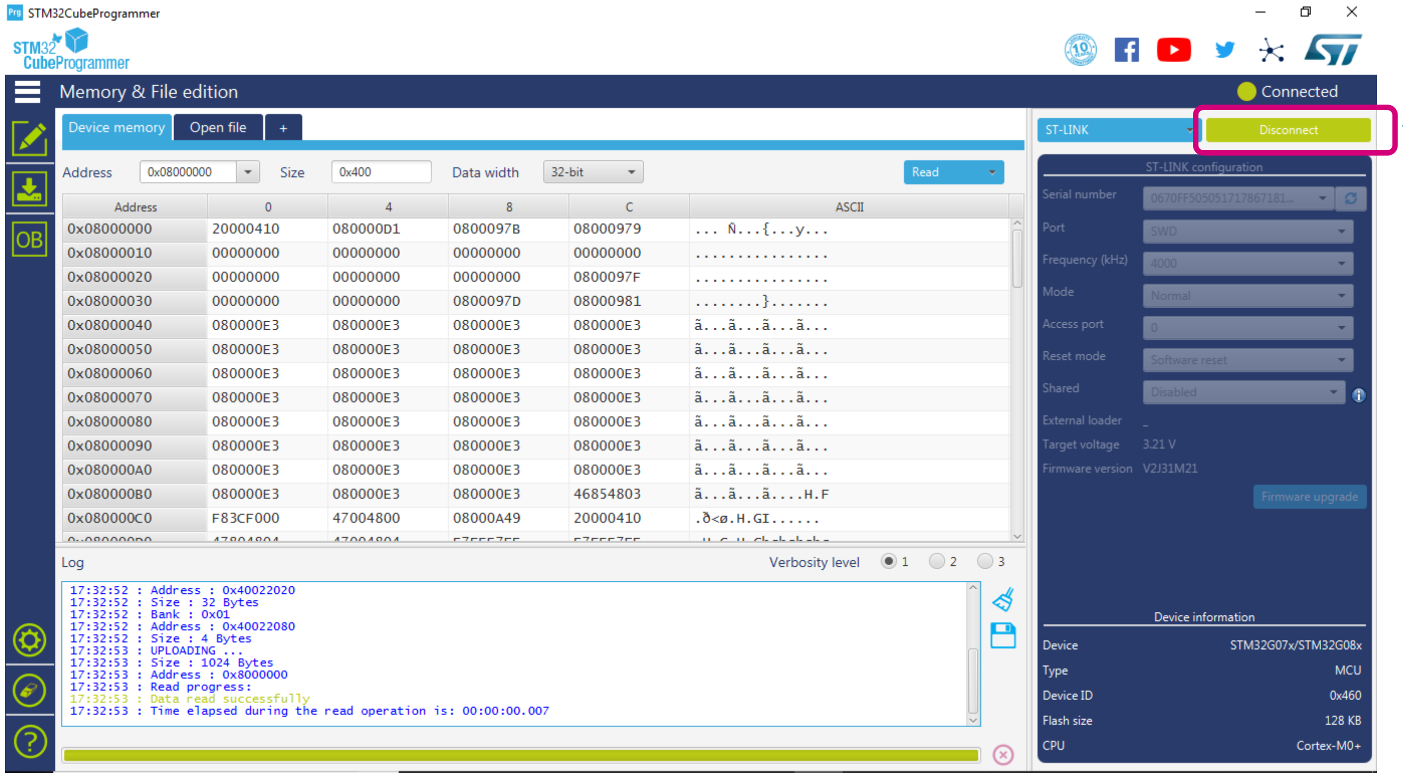
Connect to the ST-LINK through SWD



Change the option byte to boot from the Flash Memory



Disconnect to the ST-LINK



Click On “Disconnect”

Reset the board and the original code will run

Summary

1. **Efficient** (Power, Performance and Cost)
2. **Robust** (EMS, ECC, Clock Monitoring/Watchdogs, Security)
3. **Simple** (Easy to configure and develop code)

www.st.com/stm32g0

- Product Tree
- Tools & Software
- Resources
- Support & Community

We Greatly Value Your Feedback

Use your phone to scan the QR code or type the link into your browser.

https://www.surveymonkey.com/r/8WBPJFF

Thank you!

Thank you

/STM32

@ST_World

community.st.com

life.augmented

www.st.com/stm32g0