Simplifying Embedded Microcontroller Design using the STM32G0 and STM32CubeMX 5.0

STM32 G0

life.augmented

Technology Tour 2019

Dallas-Richardson, TX | March 7

Agenda

9:00 AM – 12:00 PM	Start Tools Installation
	STM32G0 Overview
	Lab: Blink an LED by software
	Lab: Use hardware (PWM timer) to blink an LED
	Lab: External Interrupt
	Lab: Low power
	Optional Lab: Power Consumption Estimation
	Optional Lab: Printf

Begin Tools Installation

- Copy the content of the USB drive to a temporary directory on your machine (i.e. C:\temp)
- Install Java
- Install Keil uVision 5

For more details on the tools installation, refer to the document "STM32G0 Workshop Installation from USB drive_v1.0.pdf" part of the files you copied from the USB drive

STM32G0 MCU series

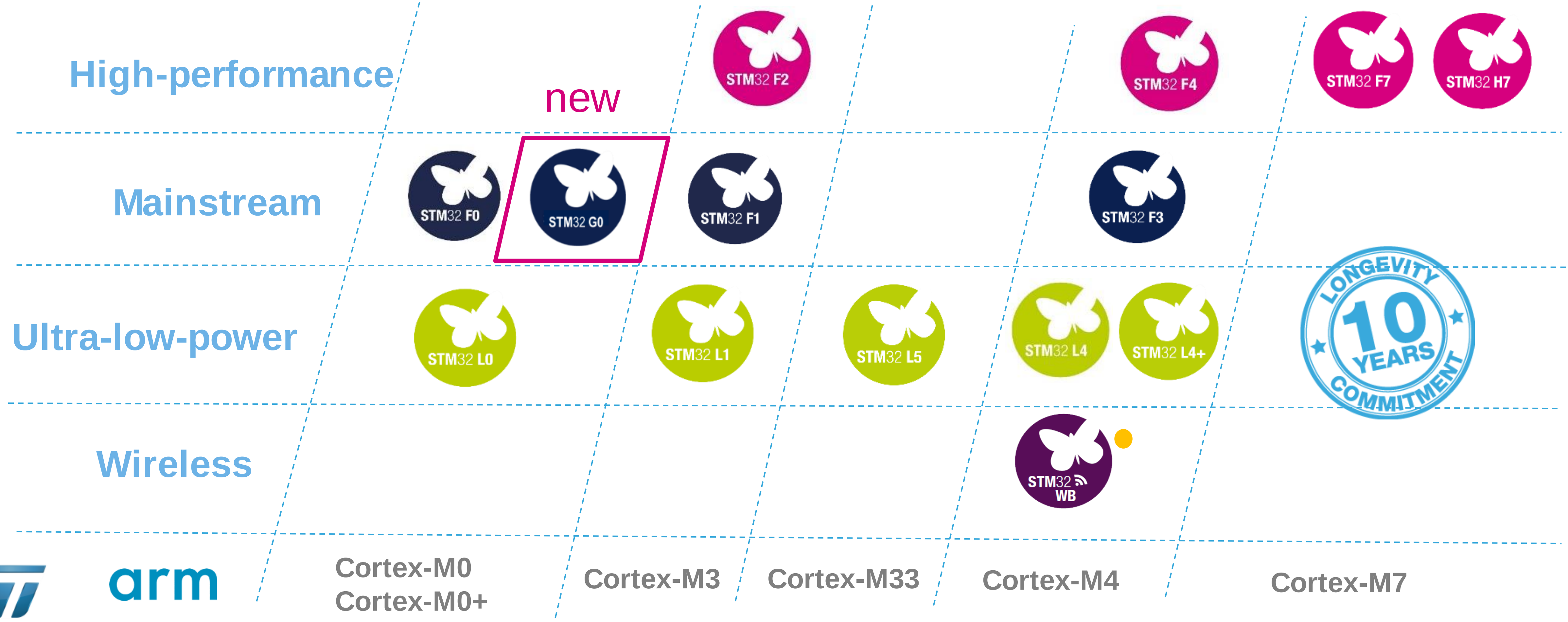
Efficiency at its best

life.augmented

STM32G0: great **investment**

Keep releasing your growing creativity

Note ● : Cortex-M0+ Radio Co-processor



arm

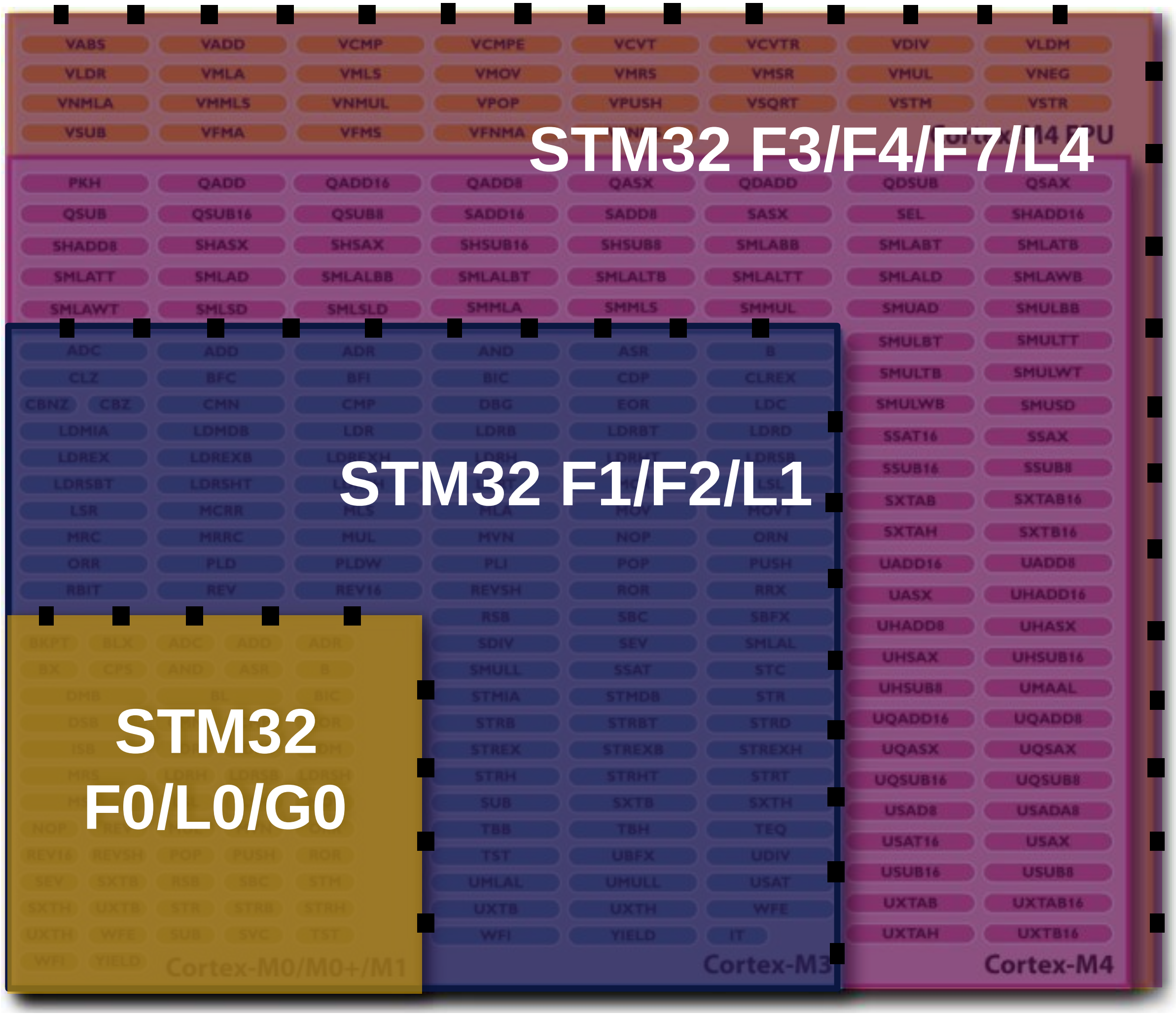
life.augmented

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

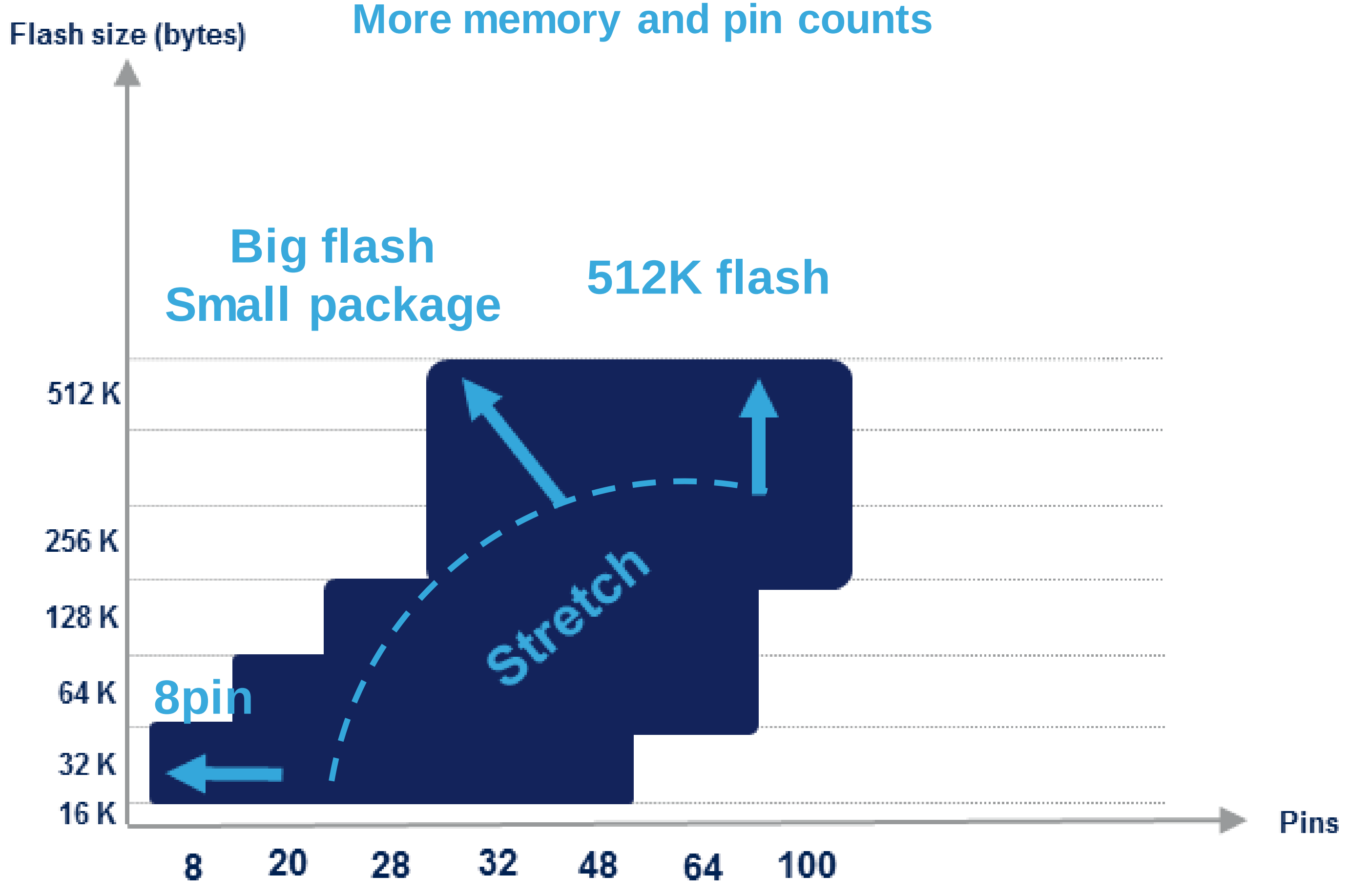
- Easy to configure thanks to the intuitive and graphical STM32CubeMX configuration tool
- Easy to develop based on the Hardware Abstraction Layer library (HAL) or the low-layer library (LL) allowing maximum re-use and faster time-to-market

Wider platform

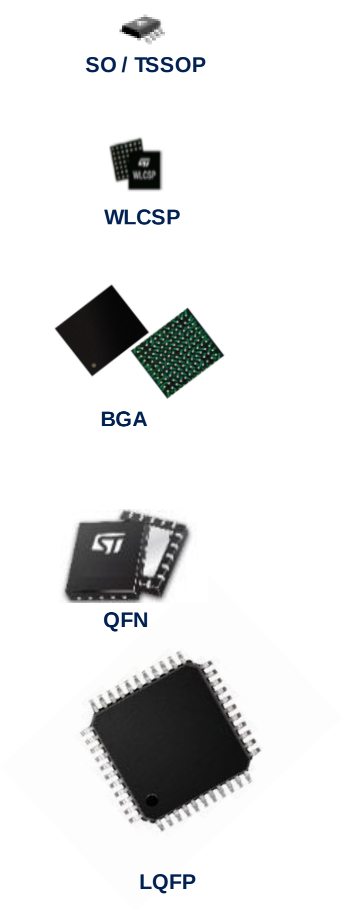
STM32 G0

Portfolio streeeeeeeeetched for efficient budget applications

More memory and pin counts



More packages



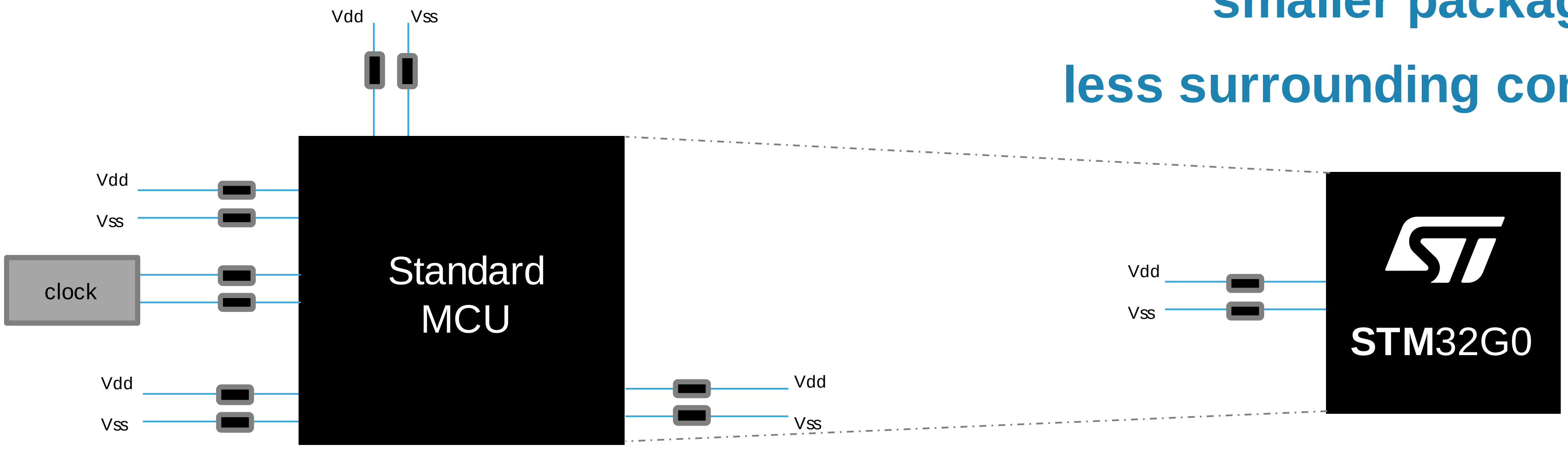
life.augmented

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**

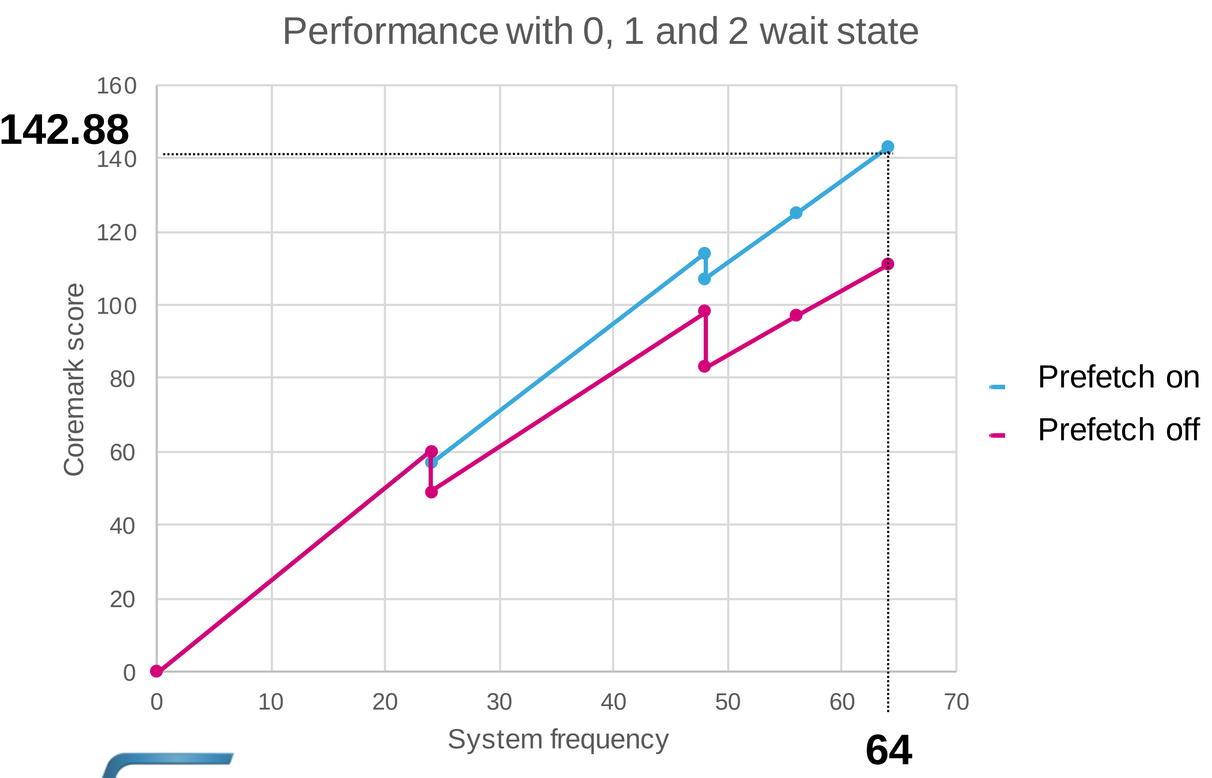
- **No external clock -10cts**
  Accurate internal high speed clock +/-1% for 0 / 90°C
- **No decoupling capacitances -4cts**
  Remove up to 6 decoupling capacitors for supply and clocks
- **Smaller PCB -1cts**
  Smaller package, less components: save on PCB area

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

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 12 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, I^2C, 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

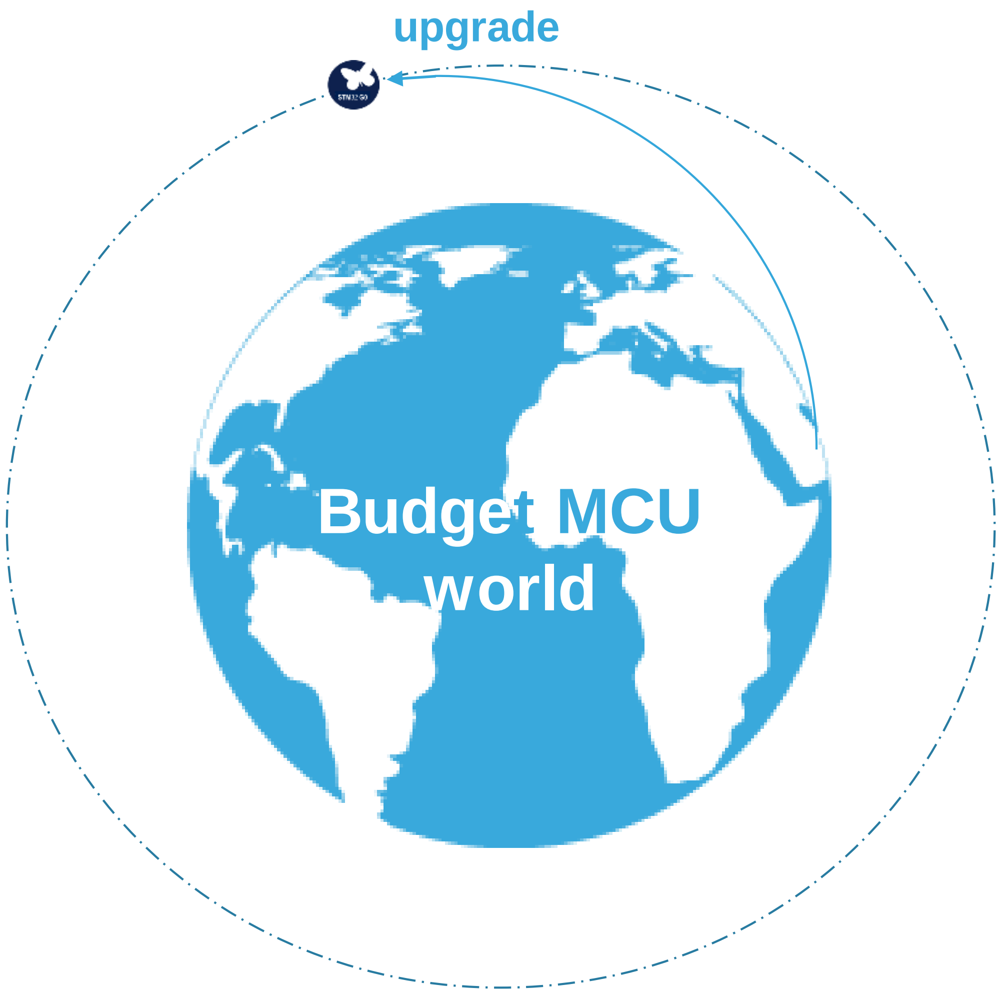
Note : * without RTC / with RTC

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

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

life.augmented

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 90°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

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

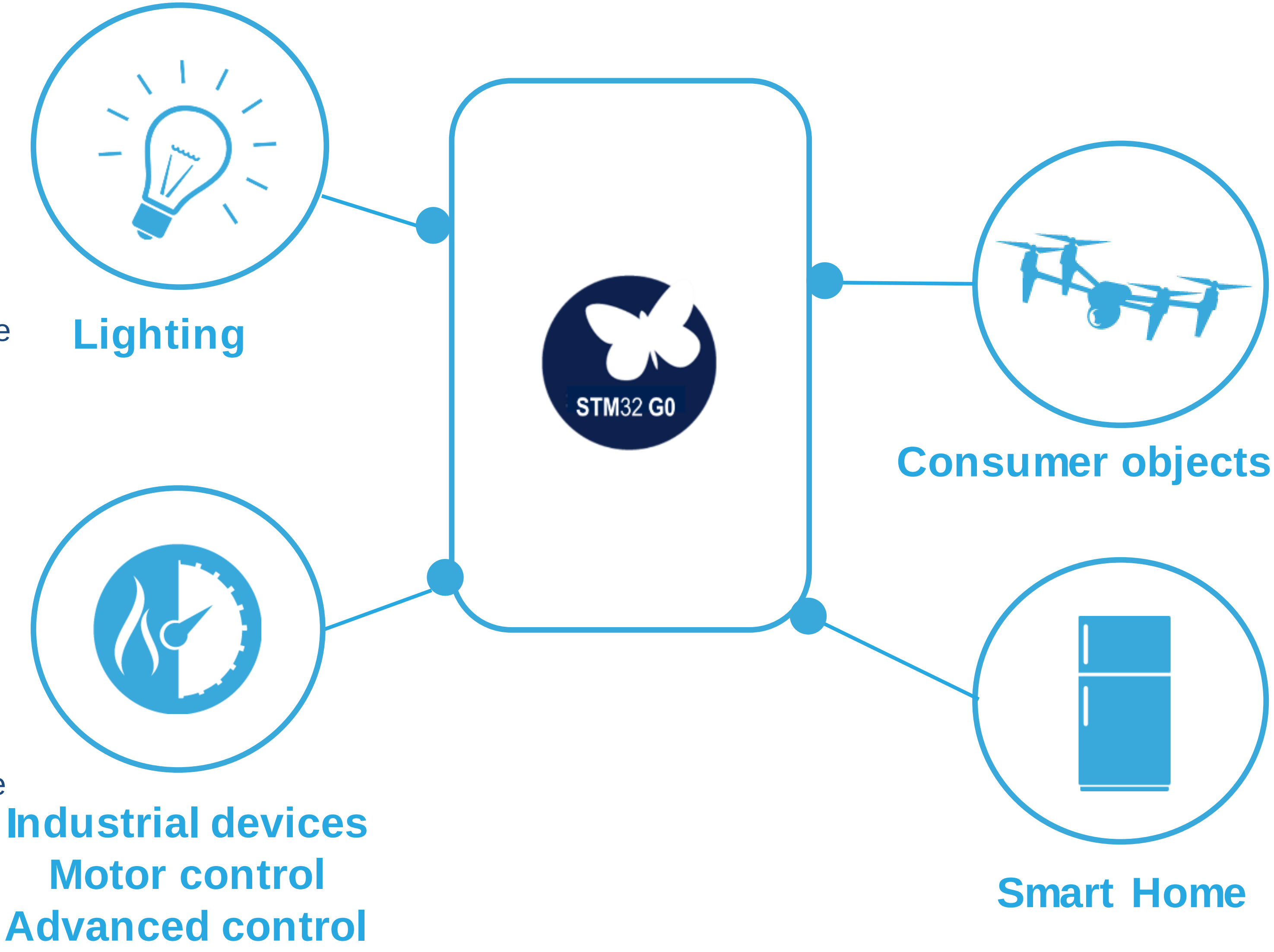
Highly immune to fast-transients
Robust IOs against negative injections

Smart applications

- 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

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



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

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

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 +85 °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

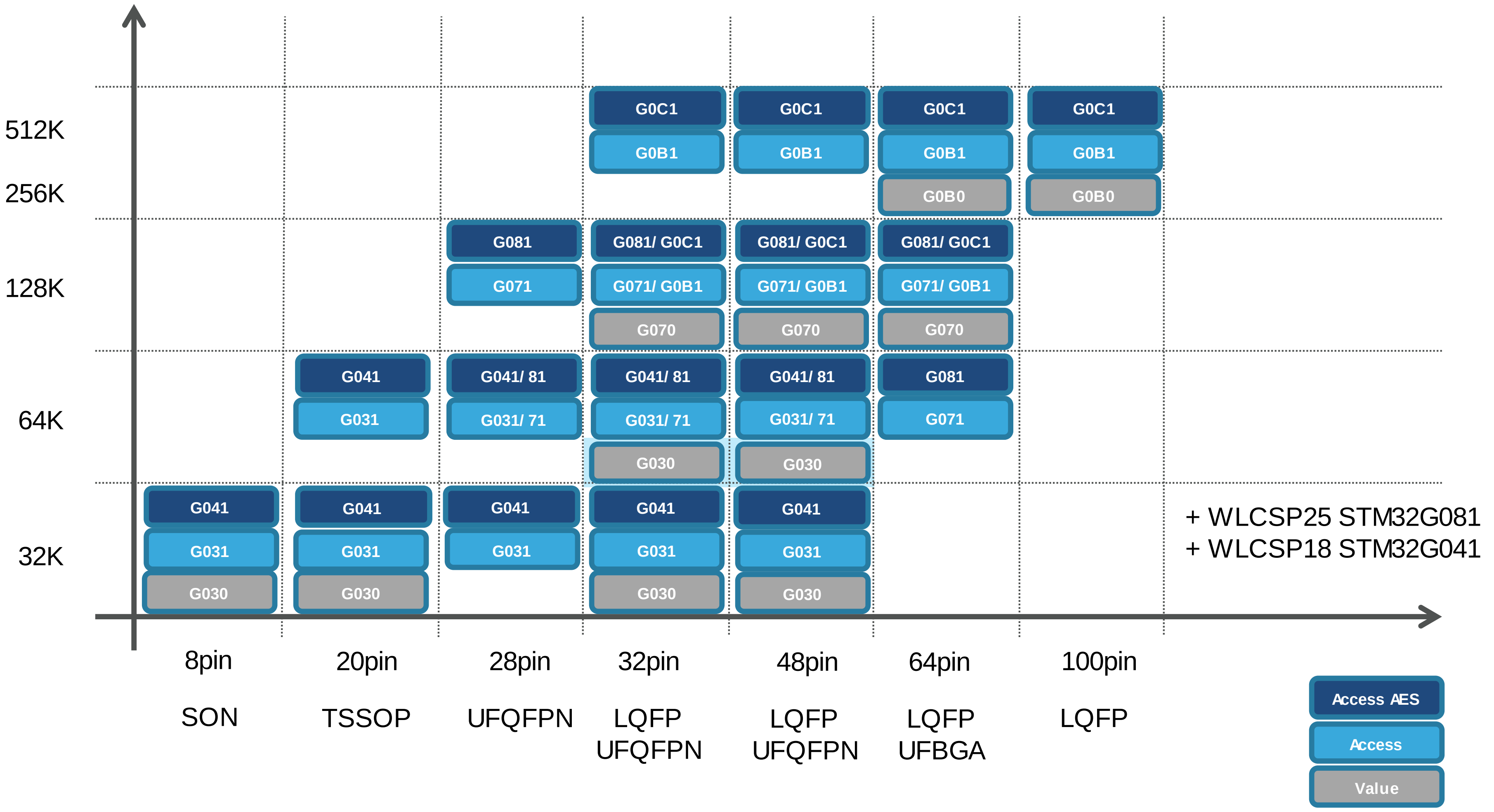
STM32G0+11 – Access Line & Encryption (ex: STM32G081)

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

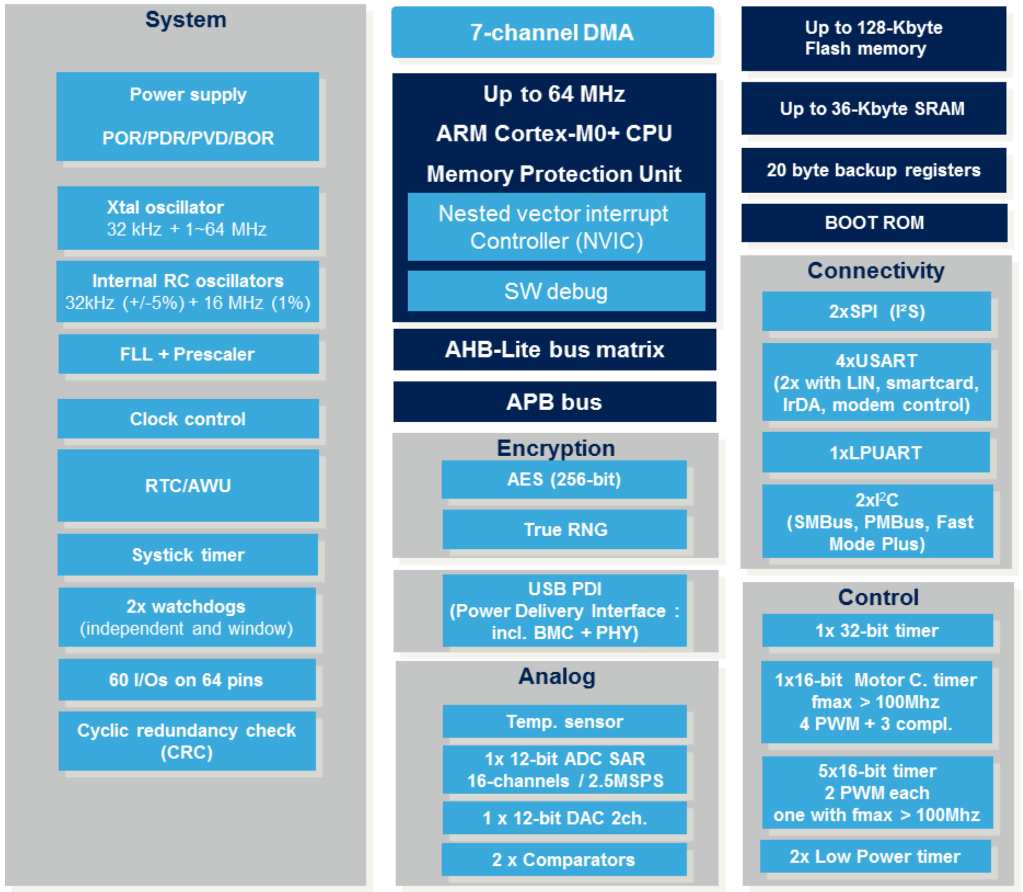
STM32G0 Portfolio



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

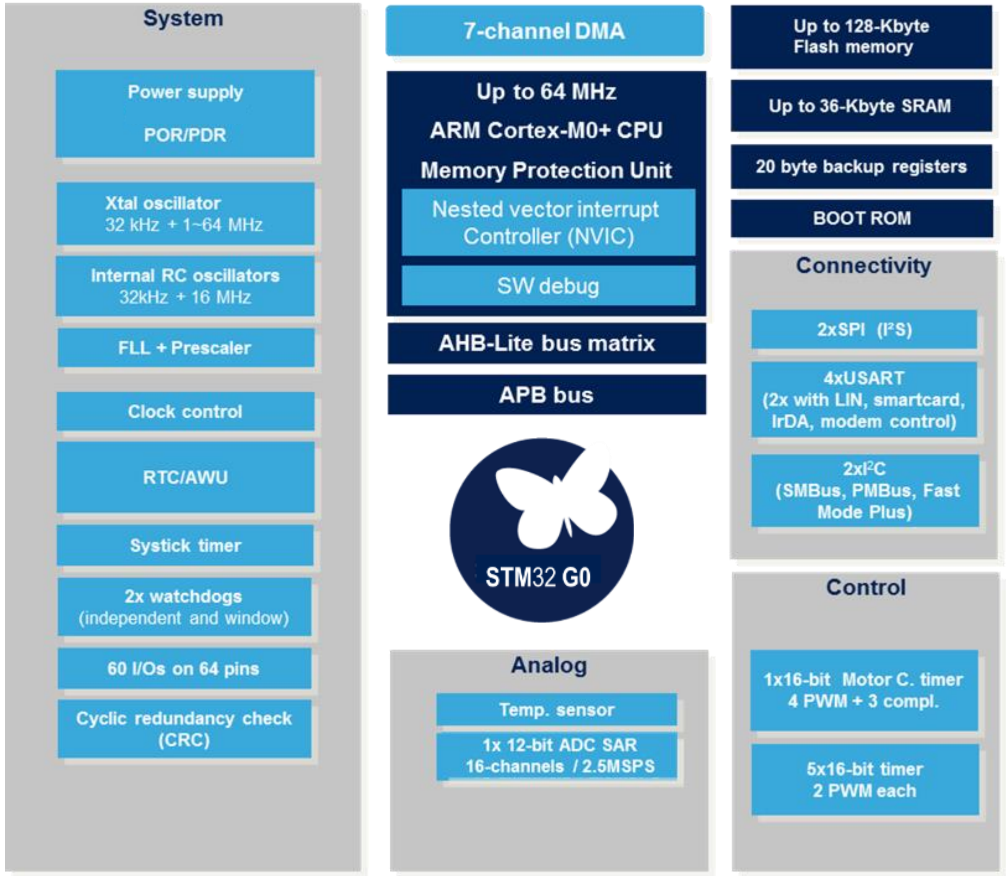


- Timers up to 2xfcpu resolution
- Real Time Clock
- I/O ports maximization
- ADC 12-bit Ultra-fast
- DAC 12-bit
- Comparators
- Safety features
- Advanced Security features

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

More **security**

Integrated security features, ready for tomorrow’s needs

Firmware IP protection

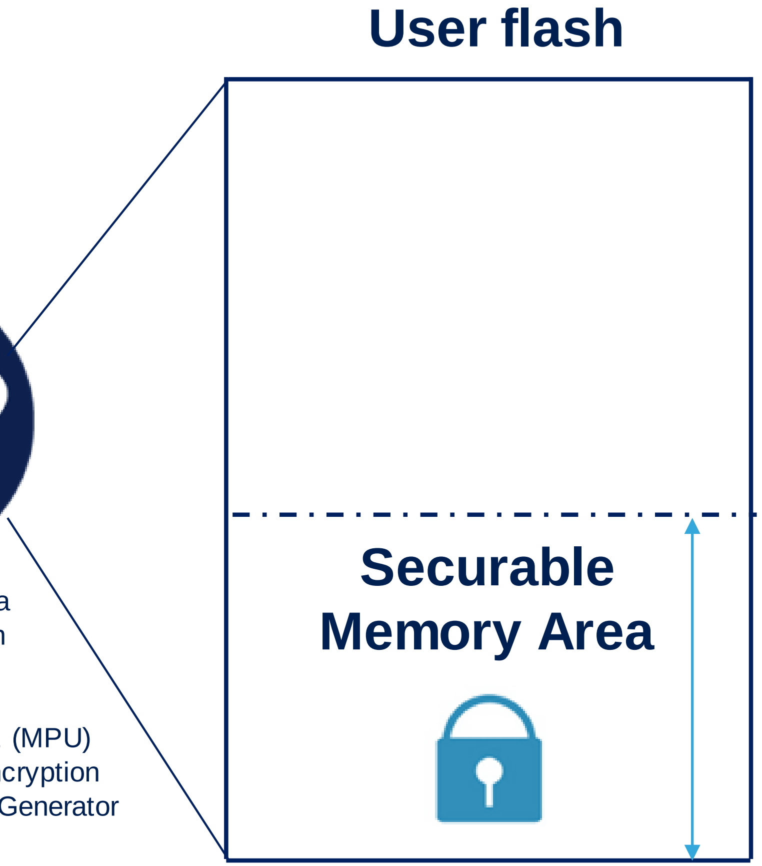
Mutual distrustful

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

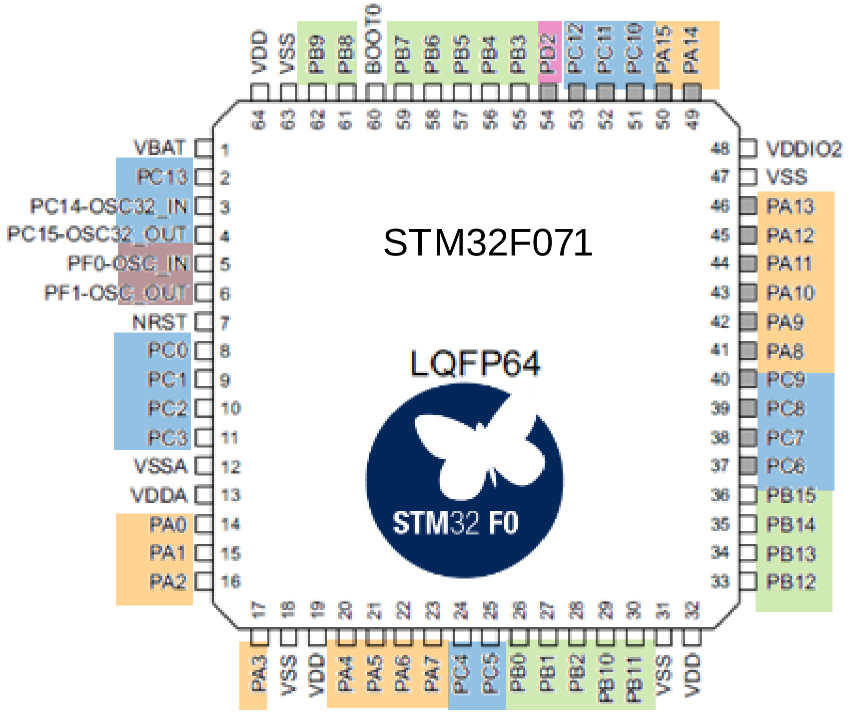
Can be secured once exiting
No more access nor debug

Configurable size

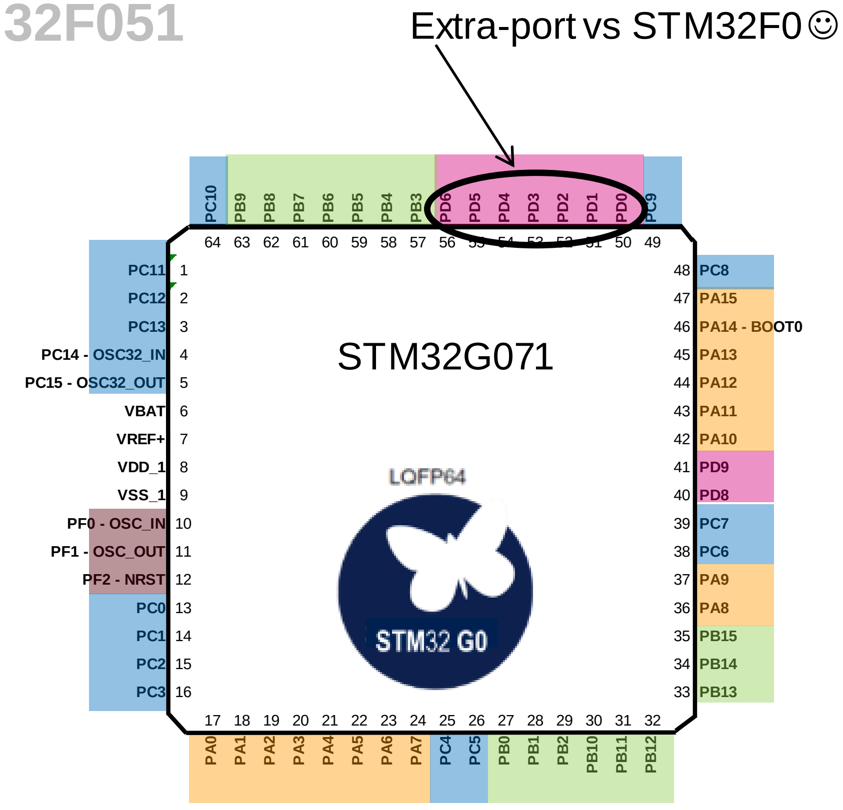
Good fit to store critical data
- **Critical routines**
- **Keys**

LQFP64 pin-to-pin comparison

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

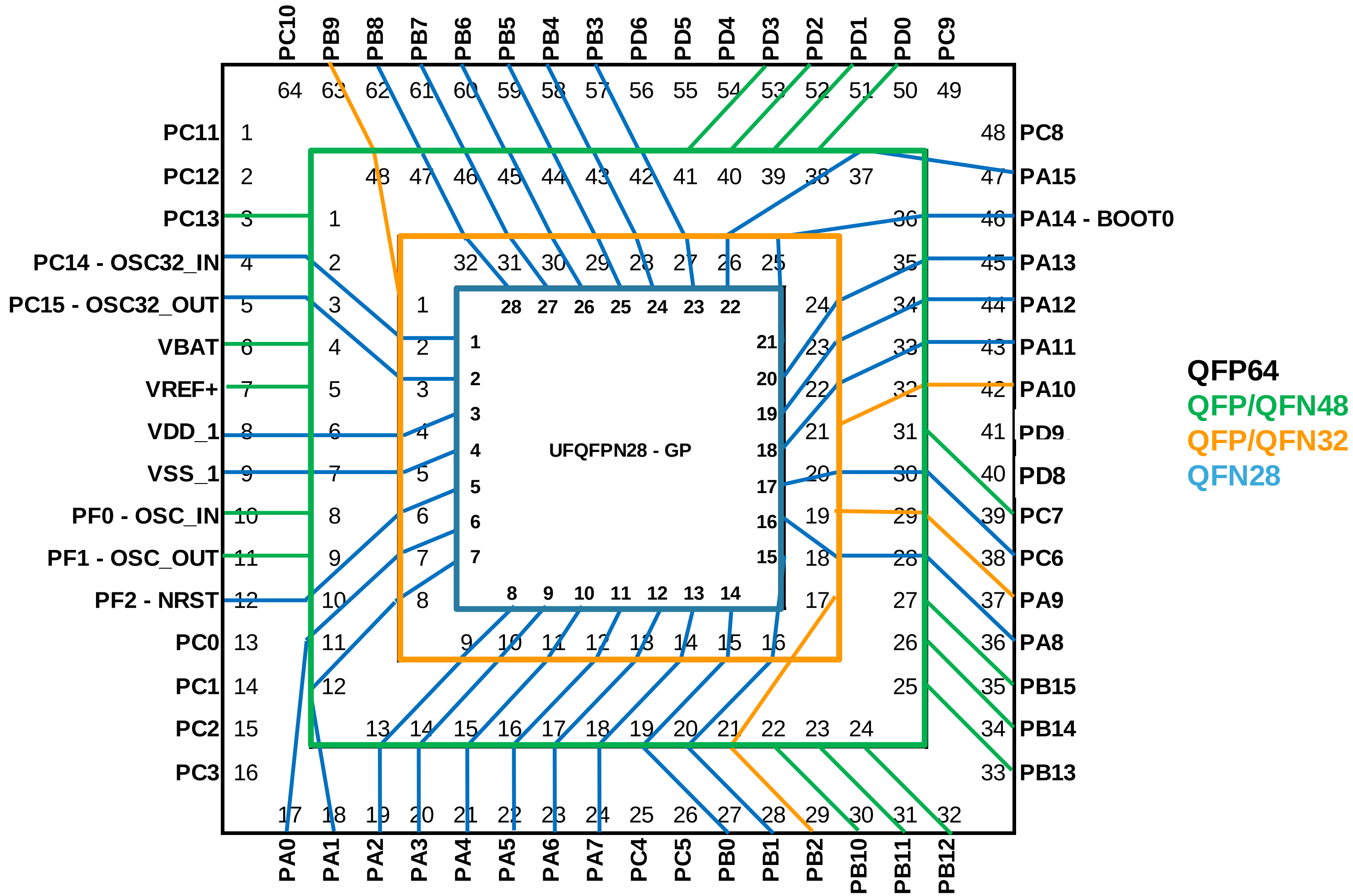


51 I/Os



60 I/Os

Consistent and optimized pinout

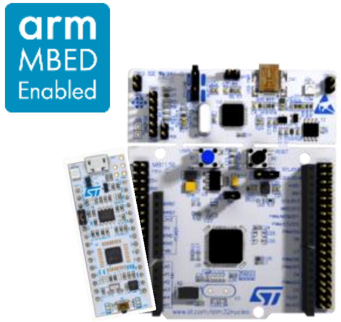


STM32G0 ecosystem

Go fast, be first

HARDWARE TOOLS

STM32 Nucleo



Flexible prototyping

Discovery kit

Key feature prototyping

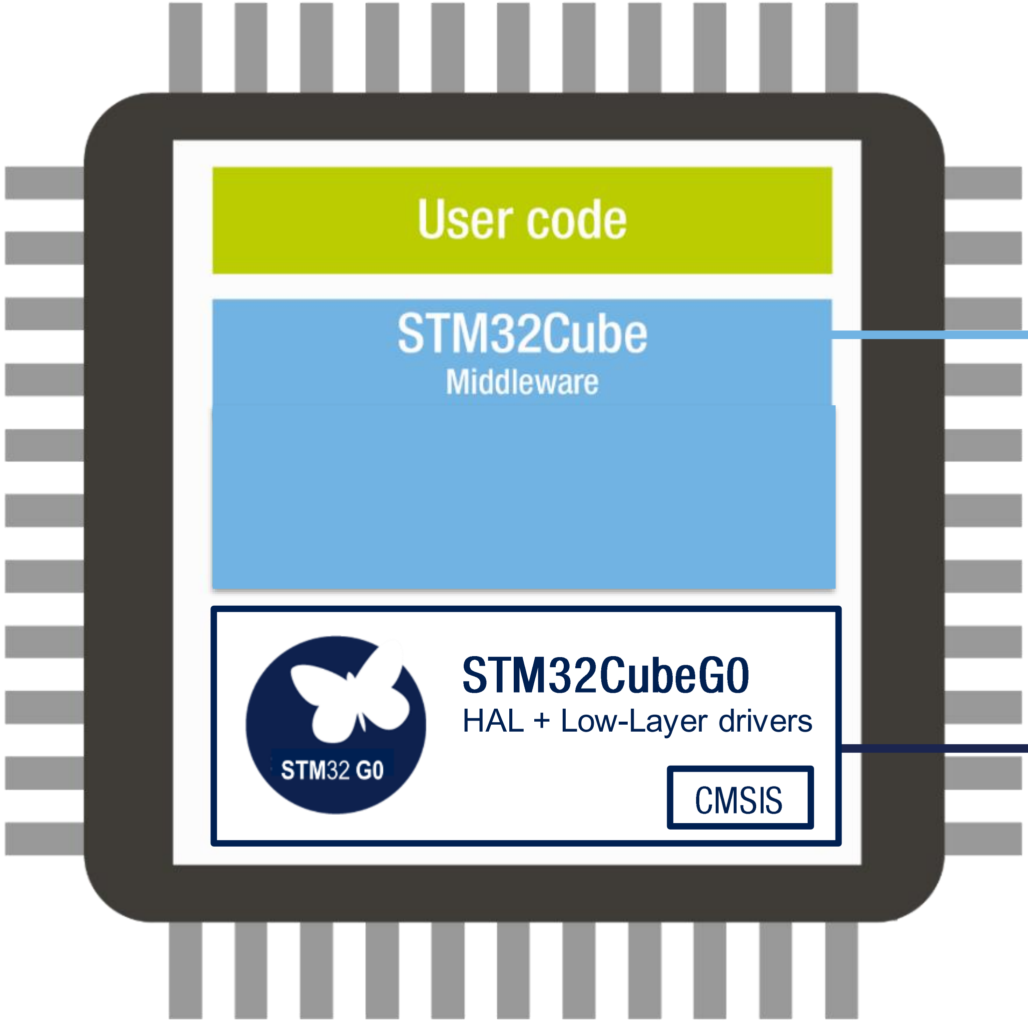
Evaluation board

Full feature evaluation

SOFTWARE TOOLS

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

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**

Continue with the tools installation

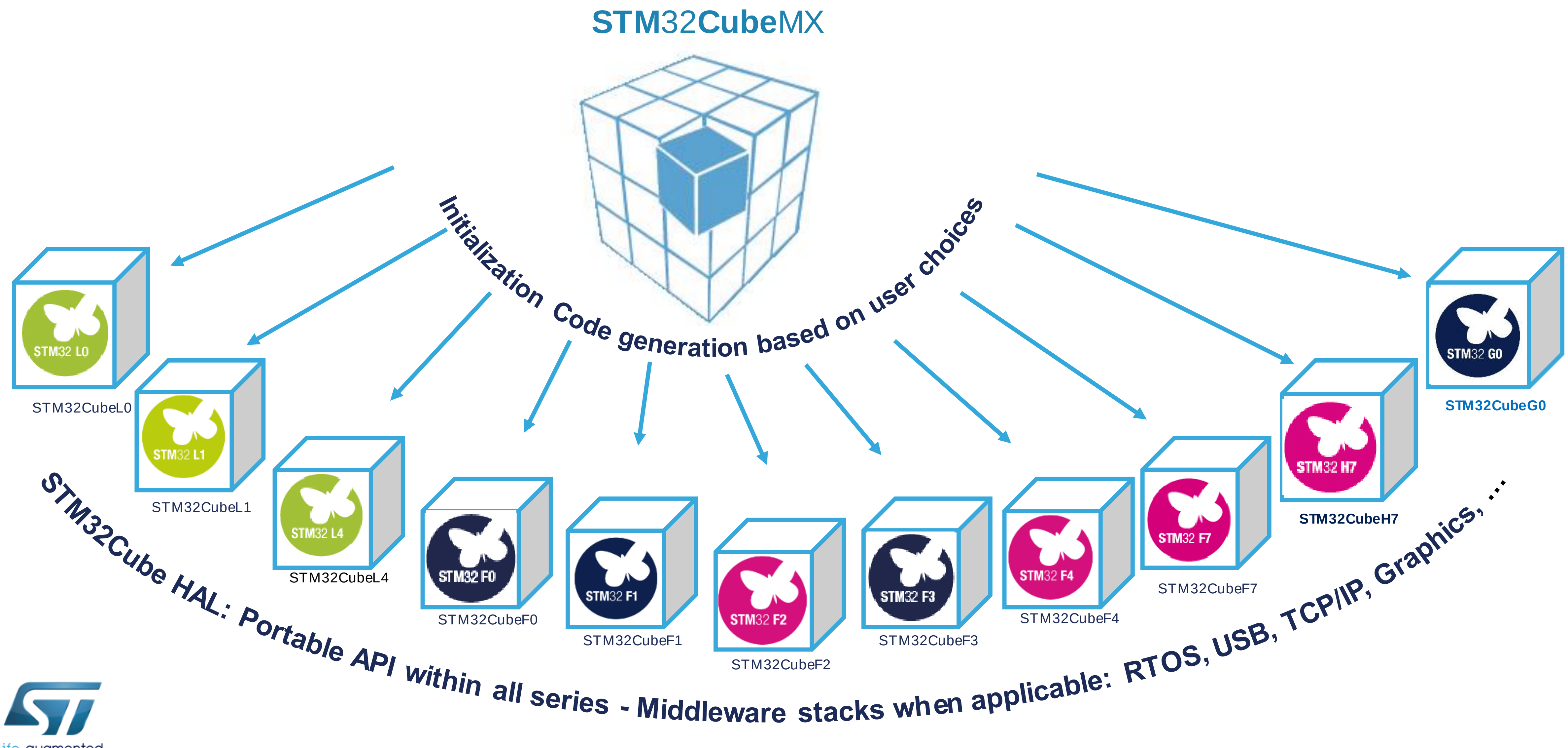
- By this time Keil uVision 5 should be installed
- Install STM32G0 Pack for Keil uVision5 (From Pack Installer)
- Activate Keil uVision5 (from Keil uVision5)

For more details on the tools installation, refer to the document "STM32G0 Workshop Installation from USB drive_v1.0.pdf" part of the files you copied from the USB drive

STM32CubeMX 5.0

STM32CubeMX graphical software configuration tool

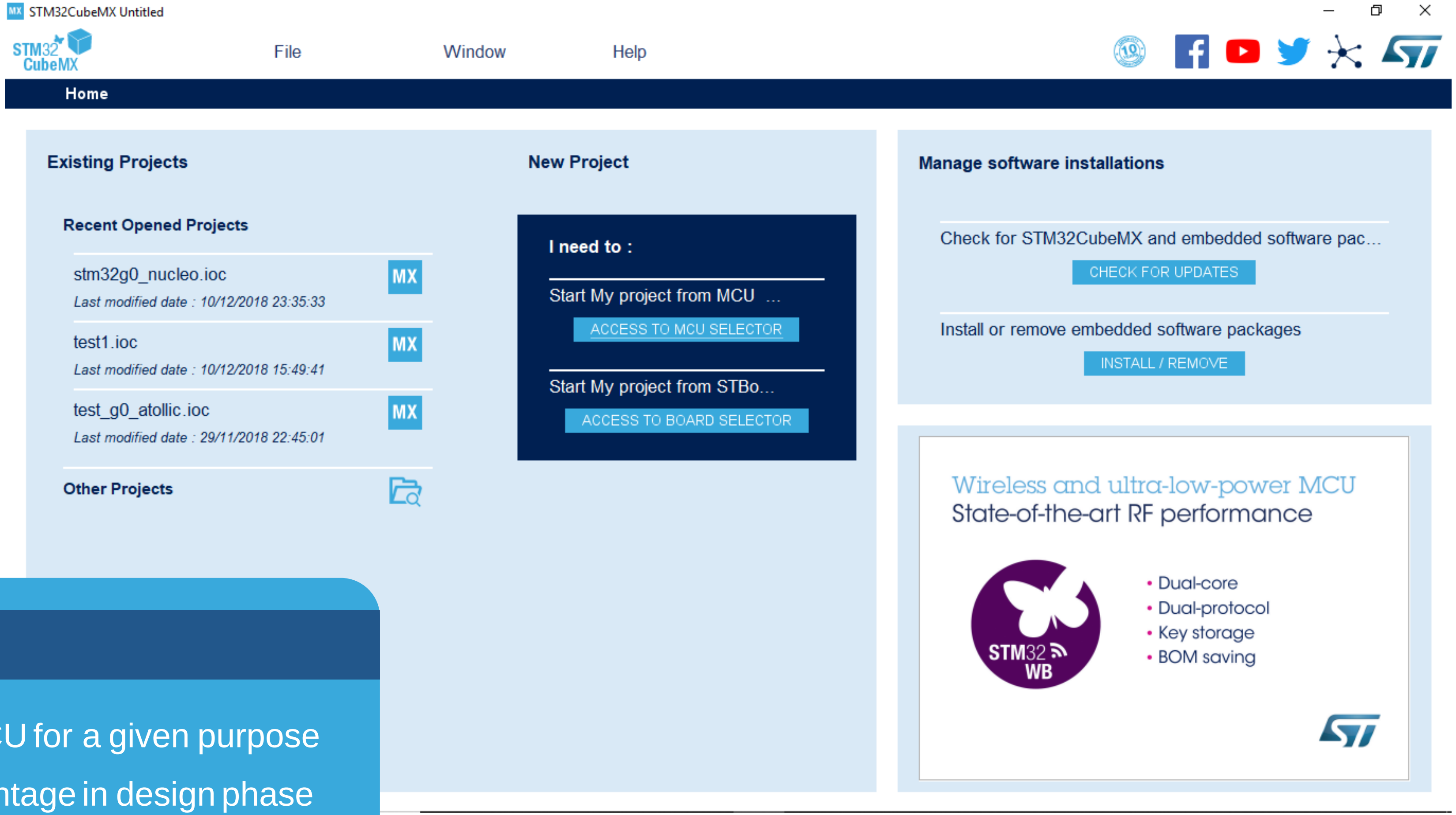
life.augmented



life.augmented

Overview

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



Application benefits

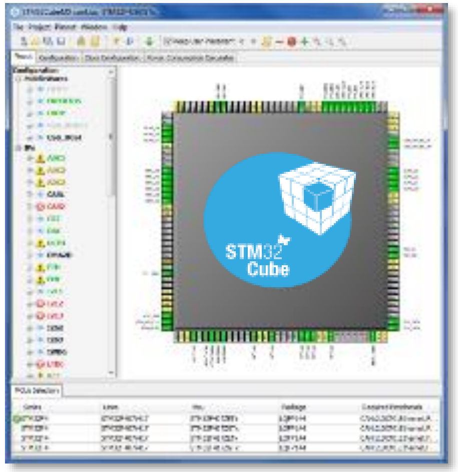
- Helps choose the correct MCU for a given purpose
- Simulation provides an advantage in design phase
- Boosts development speed with a headstart

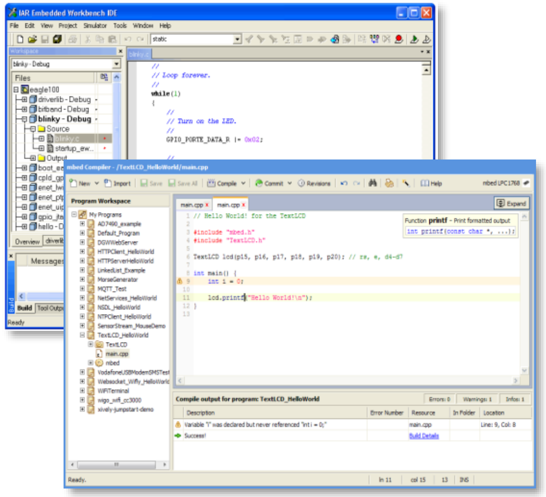
life.augmented

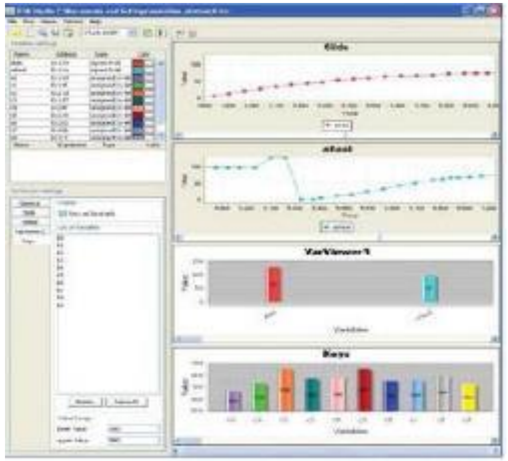
Key features

- 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	Partners IDEs	STMStudio
Generate Code	Compile & Debug	Monitor

IAR SYSTEMS | iSYSTEM | EMPROG | TASKING | KEIL LABS | SEGGER Embedded Studio

ARM KEIL Microcontroller Tools | CooCox | atollic | mbed | ac6

Free IDE

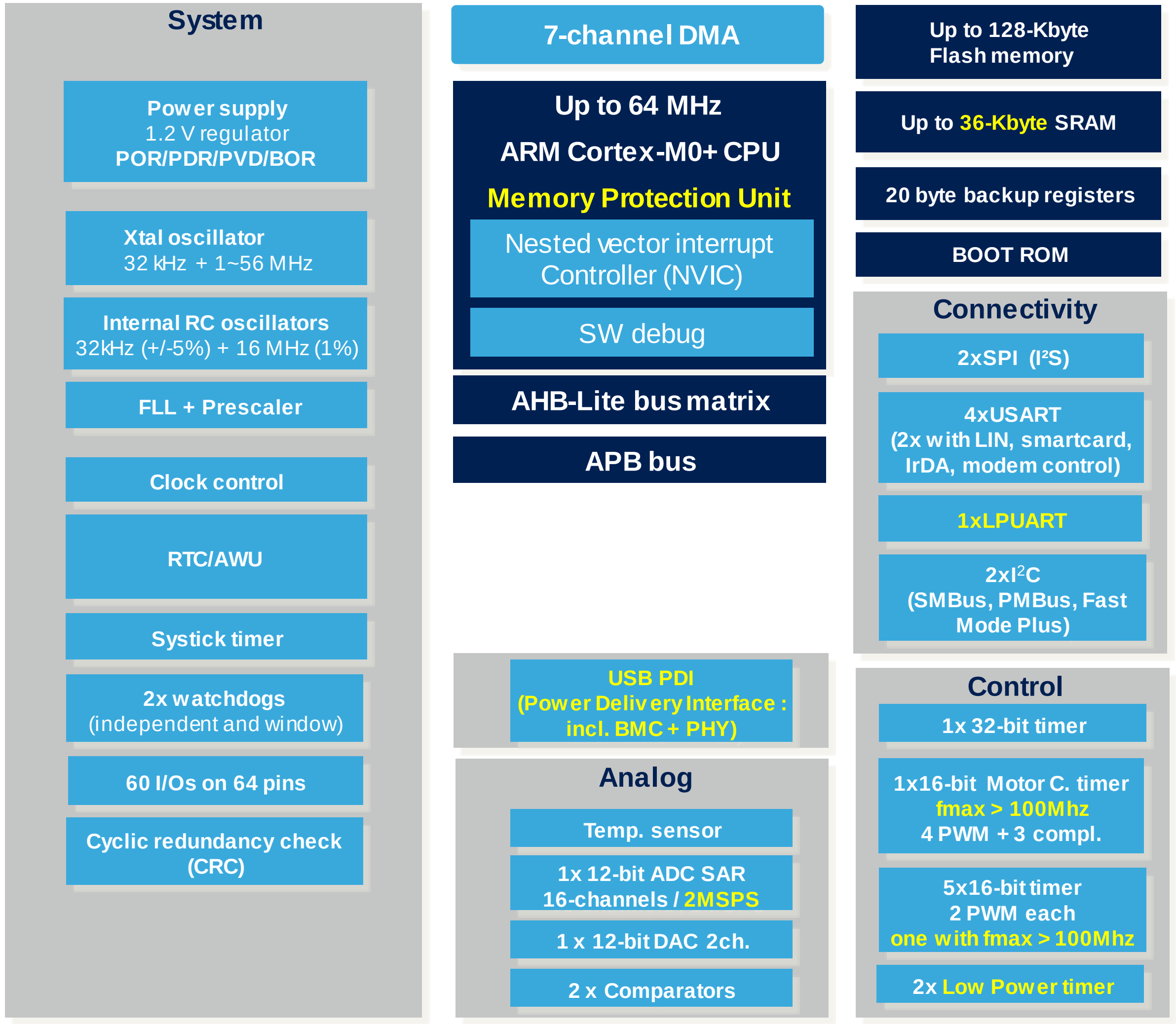
life.augmented

STM32G071 Block diagram

Main specification

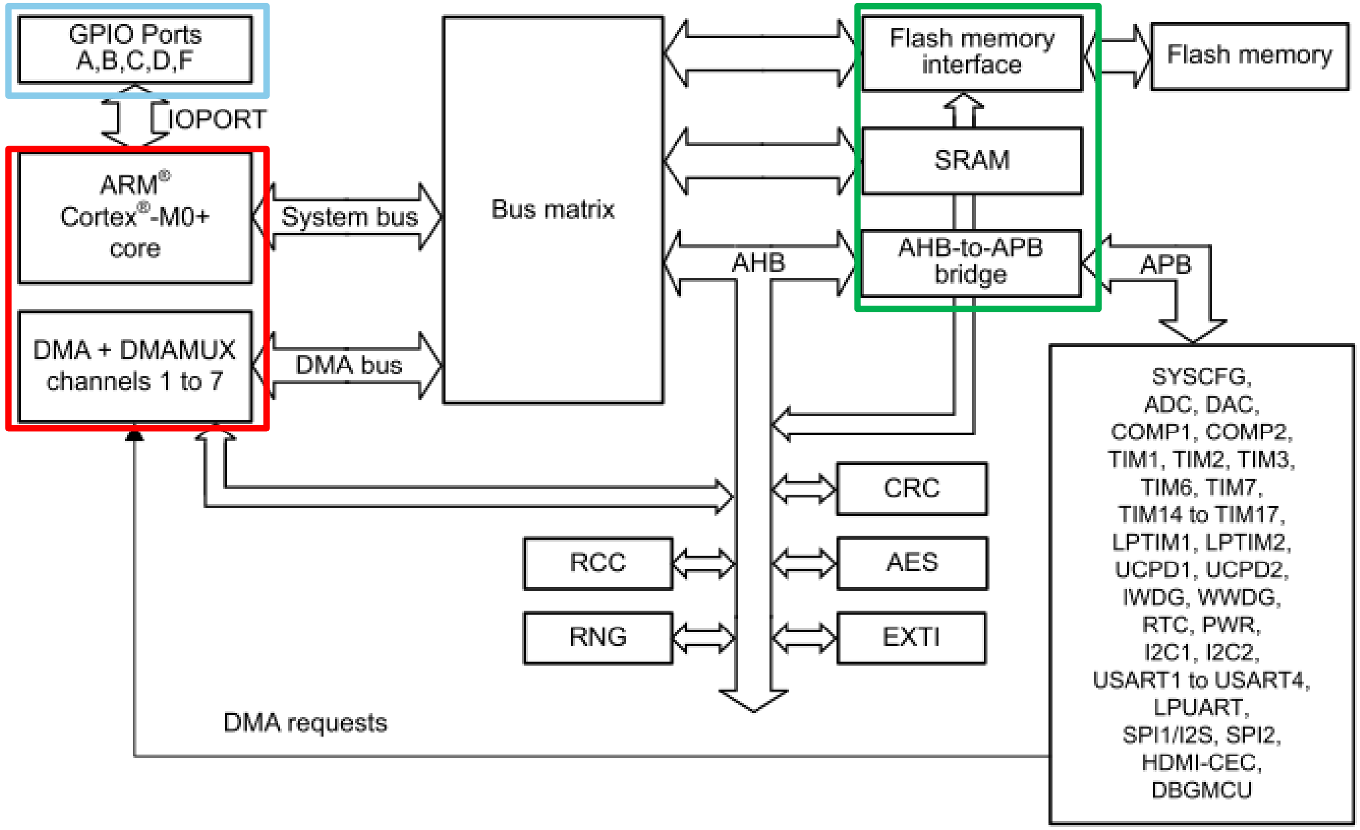
- 1.65V to 3.6V
- 0.93 DMIPS/MHz
- **Vbat** supply
- Vref+ pin
- **Max ambiant temp 125°C**
- One Supply pair
- Securable Memory Area
- High sink I/Os
- <100µA/MHz run mode
- **Stand-by** <1µA @ room temperature
- Stop 5 µA @ room temperature
- **Shutdown mode**
- Low EMI SAE (2.5@24MHz)
- Robust EMC/ESD/EMS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

features highlight



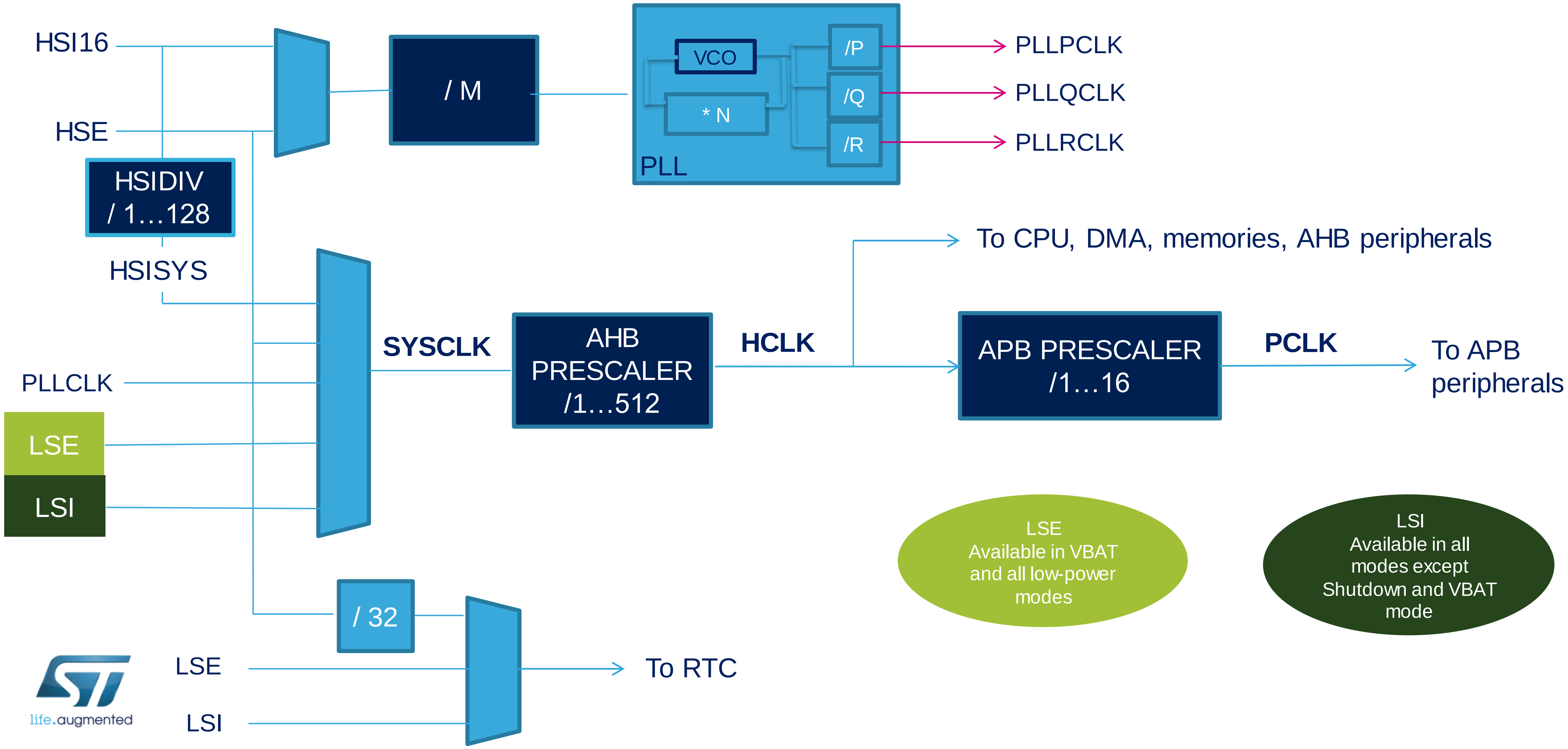
life.augmented

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

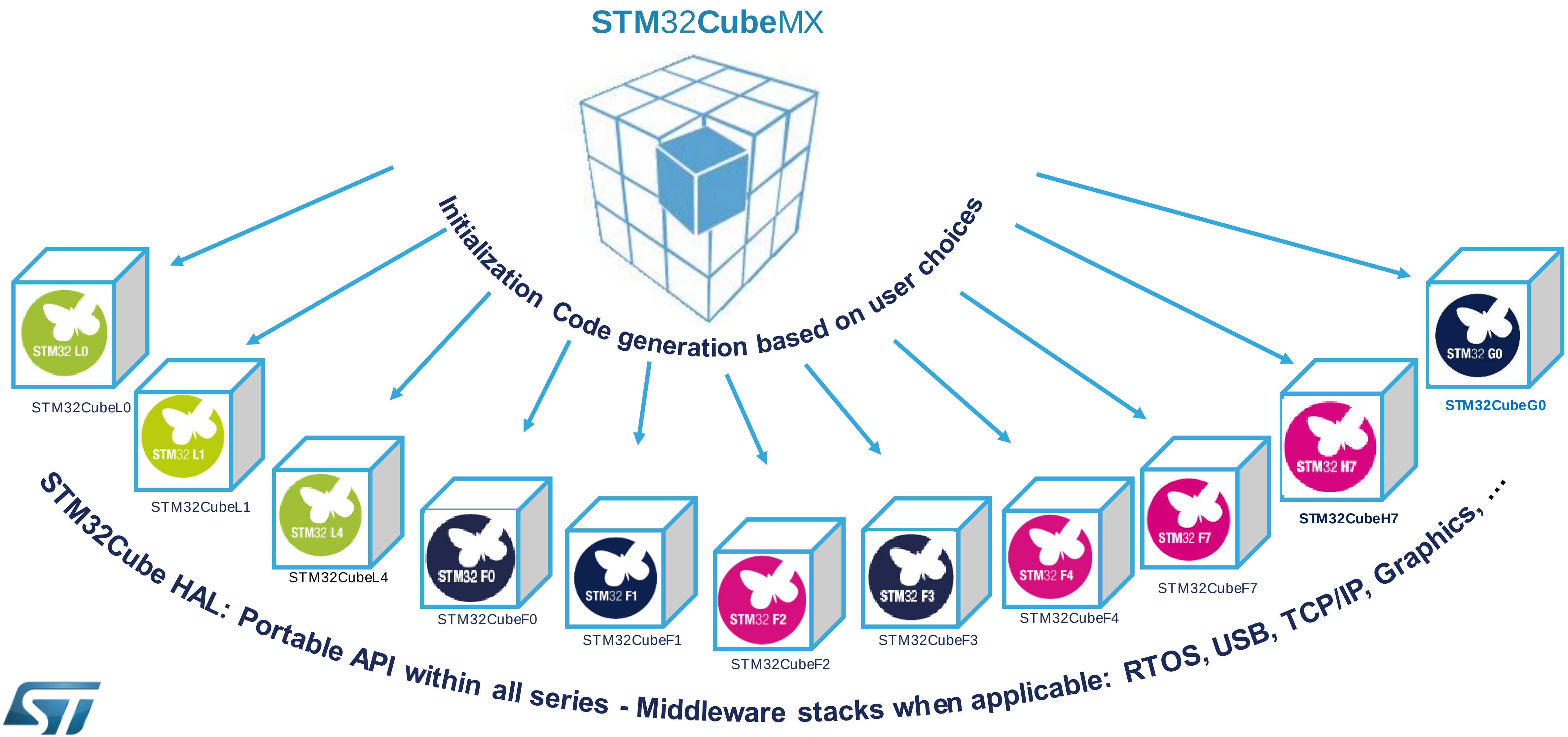
Simplified clock tree



STM32Cube and STM32CubeMX

life.augmented

Introduction

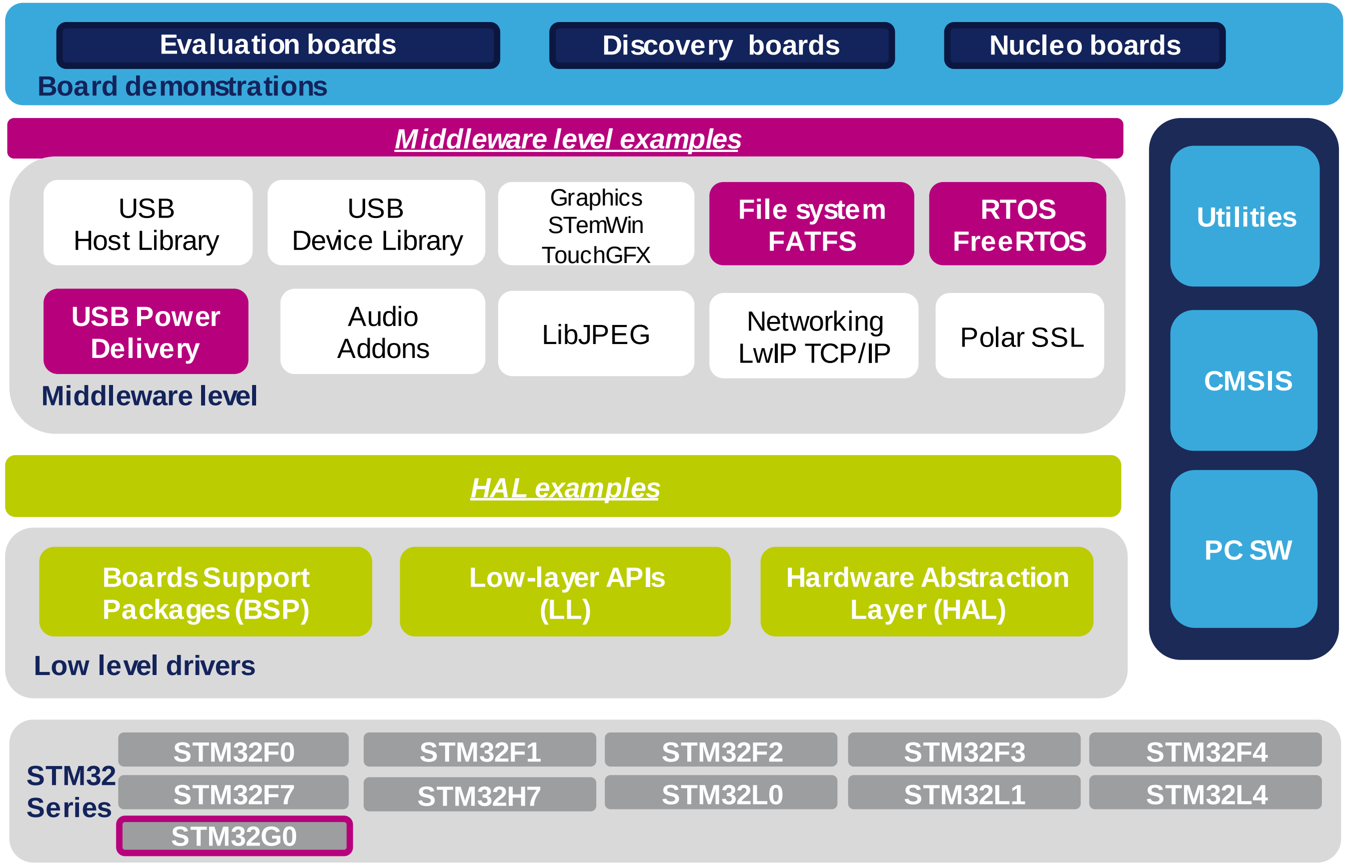


life.augmented

STM32Cube

life.augmented

Overview



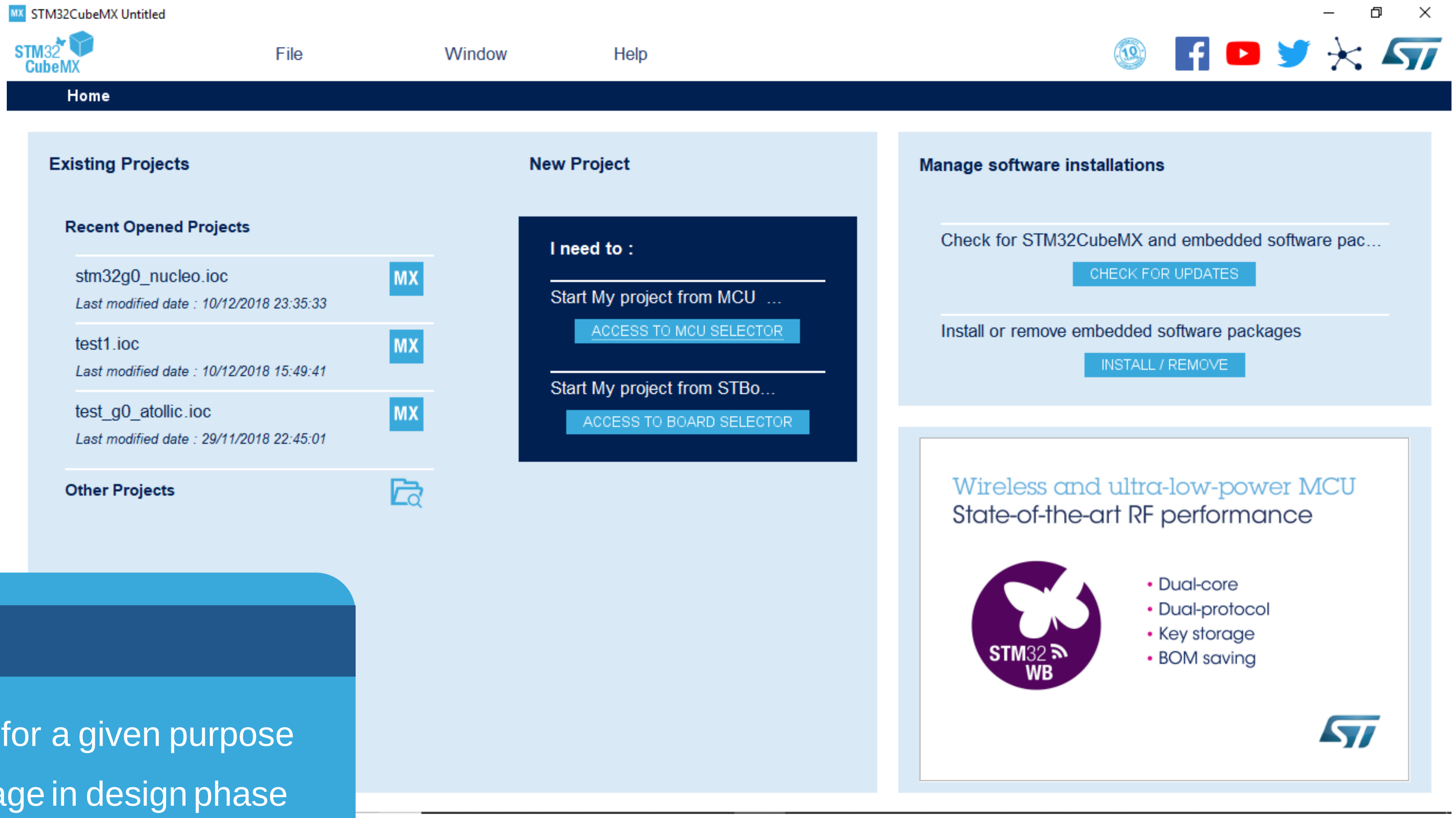
Application benefits

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

STM32CubeMX

life.augmented

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

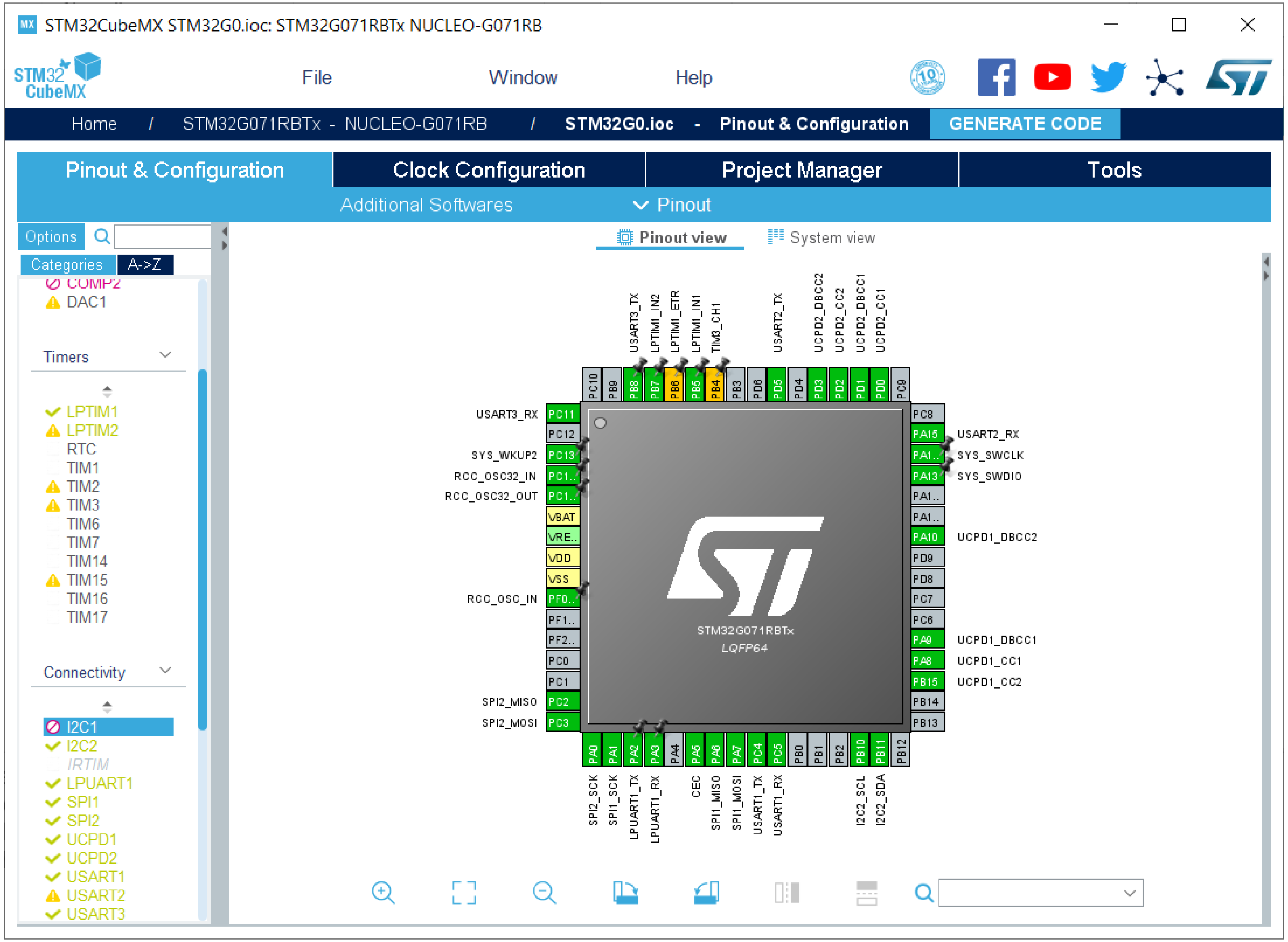


Application benefits

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

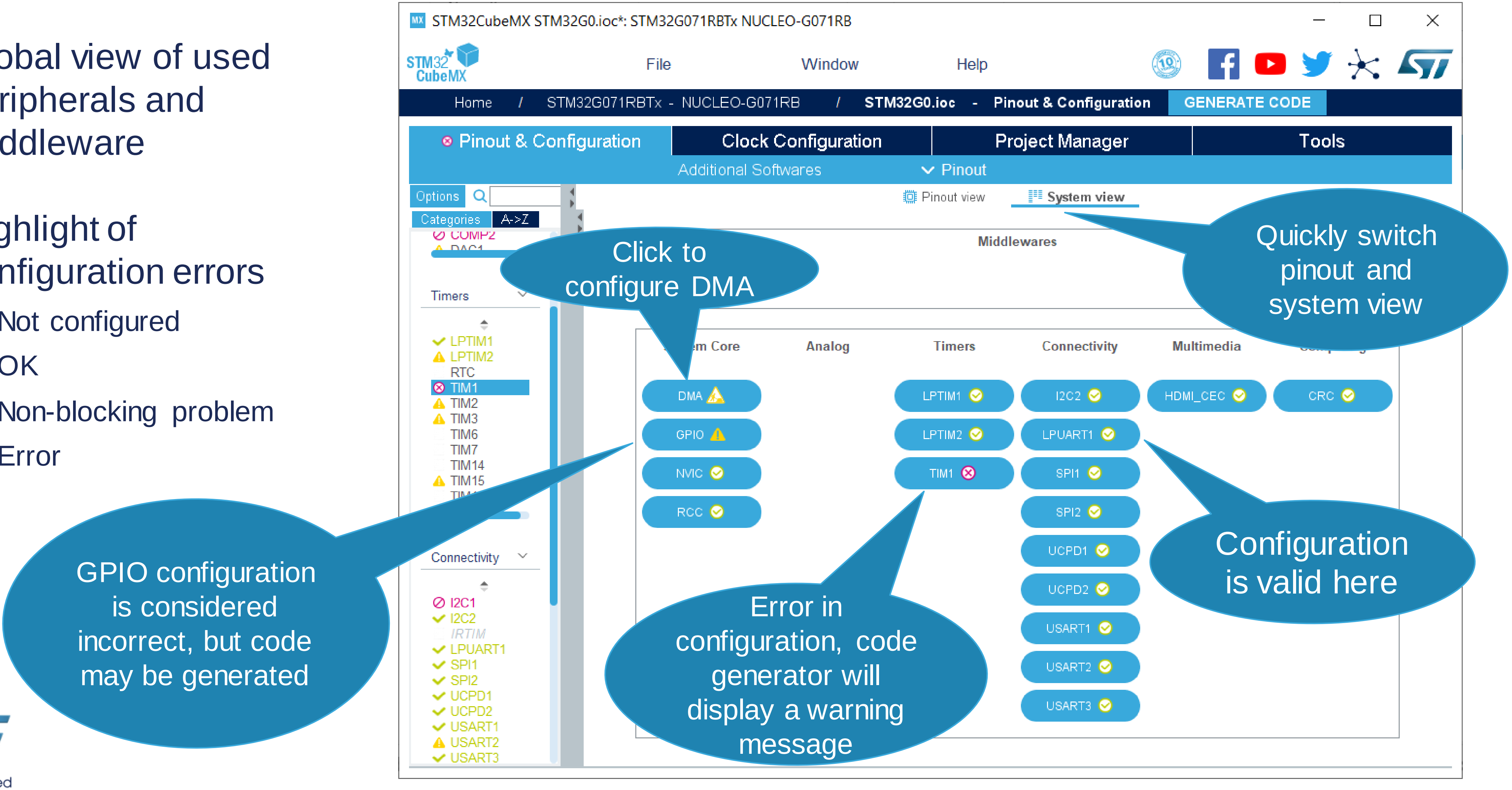
Pin assignment

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



Peripheral and Middleware configuration

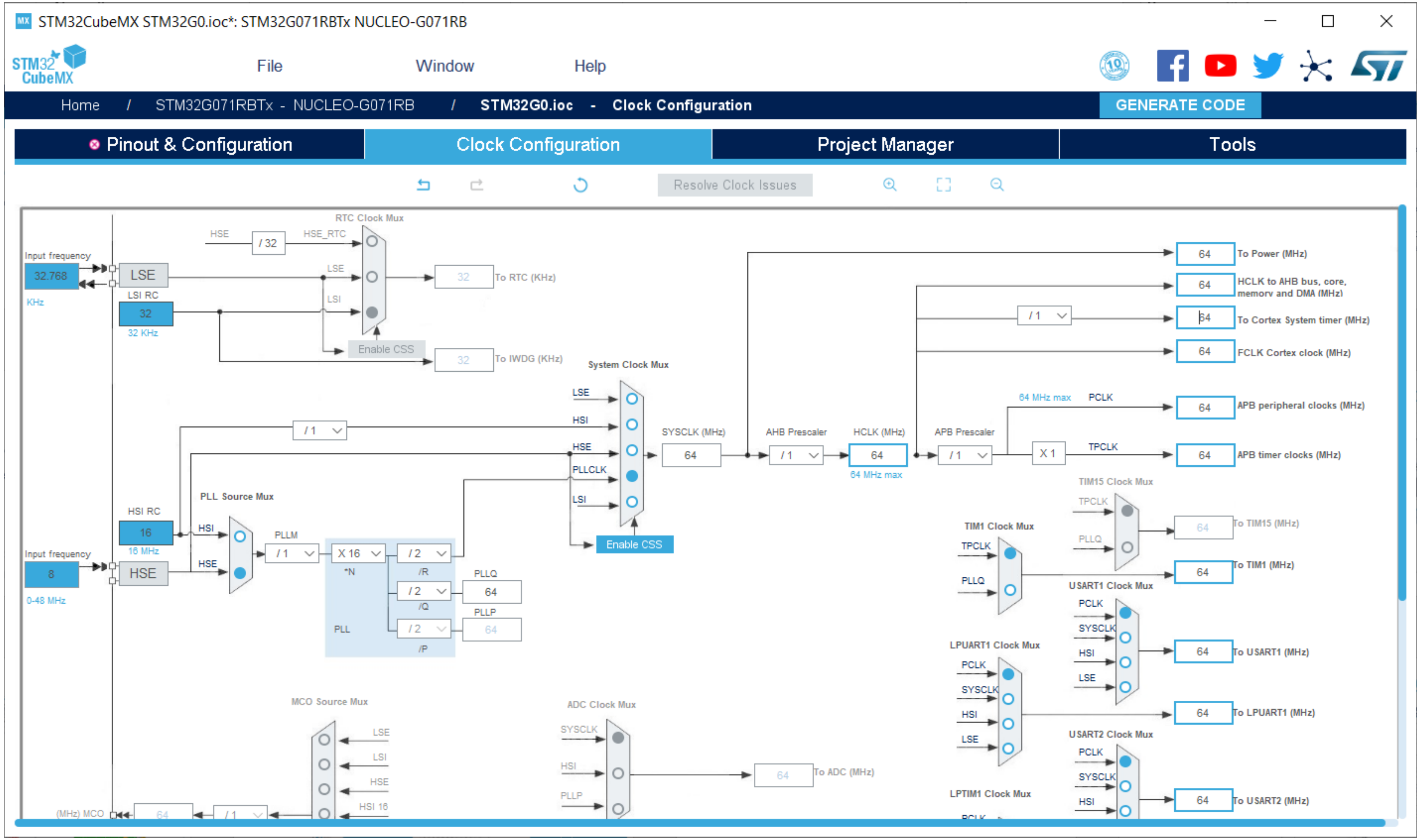
- Global view of used peripherals and middleware
- Highlight of configuration errors
  - + Not configured
  - √ OK
  - ⚠ Non-blocking problem
  - x Error



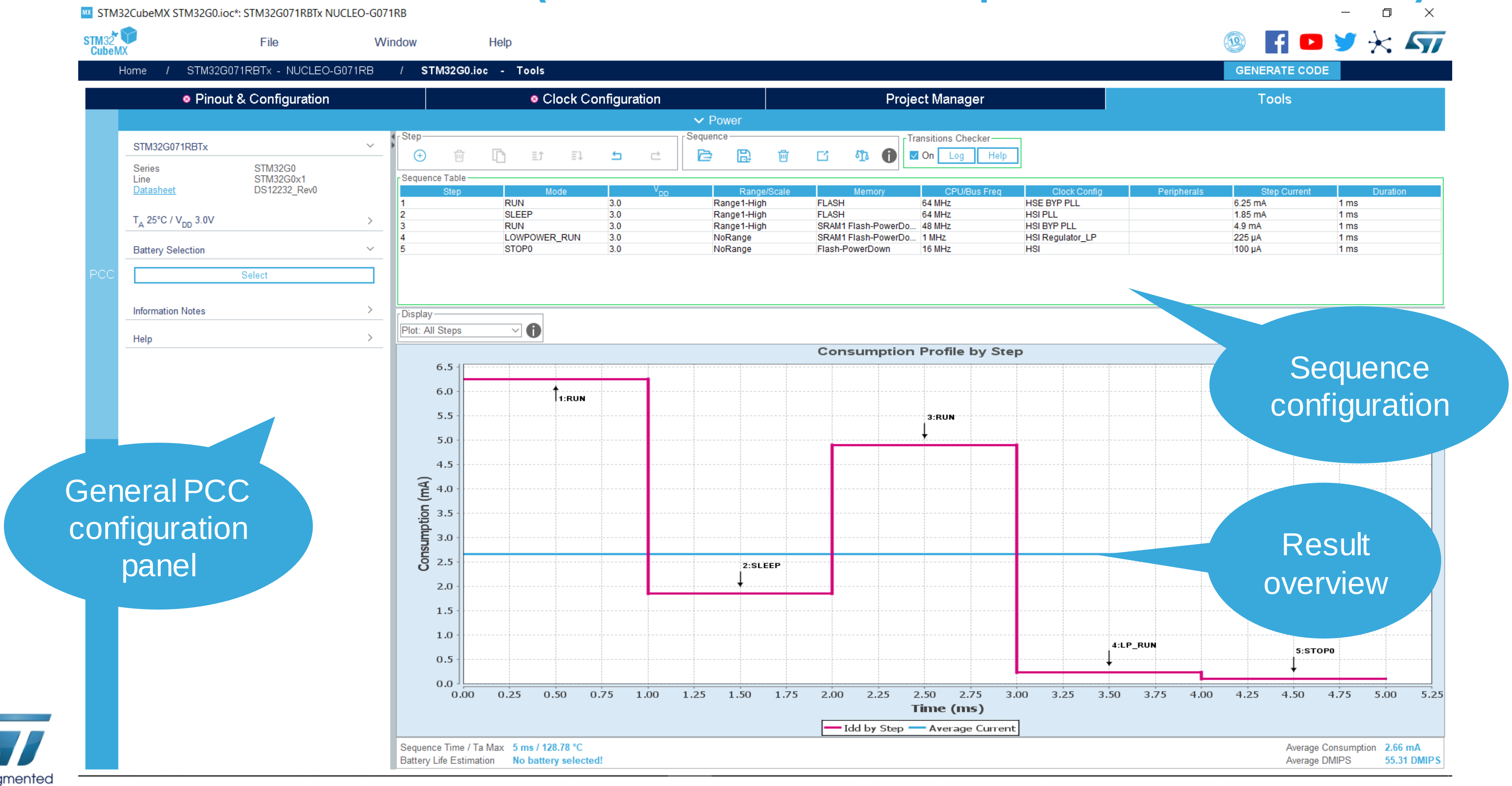
life.augmented

Clock configuration

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

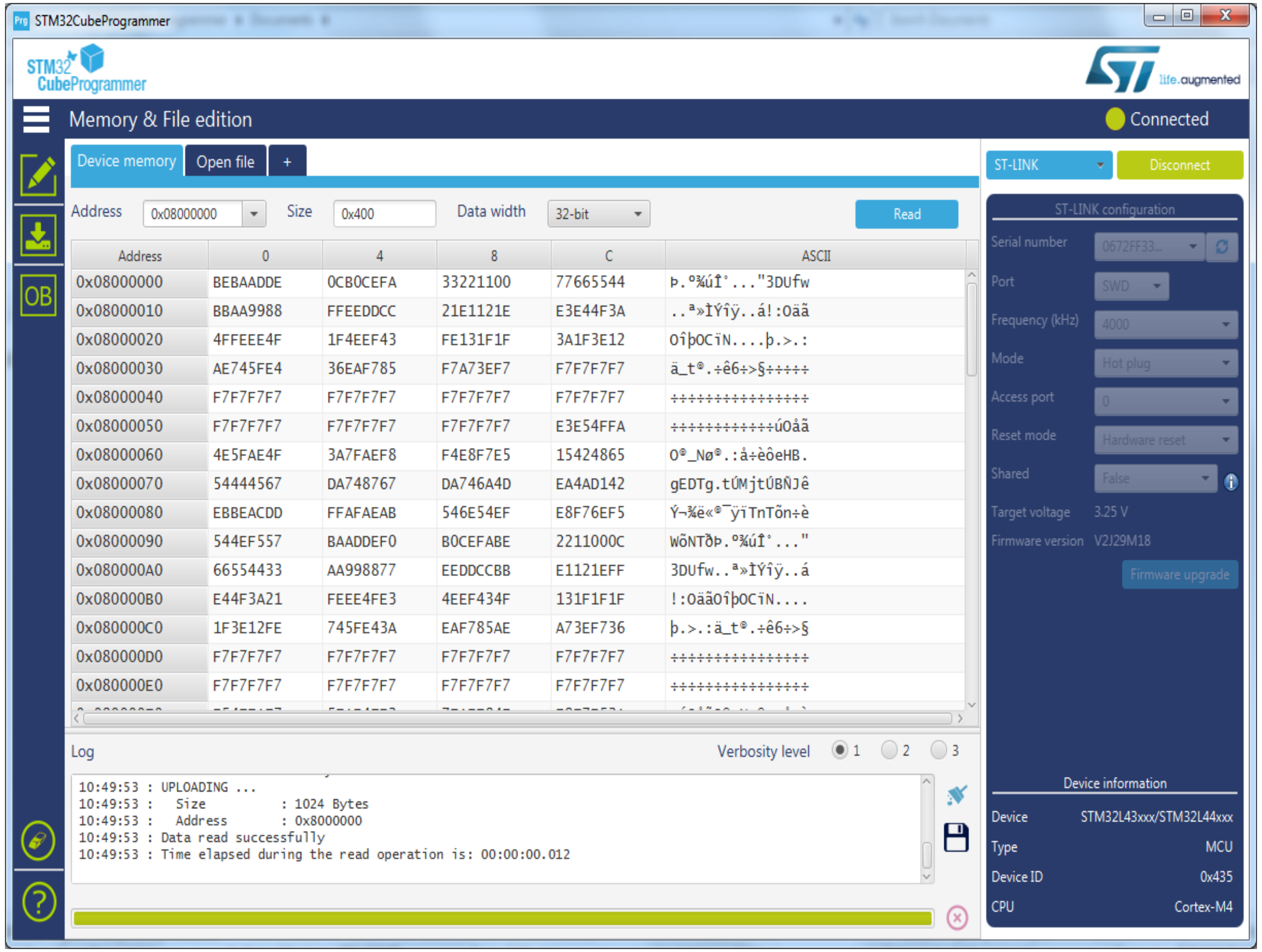


PCC (Power Consumption Calculator)



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



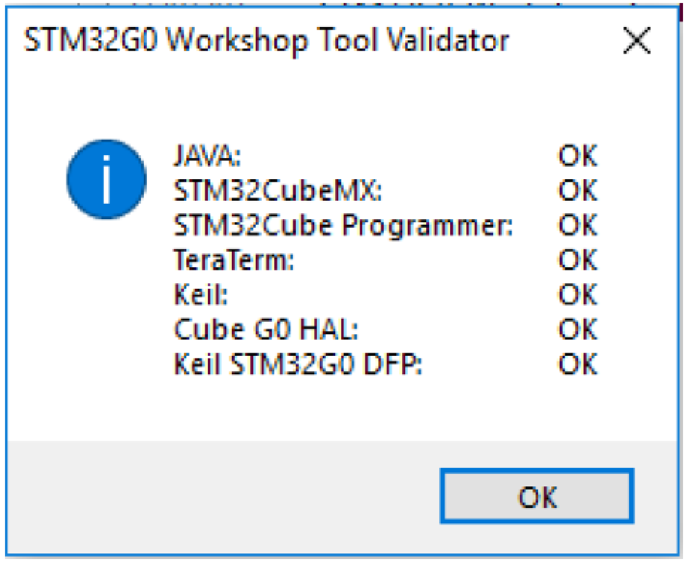
Finish the tools installation

- Install STM32CubeMX
- Install STM32CubeProgrammer
- Install STM32CubeG0 FW Library
- Tera Term

For more details on the tools installation, refer to the document "STM32G0 Workshop Installation from USB drive_v1.0.pdf" part of the files you copied from the USB drive

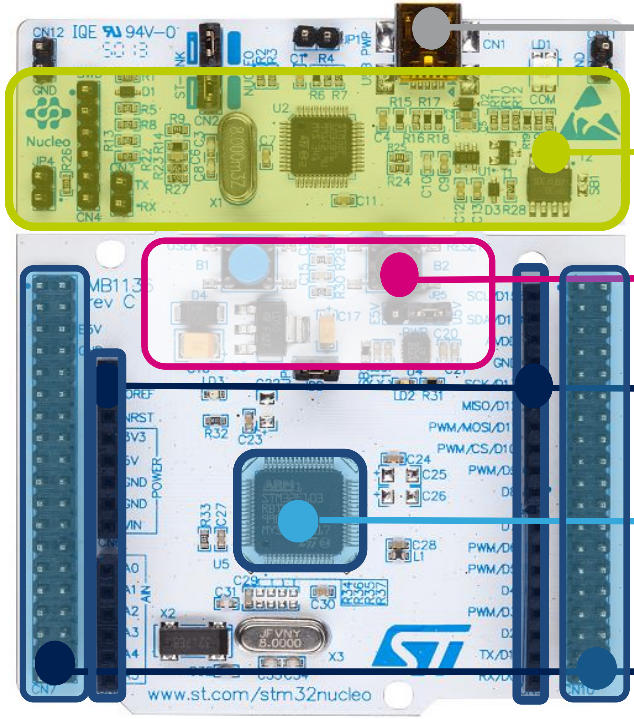
Validate the Tools Installation

- Run The Tool Validator as administrator (part of the files you copied from the USB drive) to validate the tools installation.
- When running the tool you should get:



- **If you don’t, please let us know**

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

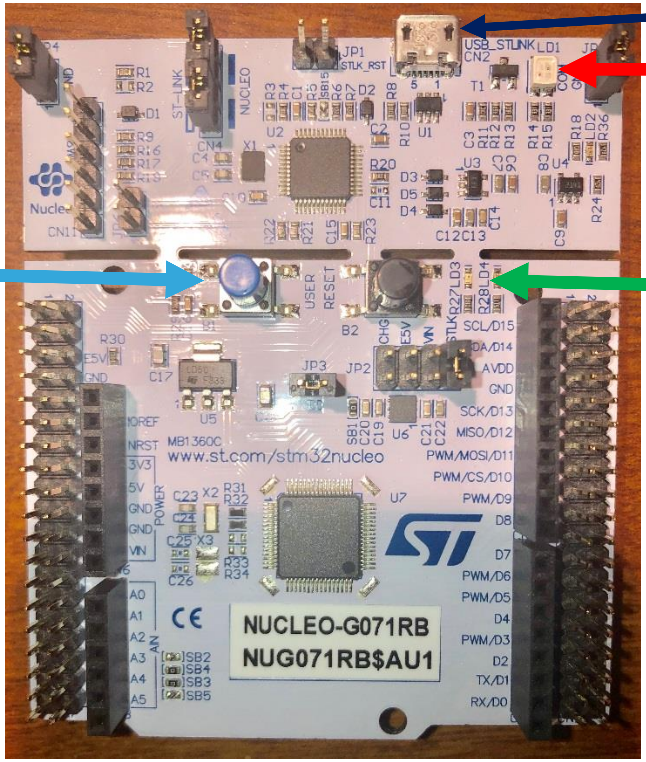
Morpho extension headers :
direct access to all MCU I/Os

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

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: Blinky

life.augmented

Objective:

- 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.

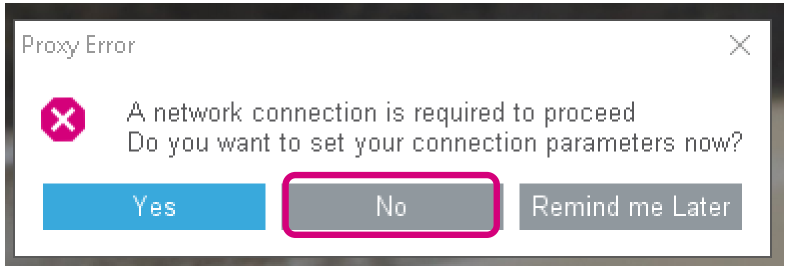
Run STM32CubeMX

1

Double Click on STM32CubeMX icon on your desktop

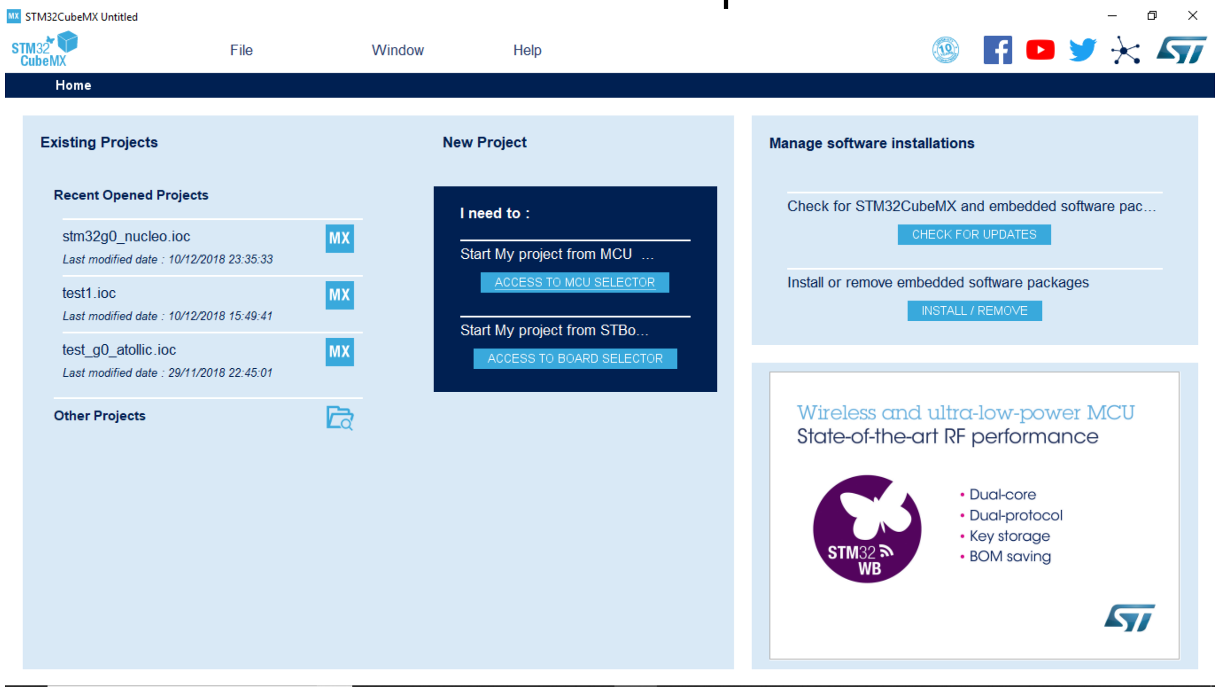
2

Click “No” when this window appears



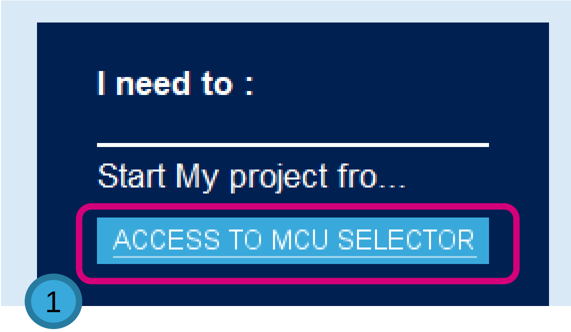
3

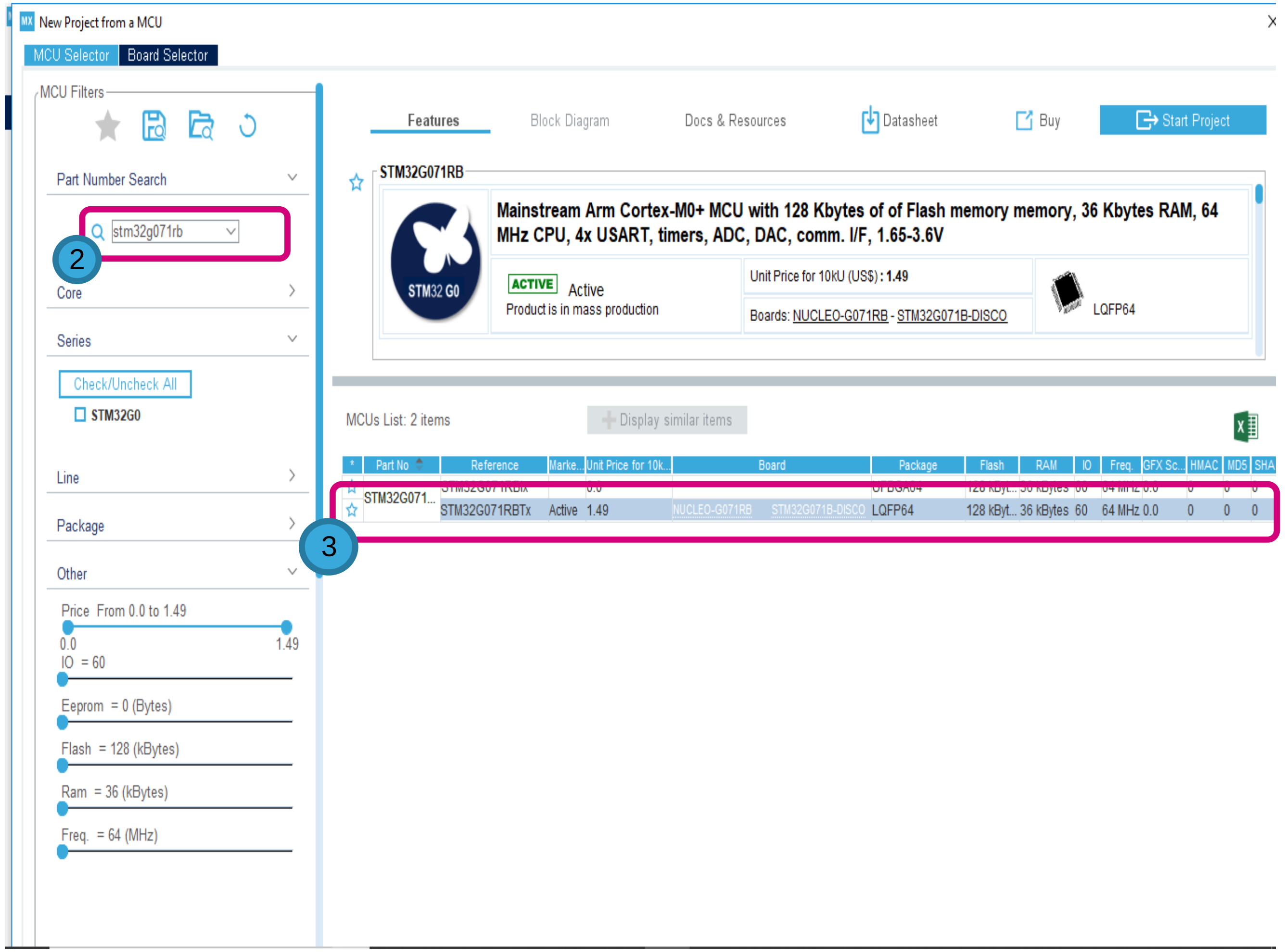
This is how STM32CubeMX will look like when open



life.augmented

Step 1: Create New Project

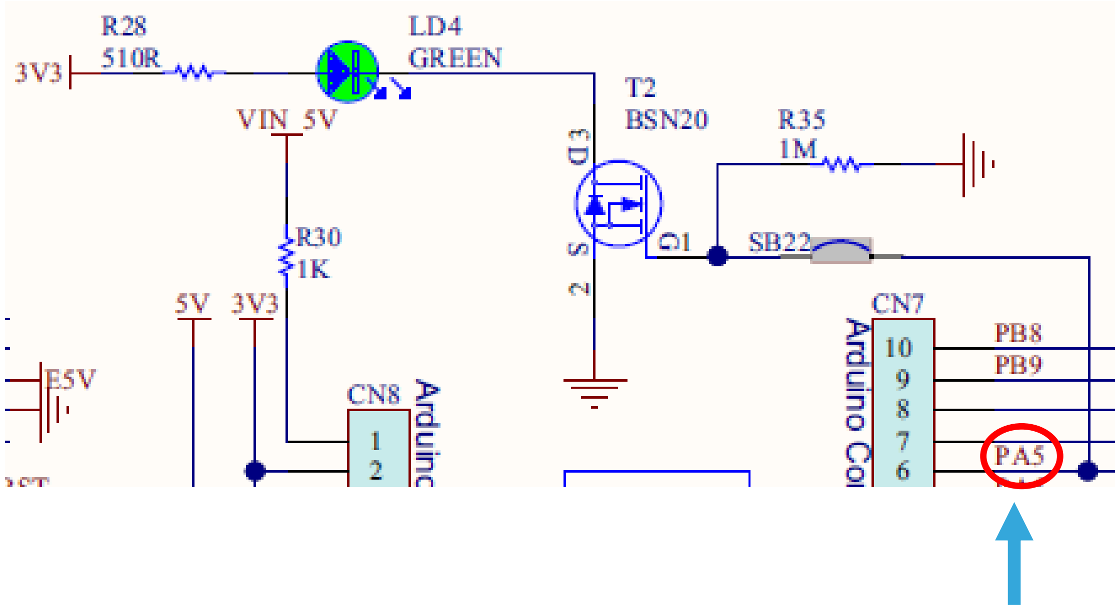


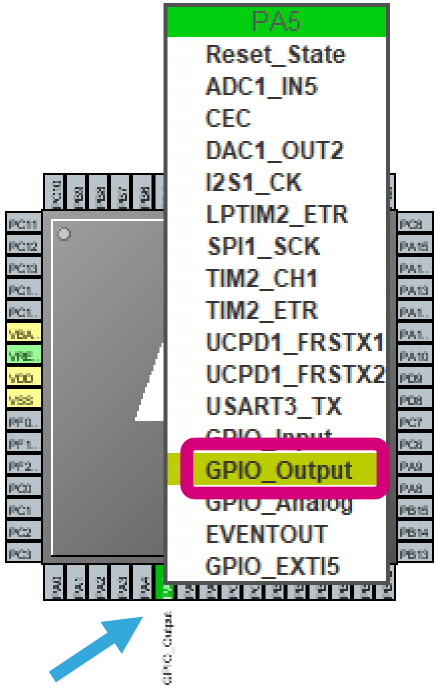


- Click **Access To MCU Selector** 1
- Type: “stm32g071rb” in the Part Number Search 2
- Select **STM32G071RBTx** 3
  - **LQFP64, 128kB Flash**
- Double Click

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
- Left-click **PA5** and set it to **GPIO_Output** mode



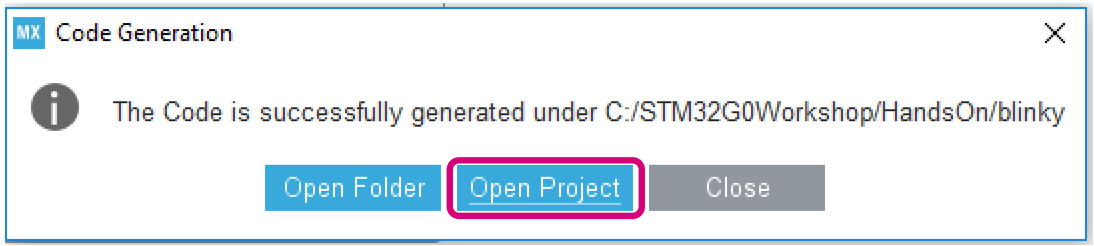


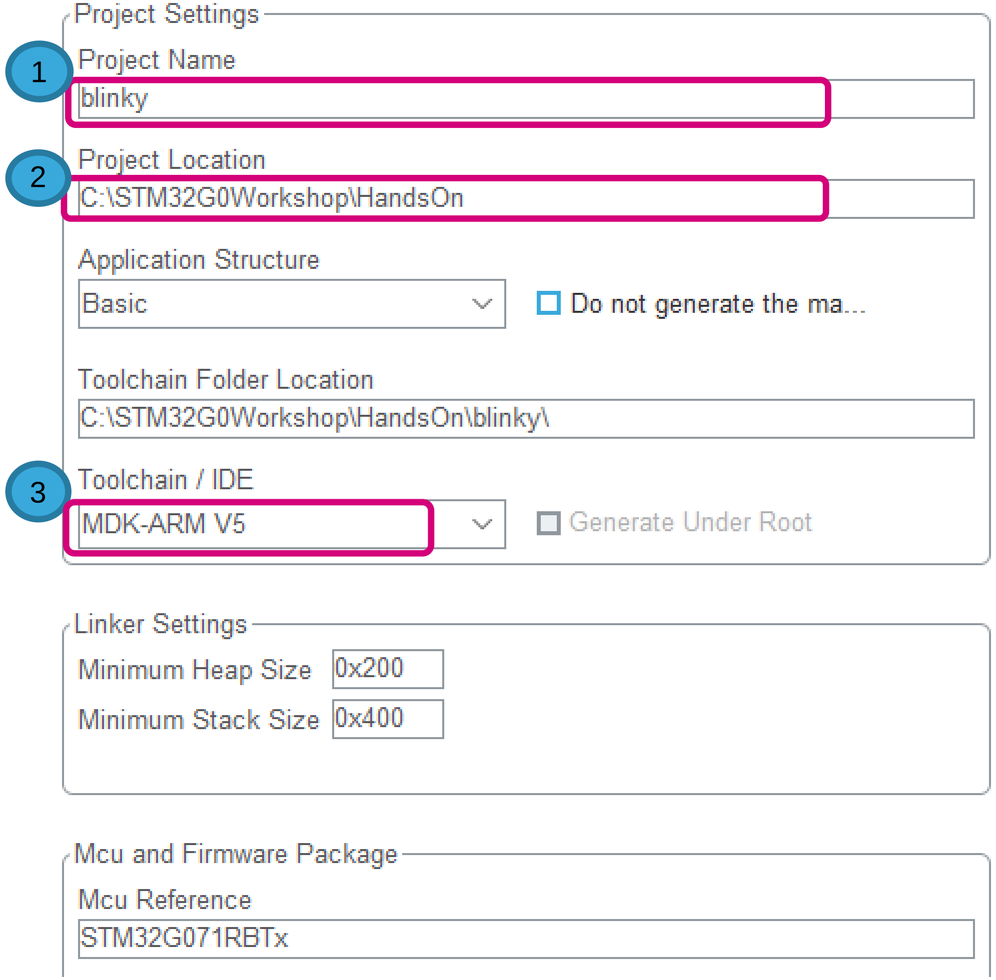
Step 3: Generate Source Code

- 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

Free MDK-ARM for ST

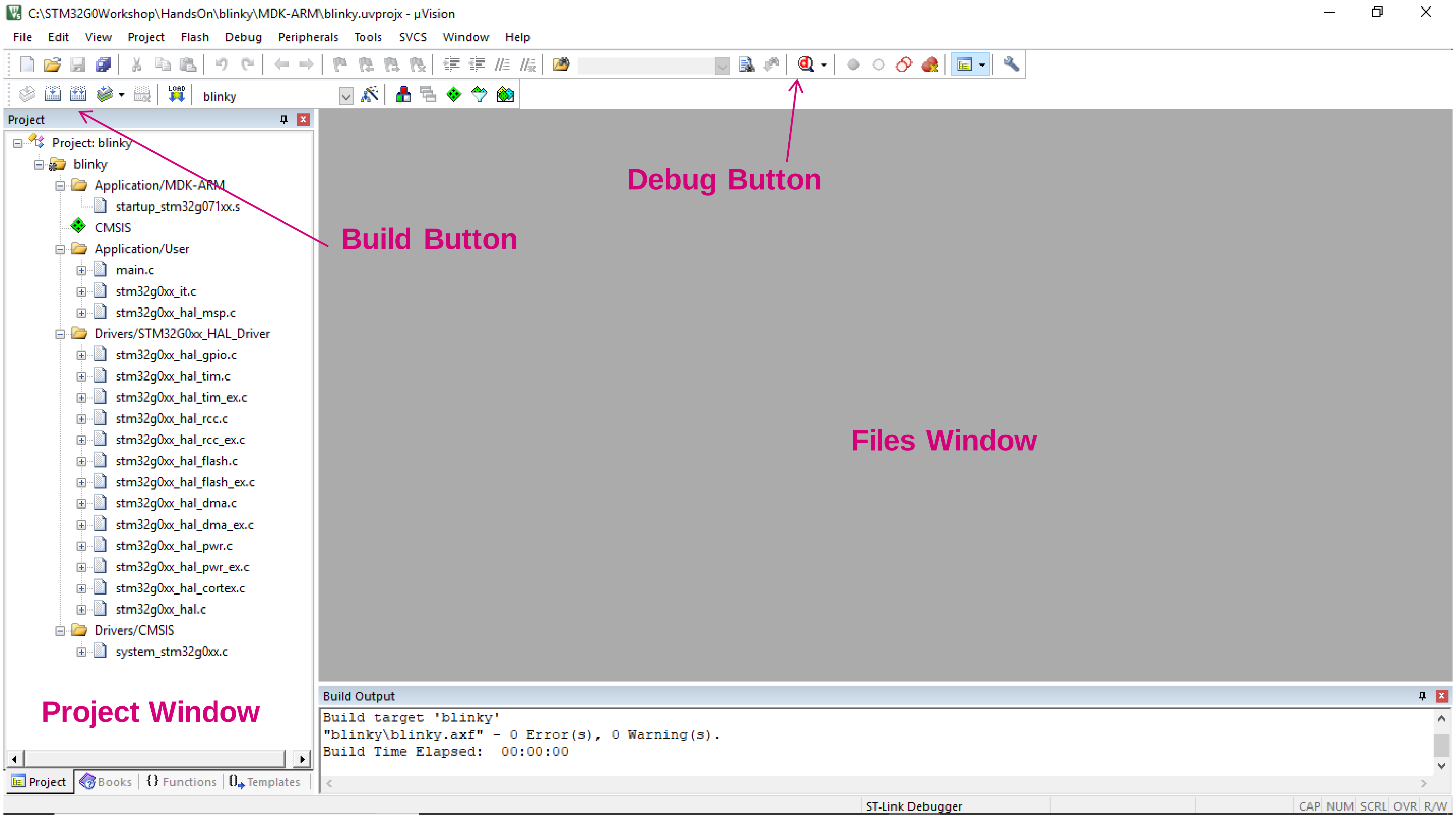
ARM KEIL Microcontroller Tools

- 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.

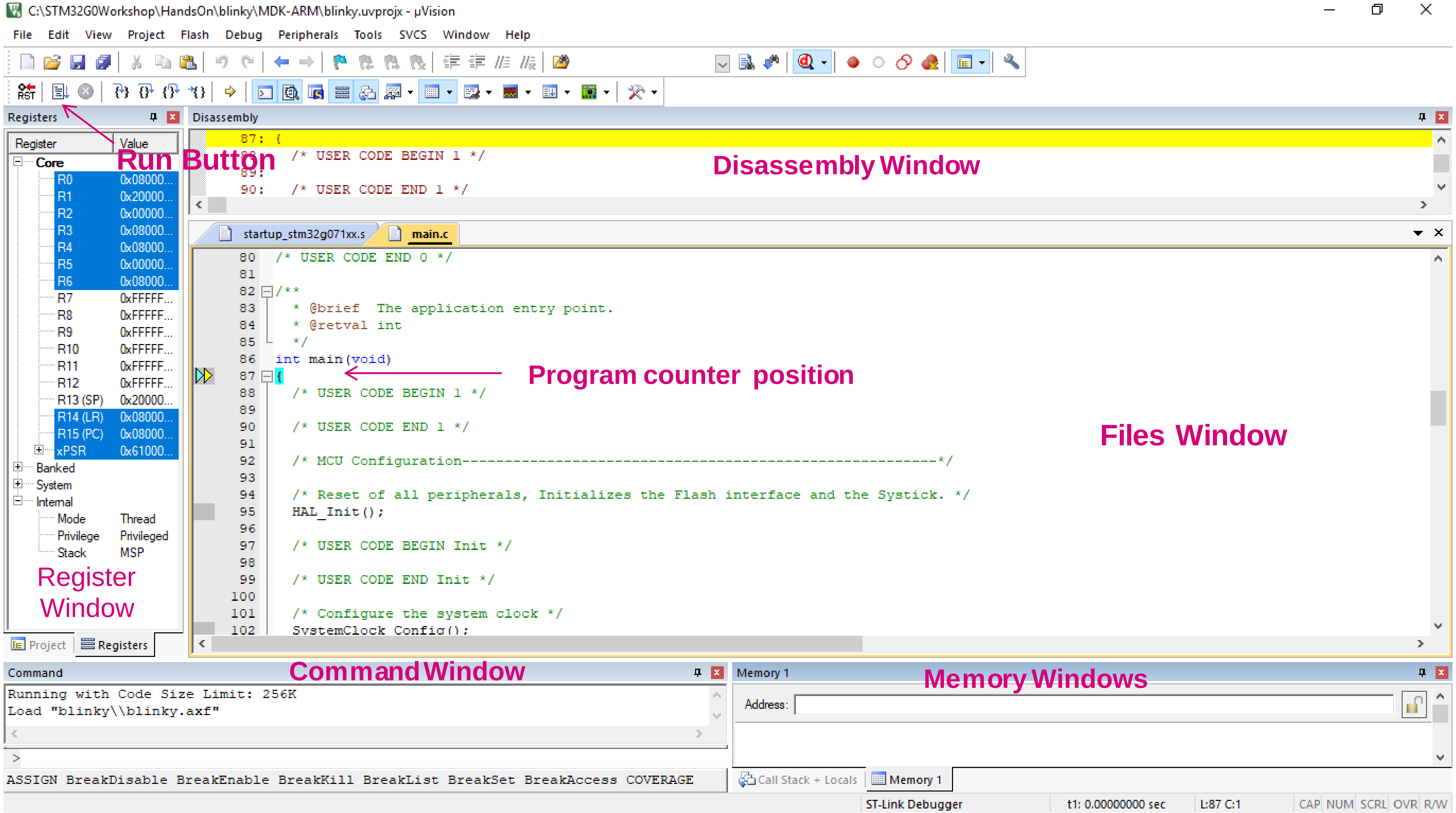
- To get a free MDK-ARM license for STM32F0,STM32L0 and **STM32G0**:
  - Go to Keil website at : www.keil.com/mdk-st
  - Download MDK-ARM tool chain.
  - Activate the free license using this Product Serial Number (PSN) :
    4PPFW-QBEHZ-M0D5M

life.augmented

Inside Keil uVision (ARM-MDK)



The MDK-ARM IDE Debugger



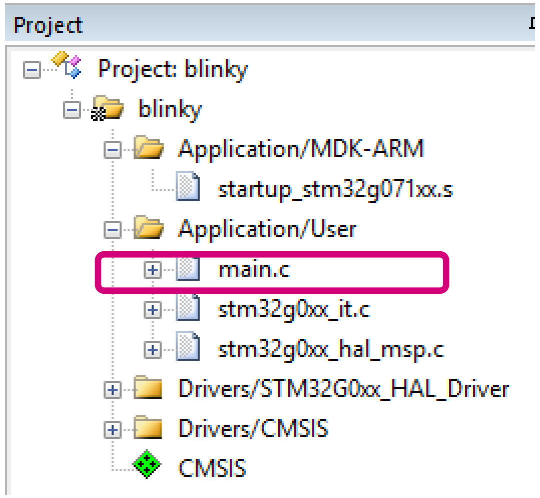
life.augmented

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 (~Line 118(*) in "main.c")

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




```
114   /* Infinite loop */
115   /* USER CODE BEGIN WHILE */
116   while (1)
117   {
118     HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
119     HAL_Delay(100);
120     /* USER CODE END WHILE */
```

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

TIP: The code to be added for the labs are located in a text called "code_to_add_vx.x.txt"

(*) Line numbers throughout the presentation are given for STM32CubeMX v5.0.1 and STM32CubeG0 Library v1.0.0 and may vary with other versions.

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)

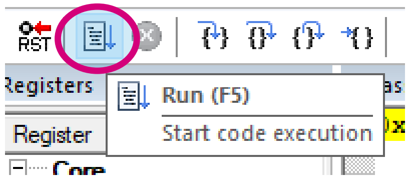
life.augmented

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)!

life.augmented

Lab: PWM (Pulse Width Modulation) Timer

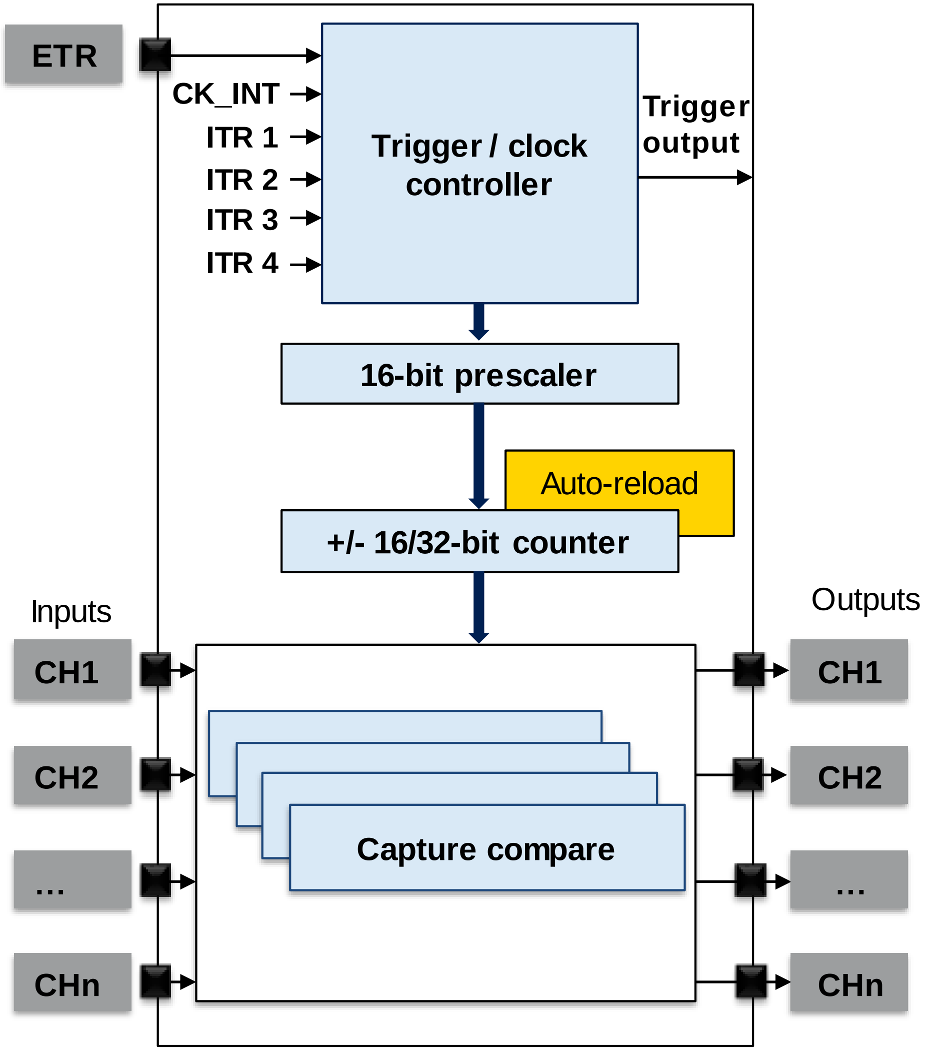
Lab: PWM Timer

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

STM32G0 timer instance features

<table>
<tr><th colspan="2" rowspan="2">Feature</th><th rowspan="2">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><th colspan="2">(General-Purpose)</th><th colspan="2">(Basic)</th><th colspan="4">(General-Purpose)</th></tr>
<tr><td colspan="2">Clock source</td><td>CK_INT
External input pin
External trigger input ETR</td><td colspan="2">CK_INT
External input pin
External trigger input ETR
Internal trigger inputs</td><td colspan="2">CK_INT</td><td>CK_INT</td><td>CK_INT
External input pin
Internal trigger inputs</td><td colspan="2">CK_INT
External input pin</td></tr>
<tr><td colspan="2">Resolution</td><td>16-bit</td><td>32-bit</td><td>16-bit</td><td colspan="2">16-bit</td><td>16-bit</td><td colspan="3">16-bit</td></tr>
<tr><td colspan="2">Prescaler</td><td colspan="9">16-bit</td></tr>
<tr><td colspan="2">Counter direction</td><td>Up, Down, Up&Down</td><td colspan="2">Up, Down, Up&Down</td><td colspan="2">Up</td><td>Up</td><td colspan="3">Up</td></tr>
<tr><td colspan="2">Repetition counter</td><td>✓</td><td colspan="2">-</td><td colspan="2">-</td><td>-</td><td colspan="3">✓</td></tr>
<tr><td rowspan="2">Synchronization</td><td>Master</td><td>✓</td><td colspan="2">✓</td><td colspan="2">✓</td><td>-</td><td colspan="3">✓</td></tr>
<tr><td>Slave</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>-</td><td>✓</td><td colspan="2">-</td></tr>
<tr><td colspan="2">Number of channels</td><td>6:
➢ CH1/CH1N
➢ CH2/CH2N
➢ CH3/CH3N
➢ CH4
➢ CH5 and CH6 output only, not available externally</td><td colspan="2">4:
➢ CH1
➢ CH2
➢ CH3
➢ CH4</td><td colspan="2">0</td><td>1:
➢ CH1</td><td>2:
➢ CH1/CH1N
➢ CH2</td><td colspan="2">1:
➢ CH1/CH1N</td></tr>
<tr><td colspan="2">Trigger input</td><td>✓</td><td colspan="2">✓</td><td colspan="2"></td><td colspan="4"></td></tr>
</table>

life.augmented

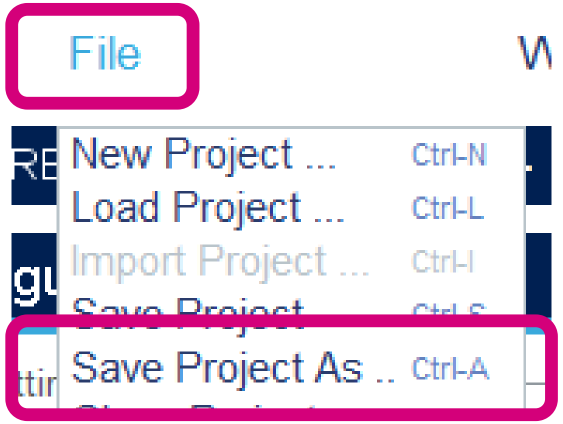
STM32G0 timer instance features

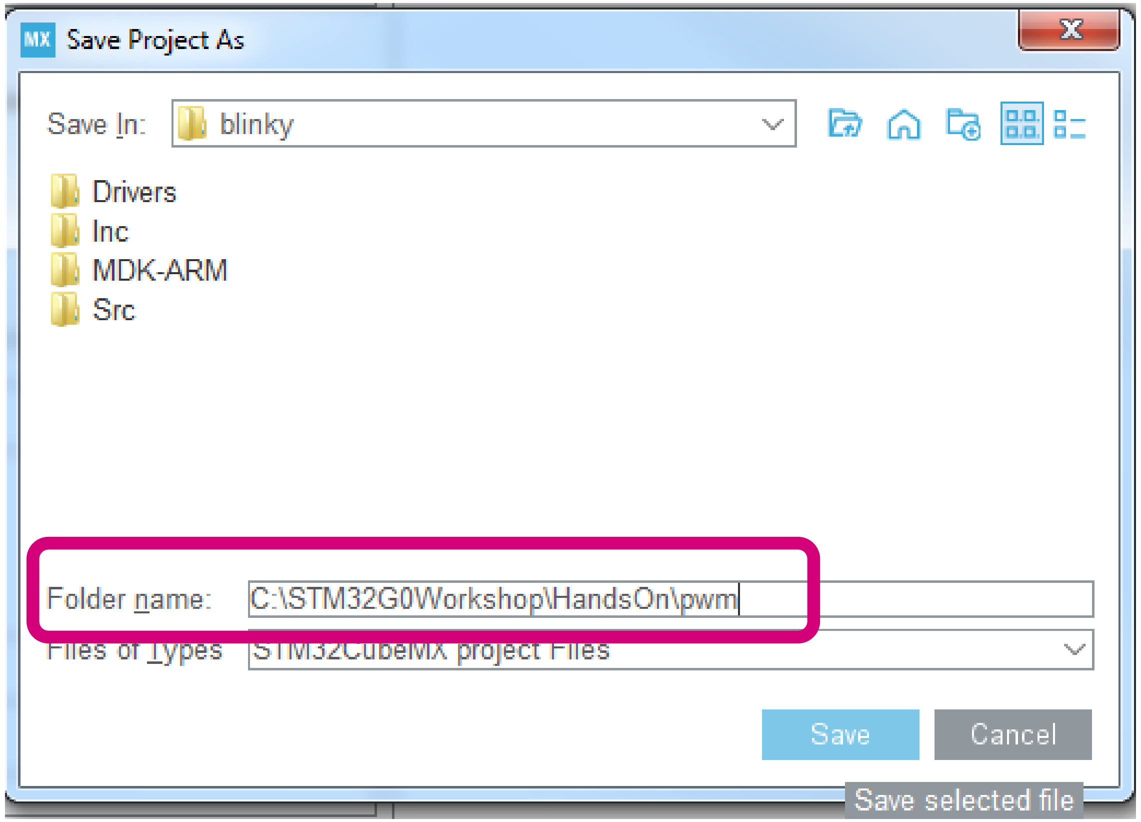
<table>
<tr><th>Feature</th><th>TIM1 (Advanced Control)</th><th>TIM2</th><th>TIM3</th><th>TIM6</th><th>TIM7</th><th>TIM14</th><th>TIM15</th><th>TIM16</th><th>TIM17</th></tr>
<tr><td>Input capture mode</td><td>✓</td><td colspan="2">✓</td><td colspan="2">-</td><td>✓</td><td colspan="3">✓</td></tr>
<tr><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>

life.augmented

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)

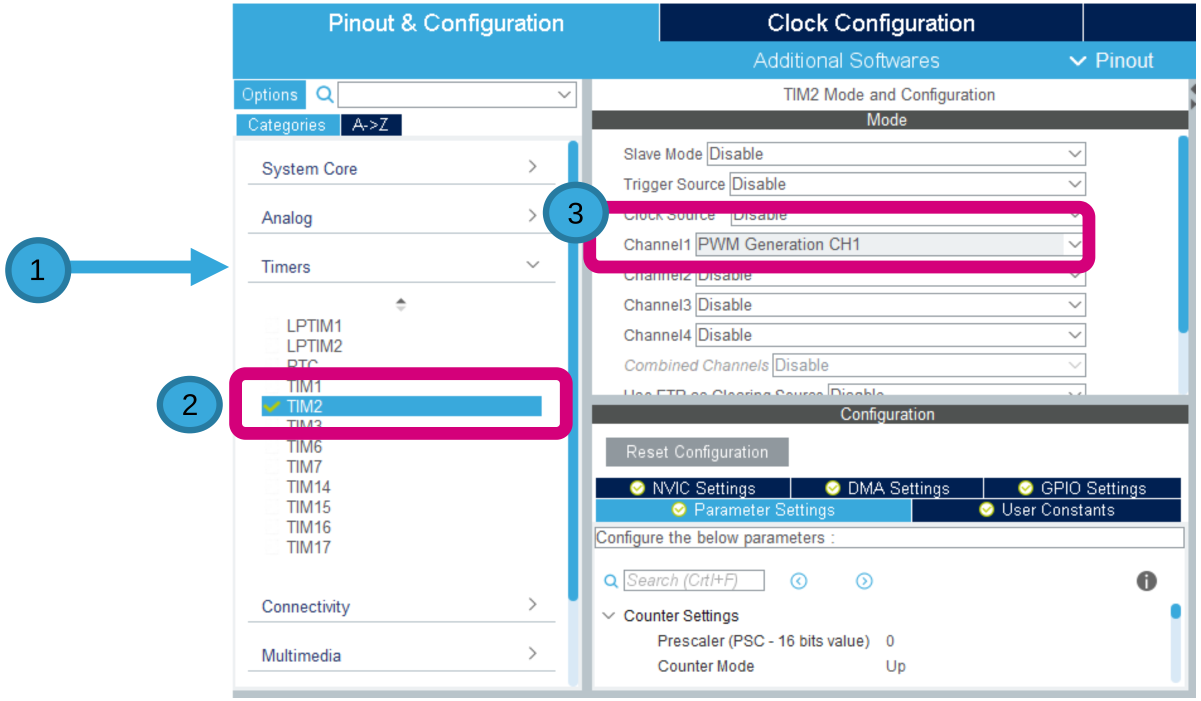


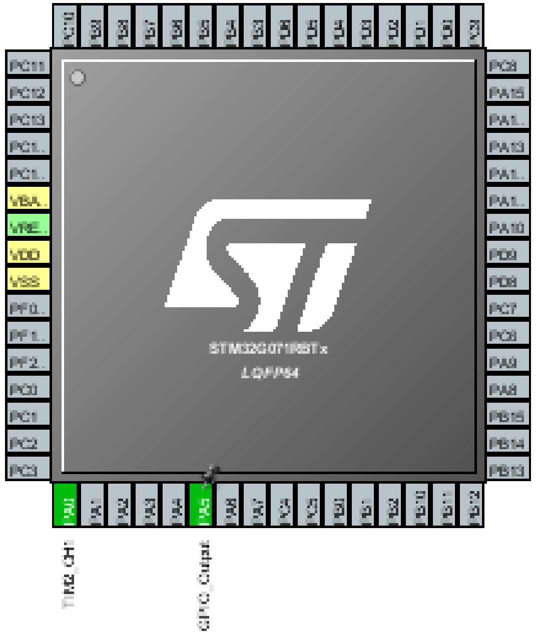


life.augmented

Lab: Timer 2 CH1 Configuration

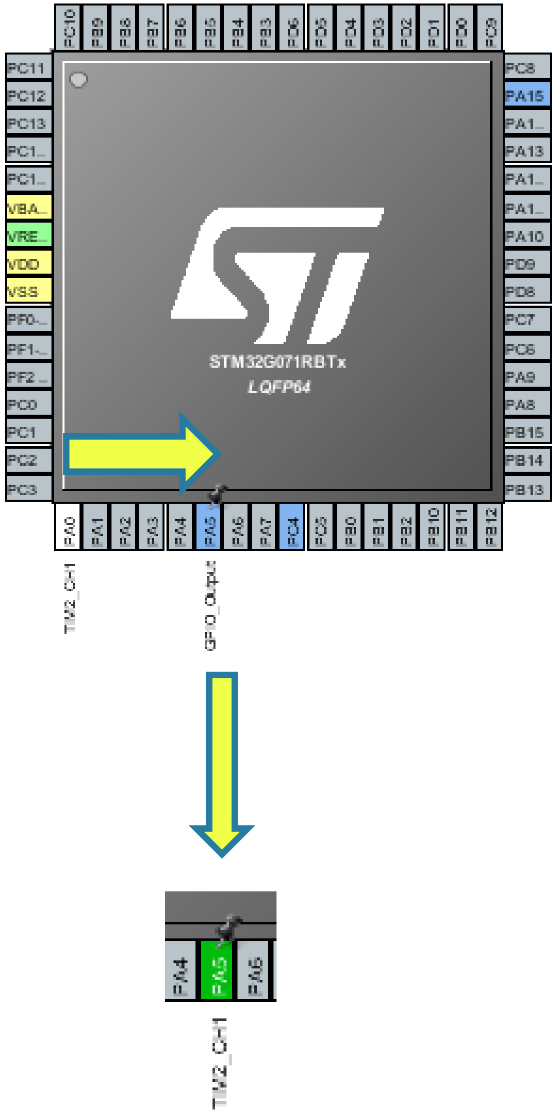
- In this STM32CubeMX project we are going to add Timer 2 Channel 1 to blink LD4 (PA5) on the Nucleo board.
- In the **Pinout & Configuration** tab, Expand **Timers** Categories, then click on **TIM2** peripheral and set Channel1 to “**PWM Generation CH1**”.





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

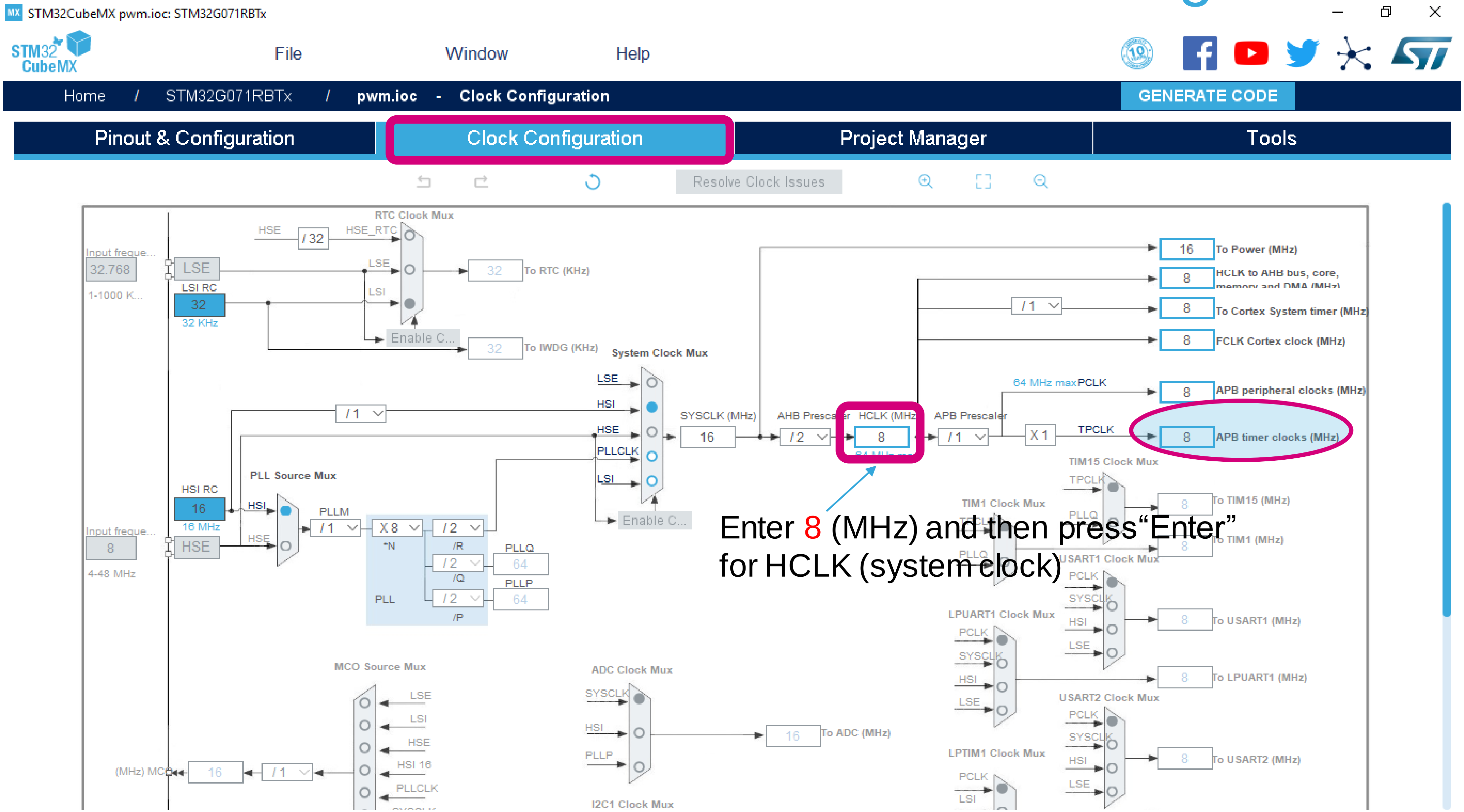


life.augmented

Timer Parameters Calculation

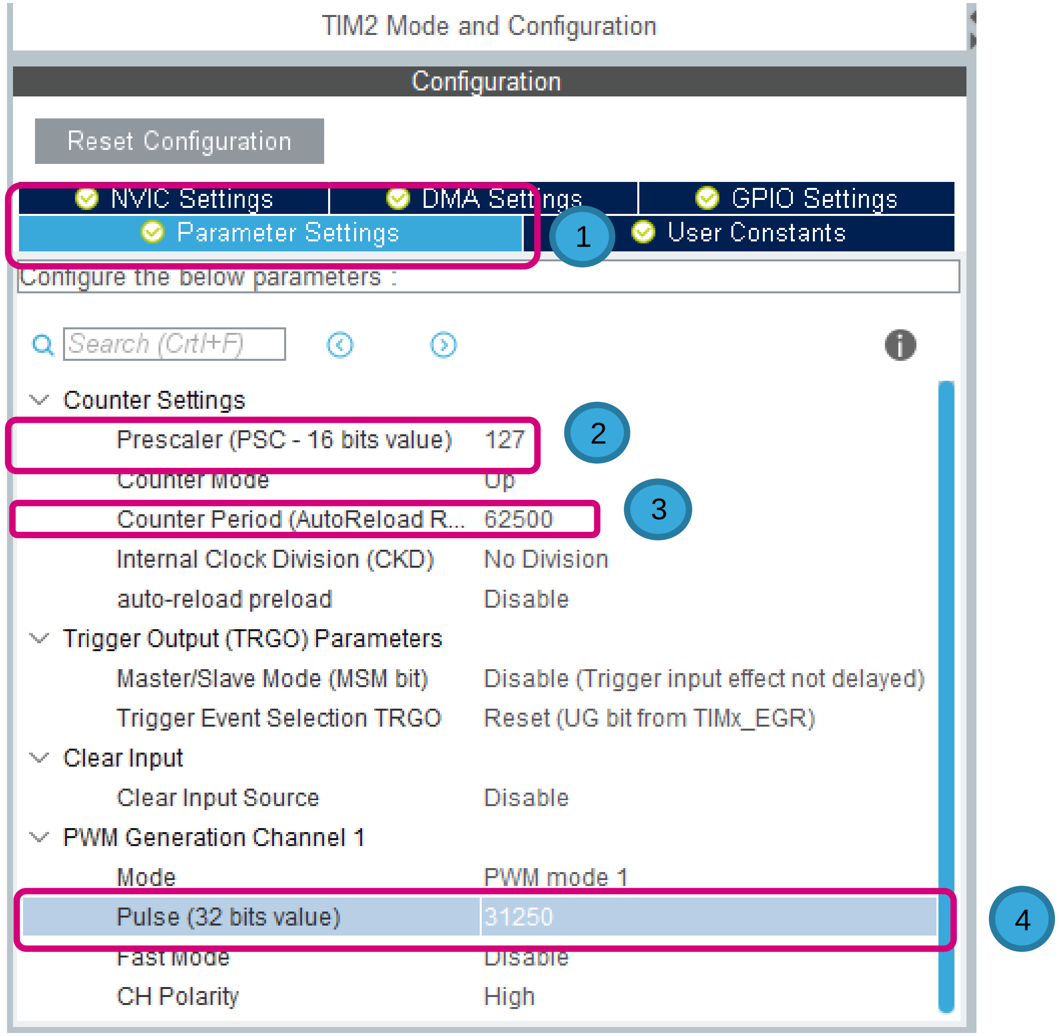
- We can calculate Timer parameters for 1 Hz period @ 50% duty cycle
- If we choose Timer input clock to be 8 MHz
- Let us say the prescaler is div-by-128; 8MHz/128 = 62500 Hz
  - So, prescaler register should be 128-1 = 127 (actual divide is PSC+1)
  - If we set Period = 62500; Pulse = 31250, we get 1 Hz Period @ 50% duty cycle

Clock Tree Configuration



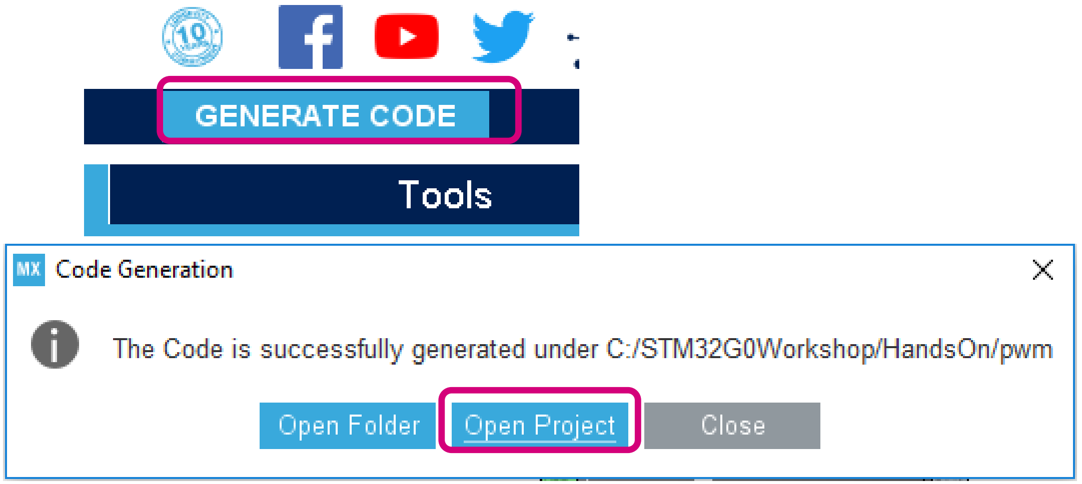
TIM2 Configuration – 4 steps

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



Generate Source Code

- **Generate Code**

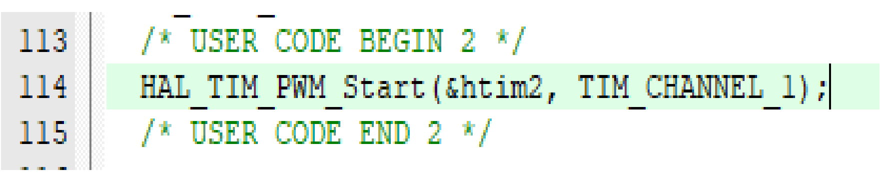


- Click **Open Project**

- 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 (~Line 114)

```
HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
```

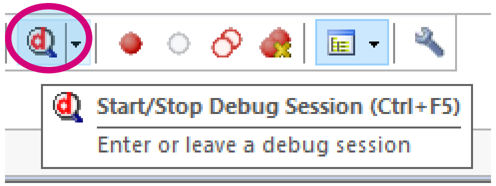


life.augmented

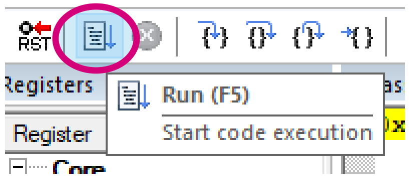
Build the Project

- 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

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.

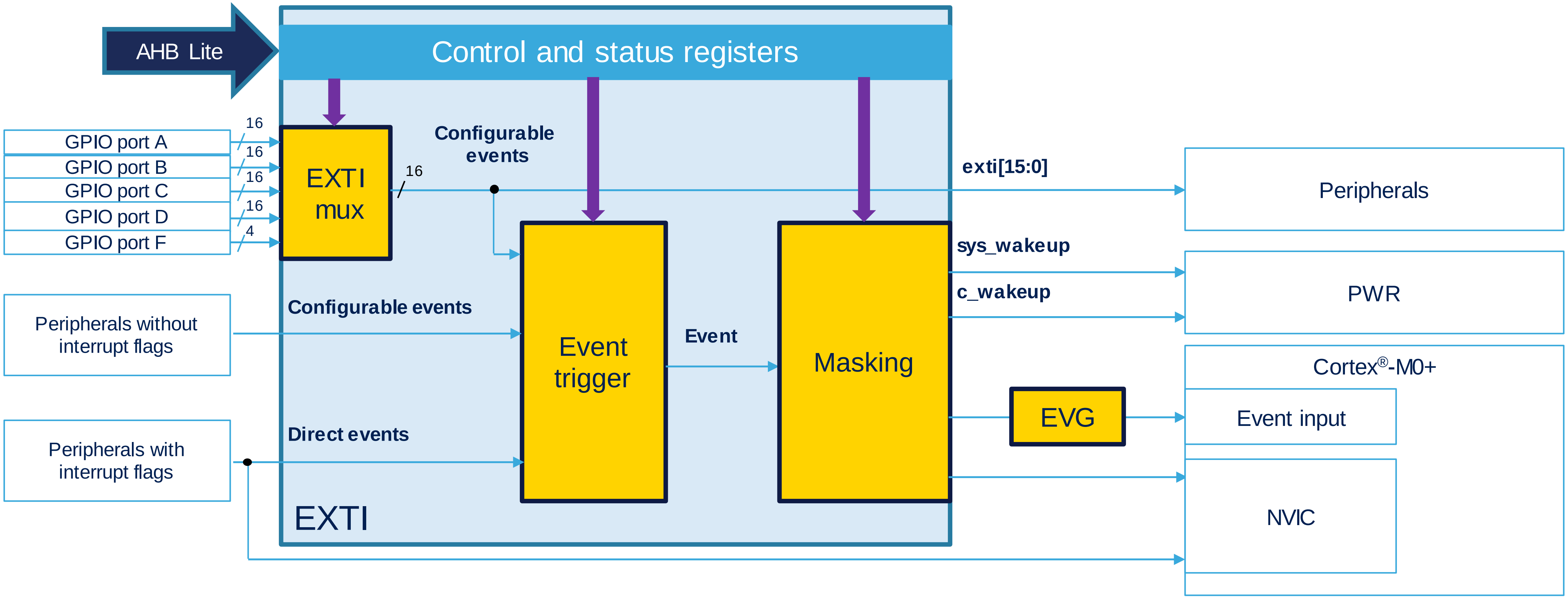
EXTI - Key features

- 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

NVIC - Overview

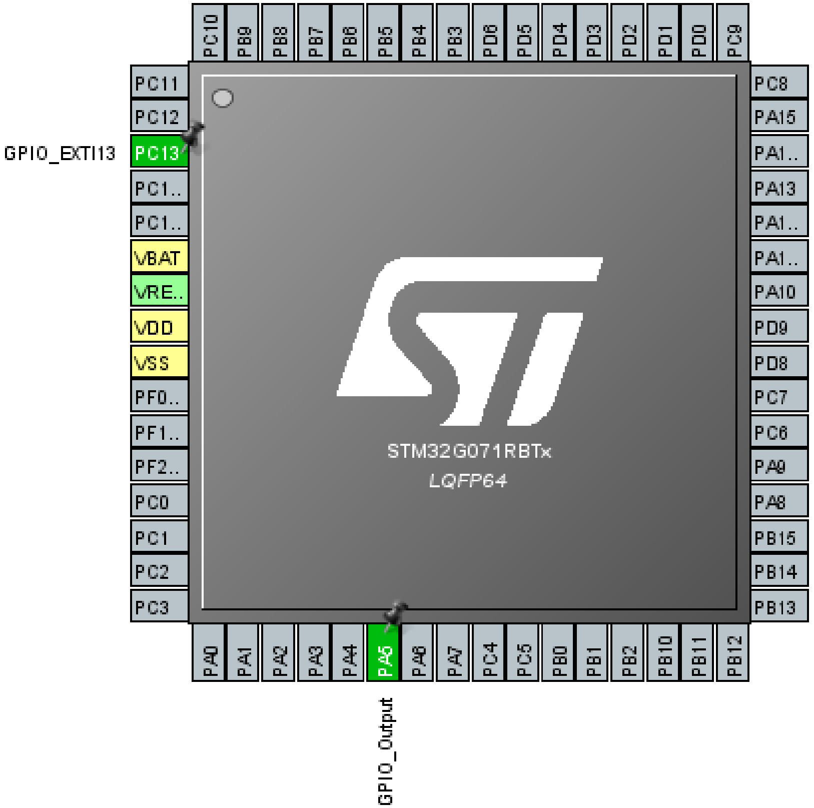
- The NVIC (Nested vector Interrupt Controller) is integrated in the Cortex®-M0+ CPU:
  - 32 maskable interrupt channels
  - 4 programmable priority levels
  - Low-latency exception and interrupt handling
  - Power management control

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.



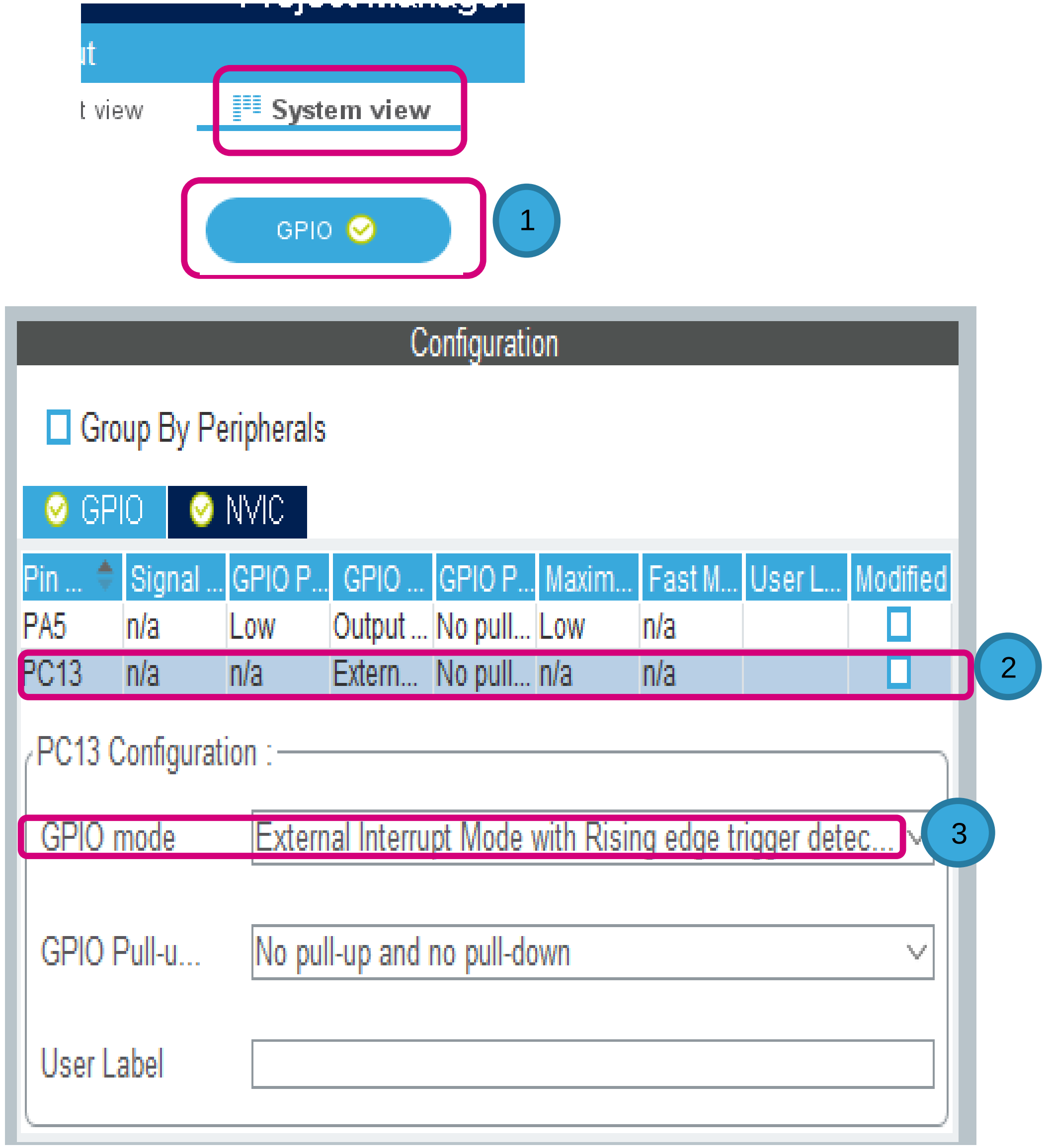
life.augmented

GPIO Configuration

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

- Click on Pin Name **PC13** 2

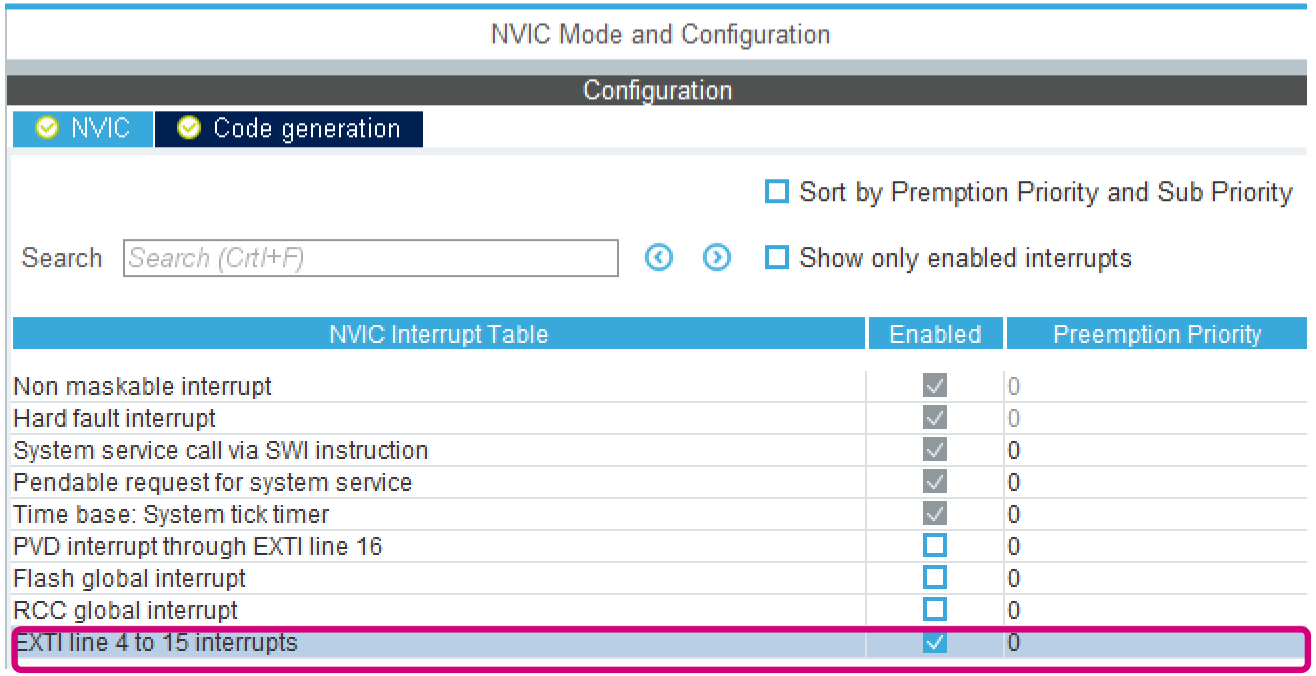
- Make sure GPIO mode is “External **Interrupt Mode** with Rising edge trigger detection” 3



NVIC Configuration

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

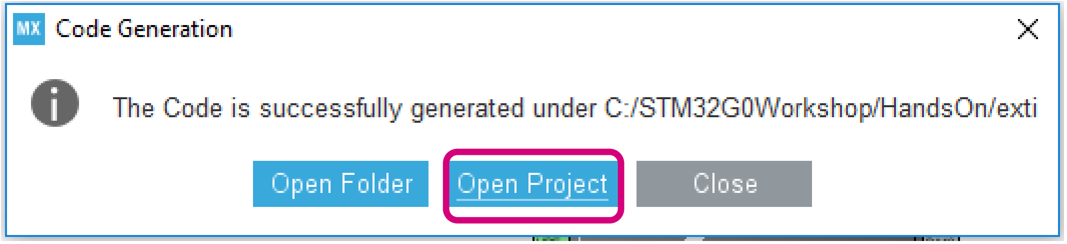
- Enable “**EXTI line 4 to 15 interrupts**” (by checking the box) 1



Generate Source Code

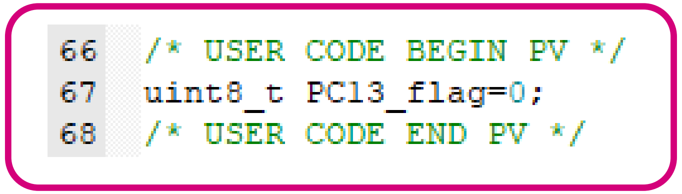
- **Generate Code**

- Click **Open Project**



- Open **main.c**, add the following code:
  - within "USER CODE BEGIN PV" / "USER CODE END PV" section (~Line 67)

```
uint8_t PC13_flag = 0;
```



Add EXTI Rising Edge Callback Function

- Also in **main.c** add the following code,
  - within “USER CODE BEGIN 4” / “USER CODE END 4” section (~line 198)

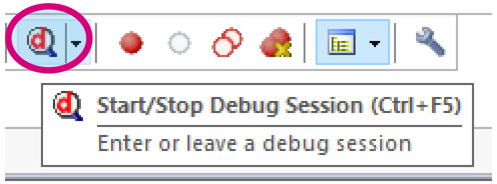
```
void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
{
  PC13_flag++;
  if ( ( PC13_flag & 0x01 ) == 0x01 )
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
  }
  else
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  }
}
```

```
197 /* USER CODE BEGIN 4 */
198 void HAL_GPIO_EXTI_Rising_Callback(uint16_t GPIO_Pin)
199 {
200   PC13_flag++;
201   if ((PC13_flag & 0x01) == 0x01)
202   {
203     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
204   }
205   else
206   {
207     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
208   }
209 }
210 /* USER CODE END 4 */
```

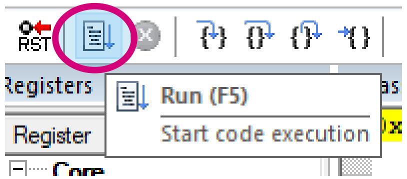
Build the Project

- 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 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

I/Os kept, and configurable

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

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

Available clocks

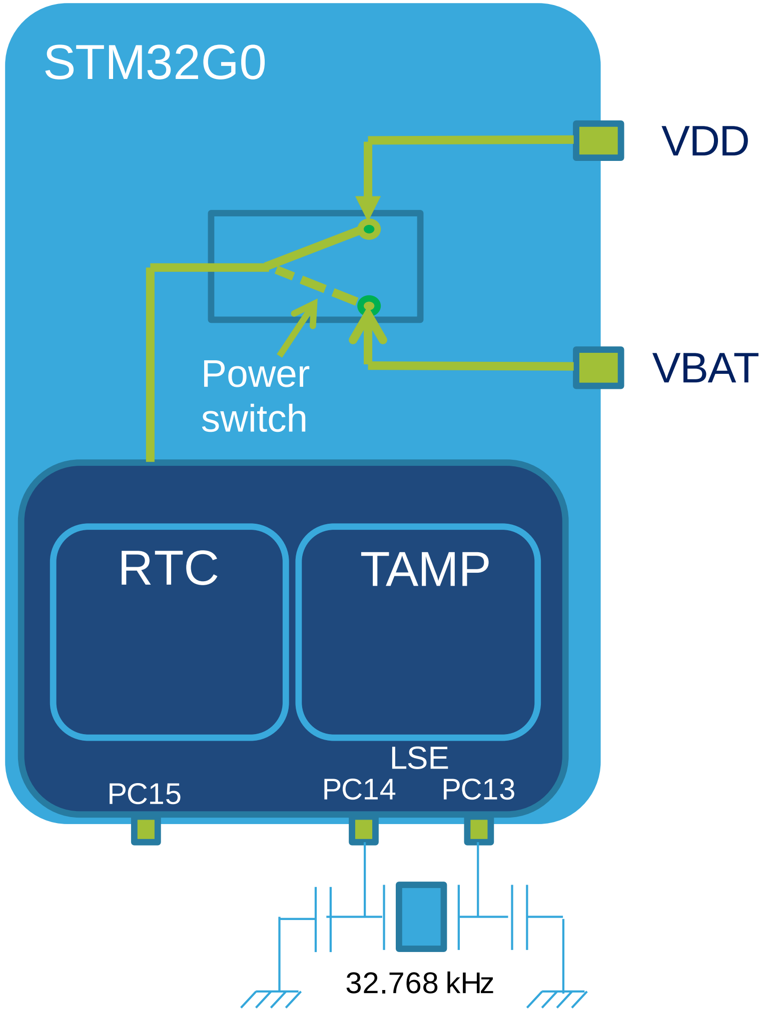
HSI16
HSE
LSI
LSE

Wake-up event

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

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

RTC - Overview

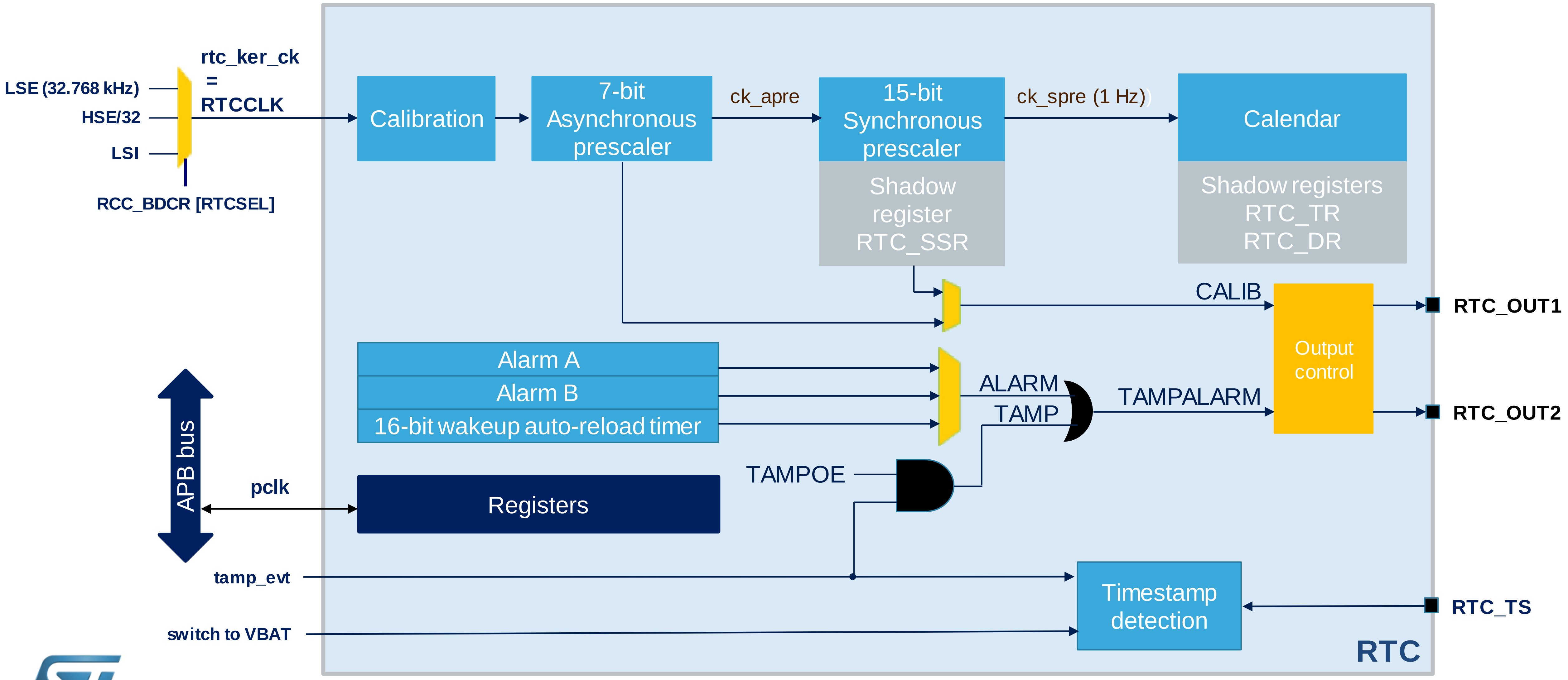


- The RTC provides an ultra-low-power hardware calendar with alarms, in all low-power modes
- It belongs to the Battery Backup Domain, so it is kept functional when the main supply is off and VBAT is present
- The TAMP peripheral features the backup registers and tamper detection

Application benefits

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

RTC - Block diagram



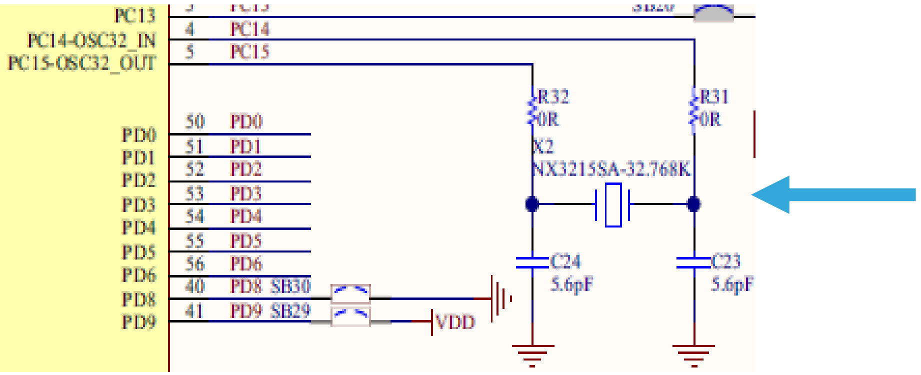
- RTC not affected by system reset when clocked by LSE

Lab: Low Power

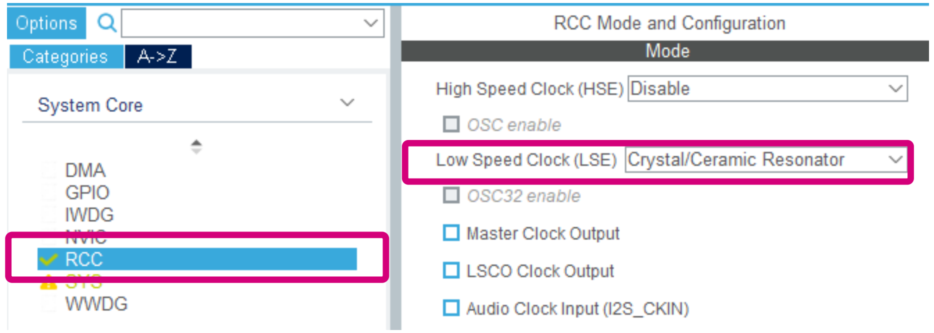
- Close Keil uVision5 IDE if it is open; In STM32CubeMX open the “**exti**” STM32CubeMX project save it as a new project like “**lowpower**”.

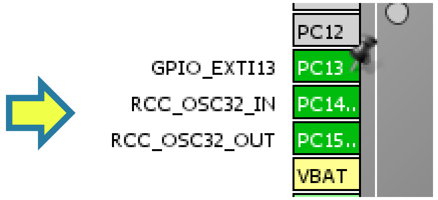
Enable LSE (Low Speed External) Clock

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



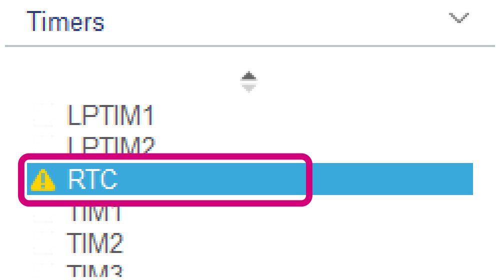
- In the pinout tab, expand RCC and choose Crystal/Ceramic Resonator for Low Speed Clock (LSE) clock:



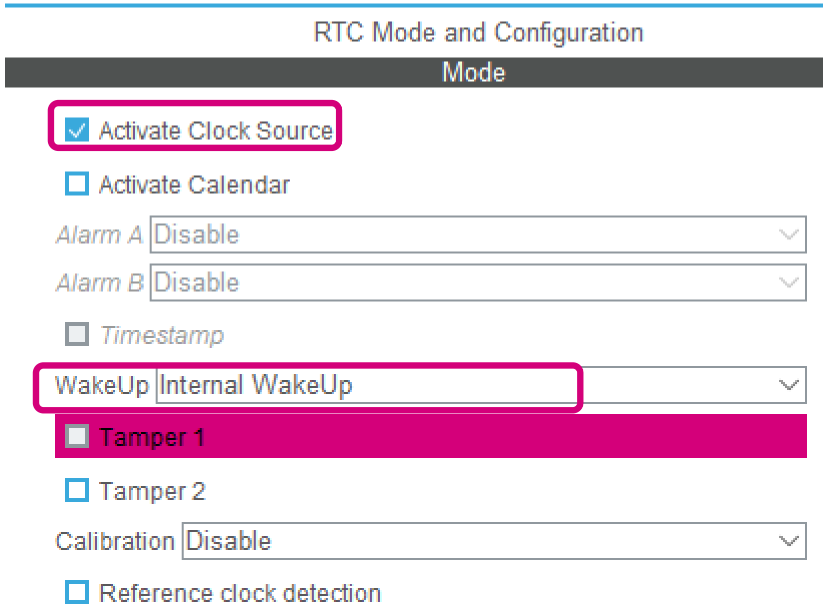


Enable and Configure the RTC

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



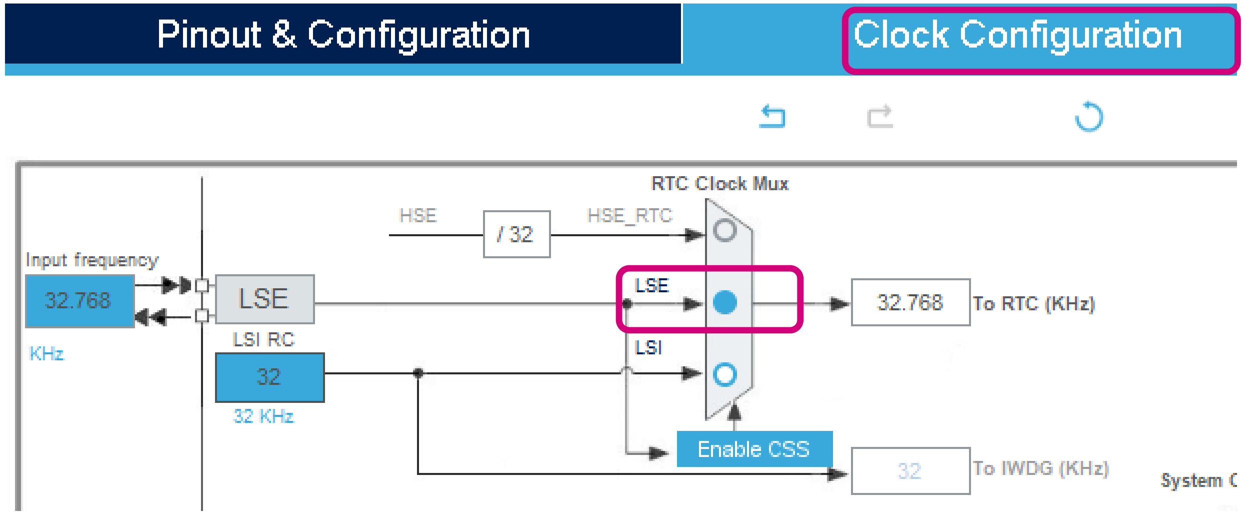
- Check the **Activate Clock Source**
- Select **Internal Wakeup** for the **Wakeup** mode



life.augmented

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

RTC configuration

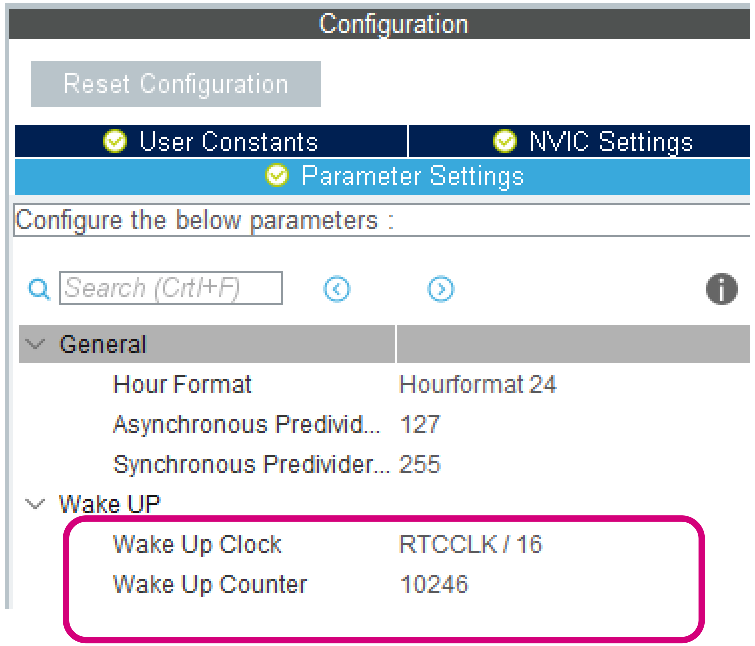
Wakeup Counter Calculation:

- 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

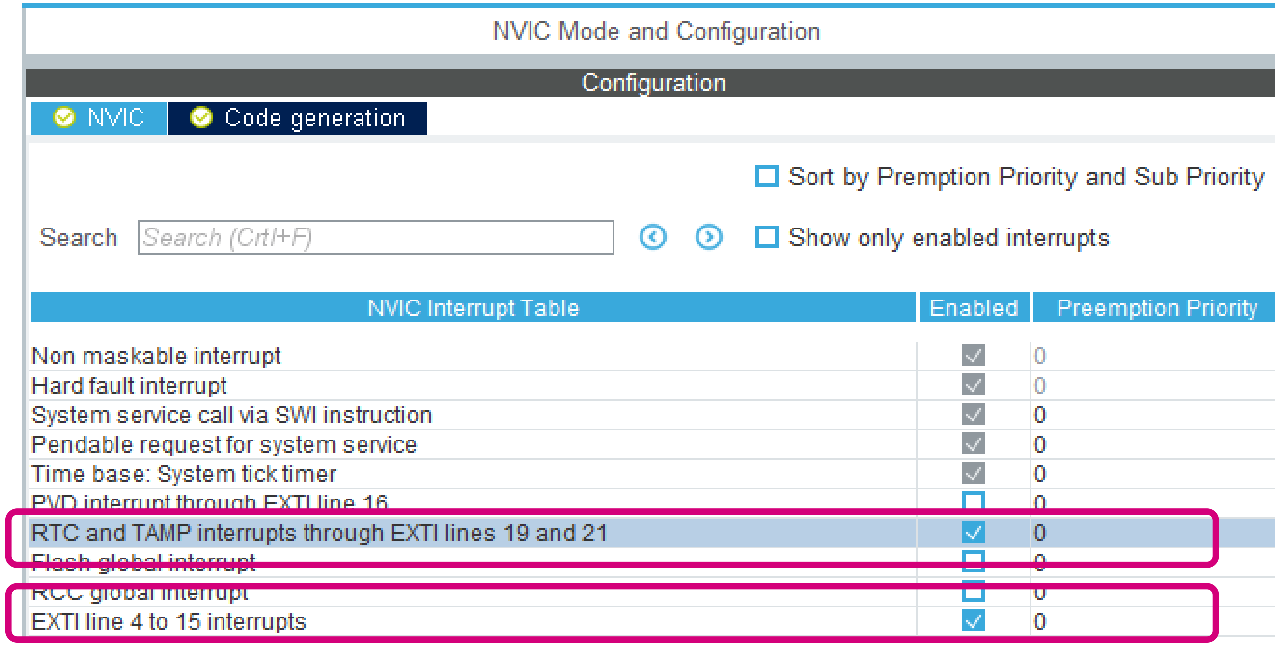
- Based on previous calculation we will configure the RTC
- In the **Pinout & Configuration** tab, click on **RTC**
- Enter the following configuration:



Enable Interrupts

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

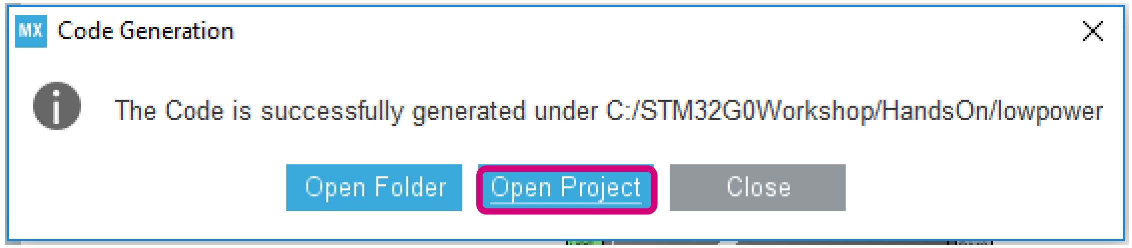
- In the “System View” in the NVIC, check that both RTC and EXTI[4…15] are enabled, if not re-enable them both:



life.augmented

- **Generate Code**

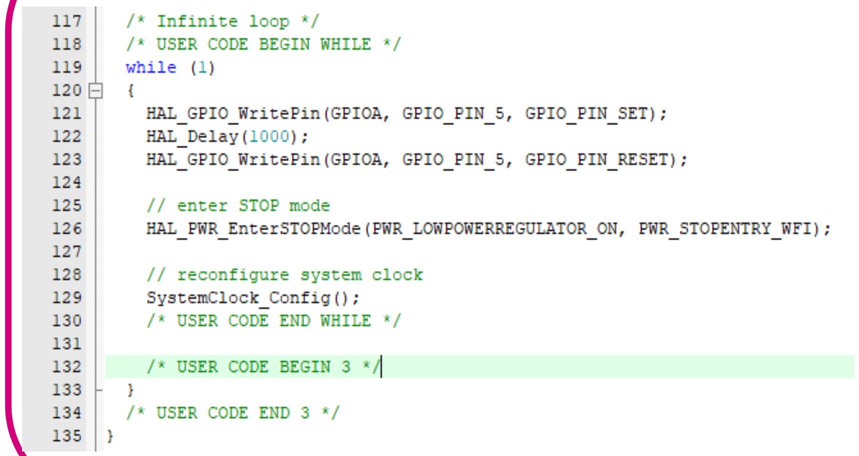
- Click **Open Project**



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 (~ line 121):

```
/* 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 */
```

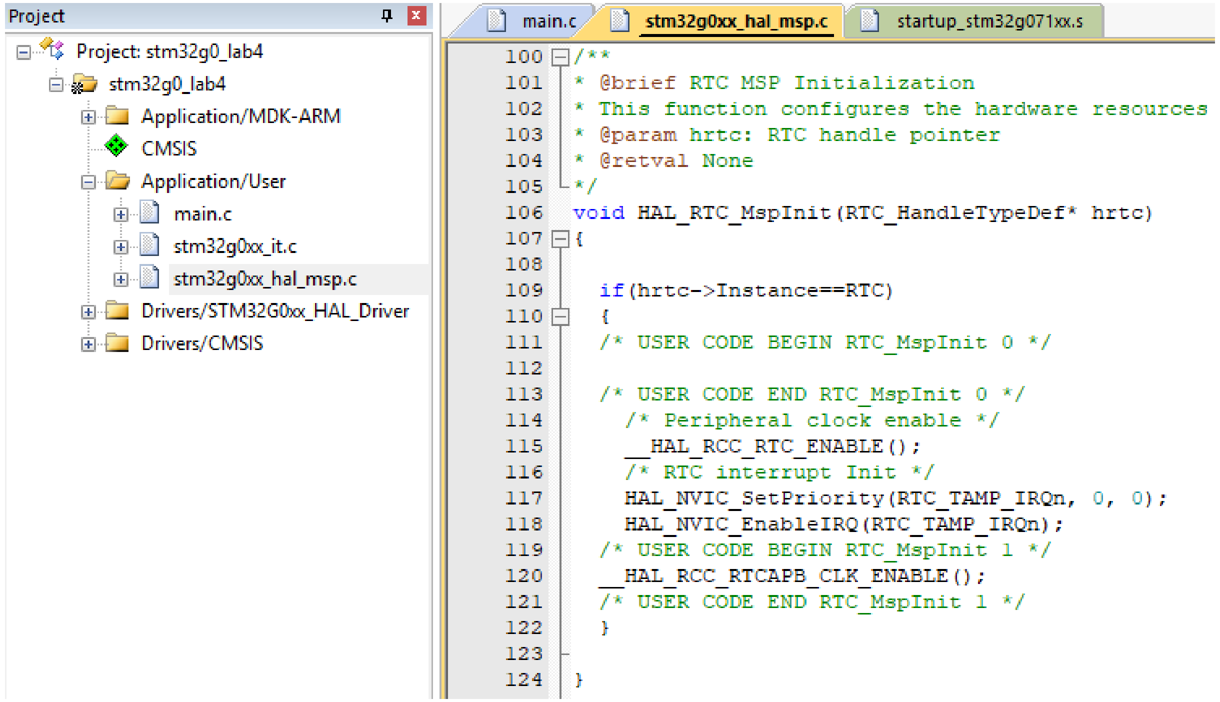


life.augmented

Add code – to init function

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

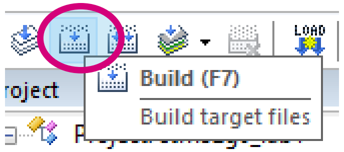
```
void HAL_RTC_MspInit(RTC_HandleTypeDef* hrtc)
{

  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 */
  }

}
```



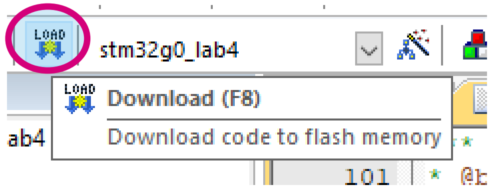
life.augmented

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

Optional Lab: Estimation of power consumption

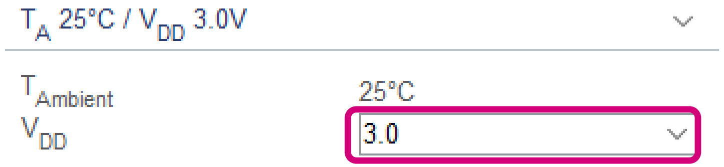
<u>Objective:</u>

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

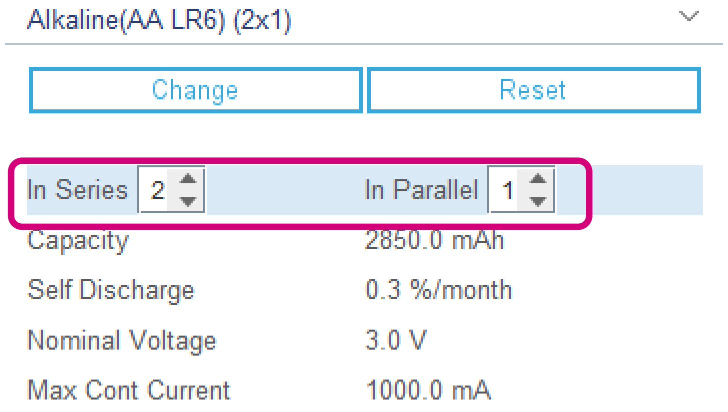
Power Supply and Power Source Selection

- Using the “**lowpower**” project in STM32CubeMX
- Click on the **Tools** tab in STM32CubeMX

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

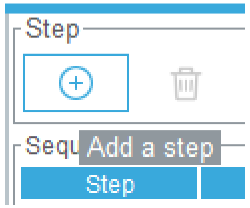


- In the **Battery Selection** section, select **AA** Alkaline batteries (2 in series, 1 in parallel) as the power source for the application



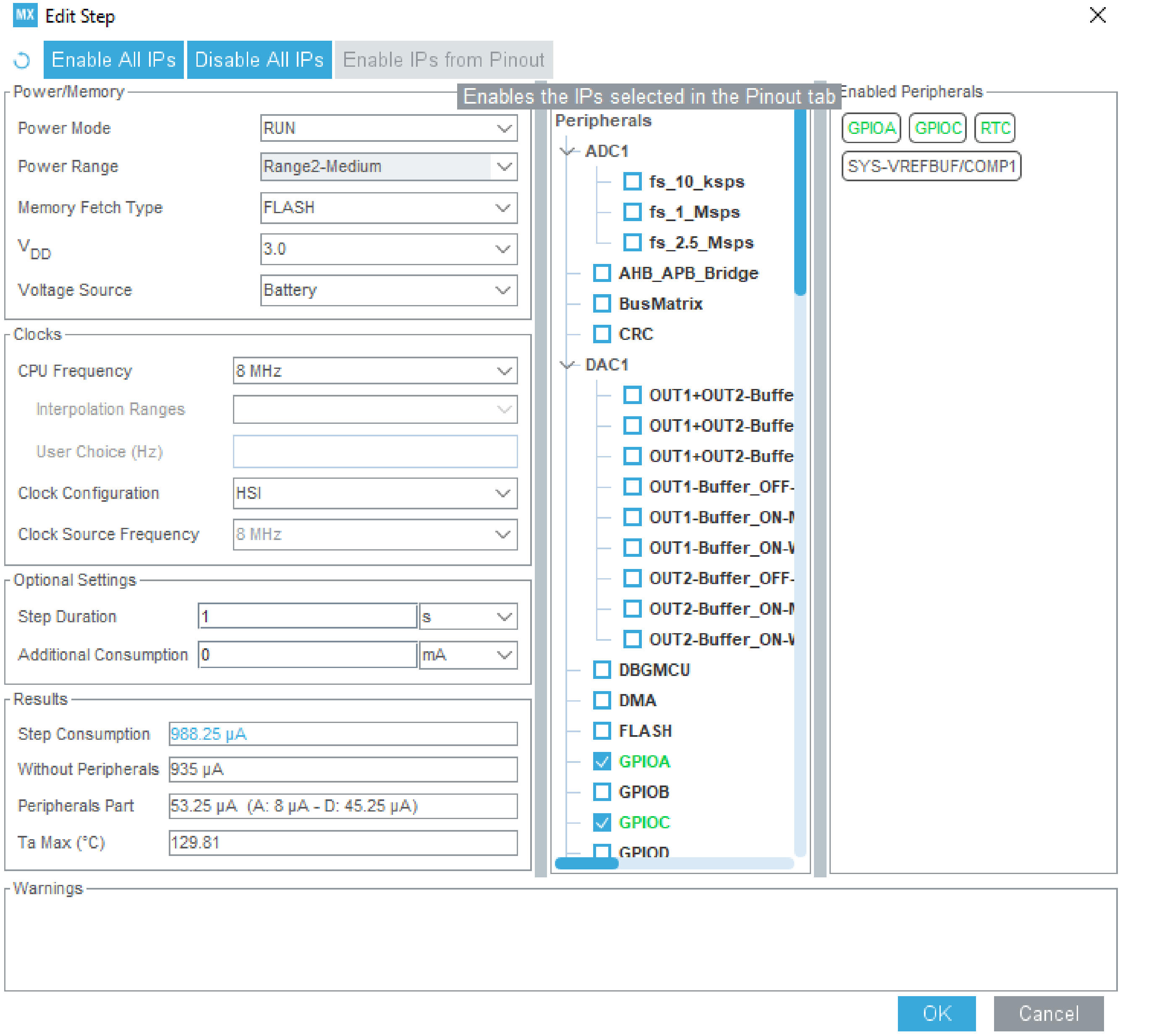
Adding a RUN mode step

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



- **Configure a first step: RUN mode**
  - 8.0 MHz Range2 RUN mode
  - from Flash at 3V from battery
  - HSI clock
  - Enable IPs from Pinout function
  - Duration: 1 second
  - Click "Add"

  - Resulting step consumption should be 988.25uA

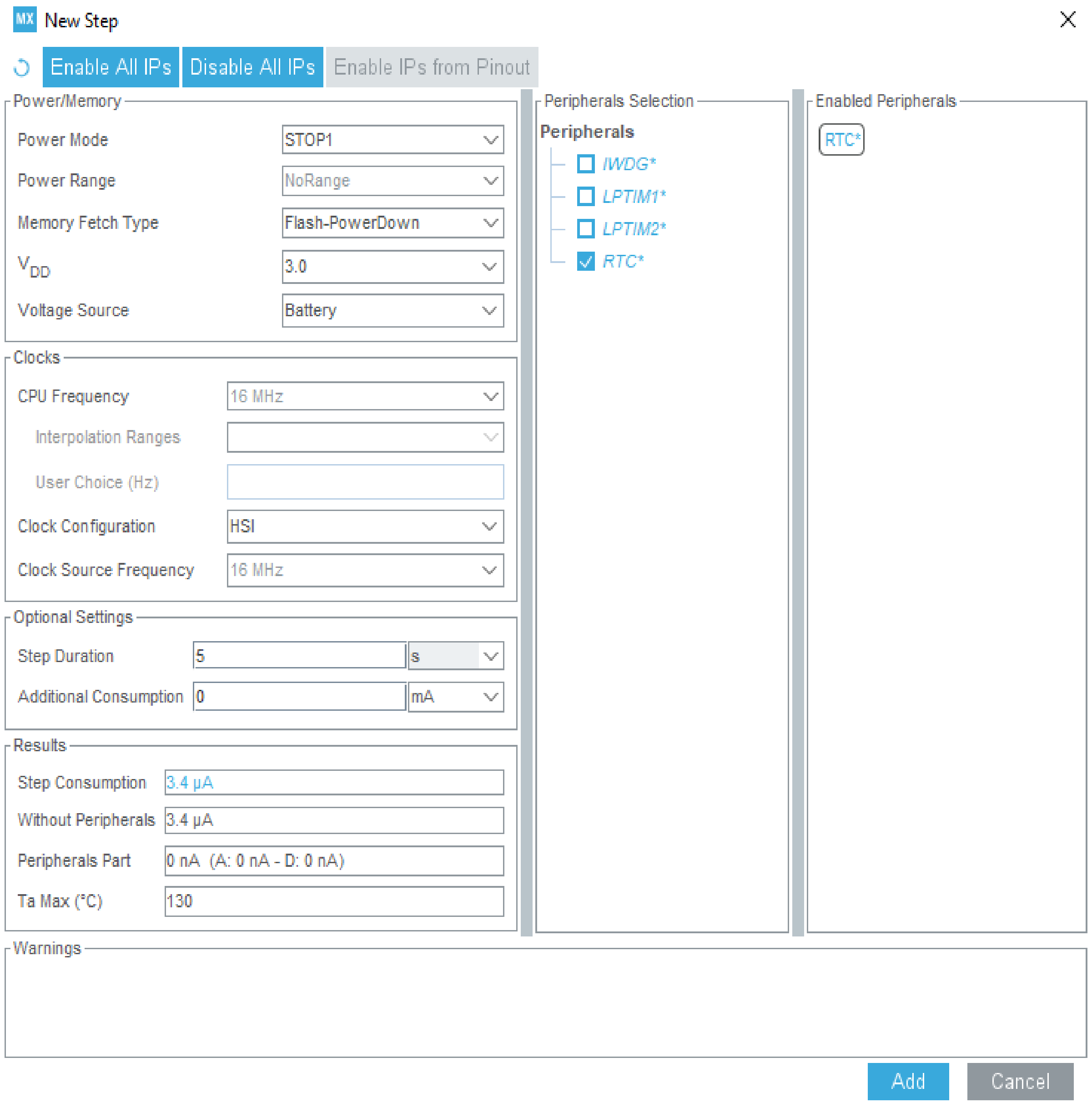


life.augmented

Add a STOP1 mode step

- **Add a second step: STOP1 mode**
  - STOP1 mode
  - Flash is Power Down mode at 3V
  - Clocks HSI 16 MHz
  - RTC enabled (to wakeup the system)
  - Duration: 5 seconds
  - Click "Add"

  - Resulting step consumption should be 3.4 uA

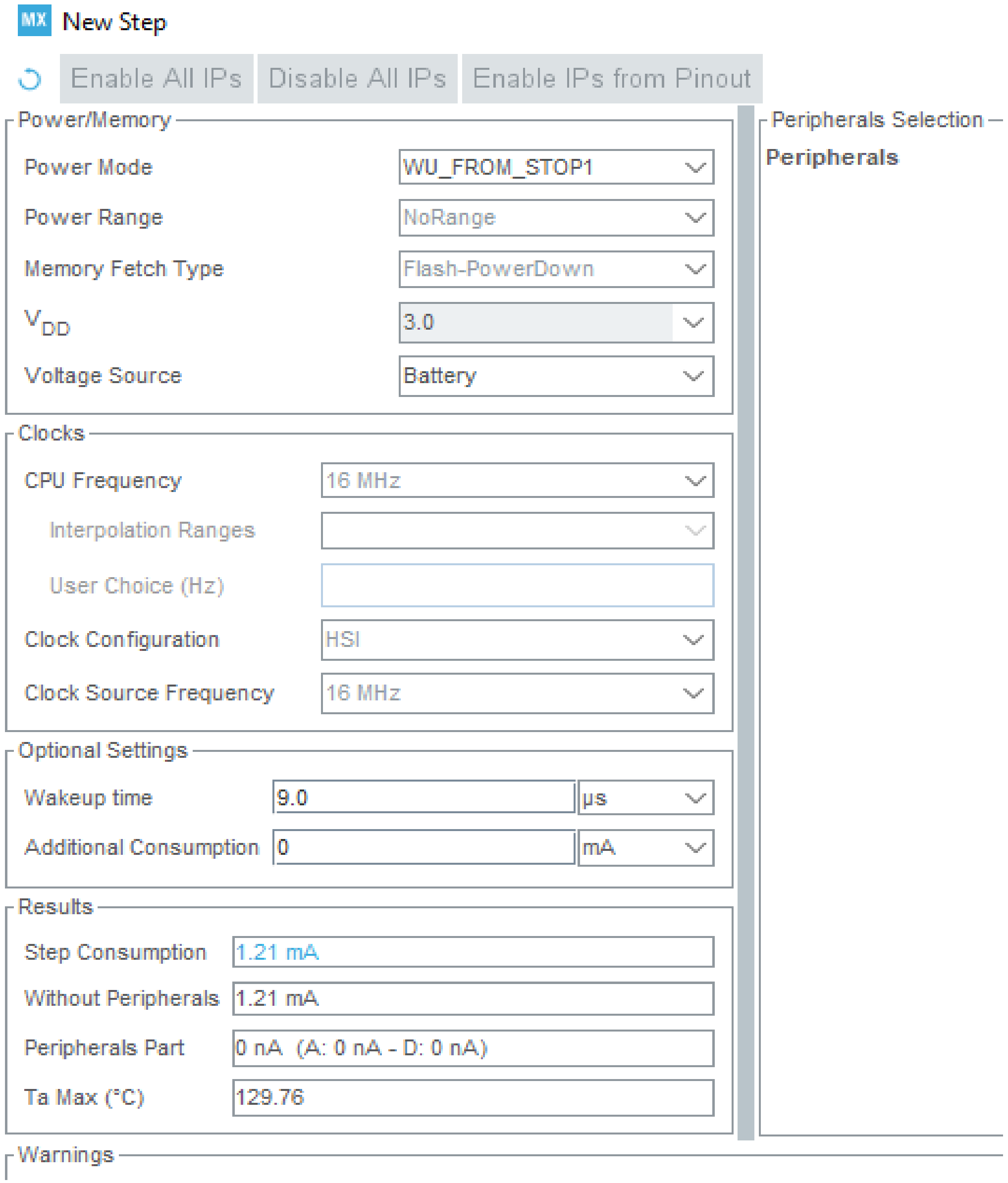


life.augmented

Add a Wakeup from STOP1 mode step

- **Add a last step: Wakeup from STOP1 mode**
  - VDD = 3V
  - Voltage source: Battery
  - Click “Add”

  - Resulting step consumption should be 1.21 mA



life.augmented

Average Current Consumption Result

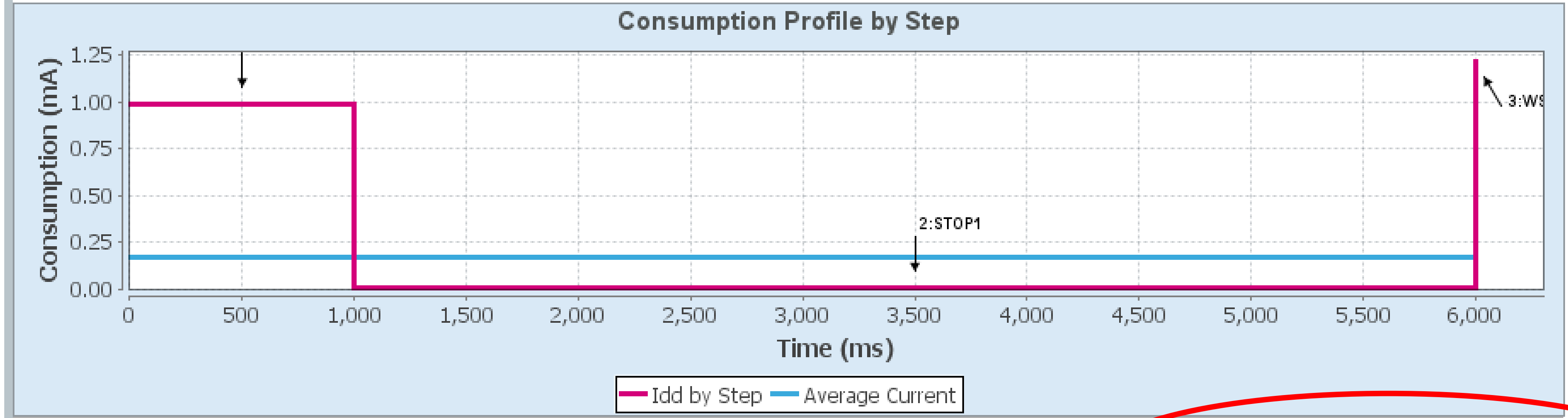
- **Note: the Current consumption numbers are for the MCU only.**

Sequence Table

Step	Mode	V_{DD}	Range/Scale	Memory	CPU/Bus Freq	Clock Config	Peripherals	Step Current	Duration
1	RUN	3.0	Range2-Medi...	FLASH	8 MHz	HSI	GPIOA GPIOC...	988.25 µA	1 s
2	STOP1	3.0	NoRange	Flash-Power...	16 MHz	HSI	RTC*	3.4 µA	5 s
3	WU_FROM_S...	3.0	NoRange	Flash-Power...	16 MHz	HSI		1.21 mA	9.0 µs

Display

Plot: All Steps



Sequence Time / Ta Max 6 s / 129.76 °C

Battery Life Estimation 1 year, 9 months, 26 days, 20 hours

Average Consumption 167.54 µA

Average DMIPS 10.0 DMIPS

life.augmented

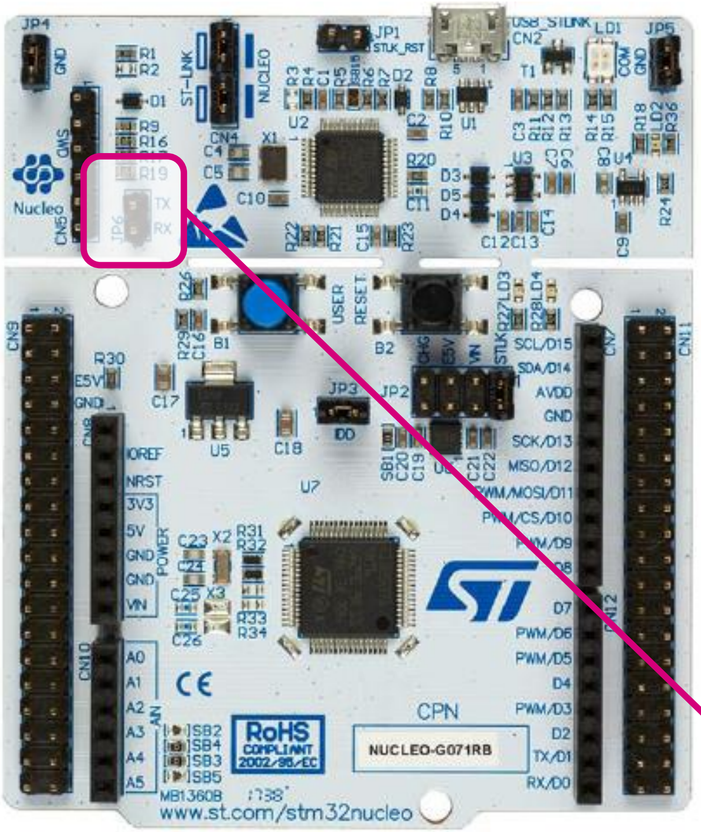
Optional Lab: printf() debugging using UART

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

<u>Set up additional GPIO / Clocks:</u>
PA2 – LPUART1, “LPUART1-TX”
PA3 – LPUART1, “LPUART1-RX”
LPUART1 Clock = PCLK1 (64MHz)

<u>LPUART1 settings:</u>
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)

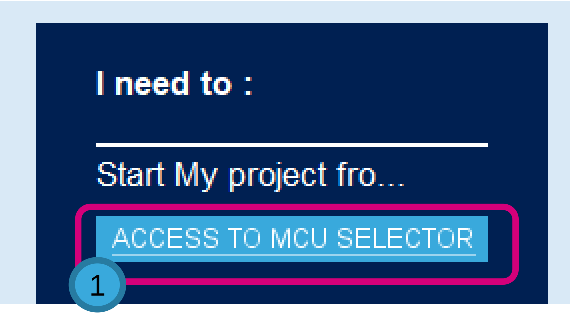
STM32G0 USART/LPUART features

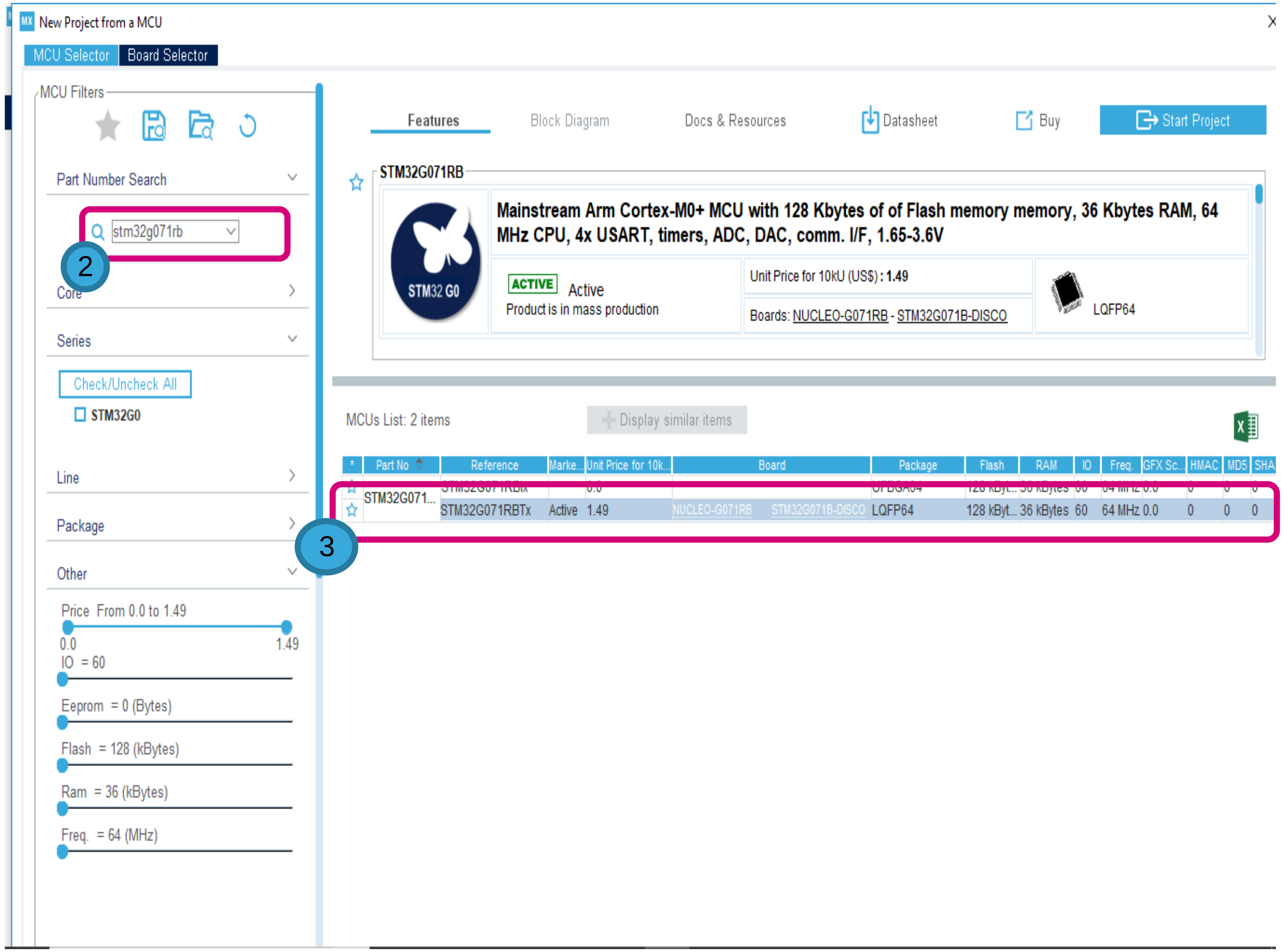
USART features	USART1/2	USART3/4	LPUART1
Hardware flow control for modem	x	x	x
Multiprocessor communication	x	x	x
Synchronous mode (Slave/Master)	x	x	-
Smartcard mode	x	-	-
Single wire half duplex communication	x	x	x
IrDA SIR ENDEC	x	-	-
LIN mode	x	-	-
Dual clock domain and wakeup from Stop mode	x	-	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

Create New Project

- In STM32CubeMX, click “Home”
- Click **Access To MCU Selector** 1



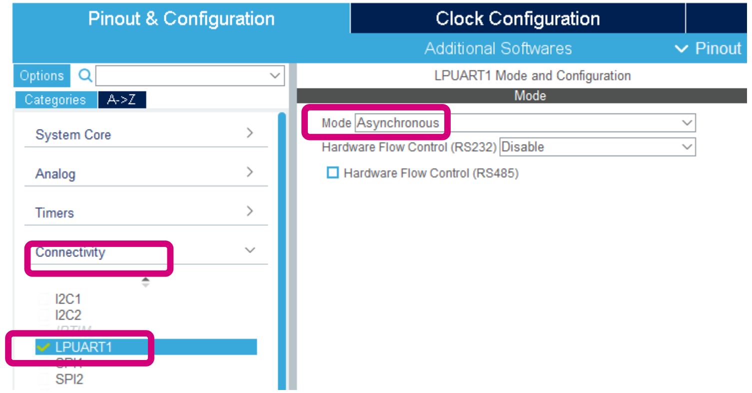


- Select **STM32G071RBTx** 2
  - **LQFP64, 128KB Flash**
- Double Click 3

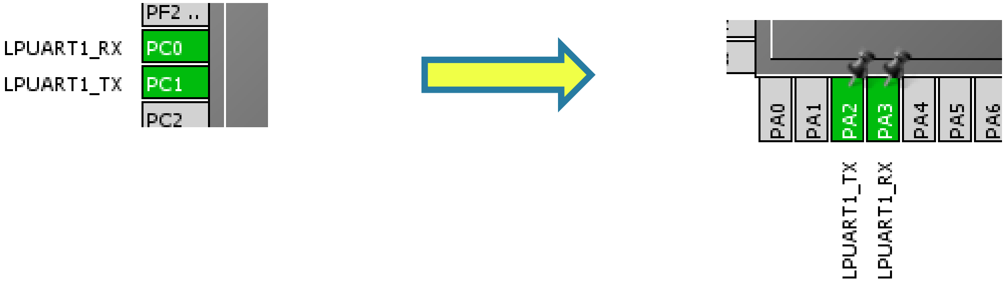
life.augmented

GPIO Configuration additions

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

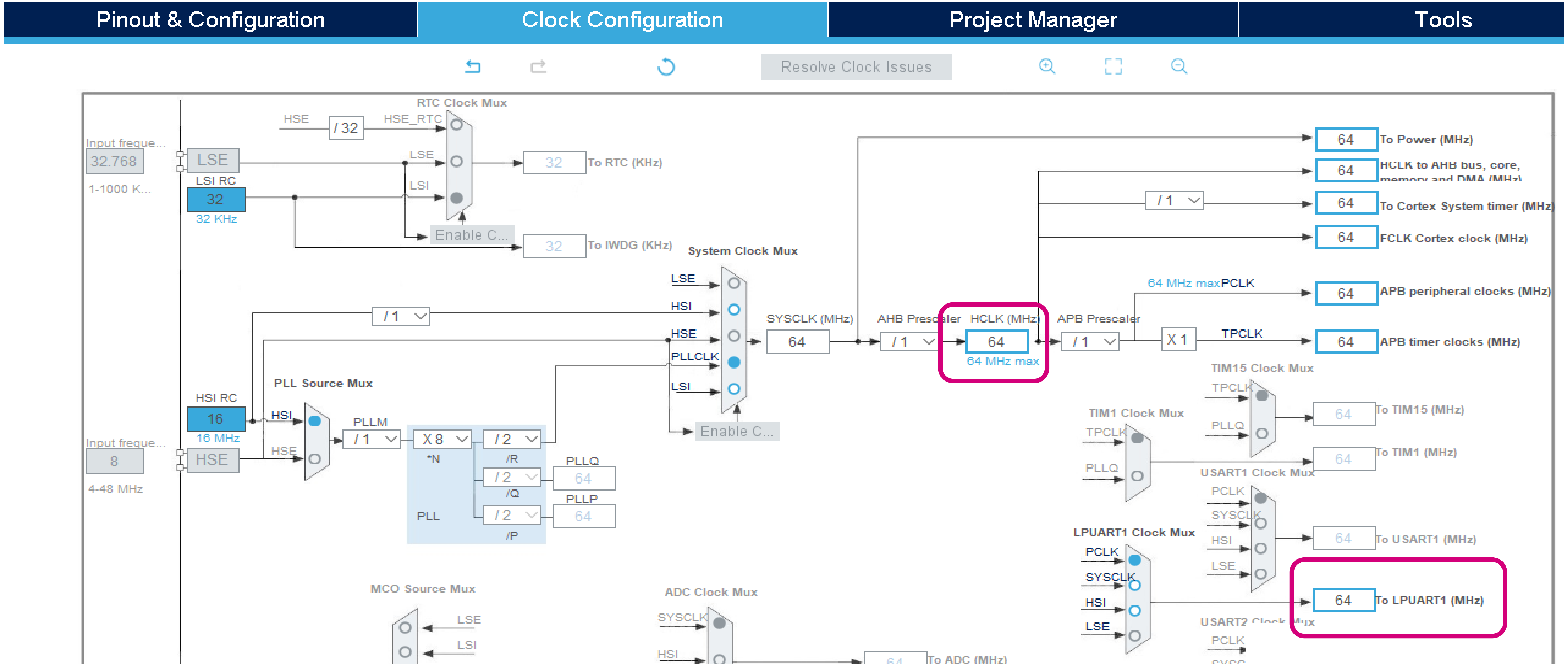


- Use PA2 & PA3 for Tx / Rx pins:
  - These are the alternate mapping pins (PC0/PC1 are default)
  - So need to remap



Clock Configuration

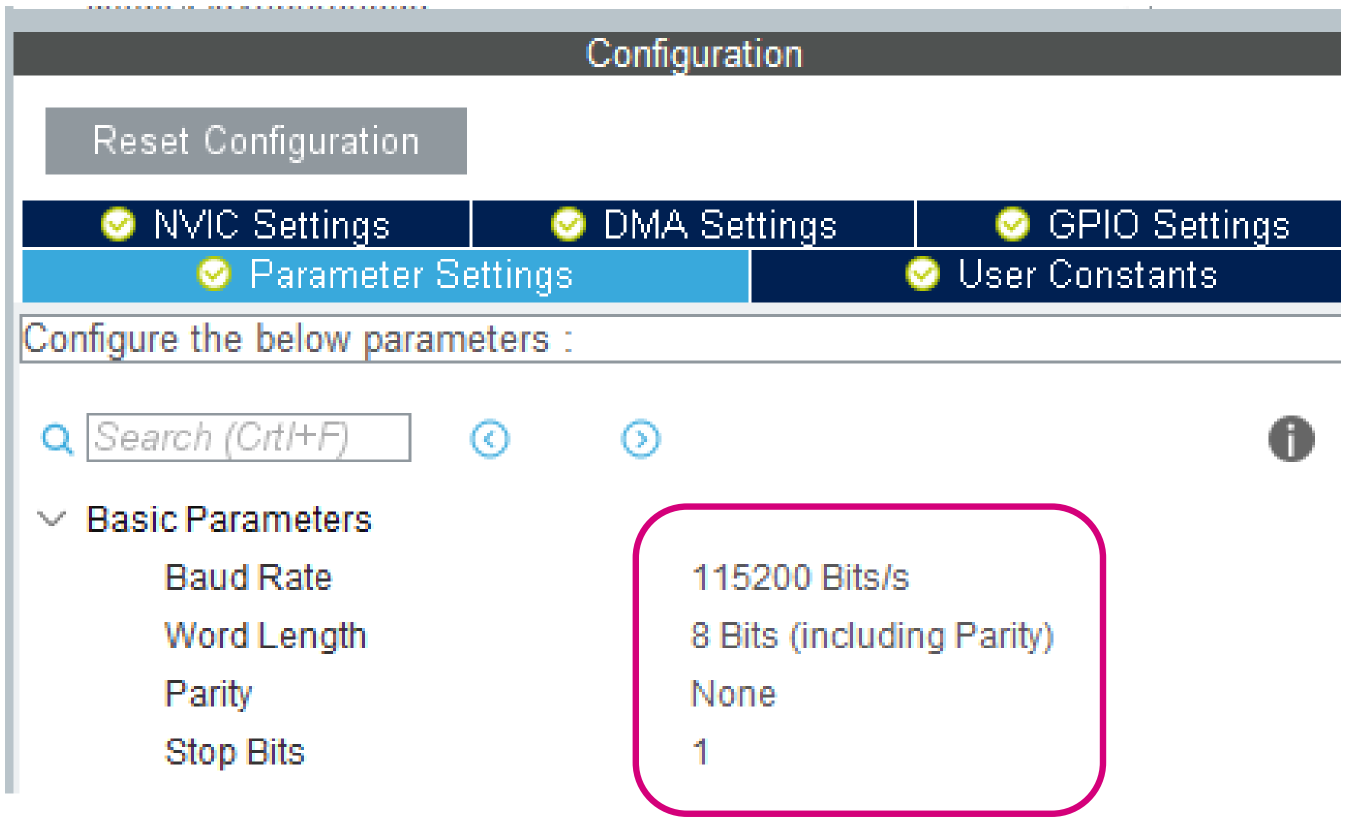
- Run the STM32G0 at 64 MHz for this lab, the LPUART1 clock also at 64 MHz.



LPUART1 Configuration

- Click on the Configuration tab and select LPUART1

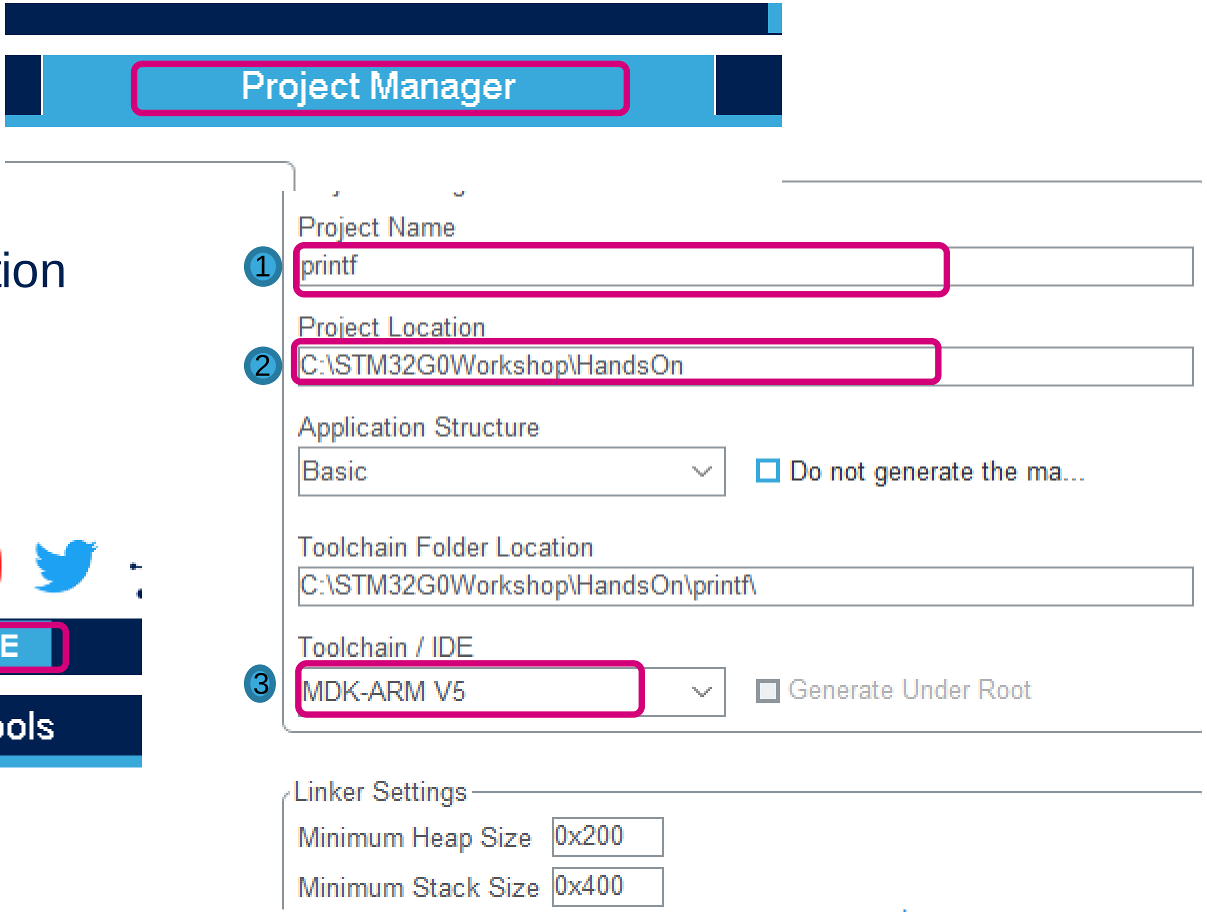
- Parameter Settings tab
  - 115200 Bits/s
  - 8-bit word length
  - No parity bit
  - 1 Stop bit
  - Keep Default settings for the rest



life.augmented

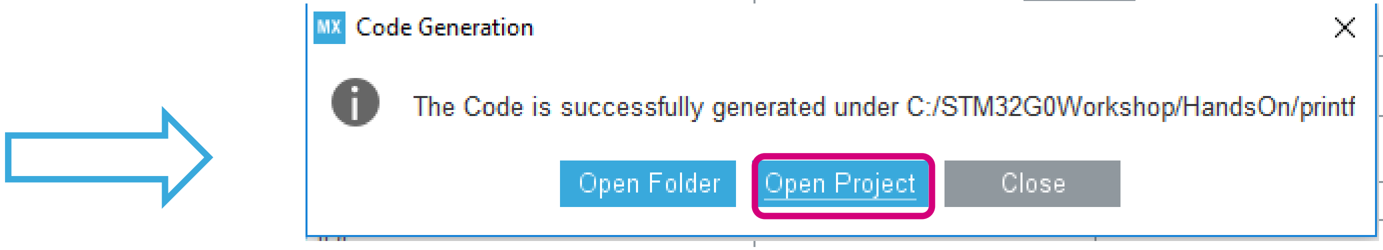
Generate Source Code

- 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 */
```

Adding application code in main.c

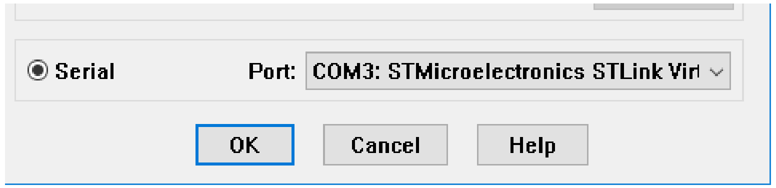
Add application code in main loop:

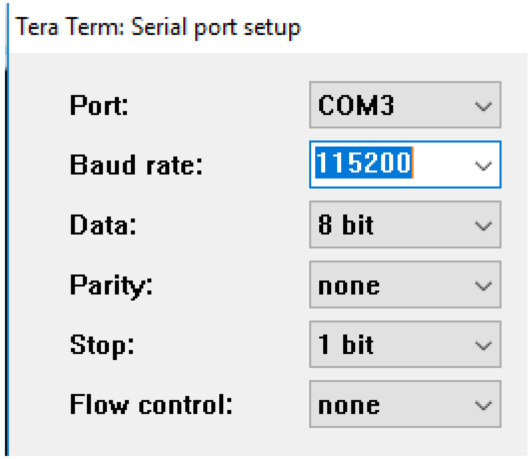
```
/* USER CODE BEGIN WHILE */
 while (1)
 {
     printf("** Hello World ** \n\r"
     HAL_Delay(1000);
 /* USER CODE END WHILE */
```

```
  /* 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.

life.augmented

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!