Development chain for STM32

STM32 microcontrollers

The STM32 family, developed by STMicroelectronics, has been designed to offer new degrees of freedom to its users. These microcontrollers, which come in a complete range of 32-bit products, not only combine high performance, real-time, low voltage and low power, but also offer full integration and easy development.

The STM32 microcontrollers are built around the ARM Cortex-M processor developed by ARM.

ARM Cortex M : Presentation

The ARM Cortex-M range belongs to a generation of processors providing a common hardware architecture for a wide range of technological demands.

In contrast to the Cortex-A and Cortex-R ranges, the Cortex-M is a complete core that has become a standard in the field of processors. It is based on a well-defined system architecture that includes interrupt management, a system clock, a debugging system and a memory address space.

The memory space is divided into separate predefined regions for code, data, devices and system. Based on a Harvard architecture, the Cortex-M thus has several buses, allowing it to perform operations in parallel. The data misaligned access mode ensures better memory usage and more efficient access to the peripheral registers. In addition, the Cortex-M has RISC (Reduce Instruction Set) instructions and specific instruction cycles.

The processors developed by the ARM company are not autonomous components but only IPs (Intellectual Properties). It is manufacturers, such as STMicroelectronics, who make autonomous components such as the STM32 by adding memory and peripherals to the Cortex-M after purchasing the licence from ARM.

A key component of the Cortex-M is its NVIC (Nested Vector Interrupt Controller), which has a standard interrupt management structure and provides up to 240 peripheral sources that can each be dedicated to one interrupt. The time between the reception of the interrupt and the start of execution of the corresponding code is optimised in part by automatic stack management in the Cortex-M firmware.

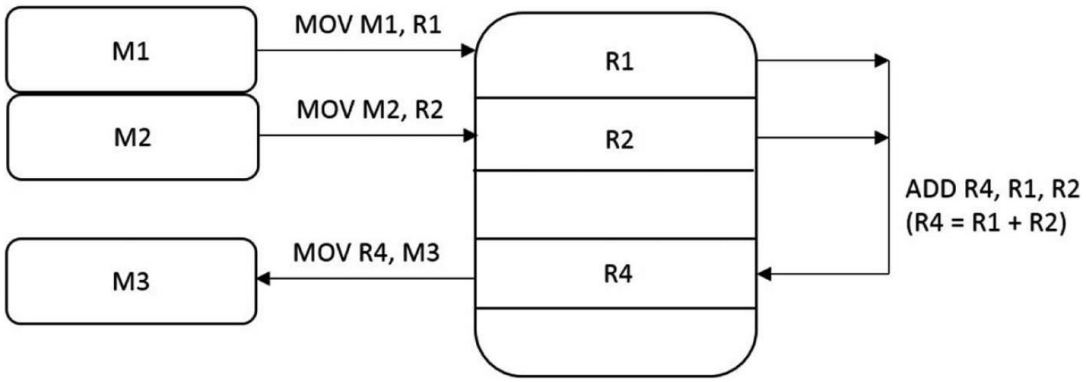
At first glance, the peripherals appear to be classics of microcontrollers: a dual analog-to-digital converter, general purpose counters, I2C, SPI, CAN, USB and a real-time counter. However, each of these devices has additional functionality. For example, the A/D converter has a temperature sensor and multiple conversion modes.

Similarly, each meter has four capture/comparison units and each meter block can be combined with the others to build more sophisticated meter tables. Another meter can control a motor, with PWM (Pulse Width Modulation) outputs. The STM32 also includes a DMA (Direct Memory Access) controller with twelve channels. Each channel can be used to transfer data to or from a peripheral register in 8, 16 or 32 bit memory.

RISC ARCHITECTURE

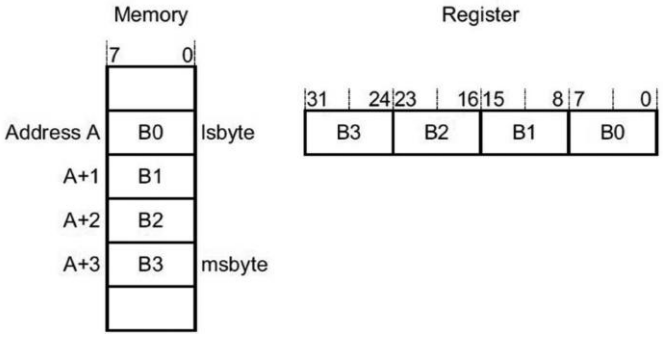
The Cortex-M has a RISC instruction set with loading and saving. When processing the data instructions, the operands are loaded into the central registers, the operation is then carried out on these registers and the results are finally saved back into the memory.

The figure below illustrates this mechanism with the addition of two operands: these are first loaded into registers R1 and R2, the result of the addition is then placed in register R4, and finally the latter is saved in memory.



MEMORY ORGANISATION

The Cortex-M sees its memory as a linear sequence of bytes numbered in ascending order from zero. These bytes are encoded in memory in the little endian format. In this format, an example of which is shown opposite, the processor stores the least significant byte of a word first and the most significant byte last.



THE CORTEX-M RANGE

The Cortex-M range is available in different models:

- The Cortex-M0/M0+/M1 handles general data as well as standard input-output tasks.
- The Cortex-M3 handles advanced data processing and bit field manipulation.
- The Cortex-M4 consists of a DSP (Digital Signal Processing) and an FPU (Floating Point Unit).
- The recent Cortex-M7 delivers a very high level of performance.

The set of instructions for each model in the Cortex-M range is bottom-up compatible.

The STM32 family

Currently, the STM32 family is divided into three main categories:

- The STM32Fxx, **F** for Fast, are used in applications where speed is a priority and power consumption is not a primary requirement.
- The STM32Lxx, **L** for Low power, are used in areas where low power consumption takes priority over application speed.
- And very recently, the STM32Wxx, **W** for Wireless.

STM32WB SERIES

The STM32WB series supports Bluetooth® LE 5.0 and IEEE 802.15.4, Zigbee® and Thread communication protocols, operating individually or simultaneously.

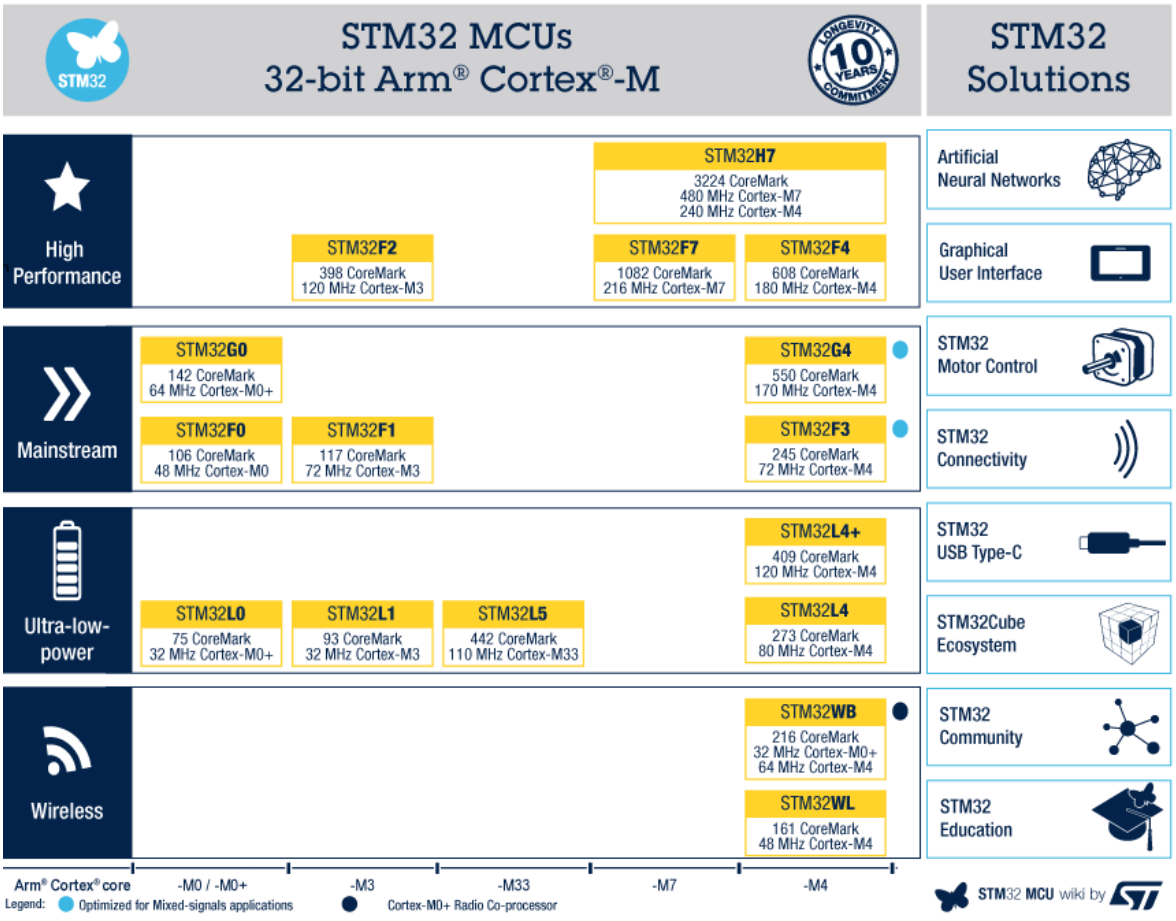
STM32WL SERIES

The STM32WL MCU series is the world's first system-on-chip compatible with LoRa® and Sigfox.

The STM32WL is fully open and supports multi-modulation, making it the ideal choice for LPWAN and IoT developments with ultra-low power consumption without compromising on performance.

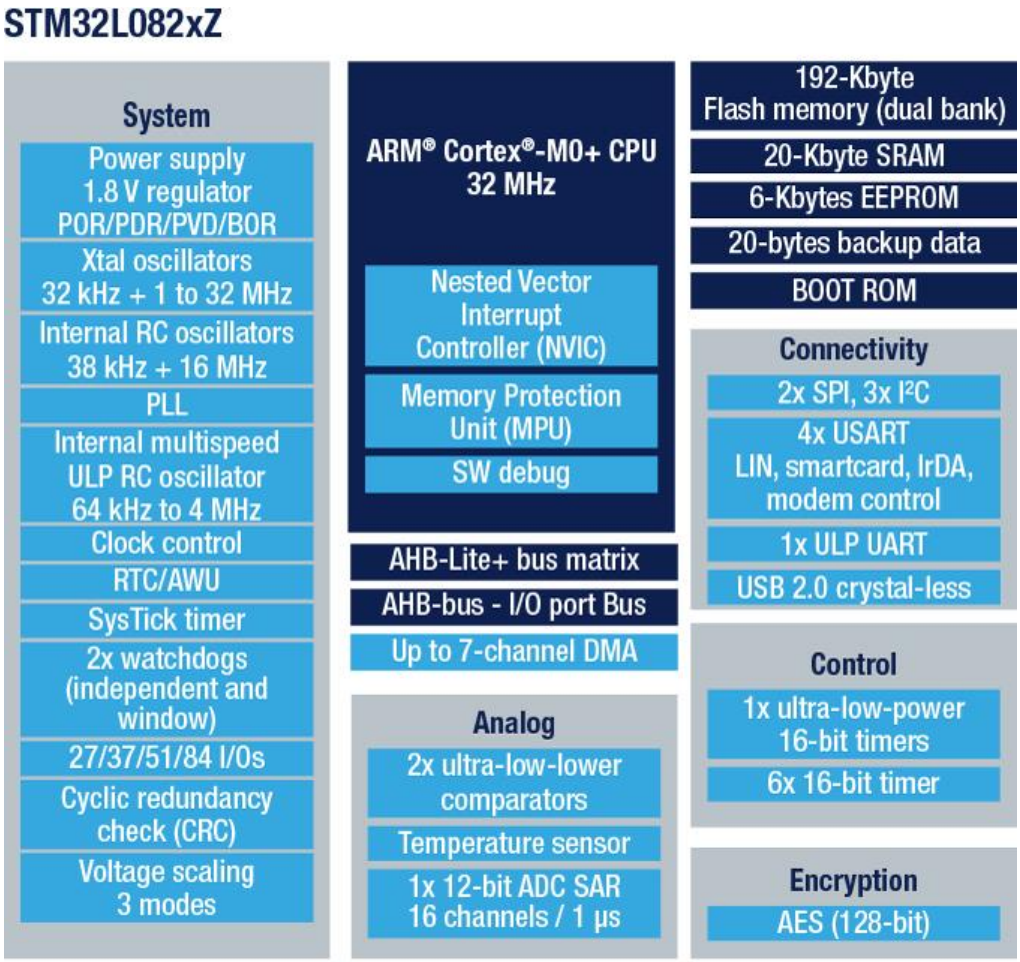
In fact, to go faster you have to consume more and, conversely, the less you want to consume the less you have to go fast. It is therefore very difficult, if not impossible, to have these two qualities, speed of execution and low consumption, in the same microcontroller.

Each of the two main categories STM32Fxx and STM32Lxx divides itself into series according to the chosen Cortex-M (figure below). Thus, an STM32F7 is composed of a Cortex-M7 and an STM32L4 of a Cortex-M4. The number after the letter F or L does not always reflect the Cortex-M used; for example, STM32L1 is composed of a Cortex-M3. In this case, please refer to the documentation to find out which Cortex-M is used to make up the chosen STM32.



For example, the STM32082KZ, which will be used in the tutorial suite, includes a Cortex-M0+ with a maximum frequency of 32 MHz, flash memory, RAM and EEPROM, an NVIC interrupt vector controller, a power management unit, internal and external clock units, two watchdogs to ensure the smooth running of the program, a CRC data transmission control system, general purpose I/Os, A/D and D/A converters, counters, interfaces for USB, SPI, I2C and USART peripherals.

The figure below shows all the blocks in the STM32082KZ.



Internal structure of the STM32

Memory zone

The Cortex-Ms have 4 GB addressable memory space organised into several sub-regions with different logical functionalities.

- The first 512 MB are dedicated to the code area.
- The next region starts at 0x20000000 and can go up to 0x3FFFFFFF. It corresponds to the SRAM of the microcontroller.
- The next 0.5 GB region is dedicated to device addressing. Each device in the selected STM32 (I2C, SPI, USART, counters...) has a dedicated area.
- The next 2 GB region is dedicated to external SRAM and flash memories. The Cortex-M can execute code from external memories, thus extending memory resources.

Clock tree

In addition to the two external oscillators LSE (Low Speed External Oscillator) and HSE (High Speed External Oscillator), the STM32 has two internal RC oscillators. After circuit initialisation, the clock source for the Cortex-M is the HSI (High Speed Internal Oscillator). The second internal oscillator is the LSI (Low Speed Internal Oscillator), which operates at a nominal frequency of 32,768 kHz and is used for real-time clocks and watchdogs.

The Cortex-M can have HSI or HSE oscillators or an internal phase-locked loop as clock source. This phase-locked loop can itself have the HSI or HSE oscillators as a clock source. Note that the HSI is less accurate than the HSE. When using serial communication devices or for accurate counting functions, the HSE must be used.

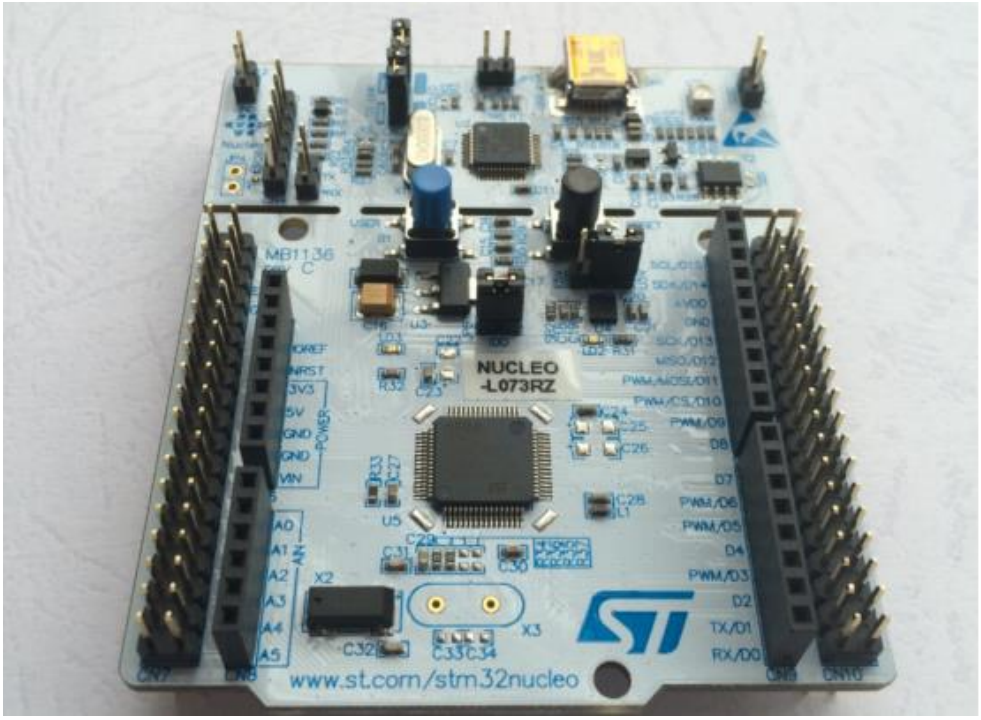
Boot mode

Depending on the STM32 chosen, the boot pin(s) (external boot) is/are used to select which region of the memory will be used for booting. This can be flash memory, bootloader (internal bootloader) or SRAM. For normal operation, pin BOOT0 must be connected to ground. Thus, booting will occur in flash memory from address 0x080000.

Focusing port

A debug (debugging) port is used to debug the code and then program the target STM32. The Cortex-M supports two standard connections: a five-pin JTAG port and a two-pin SWD (Serial Wire Debug) port. Both of these configurations sacrifice pins so that the focus port is available. After initialization of the STM32, the Cortex-M configures the dedicated SW pins in this mode for user access.

The nucleo-64 cards



Nucleo boards are low-cost, easy-to-use platforms for quickly evaluating the prototyping of an application or developing a personal design. These boards are built around the STM32 microcontroller and are available in three families: Nucleo-32 with LQFP32 housing, Nucleo-64 with LQFP64 housing and Nucleo-144 with LQFP144 housing. This tutorial will focus on the most popular boards currently in use, i.e. the Nucleo-64.

Features

The Nucleo-64 boards are designed around an STM32 microcontroller in an LQFP64 package. They are available in as many models as there are microcontrollers in the STM32 family. However, the dimensions, connectors and all features, with a few exceptions, remain the same for all Nucleo-64 boards, making it easy to switch from one model to another without having to rethink the entire application. These boards have the following two types of connectors:

- a single row Arduino Uno Revision 3, compatible with Arduino expansion cards ;
- a double row STMicroelectronics Morpho, which gives access to all the resources of the STM32.

A ST-LINK/V2-1 type linker (link generator) allows the programmer to communicate with the STM32 on the board via a two-wire SWD link, as the five-wire JTAG link is not offered on Nucleo boards.

There are several choices for the power supply of the card:

- via the USB connector (VBUS) ;
- externally (7 V < VIN < 12 V) from Arduino or ST Morpho connectors ;
- externally (E5V) from Morpho ST connectors ;
- externally (3V3) from Arduino or ST Morpho connectors.

The Nucleo-64 cards contain three LEDs :

- LD1 to indicate that the USB port is operational ;
- LD2 which can be used in the ;
- LD3 which indicates the presence of the power supply.

There are also two push buttons:

- B1 for application if necessary ;
- B2 to initialise the card.

The Nucleo-64 cards contain an external 32,768 kHz oscillator used by the STM32.

The USB port can be listed in three ways :

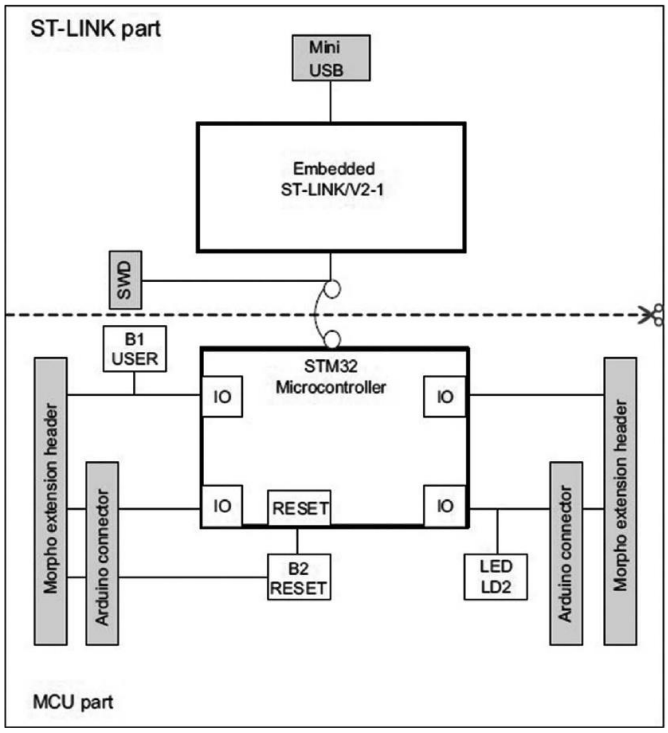
- in a virtual communication port ;
- as a mass storage device ;
- in focus port.

Finally, the Nucleo-64 cards support various IDEs (Integrated Development Environment) including IAR, Keil and STM32CubeIDE which we will see in detail.

Thanks to Arduino Uno and Morpho connectors (all the microcontroller's pins are routed). If required, ST *shields* can also be plugged onto the latter, which can prove useful, especially for low-cost prototypes with sophisticated sensors. And for pin assignment and internal circuitry, it's worthwhile looking at the UM1724 user manual.

The figure opposite shows the block diagram of a Nucleo-64 card, which is divided into two parts:

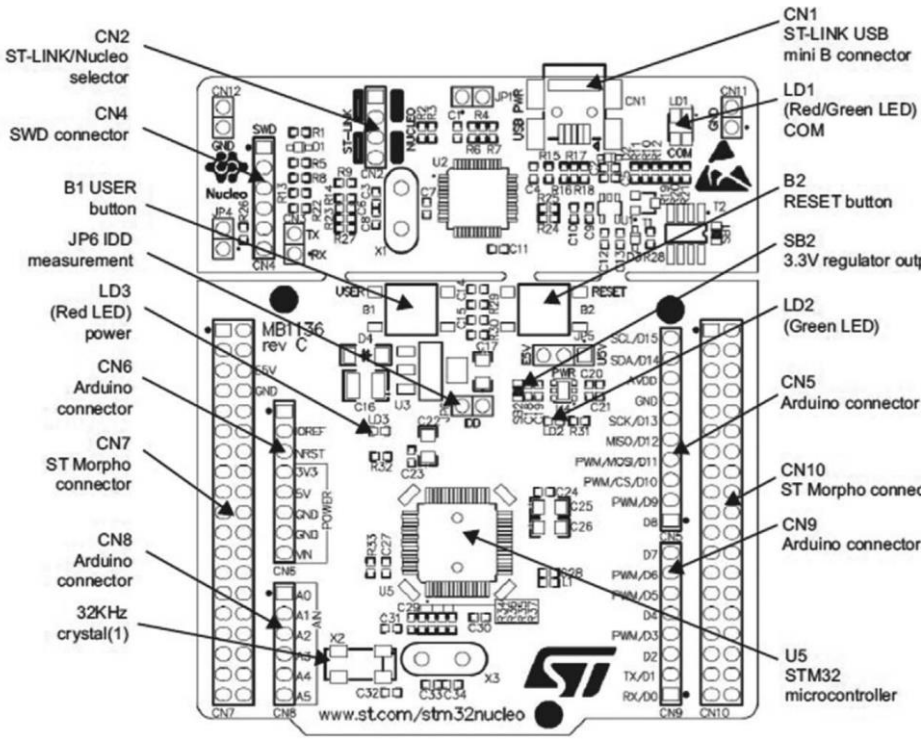
- the microcontroller part, at the bottom of the diagram, on which it is easy to recognise the push buttons B1 for the user and B2 for initialisation, the LD2 LED for the user, the Arduino and STMorpho connectors, and the STM32 microcontroller, located in the centre ;
- the programmer-linker part of type ST-LINK/V2-1, at the top of the diagram. It is also an STM32 but smaller than the main microcontroller. The ST-LINK/V2-1 is connected to the computer by a Mini-USB type connector and to the main STM32 by a two-wire link meeting the SWD (Serial Wire Debug) standard.



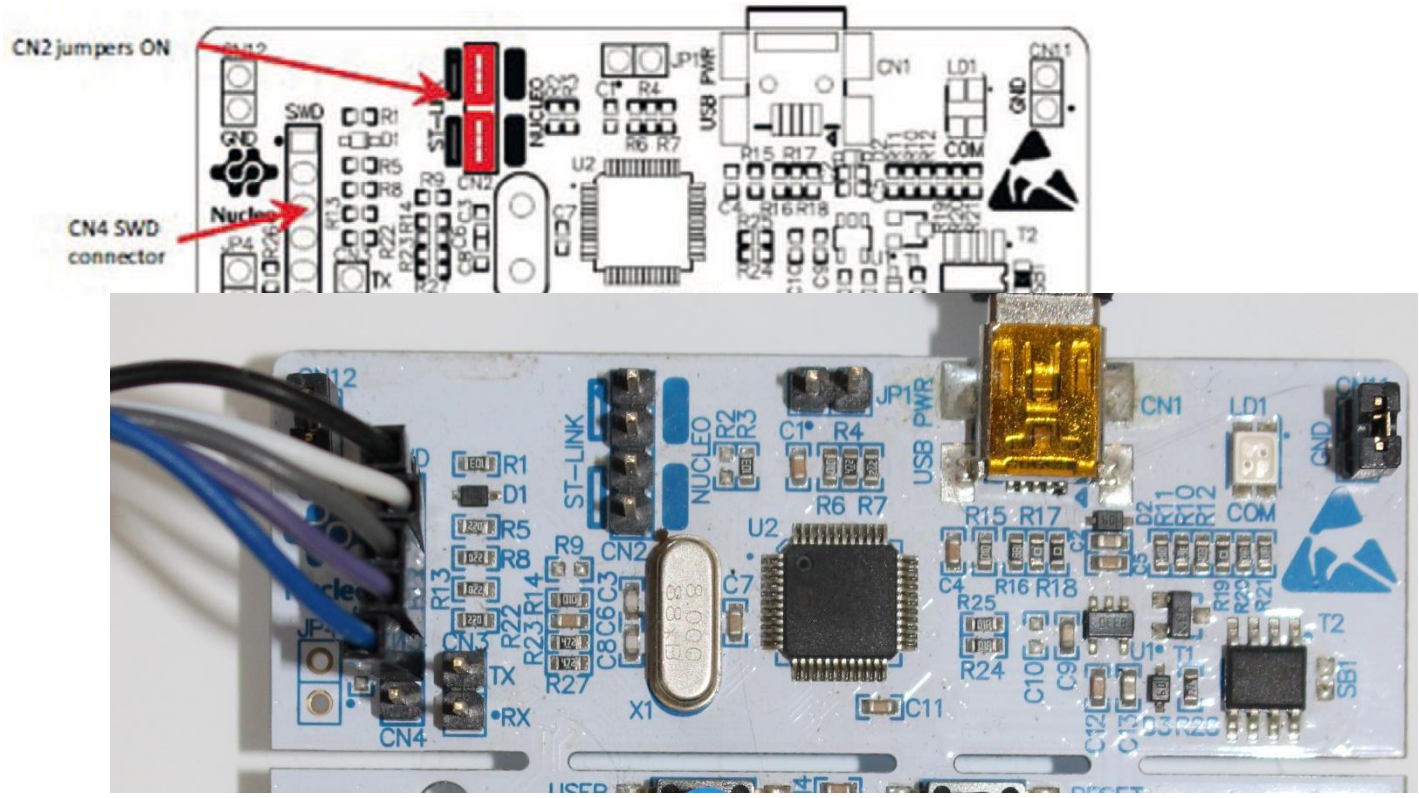
The two parts of the Nucleo board can be separated from each other, either to save space or because the application is complete and the STM32 no longer needs to be programmed. In this case, the main STM32 can be powered:

- or from the STMicroelectronics Morpho CN7 connector by VIN, E5V or 3V3 ;
- or from the Arduino CN6 connector by VIN.

Once the two parts of the board are separated, however, it is still possible to program the main STM32 from the ST-LINK/V2-1 by connecting them via CN4 wires and the SWD pins available on the STMicroelectronics Morpho connector (SWCLK CN7 pin 15 and SWDIO CN7 pin 13).



To program the main STM32, the two CN2 jumpers must be placed as shown in the figure below. In this case, the CN4 connector, which is used when the two boards are separated, should not be used, as it could interfere with the communication with the main STM32.



To use the ST-Link part to program an external board (which will be our case when programming the STM32_LP sensor board), remove the two CN2 jumpers and use the CN4 connector to connect it to the target board as shown in the table below.

Broche	CN4	Désignation
1	VDD_TARGET	VDD de la carte externe
2	SWCLK	Horloge SWD
3	GND	Masse
4	SWDIO	données entrée/sortie SWD
5	NRST	RESET du STM32 cible
6	SWO	Réservé

Only the highlighted pins are mandatory for the connection to the target.

communication usart2

The USART2 (Universal Synchronous/Asynchronous Receiver Transmitter) communication interface available on the PA2 and PA3 ports of the main STM32 can be connected to the microcontroller providing the STLINK/V2-1 function, to the STMicroelectronics Morpho connector or to the Arduino connector. The choice is expressed by positioning the corresponding SBxx solder bridges.

By default, the USART2 communication interface between the main STM32 and the STLINK/V2-1 is active to support Virtual Port Com (virtual port) communication with SB13 and SB14 ON, SB62 and SB63 OFF configuration.

If communication between the PA2 and PA3 ports of the main STM32 and its expansion board is required, then SB13 and SB14 OFF, SB62 and SB63 ON must be set to SB13 and SB14 OFF, SB62 and SB63 ON. In the latter case, it is possible to connect another USART communication interface of the main STM32 to the STLINK/V2-1 using flying leads between the STMicroelectronics Morpho connector and CN3.

Let's move on to programming

We will distinguish between two approaches to software development:

1. for rapid experimentation on Nucleo 64 (or other) type cards. This way of working is certainly simple, but it does not allow for debugging or the creation of a personal map. This is the user-friendly *Mbed* online platform, for which the card is presented as a simple USB mass storage device.
2. For an experiment to make the most of the microcontroller's capabilities and to use the professional C tools made available by ST, we will use the free IDE from ST STM32CubeIDE. But this way of working presupposes some knowledge of microcontrollers and software.

Presentation of the mbed development tool

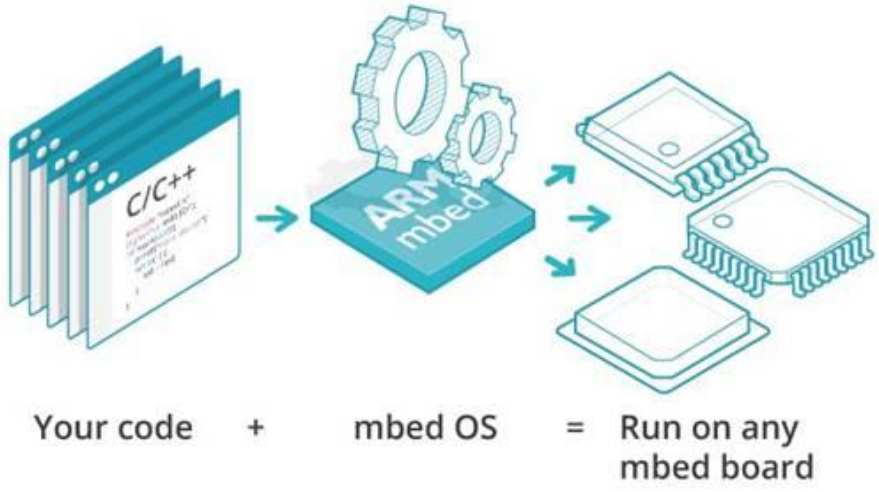
The acquisition of a new development card often comes with some difficulties:

- learning a new integrated development environment (IDE),
- need for a licence or limitation of code size,
- possibly acquisition of a programmer,
- complex microcontroller configuration,
- …

It is therefore not easy to do rapid prototyping...

ARM has developed a programming environment that allows rapid prototyping using cards based on microcontrollers from the Cortex M family.

Many manufacturers (NXP, ST, Silicon Labs, etc.) offer a variety of evaluation and prototyping boards that are compatible with the popular and easy to use ARM mbed development platform. With an online Software Development Kit, open source software libraries, hardware designs and online tools, this is the fastest way to create products based on ARM microcontrollers.



The tools compatible with mbed development boards allow most of the low-level work usually associated with microcontroller development to be dispensed with.

You develop your code using intuitive APIs (Application Programming Interface) that allow you to design your applications without worrying about the exact implementation in the microcontroller or its peripherals.

The online C/C++ compiler requires no downloading or installation and allows you to create and compile your programs very simply and quickly.

It can be used with Internet Explorer, Firefox, Safari and Chrome on computers running Windows, Mac or Linux. So you can use it anywhere, as long as you have an internet connection.

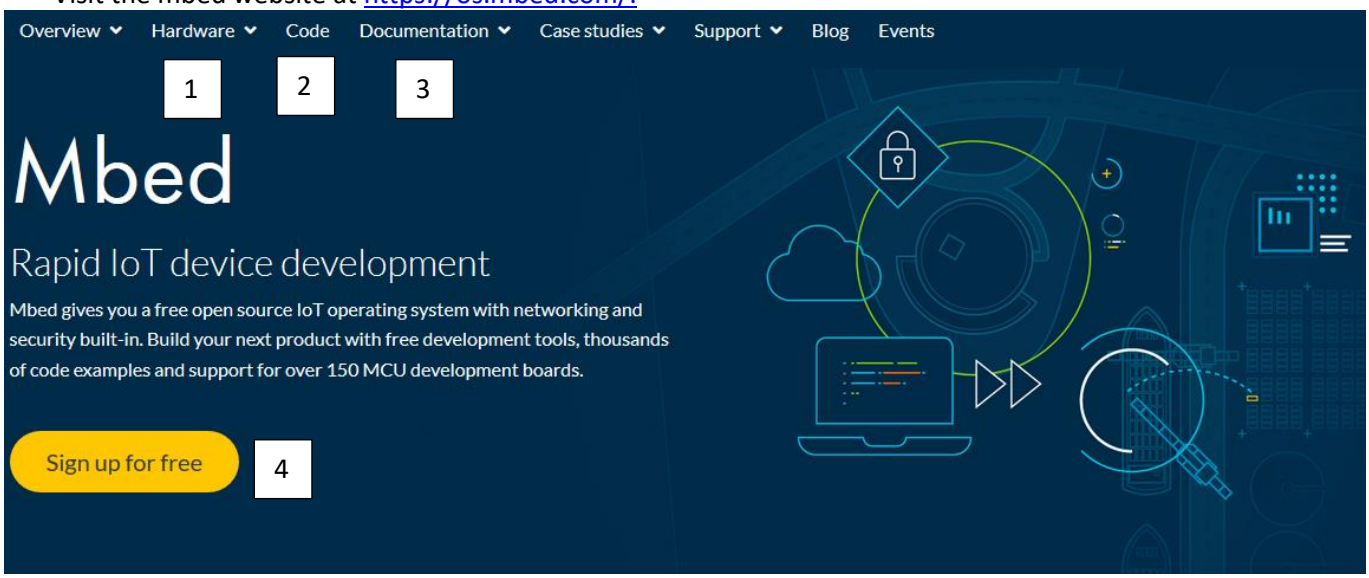
Your projects will be stored online and you will find them wherever you are.

The mbed platform is supported by an active international community of skilled developers who collectively contribute to accelerating the completion of prototypes. Best of all, they are building a collection of information and tutorials called "Cookbook" at https://os.mbed.com/cookbook/Homepage.

Presentation of the development tools.

To start, connect the card to a USB port on your PC, it will be recognised as a USB key.

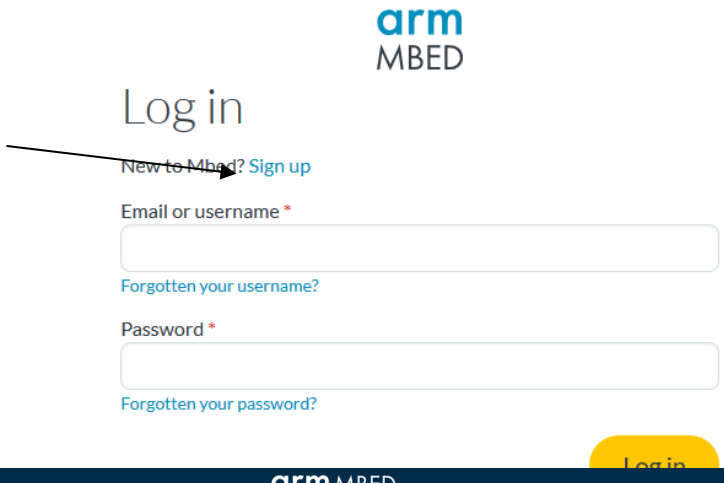
Visit the mbed website at https://os.mbed.com/.



The site is very important and will contain a lot of information to help you progress. Visit the different menus.

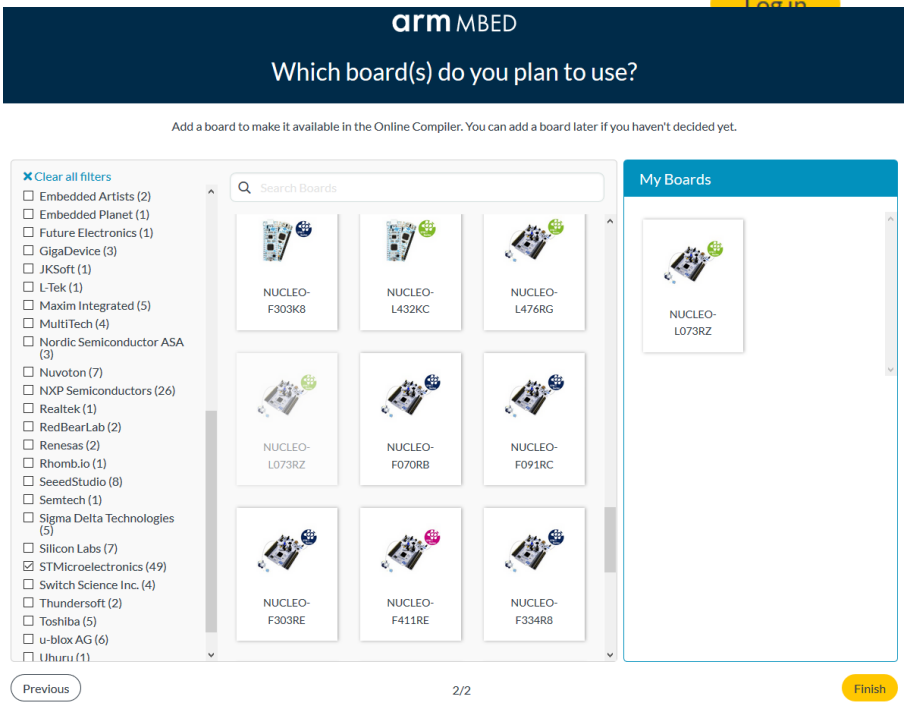
- In 1, you will have the choice between :
  - Boards that will give you all compatible boards
  - Components where you will find by category, (display, sensors, etc...) the various components or modules already used by other users, who have shared their code. This part will allow, by using libraries, to develop more quickly
- In 2, you will find a large amount of code that works and is shared by the community.
- In 3, you will have the choice between :
  - Documentation that will give you the user manual of the programming language, for each resource (input/output, analog I/O, etc.), all the associated commands and how to use them. You will even find an example of a program that you can import.
- In 4, you will finally be able to log in or create your own account to get started.

You will be able to login or create an account. Create your account and we will create a first programme.



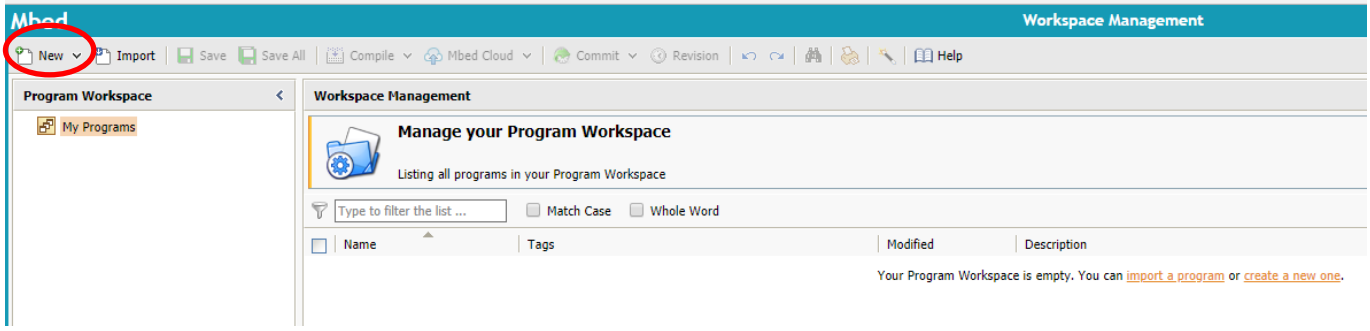
You must then choose one or more cards in your possession in order to be able to work. We choose to restrict the choice to boards from Board vendor ST. For example, we select the L073RZ Nucleo board. To finish click on Finish.

You must also confirm your email address by clicking on the link you must have received by email.

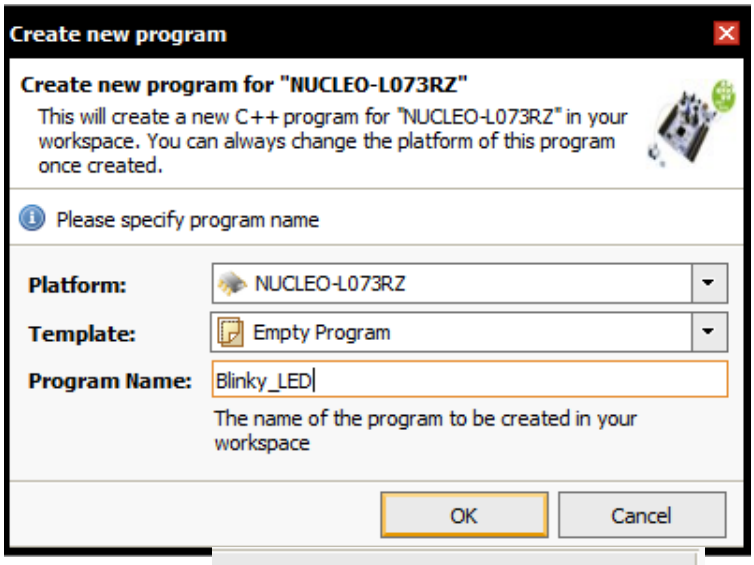


Click on the Compile button to access your workspace.

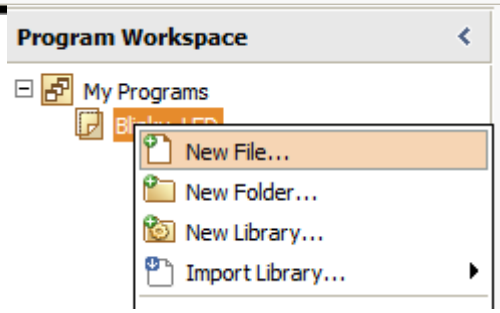
At the top left, click on new (for new programme)

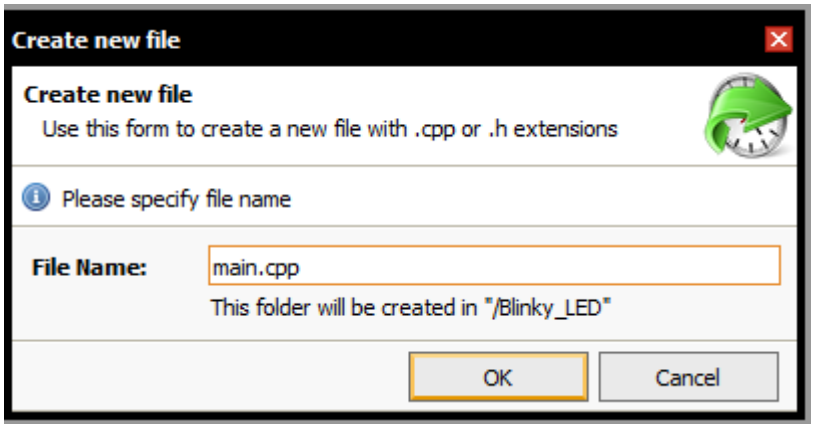


Finally we can create a new programme by clicking on " new " .
We check that the chosen target is NUCLEO L073RZ and select empty program to start from scratch. Finally, we have to give a name to our project (here Blinky_LED) and click on OK.

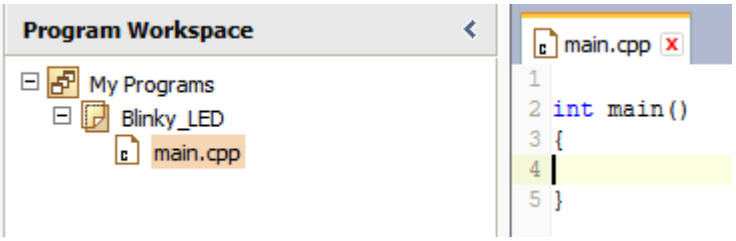


You must first add a file in this new folder, right-click on the name of our "program" and then on New File, which we will name main.cpp.



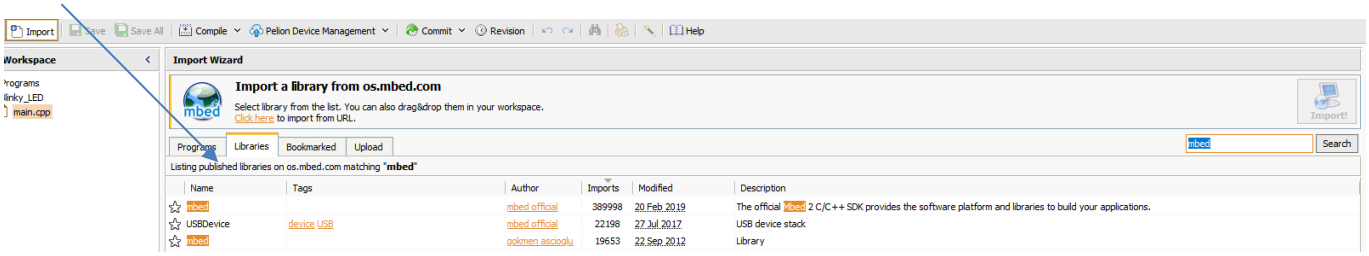


Trivially, a program in C++ must contain :



Then you need to add the mbed library to work with the APIs written for the card.

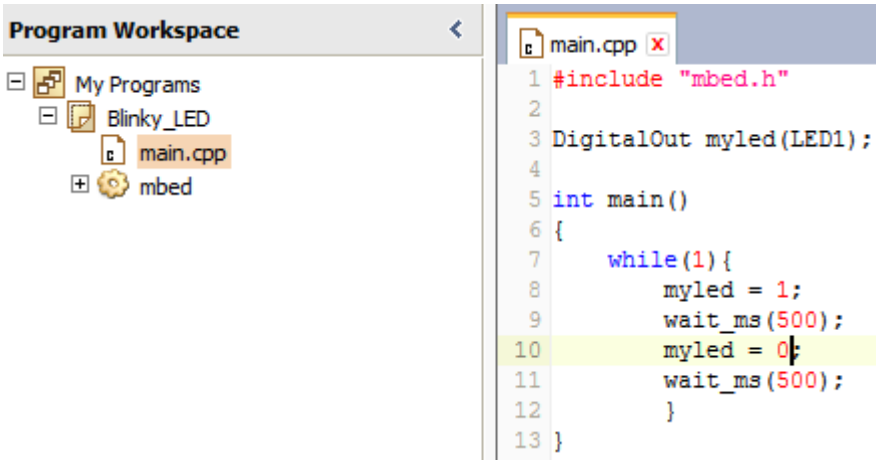
Click on "import" to open a window in which you can search for the mbed library. To include this official library, just double click on its name :



Then type the program whose algorithm is :

Blinky_LED algorithm

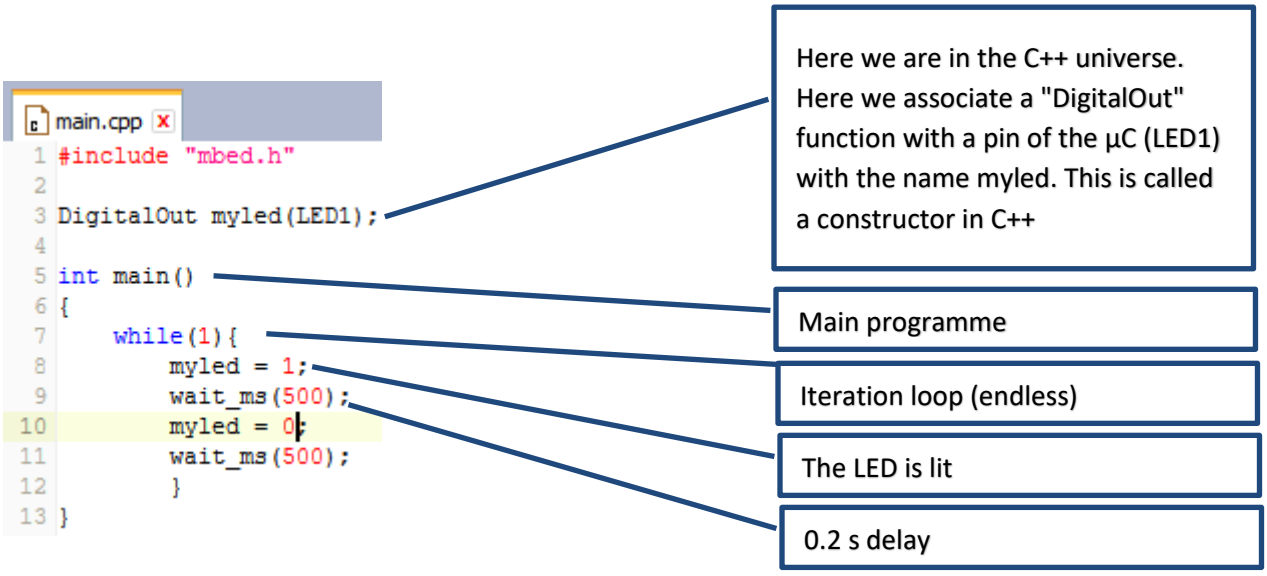
Start
  Go to
    Switch on LED
    Wait 500 ms
    Switching off LEDs
    Wait 500 ms
  End Iterate
End



Analysis of the programme :

You recognise the classical architecture of a C-language programme.

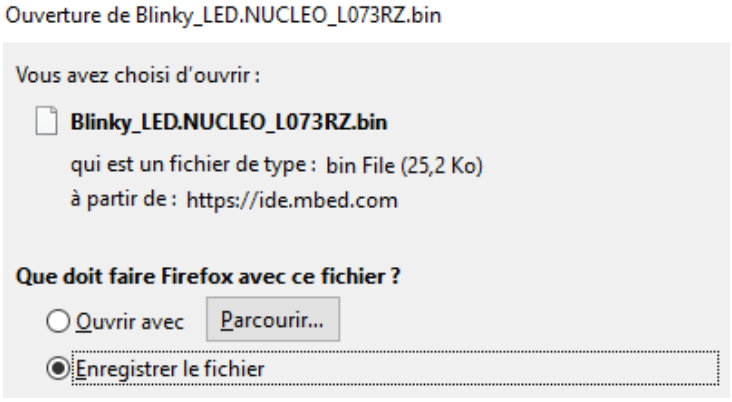
The only novelty here is the appearance of the notion of constructor from the C++ language (object-



oriented).

Then compile the program by clicking on "Compile".

When the job is finished, a Blinky_LED.NUCLEO_L073RZ.bin file is created. This file must now be transferred to the USB reader corresponding to the mbed card (copy-paste or drag and drop...).

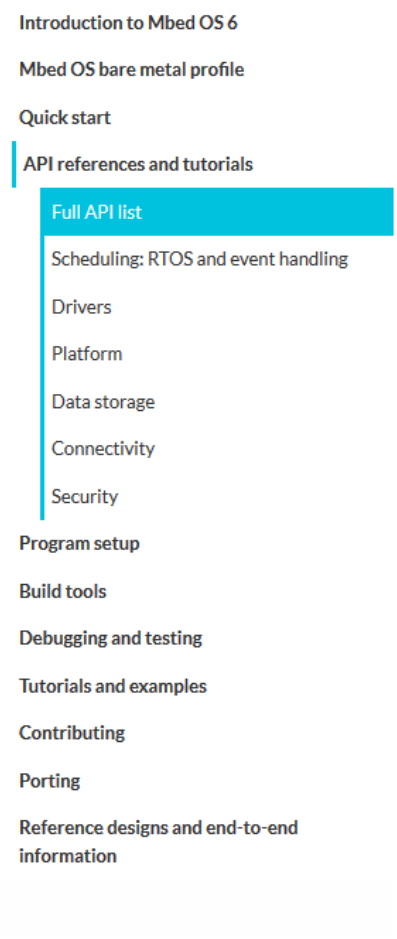


Help on the mbed system and its APIs

The mbed site is rich in resources.

The most complete access to this information can be done from the home screen of the site by clicking "Documentation Mbed→ OS".

You then have all the documentation relating to the "mbed operating system (OS)". It's all there, so this information will certainly be difficult to understand at first and its abundance may scare you. But don't hesitate to come and consult it to understand the mbed system.



SPISlave	✓	✓

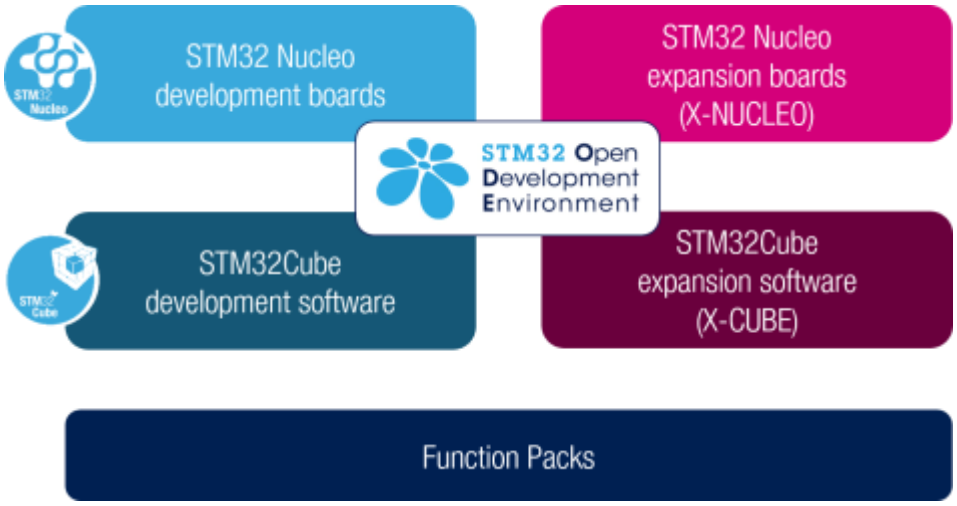
Input/Output drivers

API	Full profile	Bare metal profile
AnalogIn	✓	✓
AnalogOut	✓	✓
BusIn	✓	✓
BusOut	✓	✓
BusInOut	✓	✓
DigitalIn	✓	✓
DigitalOut	✓	✓
DigitalInOut	✓	✓
InterruptIn	✓	✓
PortIn	✓	✓
PortOut	✓	✓
PortInOut	✓	✓
PwmOut	✓	✓

The development environment for STM32

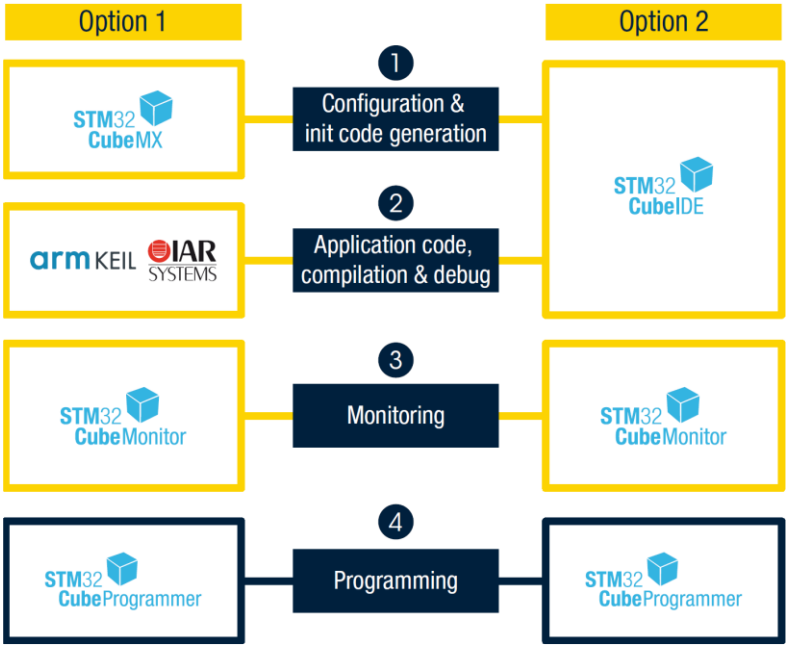
The development environment for STM32 is a flexible platform that facilitates and accelerates the coding of applications and systems designed around the STM32 microcontroller. It is called STM32 ODE (Open Development Environment).

The STM32 ODE combines the *STM32Cube* software environment, embedded libraries including Hardware Abstraction Layer (HAL) APIs, and STM32Cube extension software (*X-CUBE*). The STM32 ODE environment facilitates the design of embedded projects by providing easy-to-use hardware and open source function blocks, as summarised in the simplified synopsis below.



The STM32Cube is a royalty-free package that allows development on the STM32 family and consists of a high-level software platform. The STM32CubeMX graphical configurator makes it easy and fast to configure the STM32 for the target application.

The STM32Cube associated with the chosen STM32 includes a HAL (Hardware Abstraction Layer) for easy porting from one STM32 to another, as well as a set of software bricks or middleware, the main ones being : USB, TCP/IP, graphics, FAT file system, RTOS, STMTouch system. A BSP (Board Support Package) is also included in the STM32Cube to activate the different functions of the Nucleo board (such as push buttons, LEDs...). Finally, hundreds of sample programmes are also provided for the majority of the functions of the target STM32.



STM32CubeMX is a graphical tool for STM32 microcontrollers based on 32-bit ARM Cortex-M cores. This environment is part of the STMCube initiative and is available in standalone mode (standalone option 1) or integrated mode in the Eclipse STM32CubeIDE development environment (option 2).

The development tool STM32CubeIDE

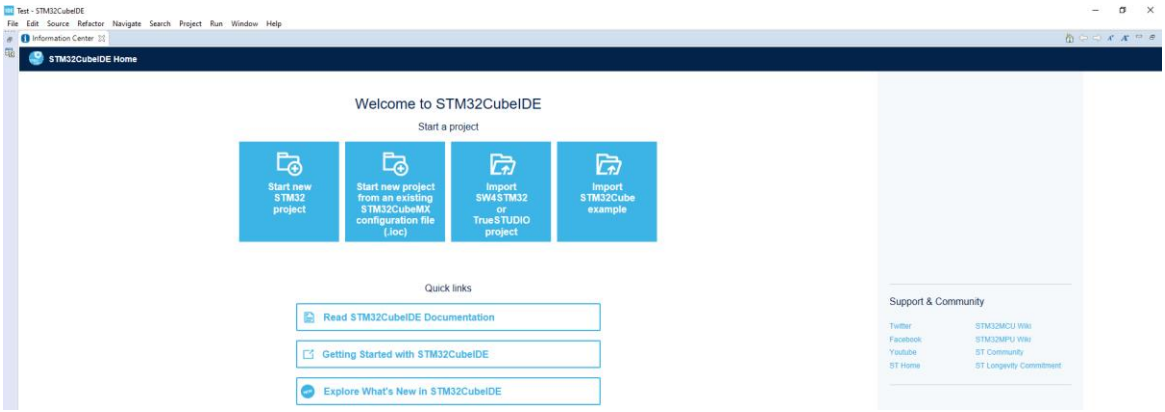
Available at this address
https://www.st.com/content/st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-ides/stm32cubeide.html

STM32CubeIDE is an all-in-one multi-OS development tool, part of the STM32Cube software ecosystem.

STM32CubeIDE is an advanced C/C++ development platform with device configuration, code generation, code compilation and debugging functions for STM32 microcontrollers and microprocessors.

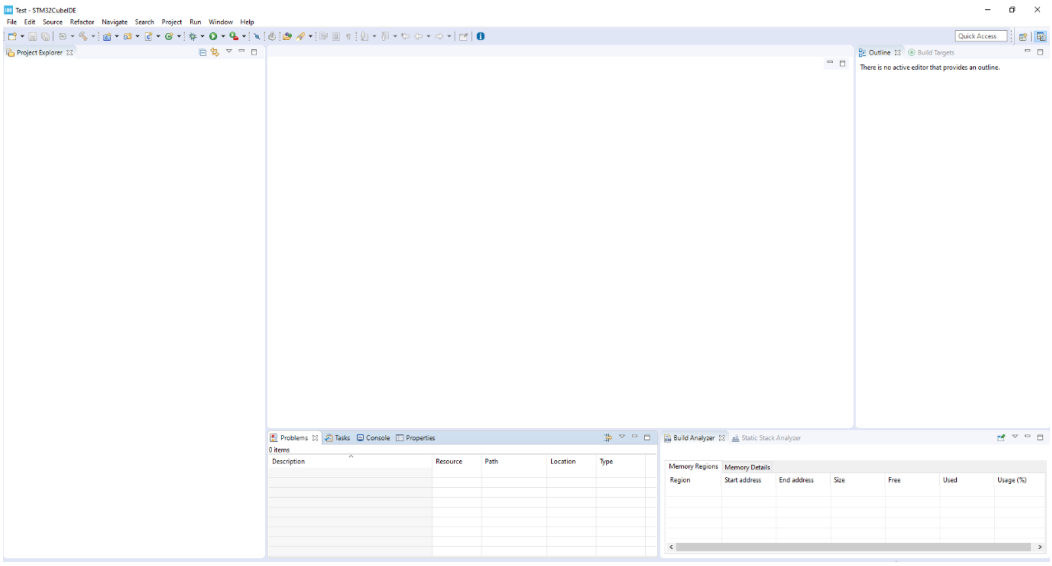
STM32CubeIDE integrates the configuration and project creation features of STM32 from STM32CubeMX to provide an all-in-one tool and save installation and development time.

After selecting an MCU or an unconfigured STM32 MPU, or a microcontroller or microprocessor from board selection or example selection, the project is created and the initialisation code generated. At any time during development, the user can return to the initialization and configuration of the peripherals or middleware and regenerate the initialization code without impacting the user code.



The figure above shows the window as it appears the first time you use it with an "Information Center" tab open.

The STM32CubeIDE software is based on the Eclipse environment, it inherits features that may confuse new users. Eclipse works with multiple perspectives (views). The first is the code editing perspective (see figure on next page). Another interesting perspective is that of debugging.

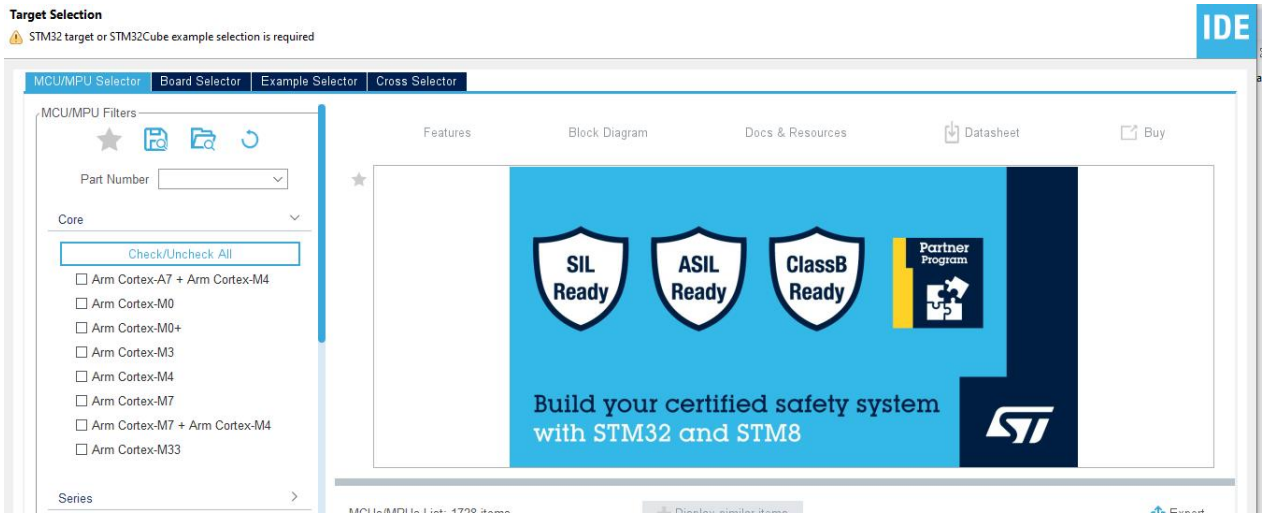


An important element to know is the notion of workspace and projects:

The first step is to create a project, which requires first introducing the concept of workspace. A workspace contains project folders or information about projects and a .metadata folder that contains information about projects.

A workspace is simply a folder on a hard drive, which can be located anywhere in the storage medium. When STM32CubeIDE starts, it asks which workspace is to be used. This can be changed at any time by selecting "File →Switch Workspace" and choosing another folder.

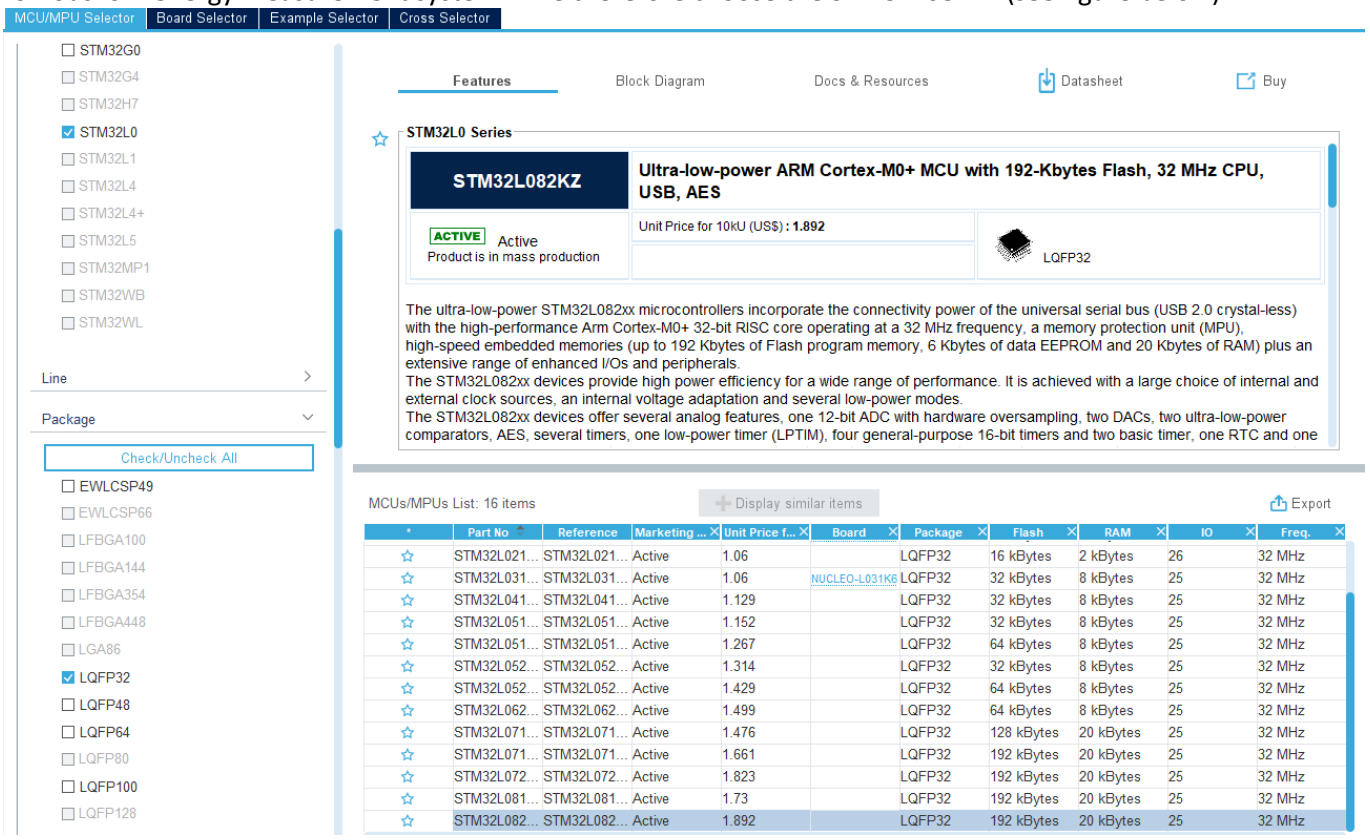
To create a new project, simply click on "File New →→STM32 Project". This opens the target selection window :



You can then choose either :

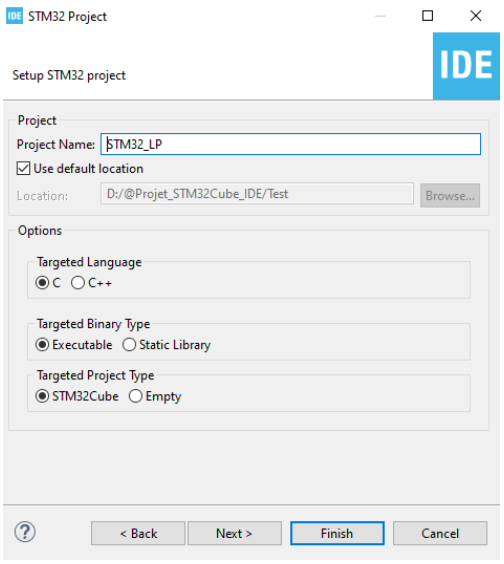
- a type of microcontroller ("MCU/MPU Selector" tab) in the case of custom board development is used if the target board has not been developed by STMicroelectronics. Here the user can select which of the full range of STM32s he is using. Filters are applicable to facilitate selection: STM32 series, package, peripherals, memory size, number of OIs. ,
- a board developed by STMicroelectronics ("Board Selector" tab). It allows you to filter according to the type of board (Discovery, Nucleo...), the STM32 series or the peripherals. The tool proposes for each board its default configuration. If alternative hardware configurations are required, it will be necessary to modify the initialisation C code generated by the tool.
- An example project ("Example Selector" tab). It allows you to filter according to the type of card (Discovery, Nucleo...), the STM32 series, the type of project, the software used...

In the rest of this presentation, we will use the MCU's choice of MCU for our application-specific card "Autonomous low-energy measurement system". We therefore choose the STM32L082KZ (see figure below).



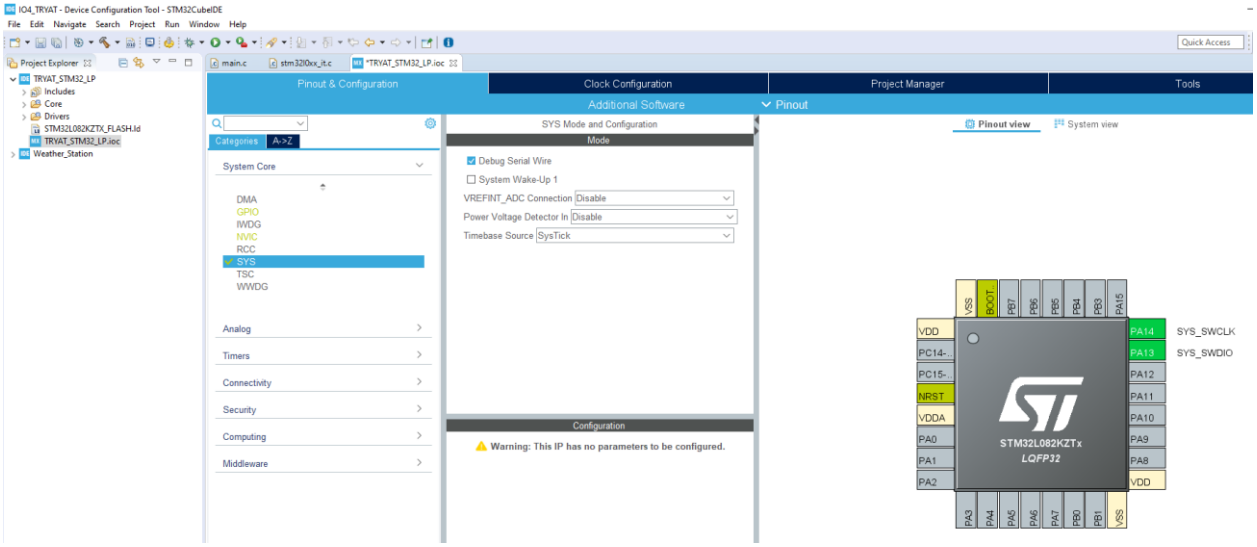
The project name, the language used, the type of output file and the type of project must then be filled in. In our example, just specify the name of the project and click on Finish.

The software requires a change of perspective that one accepts. The IDE then loads all the APIs necessary to develop our project and prepares the next operations, namely the choice of the different peripherals that we will use for our application.



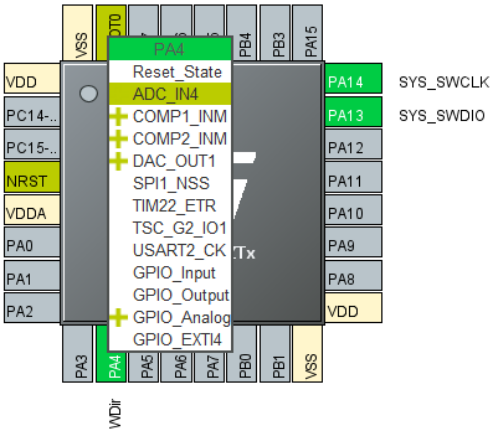
Pin validation for in-situ programming

For programming, in the " System Core " tab, select " SYS " then validate the " Debug Serial Wire " box (see below).

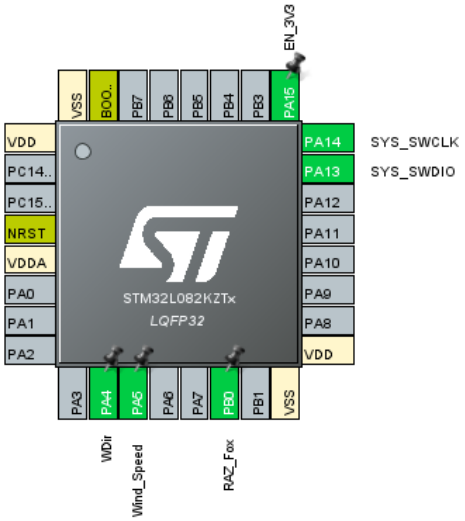


Inputs and outputs

Pin PA4 is an analogue input (wind direction measurement).➔ Click on this pin and choose ADC_IN4.



The PA5 pin is an interrupt input (wind speed). ➔Click on the PA5 pin and select GPIO_EXTI5 :



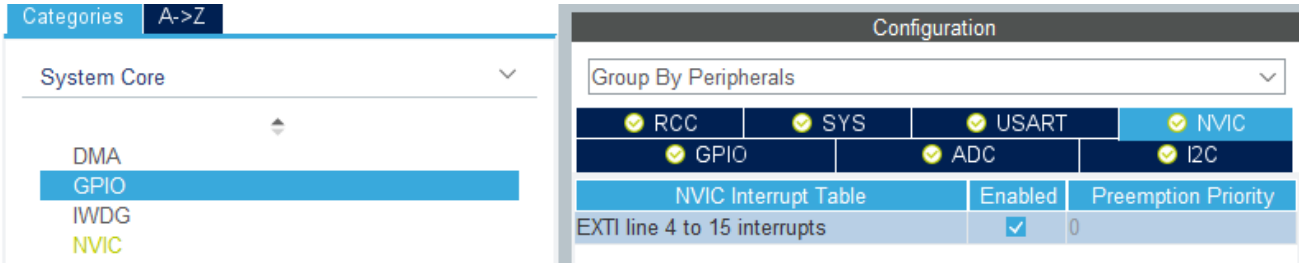
For a better readability, it is interesting to name the pins more explicitly :

- right click and "Enter User_Label" name the Wind_Speed for the PA5 spindle and WDir for the PA4 spindle.

Proceed in the same way for :

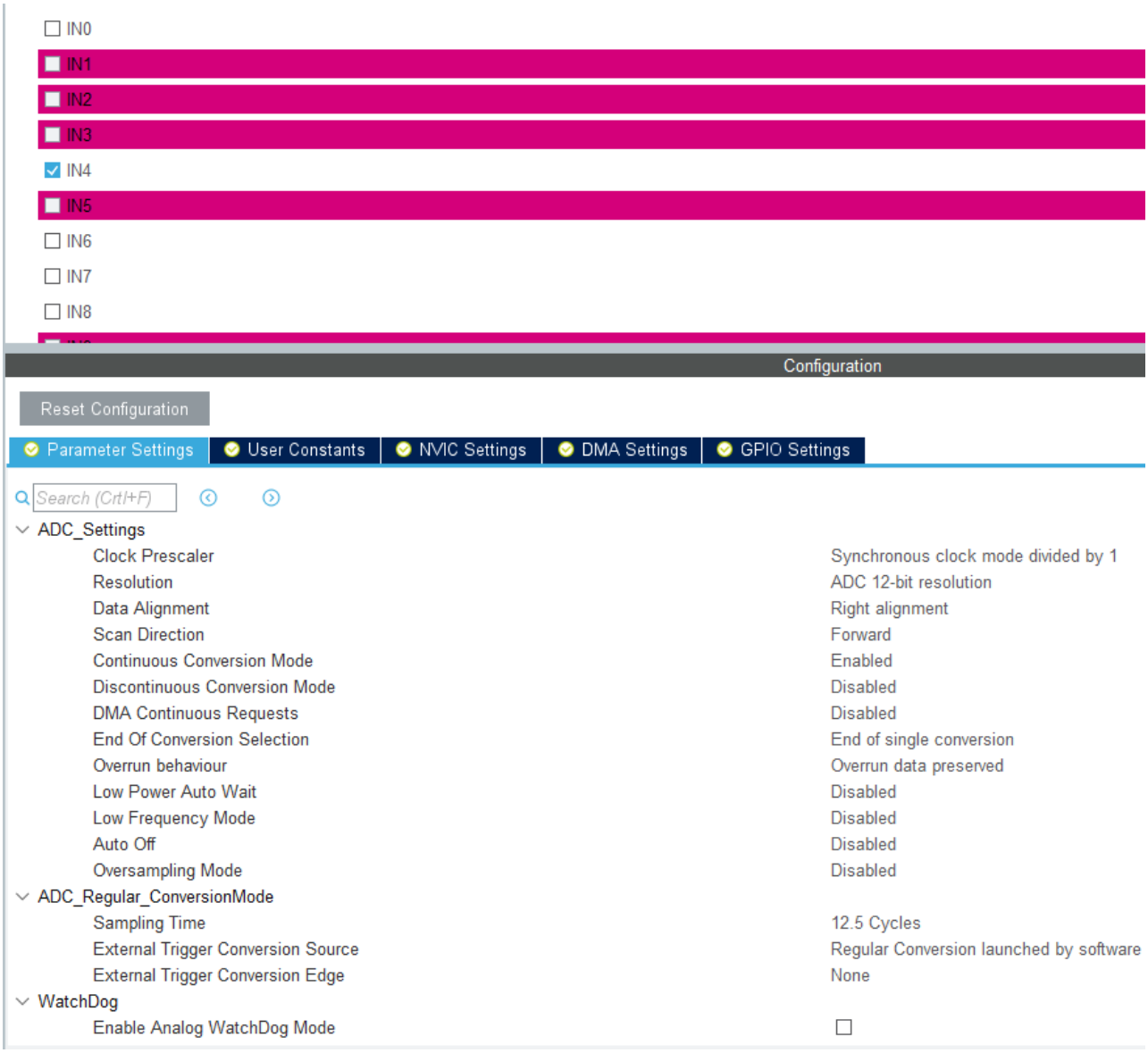
- PB0 which is an output named RAZ_Fox
- PA15 which is an output named EN_3V3

Do not forget to validate the interruptions for wind speed measurement see in the NVIC tab below :



Configuration of the Analogue Digital Converter (ADC)

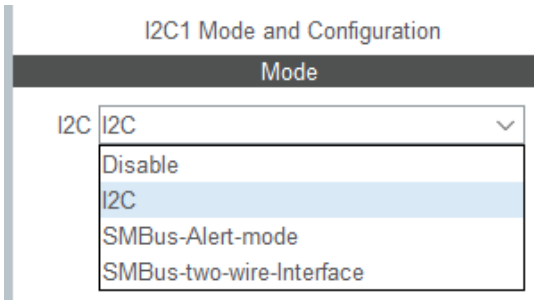
The continuous conversion mode must be enabled and a sampling rate of 12.5 cycles must be selected.

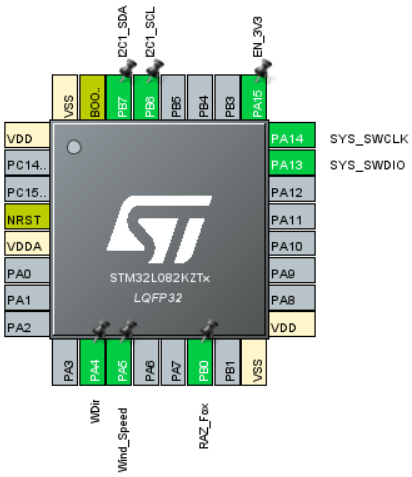


Communication bus

I2C bus

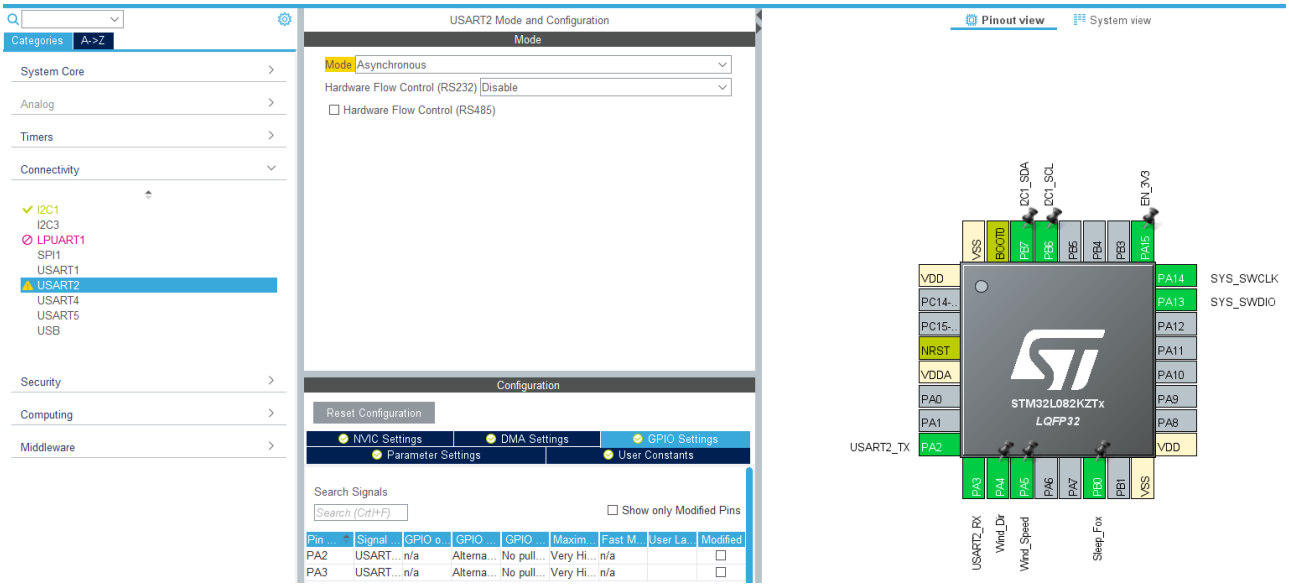
The I2C bus is used to communicate with all the sensors. It is wired on pins PB6 and PB7. Simply click on each pin and choose I2C1_SDA or I2C1_SCL .





Asynchronous serial link: USART

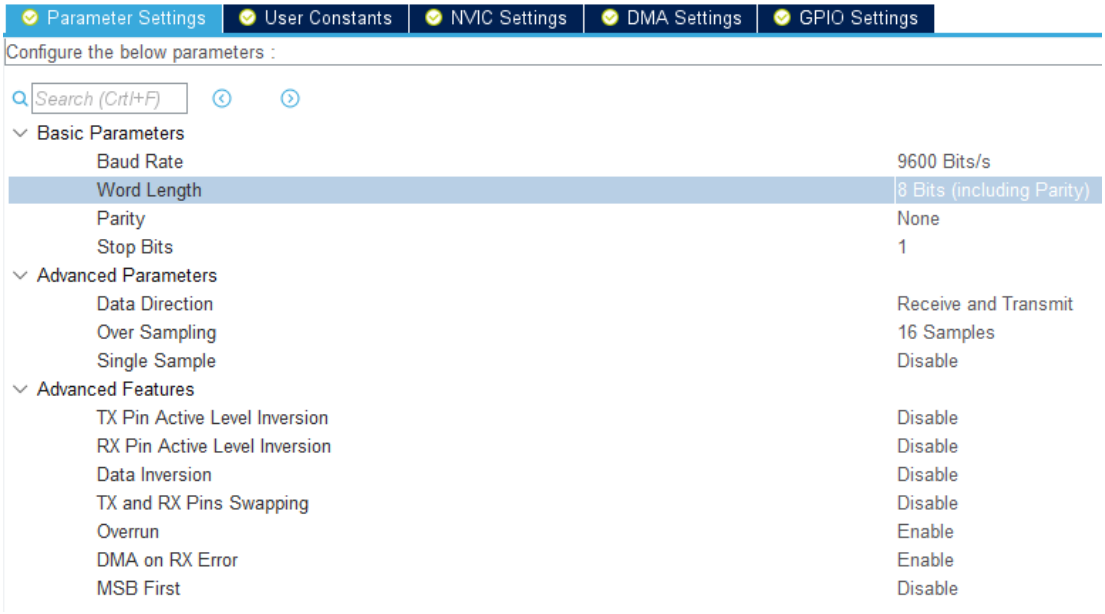
The Sigfox module dialogues with the microcontroller on the UART 2 connected to pins PA2 and PA3 (TX and RX respectively). Simply click on each of the pins and select USART2_TX or USART2_RX. Alternatively, select "Connectivity" on the left side of the window and then select USART2 and choose "Asynchronous" mode.



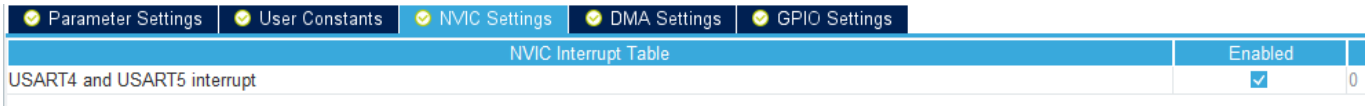
The procedure is identical for the connection between the GPS module and the microcontroller using the USART5 (PB3 - TX and PB4 - RX).

Configuration

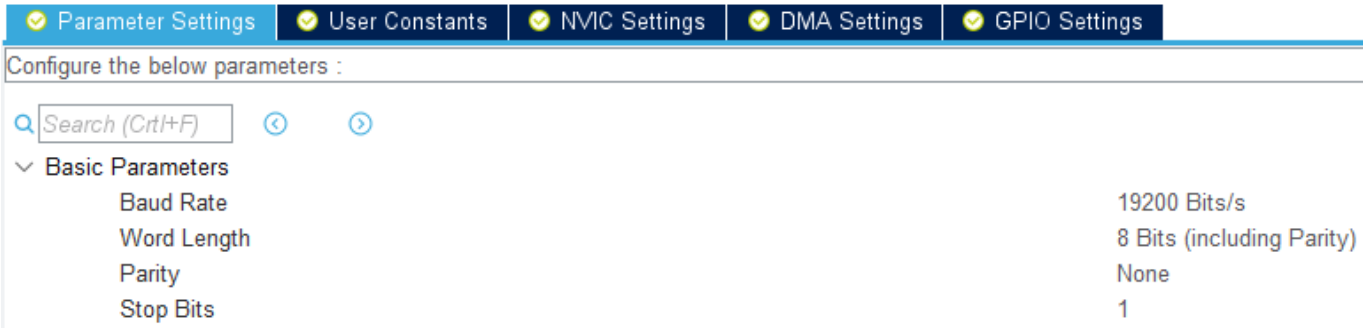
For the connection to the GPS, the bit rate must be set to 9600 bits/s ...



And validate the interruptions (in the NVIC tab).



For connection to the Sigfox module, the bit rate must be set to 19200 bits/s (no need to operate in interrupt mode for this connection).



Use of RTC

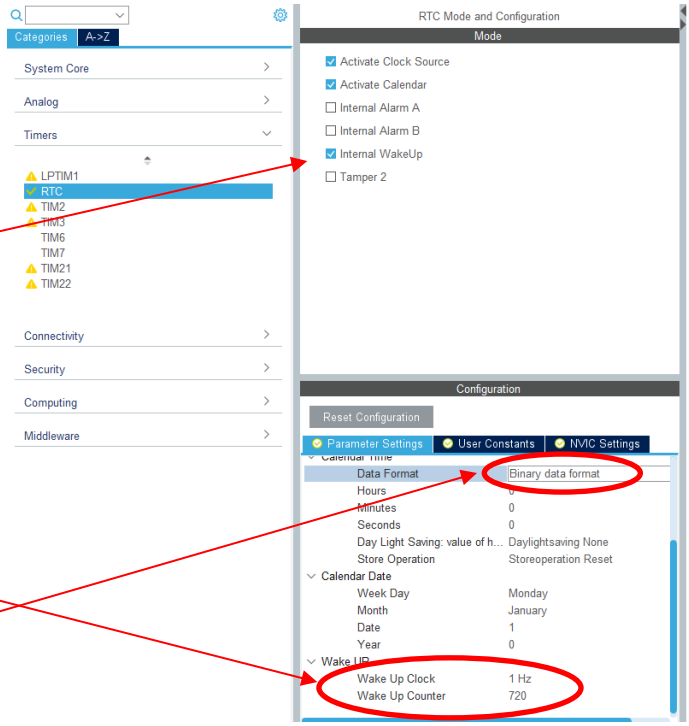
The real time clock is useful for obtaining the current time and date (after synchronisation via GPS), but above all, it allows us to set the sleep/wake periods in order to save as much energy as possible.

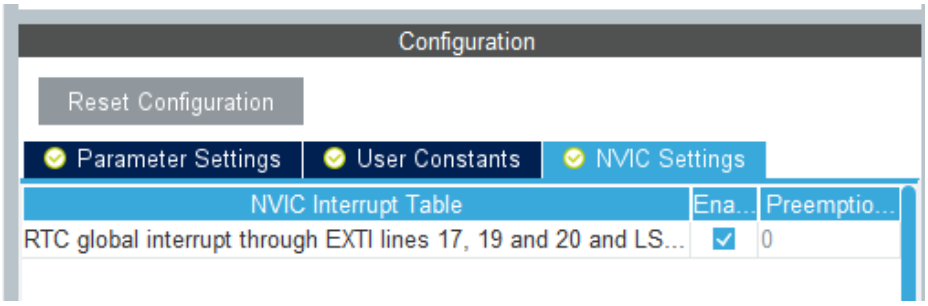
The clock, calendar and internal wakeup are activated to schedule an interrupt and wake up the system at regular intervals.

Of course, you must specify the length of sleep and therefore the times when the programme is resumed. In our application, we are constrained by the subscription to the Sigfox network to a maximum number of messages per day. In our case (subscription included when purchasing the module) the limitation is 140 messages per day (this is the subscription with the maximum number of messages), i.e. 1 message every 617 s. We have chosen to send messages only every 720 seconds.

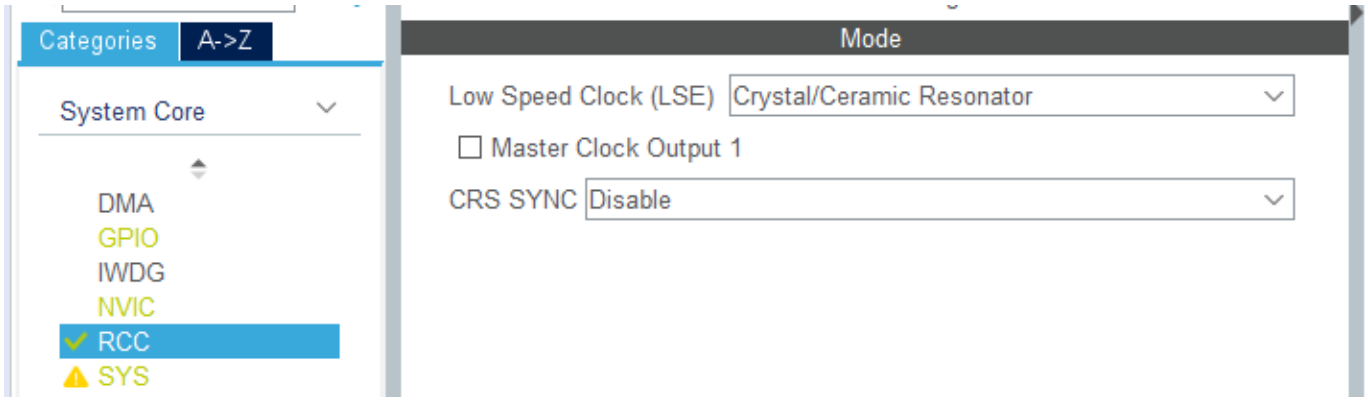
Also select "Binary data format" as the data format.

Validate the use of interruptions to be able to wake up the microcontroller after it has been put to sleep:

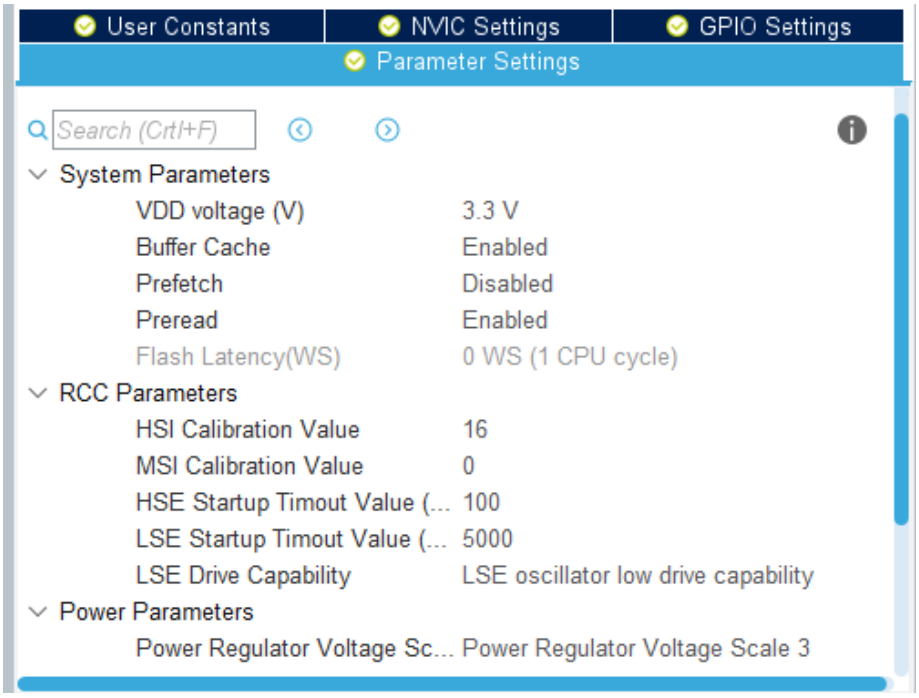




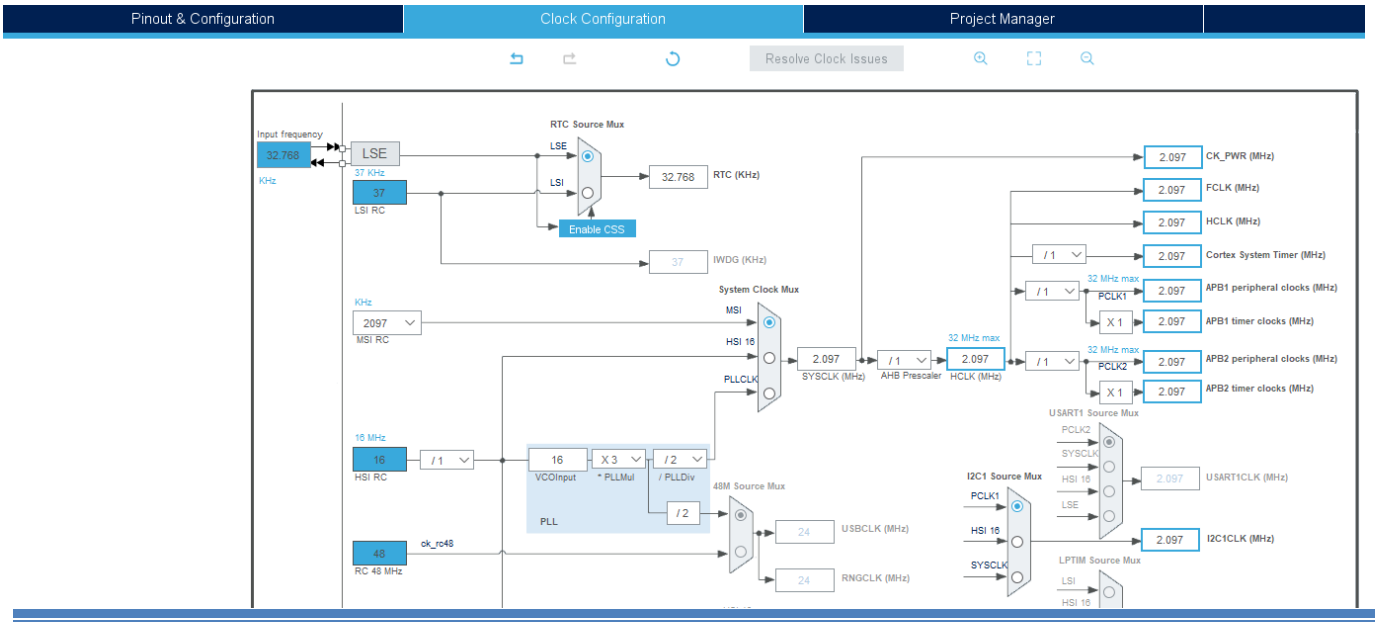
You must then validate the use of the watchmaker's quartz (32,768 kHz) present on the map:



Always at the RCC component (Reset and Clock Configuration). In the ***Parameter Settings*** tab (figure below), select Power Regulator Voltage Scale 3. This is the most energy efficient mode.

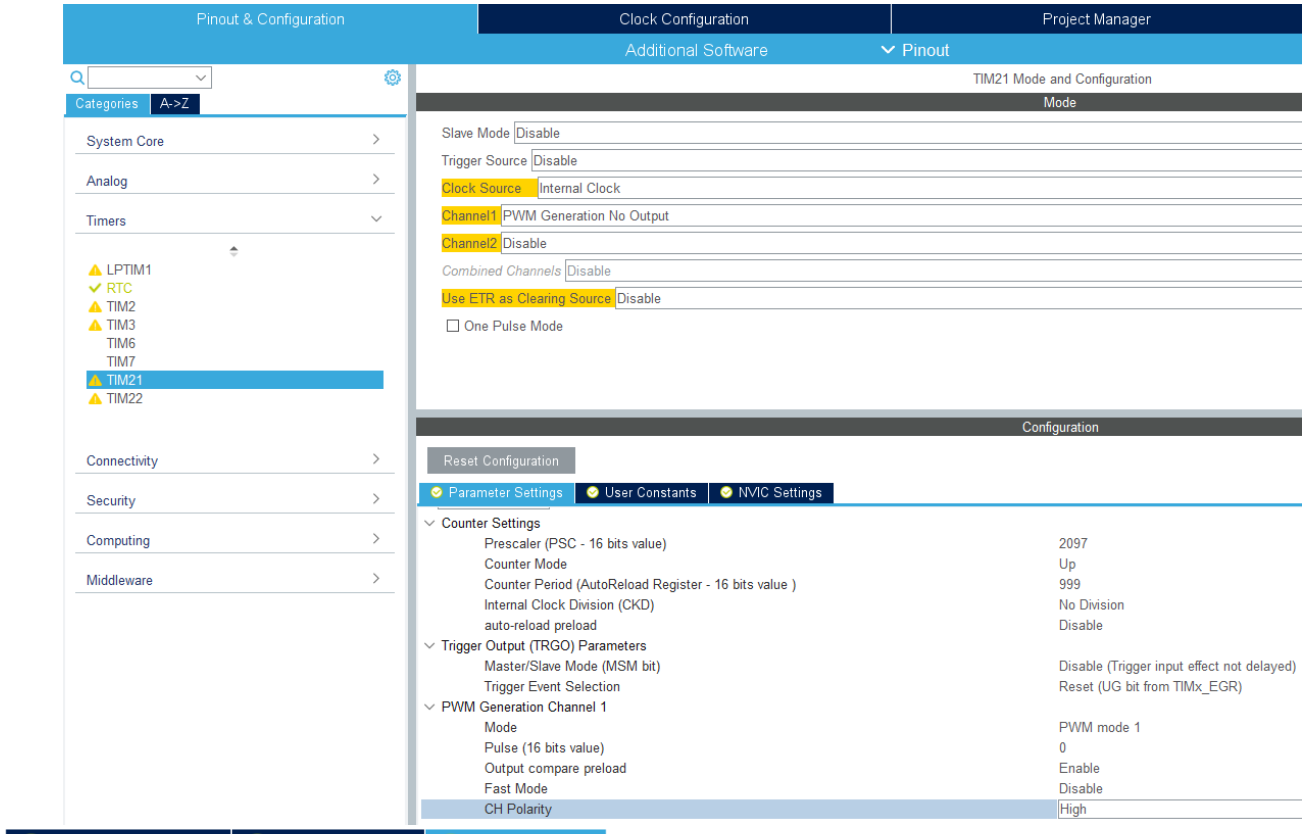


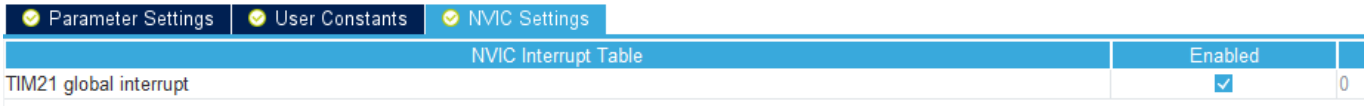
Do not forget to validate this also in the clock management:



Using the timer

We will use timer 21 to generate interruptions every second. As the timer clock is at 2.097 MHz, dividing by 2097 will return to a rate of one millisecond (1kHz) and then counting up to 999 will result in one event every second.

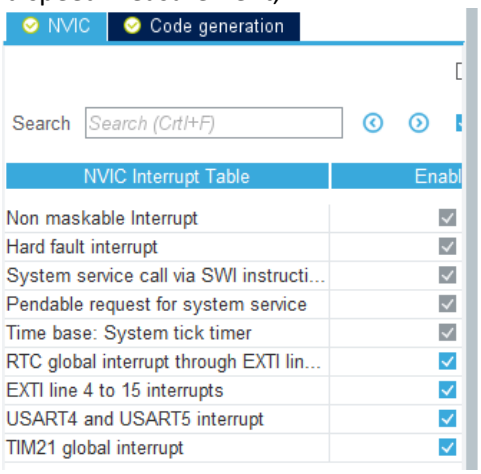


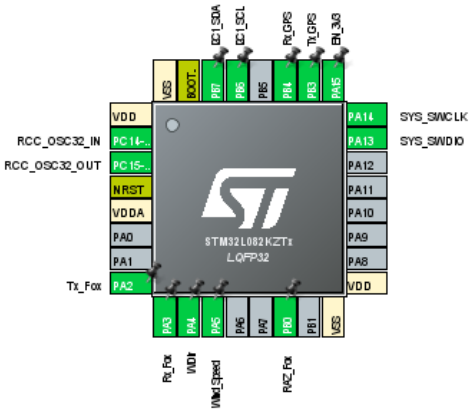


Interruptions

Verification of the use (validation) of interruptions useful to the project. Select the NVIC component in the "System Core" category and check that the following interrupts are validated:

- interruption for RTC (periodic wake-up of the microcontroller),
- interruption for wind speed measurement,
- periodic interruption every second (timer 21),
- interruption for the asynchronous link with the GPS,





Code generation

All we have to do is save our work, the software asks us if we want to generate the code and then signals a change of perspective, answer yes to both proposals.

When generating the initialization C code, STM32CubeMX created the main.c file in which the main function of the int main(void) program is located.

Inside this hand, some other indispensable functions are called :

- HAL_Init() initializes the flash memory .
- SystemClock_Config() initialises all the clocks of the STM32L082RZ according to what has been set in the ***Clock Configuration*** tab of STM32CubeMX.
- Then the initialization of each of the peripherals as we configured them in the previous step.

STM32_LP.ioc | *main.c | main.h

```
80 int main(void)
81 {
82   /* USER CODE BEGIN 1 */
83
84   /* USER CODE END 1 */
85
86   /* MCU Configuration--------------------------------------------------------*/
87
88   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
89   HAL_Init();
90
91   /* USER CODE BEGIN Init */
92
93   /* USER CODE END Init */
94
95   /* Configure the system clock */
96   SystemClock_Config();
97
98   /* USER CODE BEGIN SysInit */
99
100  /* USER CODE END SysInit */
101
102  /* Initialize all configured peripherals */
103  MX_GPIO_Init();
104  MX_ADC_Init();
105  MX_I2C1_Init();
106  MX_RTC_Init();
107  MX_TIM21_Init();
108  MX_USART2_UART_Init();
109  MX_USART5_UART_Init();
110  /* USER CODE BEGIN 2 */
111
112  /* USER CODE END 2 */
113
114  /* Infinite loop */
115  /* USER CODE BEGIN WHILE */
116  while (1)
117  {
118    /* USER CODE END WHILE */
119
120    /* USER CODE BEGIN 3 */
121  }
122  /* USER CODE END 3 */
```

Anything between /* and */ is a comment in C language and is therefore not executed. Some types of comments are special:

/* USER CODE BEGIN */

…

/* USER CODE END */ possibly with numbers or names after BEGIN and END.

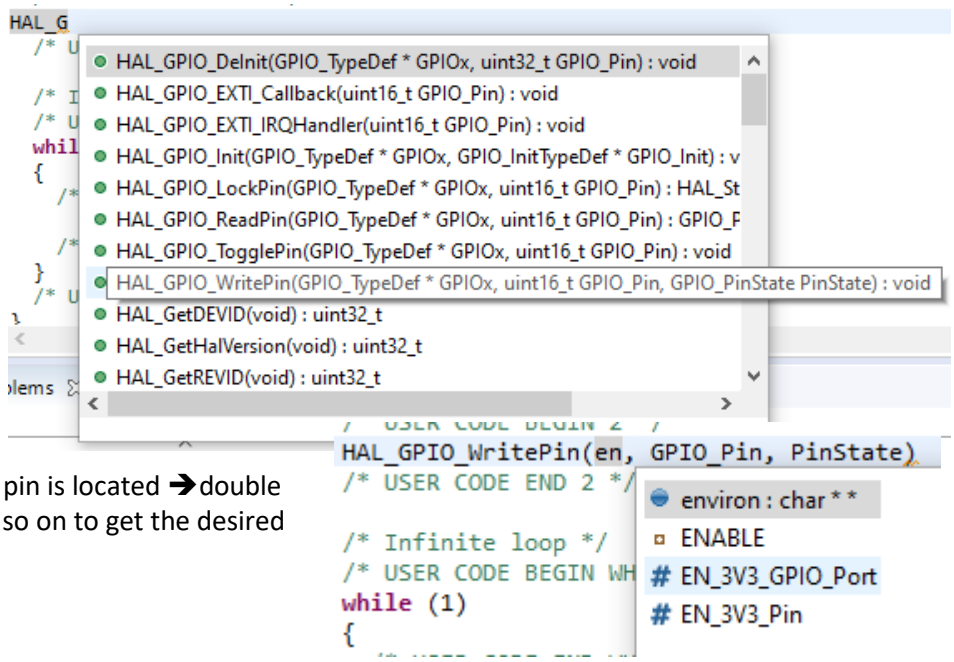
This type of comment has a special meaning for STM32CubeMX, which interprets them as tags (labels or beacons). It is advisable not to **delete them**, but rather to **insert our code inside these tags** because, if we have to generate the initialisation C code again with STM32CubeMX after adding a device for example, the code placed between BEGIN and END will be kept. On the other hand, if our code has been added outside these tags, it will simply be deleted.

The generated hand function contains the instruction while (1) followed by an opening and a closing brace. It is inside these two brackets, and if possible inside BEGIN and END tags, that we will insert our code which will be executed in a loop.

For example, at the beginning of the programme between the /* USER CODE BEGIN 2*/ and /* USER CODE END 2 */ tags, insert the code to set the bit of the port that controls the operation of the sensors to 1 (pin that we have named EN_3V3).

To do this, first type hal_g and then press the ctrl and space keys at the same time to open suggestions for operation by the software. We choose `HAL_GPIO_WritePin(GPIOx, GPIO_Pin, PinState);`

This function needs to be complemented with the right arguments. Here again the suggestions of the software can help us start typing the name of the concerned pin here just in then "Ctrl + space" the first element being the port where the pin is located ➔double click on EN_3V3_GPIO_Port... and so on to get the desired code :



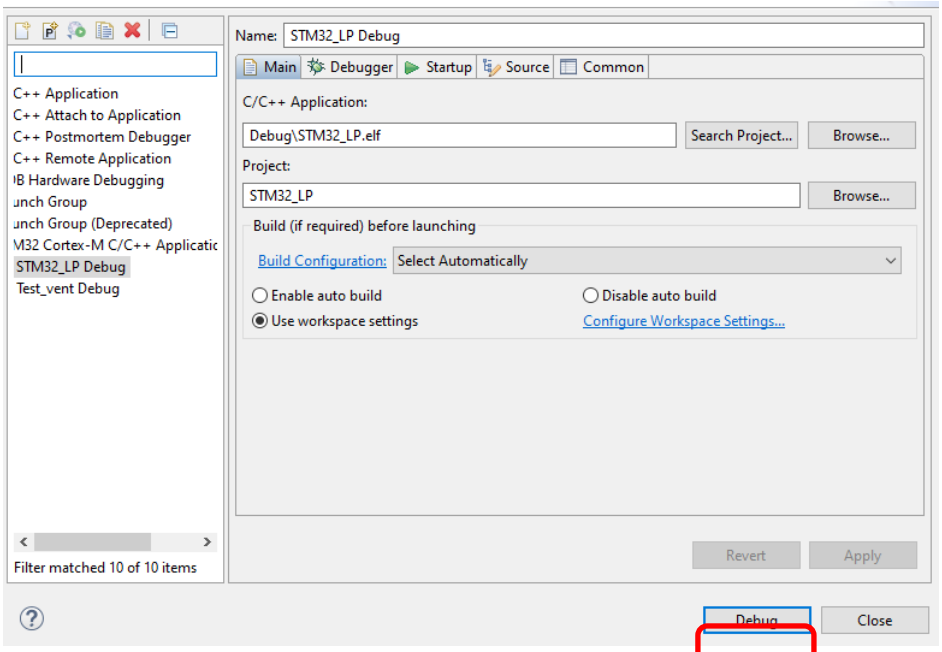

```
HAL_GPIO_WritePin(EN_3V3_GPIO_Port, EN_3V3_Pin, GPIO_PIN_SET);
```

Code execution and debugging

Tools for analysing and debugging embedded systems fall into two main categories. The first is in the hardware area: tools such as oscilloscopes, logic analysers, data sheets. All of these tools can work together to help you evaluate your systems and solve problems related to your designs. These tools are frequently used.

The second category of tools focuses on software debugging, with breakpoints, variable visualisations and registers internal to the microcontroller, etc. All these tools work together to help you diagnose and understand what your embedded application is doing.

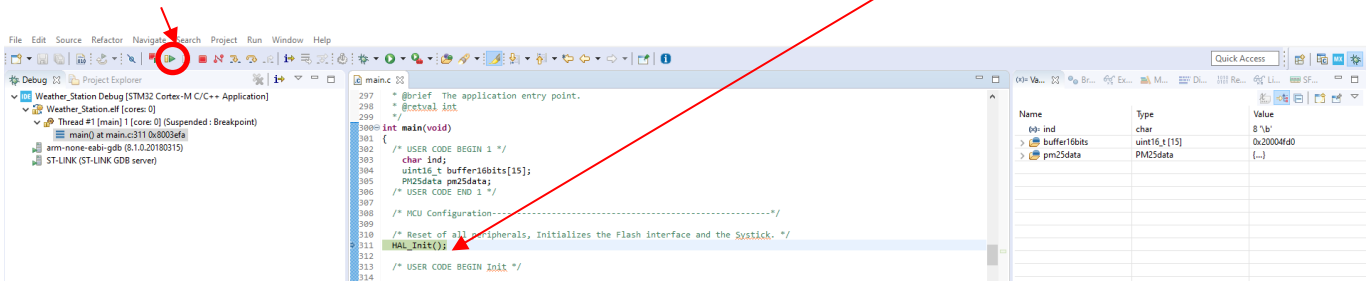
Register your code. Click Project > Build Project. Your code must be compiled and linked to the appropriate libraries. When this is done (and you see a message indicating 0 errors in the console panel at the bottom), click Run > Debug As > STM32 MCU C/C++ Application. It is also possible to click on the icon 



You should get a pop-up window asking you to set the debugging configurations. Leave everything as default and click on DEBUG.

When you are asked about changing perspective, click on "Switch". You should get a new perspective with a new toolbar at the top of your IDE.

The program has been transferred to the target, and the debugging probe has taken control of the program flow IDE highlights the function that is being executed here the first instruction of the program To start the code execution you must click on the Click the button "Resume".



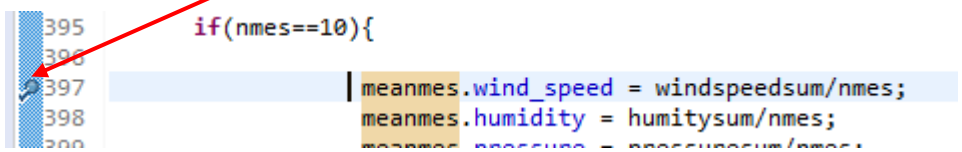
It is possible to suspend the execution of the programme by clicking here.

Or stop it by clicking here

Break point

In addition to performing and suspending performance, we may also request the programme to suspend at a time of our choosing. This is called the creation of a stopping point.

In STM32CubeIDE, you do this by double-clicking on the blue bar next to the line numbers, which causes a small blue dot to appear indicating a breakpoint.

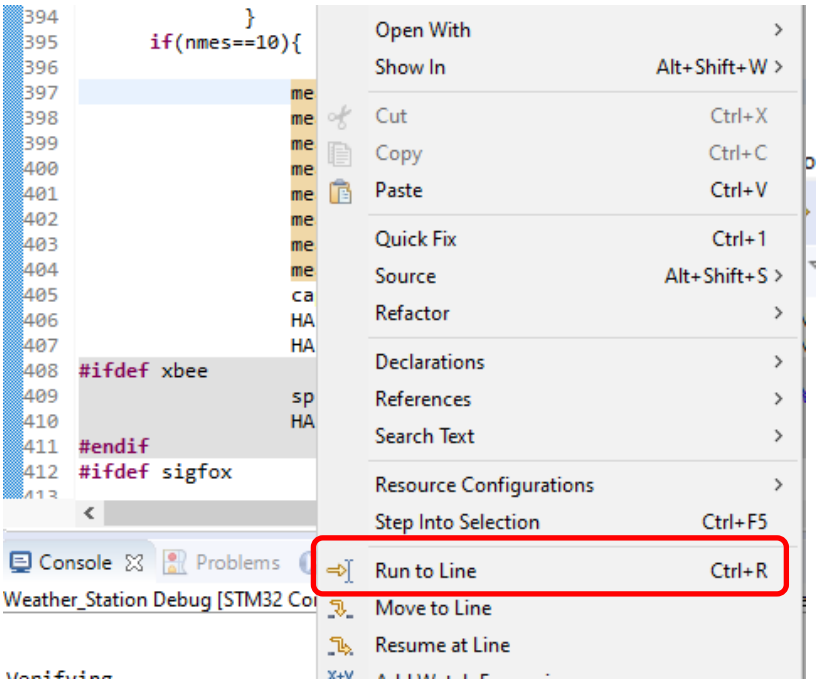


Now, without changing anything else, restart the debug mode.

This time, when you press "resume", you will notice that the programme runs and then stops automatically when it reaches your stop point.

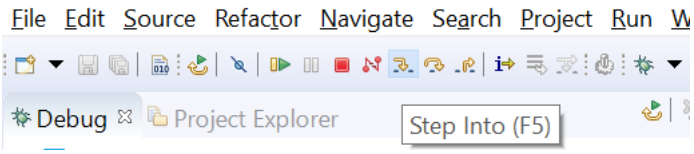
Another method of executing the code up to an instruction (a line) of the code is to position the cursor at the line to be reached and right click.

Then click on "Run to Line".

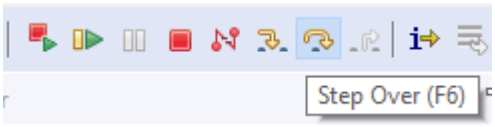


You can then use the "Step Into", "Step Over" and "Step Return" buttons (to the right of the "Stop" button).

To move from one line of code to another, step by step.



- "Step Into": when you are positioned on a function call, a click on this icon allows you to enter the definition of this function to execute lines of code one at a time. If you are not on a function call, execute the line of code.
- "Step Over": a click on this icon allows you to execute the line of code, even if it is a function call.



- "Step Return": If you are currently in the definition of a function, clicking on this icon will execute the rest of the code in that function and return to the next instruction in the main program.

When executing step-by-step instructions, it is possible to view the variables. To add a variable whose evolution you want to follow, you can right-click on the variable for example "meanmes" and select "Add Watch Expression...".

If the variable is changed when executing an instruction, the changed variable is highlighted.

Expression	Type	Value
meanmes	mesurescapteurs	{...}
(x)= temperature1	float	24.1353607
(x)= humidity	float	61.1755371
(x)= pressure	float	98392.25
(x)= wind_dir	uint8_t	0 '\0'
(x)= wind_speed	float	0
(x)= part10	uint16_t	939
(x)= part25	uint16_t	499
(x)= part10env	uint16_t	1356
(x)= part25env	uint16_t	4029