

SEMINAR ON STM32 MICROCONTROLLERS

Dott. Daniel Rossi

daniel.rossi@unimore.it

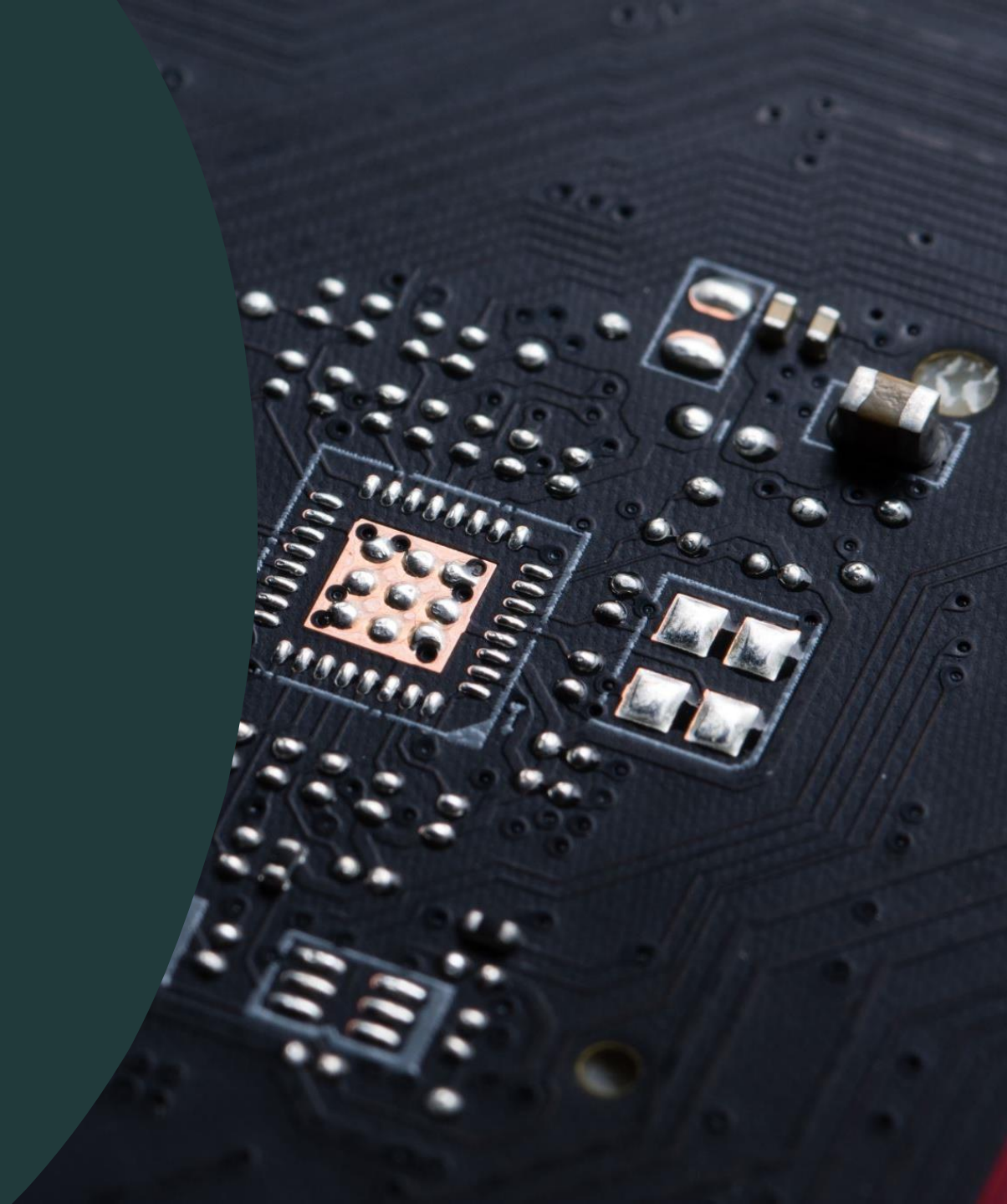


UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

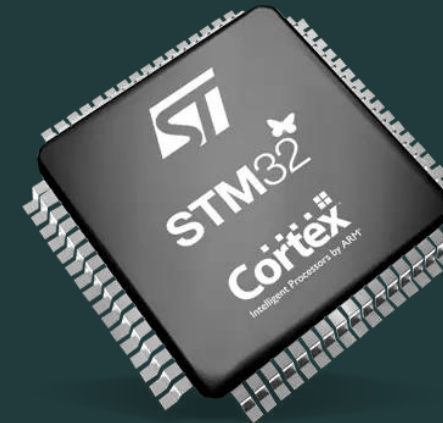
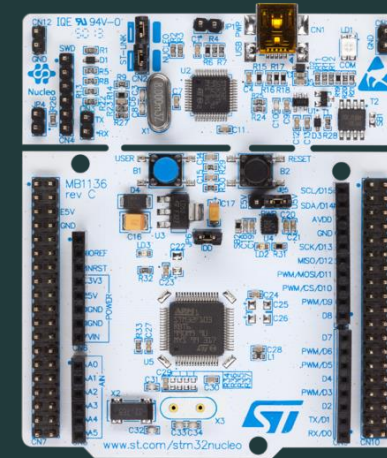
ST MICROELECTRONICS

STM32 & ST Nucleo



WHAT ARE STM32S AND ST CORES?

- STM32s are a family of microcontrollers manufactured by ST Microelectronics.
- They are based on the ARM Cortex-M architecture
- ST Nucleos are DEVELOPMENT boards developed and supplied by ST
- Typically a circuit developed with ST Nucleo is designed and programmed, then the prototype PCB design is created



WHERE ARE STM32S BEING USED?

Typically, STM32s are used in industry. Some examples of industries may be:

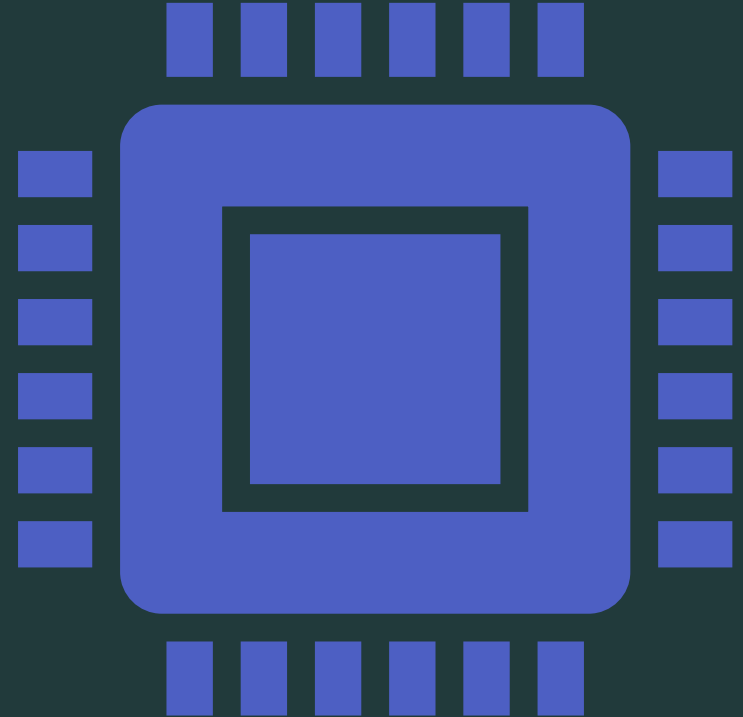
- Industrial Automation
- Consumer Electronics
- Automotive
- Medical Industry
- Energy and Energy Management
- Aerospace and Defense
- Power Electronics
- Instrumentation and Measurement



SUPPORTED PROTOCOLS

STM32 microcontrollers support:

- I2C
- SPI
- CAN/CAN-FD
- UART
- LIN
- MODBUS
- RS485
- SDIO
- ecc ...



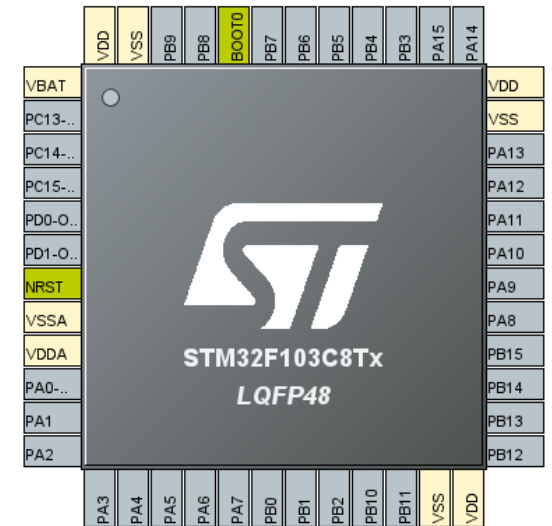
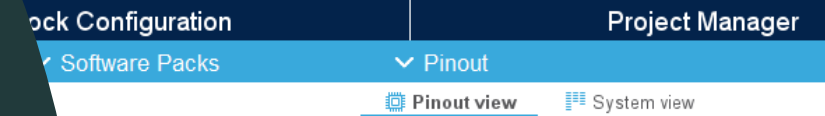
WHY DON'T WE USE STM32 INSTEAD OF ARDUINO?

The development environment is designed to meet the needs of the industry.

Some problems for inexperienced students may be:

- Knowing how to consult the datasheets of a microcontroller
- Defining the layout of the pins to be used
- Working at 3.3V, and not 5V
- Low-level programming of the tools provided by the micro

It is not an easy environment, so Arduino is an excellent starting point for building the experience needed to understand the ST ecosystem.





HOW DOES ST MICROELECTRONICS' ECOSYSTEM WORK?



NOMENCLATURE (1)

ST manufactures microcontrollers for a variety of industries.

An example of a microcontroller identifier might be:

"STM32F103C8T6"

STM32 MCUs

32-bit Arm® Cortex®-M

High Performance

STM32F7

1082 CoreMark
216 MHz Cortex-M7

STM32H7

Up to 3224 CoreMark
Up to 550 MHz Cortex-M7
240 MHz Cortex-M4

STM32F2

398 CoreMark
120 MHz Cortex-M3

STM32F4

608 CoreMark
180 MHz Cortex-M4

STM32H5

Up to 1023 CoreMark
250 MHz Cortex-M33

Mainstream

STM32G0

142 CoreMark
64 MHz Cortex-M0+

STM32G4

569 CoreMark
170 MHz Cortex-M4

STM32C0

114 CoreMark
48 MHz Cortex-M0+

STM32F0

106 CoreMark
48 MHz Cortex-M0

STM32F1

177 CoreMark
72 MHz Cortex-M3

STM32F3

245 CoreMark
72 MHz Cortex-M4

Optimized for mixed-signal applications

Ultra-low-power

STM32L4+

409 CoreMark
120 MHz Cortex-M4

STM32U5

651 CoreMark
160 MHz Cortex-M33

STM32L0

75 CoreMark
32 MHz Cortex-M0+

STM32L4

273 CoreMark
80 MHz Cortex-M4

STM32L5

443 CoreMark
110 MHz Cortex-M33

Wireless

STM32WL

162 CoreMark
48 MHz Cortex-M4
48 MHz Cortex-M0+

STM32WB0

64 MHz Cortex-M0+

STM32WB

216 CoreMark
64 MHz Cortex-M4
32 MHz Cortex-M0+

STM32WBA

407 CoreMark
100 MHz Cortex-M33

Cortex-M0+ Radio co-processor

NOMENCLATURE (2)

The identification code of the microcontroller gives us so much information regarding what it can offer us

"ST | M | 32 | F | 103 | C8 | T6"

ST: indicates the **manufacturer**

M: indicates that the microcontroller is an **ARM Cortex M**

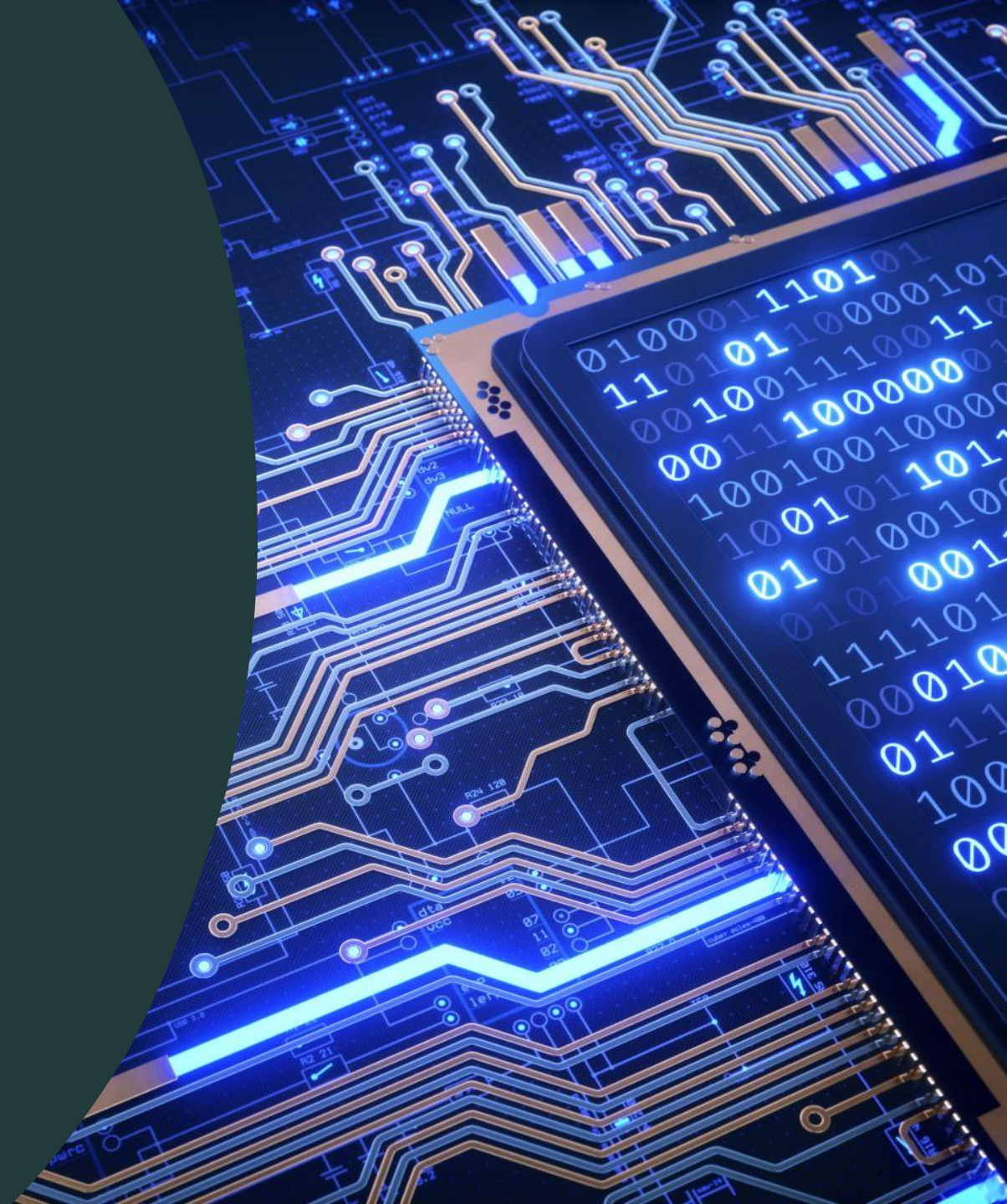
32: indicates that the microcontroller works in **32 bits**

NOMENCLATURE (3)

«ST | M | 32 | F | 103 | C8 | T6»

The classes:

- **F:** Full-Featured
- **L:** Low Power
- **H:** High Performance
- **G:** Motor control and power drives
- **W:** Wireless Microcontrollers
- **MP1:** Embedded Linux Applications
- **U:** Ultra-Low Power
- **B:** Security and Encryption
- **X:** Specific Application Microcontrollers (encryption, security, ecc...)
- **C:** Automotive



NOMENCLATURE (4)

«ST | M | 32 | F | 103 | C8 | T6»

The remaining information in the microcontroller code indicates:

- XXX (103): identifier of the microcontroller in that family (F in this case)
- XX (C8): additional identifier to distinguish microcontrollers of the same type but with different specifications (e.g., amount of memory, clock speed, etc...)
- XX (T6): package or enclosure type of the microcontroller (this is useful information for PCB design)

ST's microcontroller selector can be found [here](#).

DEVELOPMENT ENVIRONMENT

Different development environments can be used; we will use ST's:

- STM32CUBEIDE: <https://www.st.com/en/development-tools/stm32cubeide.html>

Others could be:

- Keil μ Vision: <https://www.keil.com/download/>
- Atollic TRUESTUDIO: <https://www.st.com/en/development-tools/truestudio.html>
- IAR Embedded Workbench: <https://www.iar.com/ewarm>

Often with third-party IDEs you also need to install:

- STM32CUBEMX: <https://www.st.com/en/development-tools/stm32cubemx.html>

CASE EXAMPLE: STM32F103C8T6

Alias “BluePill” used within the “MOVE
MOTION” PlayStation 3 controller

Available [here](#) with [STLink](#)

筐体の裏側



- ①○×△□ボタンや「Moveボタン」部分(点線で囲んだ部分にボタン・シートがかぶさる)
- ②「PSボタン」部分
- ③3軸の加速度センサ(Kionix社製,「KXSC4103492910」の刻印)
- ④xy軸ジャイロ・センサ(ソニー「O81P8」の刻印)
- ⑤z軸ジャイロ・センサ(「Y5250H2033MPFGZ」の刻印)
- ⑥3軸の地磁気センサ(旭化成エレクトロニクス製,「AKM8974028B」の刻印)
- ⑦Bluetoothモジュール(アルプ電気製,「701A09AALPS」の刻印)
- ⑧水晶発振子
- ⑨Bluetooth送信IC(CSR製,「BC4REA16U027CD」の刻印)
- ⑩LED用フラット・ケーブルとの接続部分
- ⑪32ビット・マイコン(STmicroelectronics社製,「STM32F103」などの刻印)

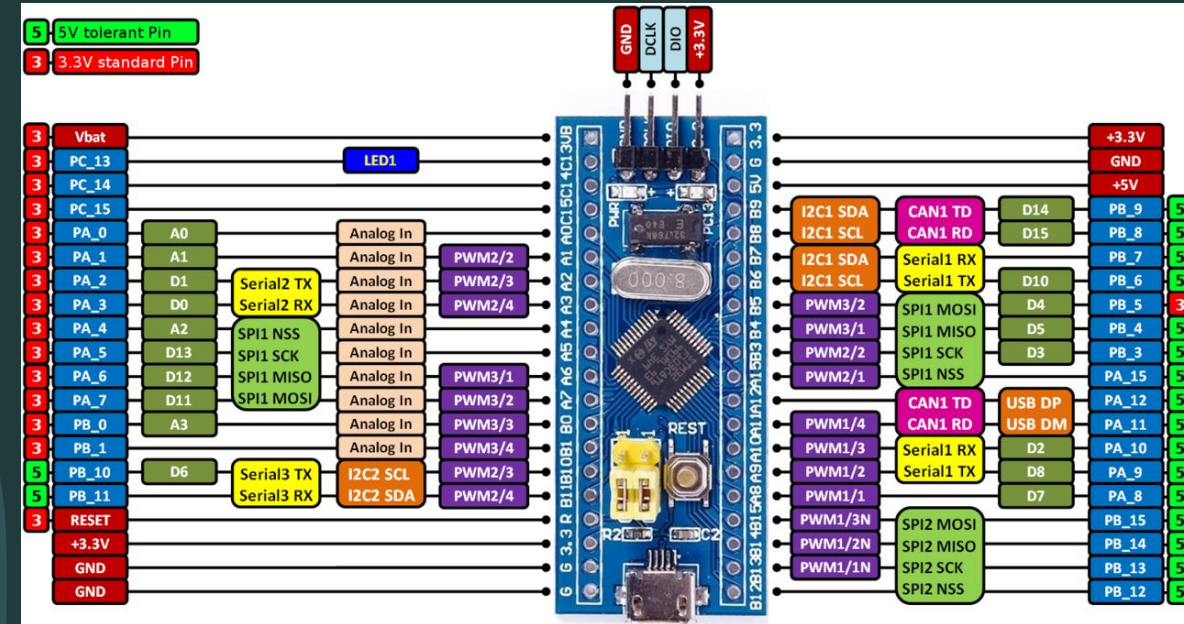
STM32F103C8T6 PINOUT

Each board has its own PinOut, and each peripheral can have outputs on different pins

!Tip: search on google

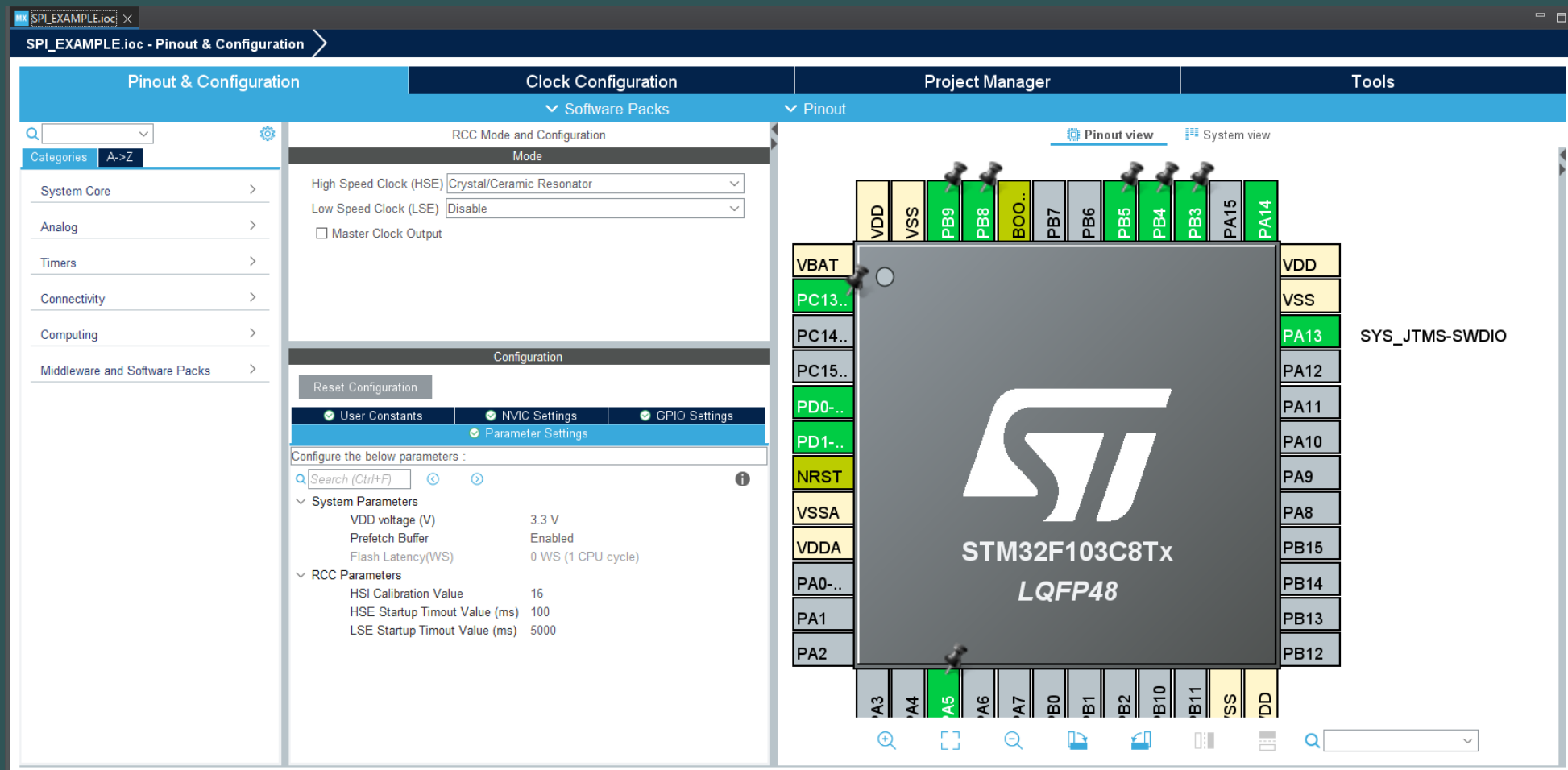
"board_name PinOut mbed"

At [MBED](https://www.mbed.org/) website you can find the pinout of different classical STM32 boards (such as [STM32F401RE](https://www.st.com/en/microcontrollers/stm32f401re.html))



STM32 CUBE IDE: IOC

- Here you can configure GPIO, Communication ports, Timers, RCC and so on.
- By saving this configuration (ctrl+S), the code will be generated



STM32 CLOCK CONFIGURATION

- You must tweak manually the clock sources and prescalers
- A simpler way to set the clock is to set the preferred frequency in HCLK and then manually tweak the APBx prescalers if needed

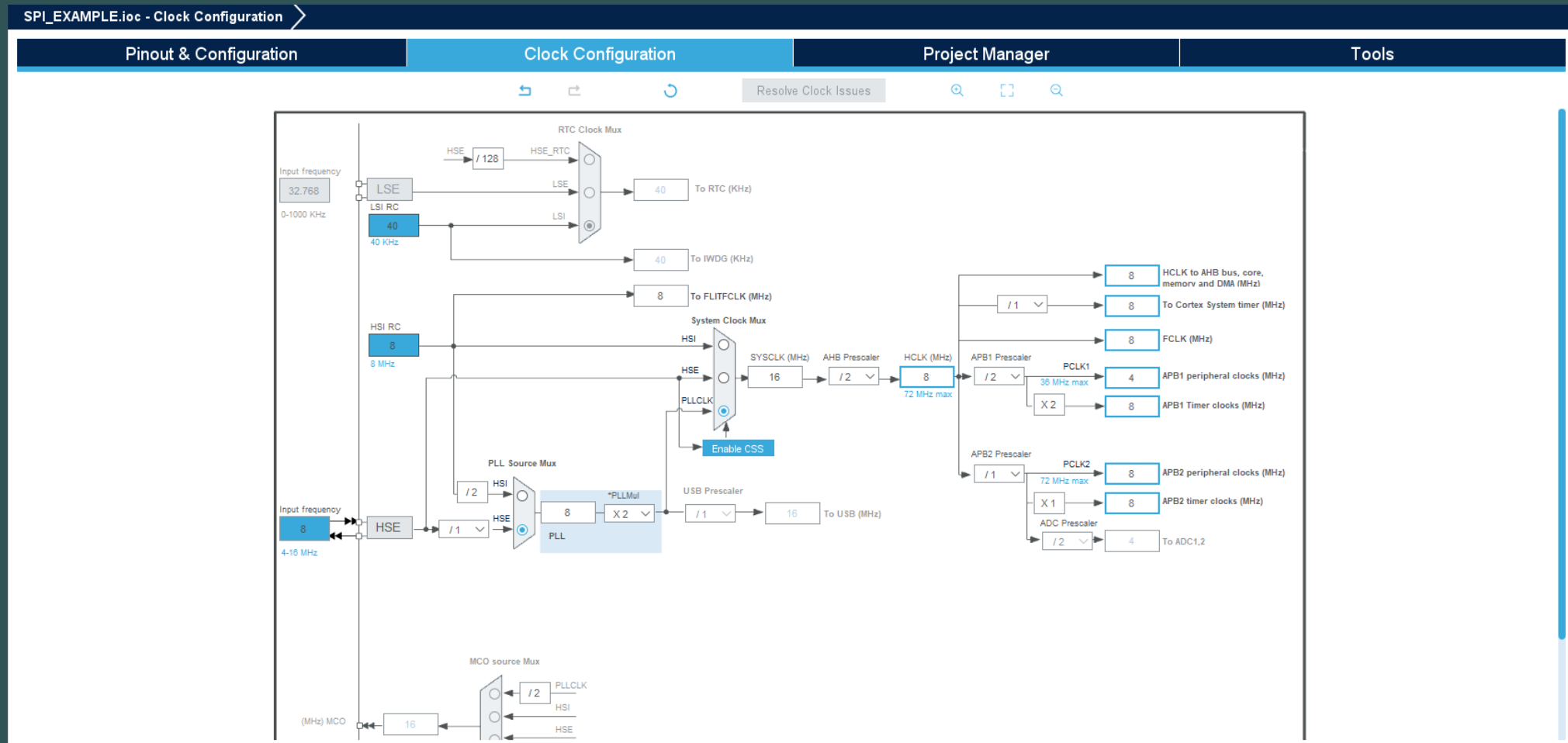
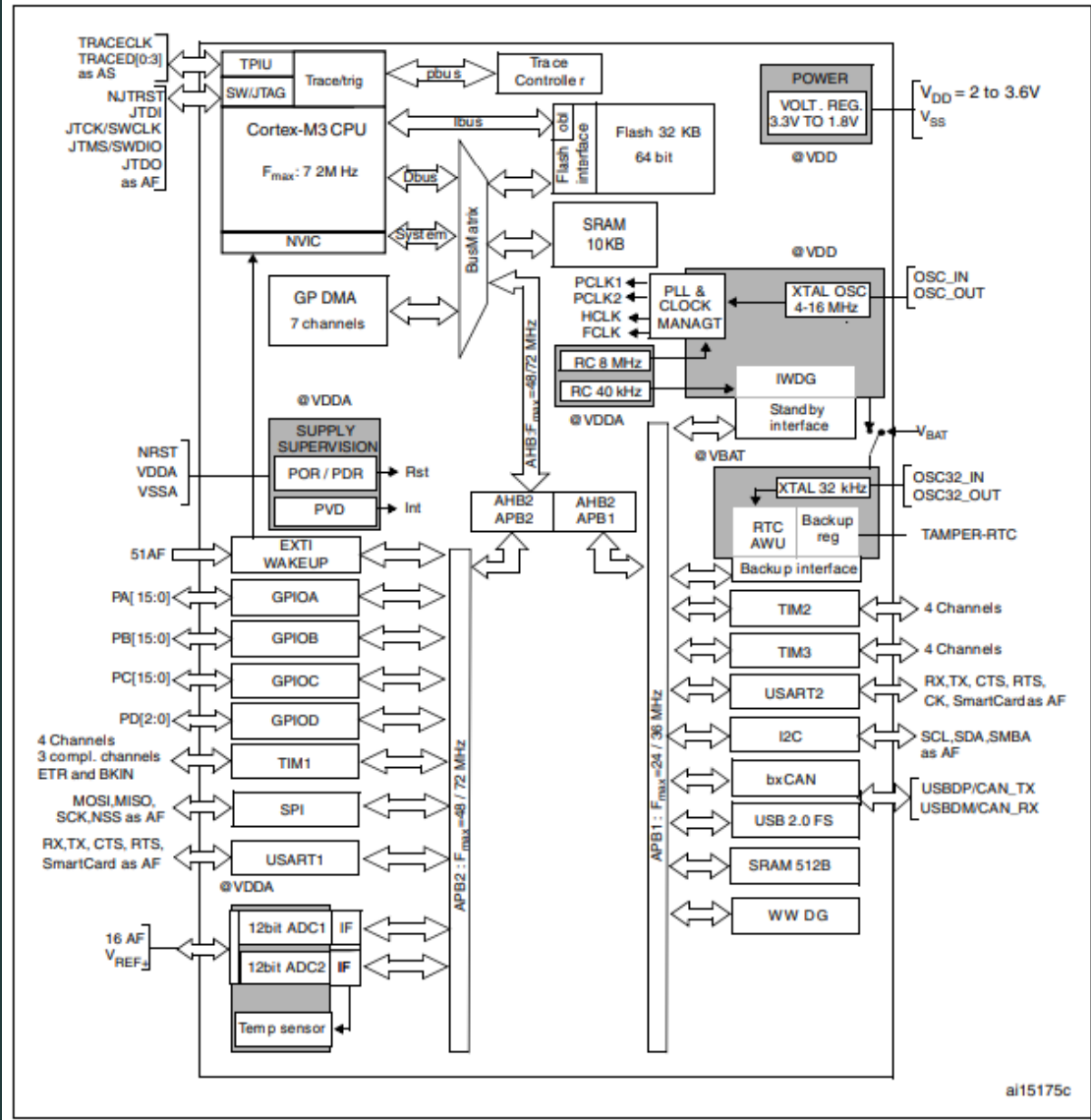


Figure 1. STM32F103xx performance line block diagram



FREQUENCY OF PERIPHERAL DEVICES

- You can find the bus configuration within the [datasheets](#).
- i.e. if you have to set the frequency of HSPI1 to 4 MHz, and you have set your HCLK to 64MHz, you should use a prescaler of 16.
- If you want to communicate with an external device you have to check the protocol details on its datasheets

MANDATORY SETTINGS

- In RCC you must select a clock source
- In SYS you have to set the communication between the PC and the Micro (typically through the ST-Link)

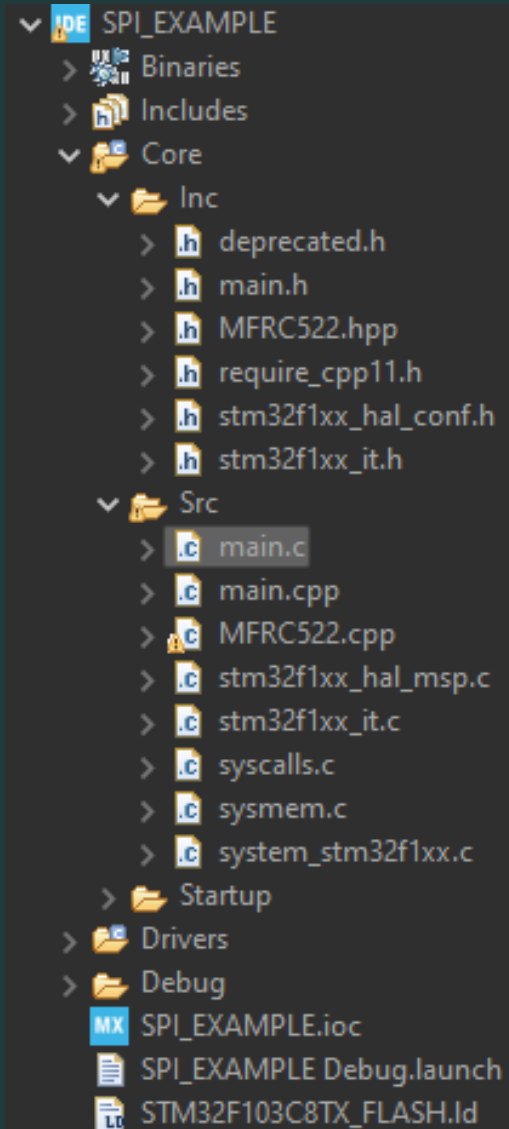
The image displays three screenshots of the STM32CubeMX IDE interface, illustrating mandatory settings for the RCC and SYS peripherals.

Left Screenshot (SYS Tab): The left sidebar shows the 'Categories' list with 'SYS' selected. The main panel shows the 'Debug' section with 'Serial Wire' selected, 'System Wake-Up' unchecked, and 'Timebase Source' set to 'SysTick'.

Middle Screenshot (RCC Tab): The left sidebar shows the 'Categories' list with 'RCC' selected. The main panel shows a warning: "Warning: This peripheral has no parameters to be configured."

Right Screenshot (RCC Tab): The left sidebar shows the 'Categories' list with 'RCC' selected. The main panel shows the 'Mode' section with 'High Speed Clock (HSE)' set to 'Crystal/Ceramic Resonator' and 'Low Speed Clock (LSE)' set to 'Disable'. The 'Master Clock Output' checkbox is unchecked. The 'Configuration' section shows a 'Reset Configuration' button and a list of settings: 'User Constants', 'NVIC Settings', 'GPIO Settings', and 'Parameter Settings'. The 'Parameter Settings' section is expanded, showing the following parameters:

Parameter	Value
VDD voltage (V)	3.3 V
Prefetch Buffer	Enabled
Flash Latency(WS)	0 WS (1 CPU cycle)
HSI Calibration Value	16
HSE Startup Timeout Value (ms)	100
LSE Startup Timeout Value (ms)	5000



STM32 PROJECT

- In Core/Inc you can find header files
- In Core/Src you can find .c files
- You can choose to program in C++ during project creation (beginning). Here you have to change main.c to main.cpp
- Many of the files you see here are specific libraries generated for the STM32f103
- Debug contains the build files

MAIN

- Programming an STM32 is closer to the classical C programming rather than the Arduino env
- HAL is the main library for STM32, here you can find all directives for controlling your micro
- The void loop actually is a while true
- Never put a return within whe infinite loop! If you have to stop the computation just use another while(1)
- You **MUST** place your code in specific sections (USER CODE BEGIN --- USER CODE END)

```
int main(void)
{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_SPI1_Init();
    /* USER CODE BEGIN 2 */

    /* USER CODE END 2 */

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

```
73     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
74     HAL_Init();
75
76     /* USER CODE BEGIN Init */
77
78     /* USER CODE END Init */
79
80     /* Configure the system clock */
81     SystemClock_Config();
82
83     /* USER CODE BEGIN SysInit */
84
85     /* USER CODE END SysInit */
86
87     /* Initialize all configured peripherals */
88     MX_GPIO_Init();
89     MX_SPI1_Init();
90     /* USER CODE BEGIN 2 */
91
92     /* USER CODE END 2 */
93
94     /* Infinite loop */
95     /* USER CODE BEGIN WHILE */
96     while (1)
97     {
98
99         /* USER CODE END WHILE */
100
101         /* USER CODE BEGIN 3 */
102     }
103     /* USER CODE END 3 */
```

SPECULAR IOC SETTINGS

- The stuff you set within the IOC is then implemented within the main, at the bottom. So you could also edit your settings here

```
static void MX_SPI1_Init(void)
{

    /* USER CODE BEGIN SPI1_Init 0 */

    /* USER CODE END SPI1_Init 0 */

    /* USER CODE BEGIN SPI1_Init 1 */

    /* USER CODE END SPI1_Init 1 */
    /* SPI1 parameter configuration*/
    hspi1.Instance = SPI1;
    hspi1.Init.Mode = SPI_MODE_MASTER;
    hspi1.Init.Direction = SPI_DIRECTION_2LINES;
    hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
    hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
    hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
    hspi1.Init.NSS = SPI_NSS_SOFT;
    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
    hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
    hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
    hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
    hspi1.Init.CRCPolynomial = 10;
    if (HAL_SPI_Init(&hspi1) != HAL_OK)
    {
        Error_Handler();
    }
}
```

```
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL2;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYCLK
                                |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV2;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
    {
        Error_Handler();
    }
}
```

SOME BASIC FUNCTIONS

- `HAL_Delay(100);` -> set a delay of 100ms (same as `delay(100)`)
- `HAL_GPIO_WritePin(GPIO_Port, GPIO_Pin, GPIO_PIN_SET);` -> `digitalWrite(Pin, HIGH);`
 - `GPIO_PIN_SET = HIGH (1)` - `GPIO_PIN_RESET = LOW (0)`
 - E.g. `Pin=B1` -> `GPIO_Port=GPIOB` and `GPIO_Pin=GPIO_PIN_1` (check `main.h`)
- `GPIO_PinState state = HAL_GPIO_ReadPin(GPIO_Port, GPIO_Pin);`

I Want to use SPI to communicate with an MFRC522, but the arduino library does not work. What should I do?

It is hard to find the same arduino library implemented also for an STM32, so you have to port it!

```
/**
 * Writes a uint8_t to the specified register in the MFRC522 chip.
 * The interface is described in the datasheet section 8.1.2.
 */
void MFRC522::PCD_WriteRegister(PCD_Register reg, uint8_t value) {

    HAL_GPIO_WritePin(CS_Port, CS, GPIO_PIN_RESET);
    // MSB == 0 is for writing. LSB is not used in address. Datasheet section 8.1.2.3.
    HAL_SPI_Transmit(hspi, (uint8_t *) &reg, sizeof(reg), HAL_MAX_DELAY);
    HAL_SPI_Transmit(hspi, &value, sizeof(value), HAL_MAX_DELAY);
    HAL_GPIO_WritePin(CS_Port, CS, GPIO_PIN_SET);
} // End PCD_WriteRegister()
```

```
void MFRC522::PCD_WriteRegister(PCD_Register reg, byte value) {
    SPI.beginTransaction(SPI_Settings(MFRC522_SPICLOCK, MSBFIRST, SPI_MODE0)); // Set the settings
    digitalWrite(_chipSelectPin, LOW); // Select slave
    SPI.transfer(reg); // MSB == 0 is for writing. LSB is for reading
    SPI.transfer(value);
    digitalWrite(_chipSelectPin, HIGH); // Release slave again
    SPI.endTransaction(); // Stop using the SPI bus
} // End PCD_WriteRegister()
```

SO, WHY USING AN STM32

- Due to specific requirements:
 - ADC 12 bit, ease of peripheral frequency setting, more GPIO, more computational power
 - Code optimization due to low level API access
 - Specific peripheral ports: CAN, RS485, ...
 - Watchdog, interrupts, timers, DMA, ...



IOT PROJECT TIPS



Environmental monitoring system: An environmental monitoring system can be used to measure environmental parameters such as temperature, humidity, air quality, or noise level. The collected data can be sent to a server for analysis or visualization. Use AI to predict these parameter for the following day/week



Home automation system: A home automation system can be used to control home devices such as lights, thermostats, and locks. Devices can be controlled manually or automatically based on certain events or conditions.



Security system: A security system can be used to monitor an area or building for intrusions or other unexpected events. The data collected can be sent to a cloud for analysis or triggering alarms.



Tracking system: A tracking system can be used to track the location of people entering and exiting a building. This scenario is very interesting in the industrial context. Estimate with AI the number of people within a building



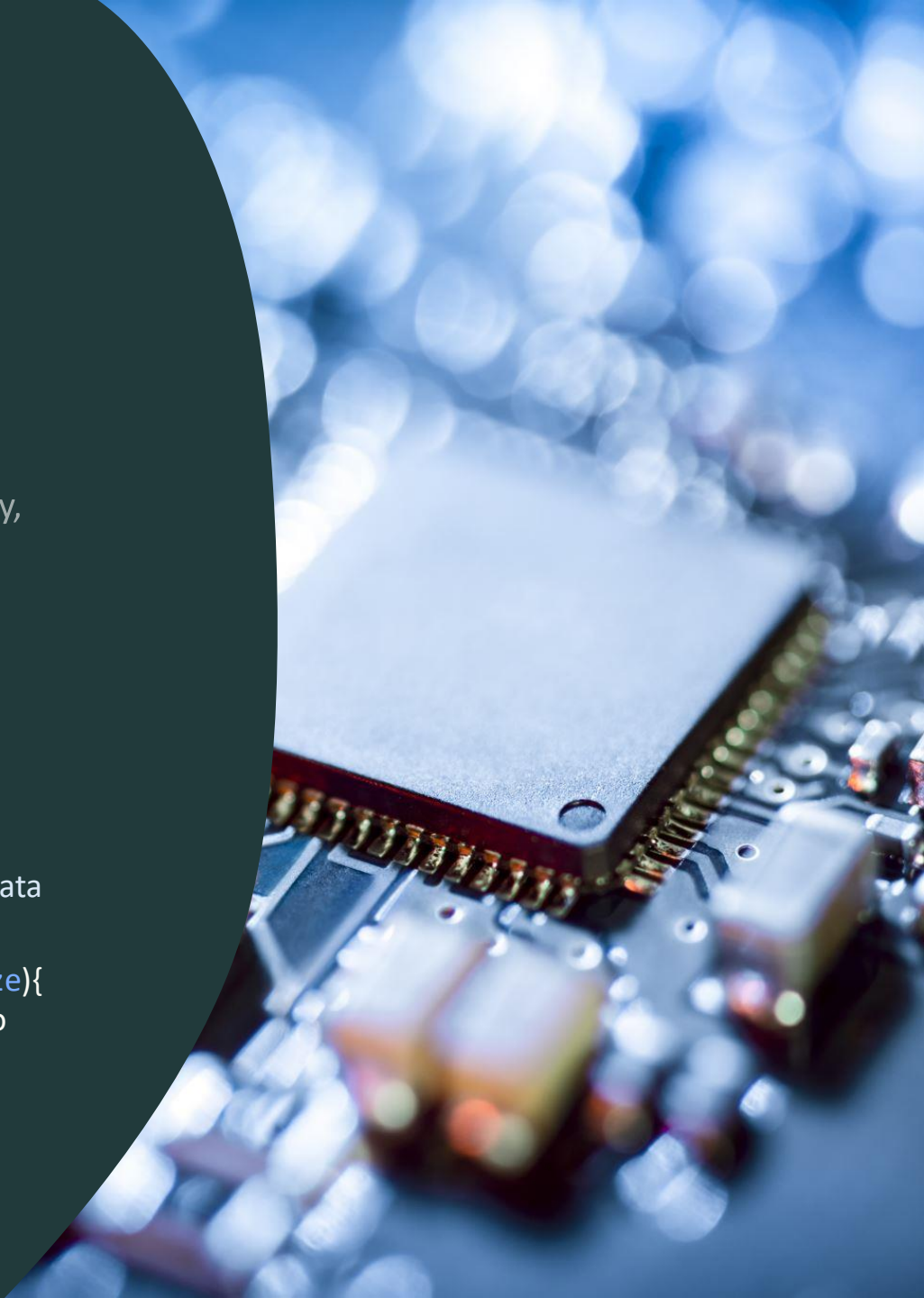
For PRO students: try to embed AI within the microcontroller

EXTRA: INTERRUPTS

- An interrupt is a signal generated externally from the CPU stream, which warns about the occurrence of an event (e.g. data has arrived on a peripheral, a button has been pressed, and so on ...)
- In the STM32 IDE, interrupts are called NVICs.
- Example:
 - `HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);`
 - `HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);` -> we enable interrupts for pins between 5 and 9
 - `EXTI->RTSR1 |= EXTI_RTSR1_RT9;` -> we enable trigger on rising edge
 - `void EXTI9_5_IRQHandler(void) {`
 - `if (EXTI->PR & EXTI_PR_PR9) {`
 - `EXTI->PR = EXTI_PR_PR9;` -> After handling the interrupt, resets the PR register bit ... then do something ...

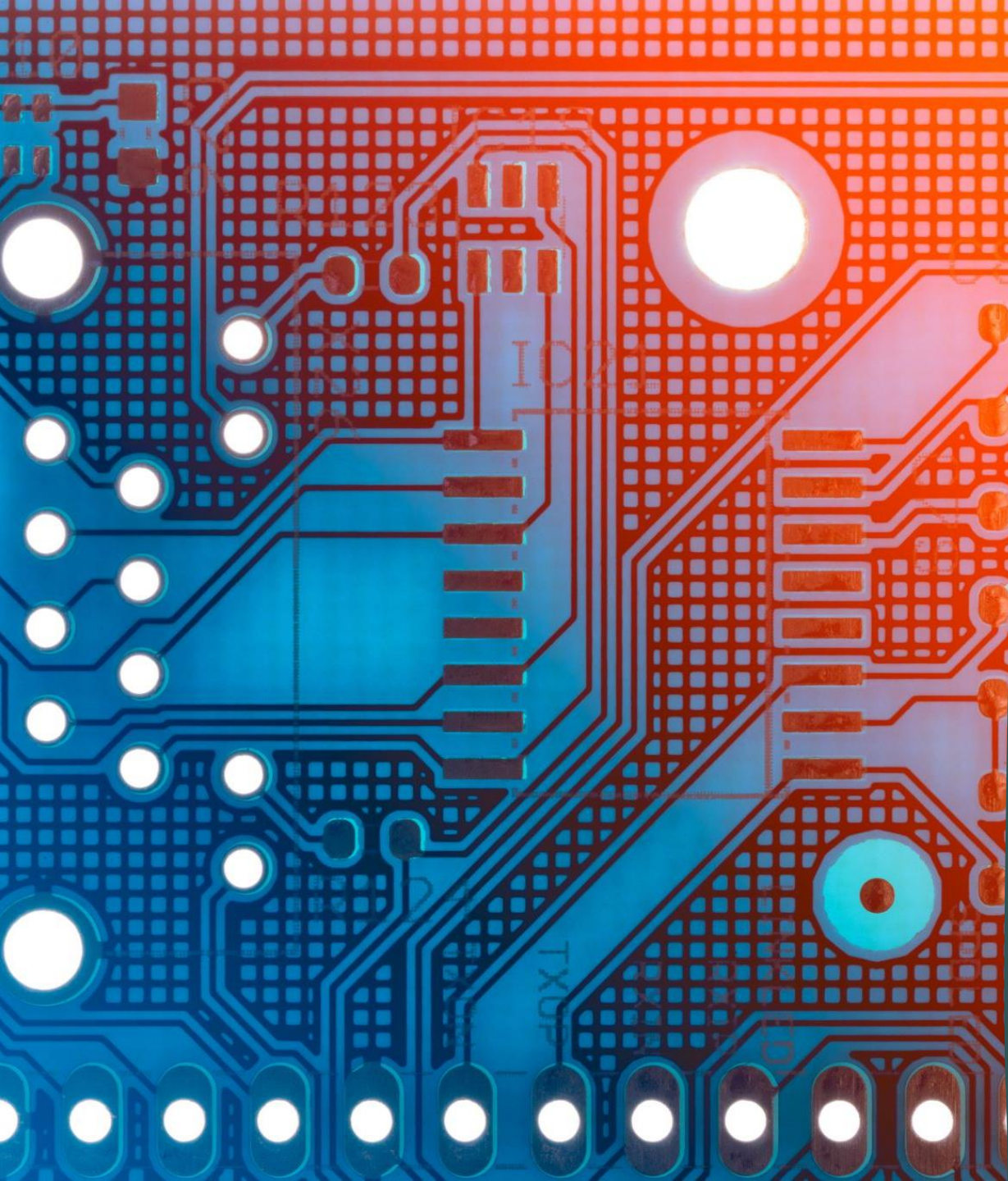
EXTRA: DIRECT MEMORY ACCESS (DMA)

- DMA is a mechanism that allows peripheral devices direct access to memory, without involving the CPU
- Example: we want to read some incoming data from the USART Port:
 - In USART1 -> DMA SETTINGS -> add USART1_RX in normal mode
 - Activate USART1 global interrupt
 - **HAL_UARTEx_ReceiveTxDle_DMA**(&huart1, (**uint8_t***) rxbuffer, MAX_STRING_LENGTH); -> we set the DMA Buffer and its size
 - **__HAL_DMA_DISABLE_IT**(&hdma_usart1_rx, DMA_IT_HT); -> disable the half data transfer interrupt
 - **void HAL_UARTEx_RxEventCallback**(**UART_HandleTypeDef** *huart, **uint16_t** size){
 - **if**(huart->Instance == USART1){ ... -> we define the DMA interrupt callback to read RX data



AI ON STM32





HARDWARE LIMITATIONS

Deploying a Neural Network on a Microcontroller is not Trivial

- Up to a few KiloBytes of RAM
- Usually less than 1 MegaByte of Internal Storage
- Limited computational power
- Typically, a floating point unit is not available

SOFTWARE LIMITATIONS

- Do I need a specific C/C++ library for running my model on the Microcontroller?
- How do I load the model within the Microcontroller?
- The model does not fit in the Microcontroller's memory, how can I reduce its size?



OPEN NEURAL NETWORK EXCHANGE

- It is recommended to watch this video
- ONNX is an open format built to represent machine learning models.
- Neural Network built upon different frameworks (such as PyTorch, Tensorflow, etc.) can be exported to this format
- It is useful when we want to optimize our network for a specific hardware platform



Everything You Want to Know About ONNX



ONNX



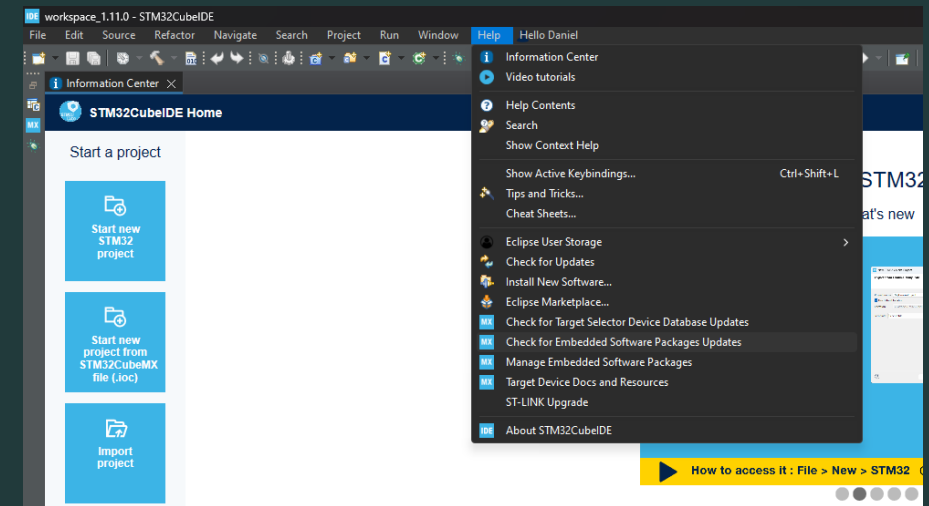
ONNX FOR STM32

On an STM32 Microcontroller it is possible to upload the following models:

- Tensorflow light
- Keras
- ONNX

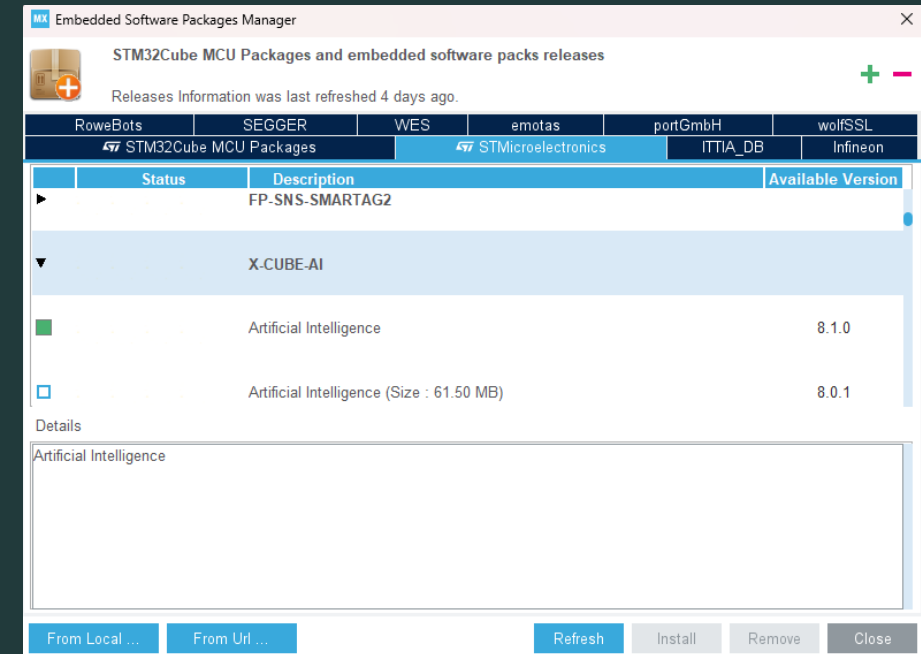
You can proceed by installing STM32CubeAI extension within your IDE

Click on: Help -> Check for Embedded Software Packages Updates



STM32 X-CUBE-AI

- Proceed by installing X-CUBE-AI extension for STM32 CUBE IDE
- This is an ST toolkit useful to load and run AI models within a microcontroller



SINNET: AN APPLICATION EXAMPLE FOR AI ON STM32

- Here we have SinNet, a Feed-Forward Neural Network which tries to approximate the Sin function
- **Hidden_size**: number of hidden neurons (256 is enough)
- **Batch Normalization**: helps to converge faster (normalizes the input data before feeding them to the following layer)
- **Dropout**: regularization is necessary to prevent overfitting (small network = small regularization)
- **Tanh activation**: we need to propagate gradient also for negative values (x in $[0, 2\pi]$; $\text{Sin}(x)$ in $[-1, 1]$)

```
SinNet.py
1  import torch
2  from torch import nn
3
4
5  class SinNet(nn.Module):
6      def __init__(self, hidden_size: int= 1024, p_dropout=0.35) -> None:
7          super(SinNet, self).__init__()
8          self.hidden_size = hidden_size
9          self.p_dropout = p_dropout
10
11         self.input_layer = nn.Linear(1, self.hidden_size)
12         self.hidden_layer = nn.Linear(self.hidden_size, self.hidden_size)
13         self.output_layer = nn.Linear(self.hidden_size, 1)
14
15         self.activation = nn.Tanh()
16
17         self.network = nn.Sequential([
18             self.input_layer,
19             self.activation,
20             nn.Dropout(self.p_dropout),
21
22             nn.BatchNorm1d(self.hidden_size),
23             self.hidden_layer,
24             self.activation,
25             nn.Dropout(self.p_dropout),
26
27             nn.BatchNorm1d(self.hidden_size),
28             self.output_layer,
29             self.activation,
30         ])
31
32     def forward(self, x: torch.Tensor) -> torch.Tensor:
33         x = self.network(x)
34         return x
```

WHAT TO DO AFTER STORING NN WEIGHTS? (QUANTIZATION)

- In order to fit the model on an STM32, we need to reduce its size!
- Quantization means changing the resolution of the network's weights
 - Typically, a network works with Float (32 bit) data and variables (let's say we have a 4KB model)
 - One way to reduce the model's size could be using 16/8 bit data and variables (16 bit means 2KB, 8 bit means 1KB)
 - This is a practical example of weights quantization (This is not the only solution)
- Quantization can be done through ONNX



QUANTIZATION



For this example we will use 8 bit Uint quantization



Weights and activation functions will be represented with 8 bit



Remember that it is not true to say that quantization reduces the accuracy!

Rarely it is possible to obtain a better accuracy since quantization is also a way to reduce overfitting



On the other hand, it is also not true that quantized models are faster, this depends upon the underlying Hardware

ONNX MODEL CONVERSION

- Before applying ONNX quantization, we must convert the model from a PyTorch representation to an ONNX representation

```
def convert_to_onnx(train_outputs: dict, args: argparse.Namespace) -> None:
    print("\nConverting model to ONNX format")

    model = train_outputs["model"] # pytorch float32 model
    model.train(False)
    model.to("cpu")

    calibration_set = torch.randn(args.batch_size, 1, requires_grad=True).to("cpu")

    onnx_model_path = os.path.join(train_outputs["run_path"], "model.onnx")
    torch.onnx.export(model, # model being run
                      calibration_set, # model input (or a tuple for multiple inputs)
                      onnx_model_path, # where to save the model (can be a file or file-like object)
                      export_params=True, # store the trained parameter weights inside the model file
                      opset_version=17, # the ONNX version to export the model to
                      do_constant_folding=True, # whether to execute constant folding for optimization
                      input_names = ['input'], # the model's input names
                      output_names = ['output'], # the model's output names
                      dynamic_axes={'input' : {0 : 'batch_size'}, # variable length axes
                                   'output' : {0 : 'batch_size'}}),
    )

    onnx_model = onnx.load(onnx_model_path)
    onnx.checker.check_model(onnx_model)
    train_outputs["onnx_model"] = onnx_model

    test_onnx_model(model, onnx_model_path, ["CPUExecutionProvider"], train_outputs, args)
    print("Exported model has been tested with ONNXRuntime, and the result looks good!")
    print(f"ONNX model stored at {onnx_model_path}")
```

ONNX QUANTIZATION

- Quantization needs a calibration set in order to better estimate the weights and activations values
- It is possible to set different variable representation and to use different calibration methods
- The quantization results in a model that we can fit into an STM32
- .pth means PyTorch model while .onnx means ONNX model






```
calibration_set = torch.rand(10**3).unsqueeze(1) * 2 * torch.pi
ort_session = onnxruntime.InferenceSession(model_fp32_path, providers=['CPUExecutionProvider'])
qdr = QuantizationDataReader(calibration_set,
                             batch_size=1,
                             input_name=ort_session.get_inputs()[0].name
                             )

quantization.shape_inference.quant_pre_process(model_fp32_path, model_prep_path)
q_static_opts = {
    "ActivationSymmetric": True,
    "WeightSymmetric": True
}

model_int8_path = os.path.join(train_outputs["run_path"], 'model_int8.onnx')
quantized_onnxmodel = quantization.quantize_static(model_input=model_prep_path,
                                                  model_output=model_int8_path,
                                                  calibration_data_reader=qdr,
                                                  extra_options=q_static_opts,
                                                  weight_type=QuantType.QUInt8,
                                                  activation_type=QuantType.QUInt8,
                                                  calibrate_method=CalibrationMethod.MinMax
                                                  )

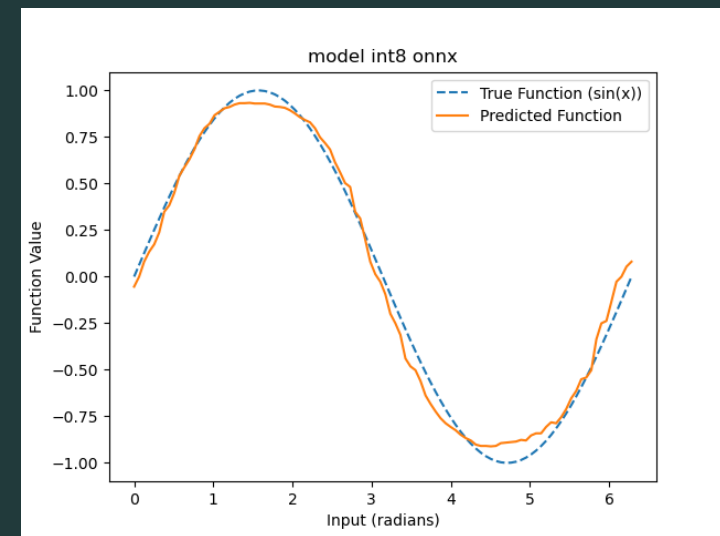
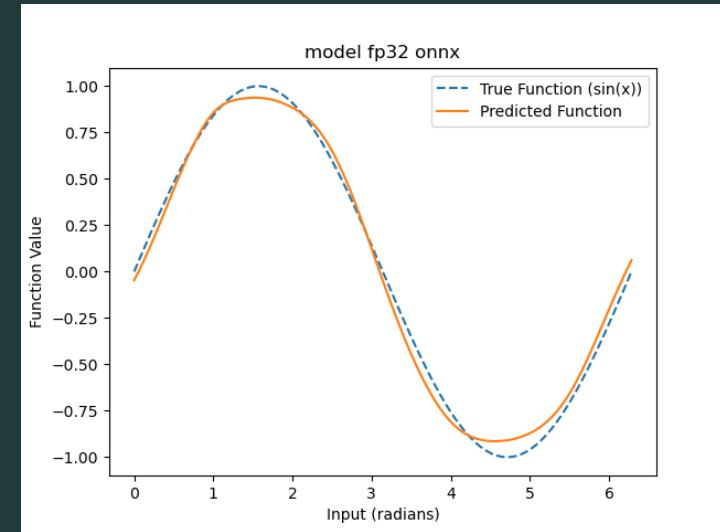
onnx.checker.check_model(model_int8_path)
test_onnx_model(model, model_int8_path, ["CPUExecutionProvider"], train_outputs, args)
train_outputs["onnx_model_int8"] = quantized_onnxmodel

print("Quantization done!")
```

 model_int8.onnx	05/12/2023 16:27	File ONNX	83 KB
 model_int8.pth	05/12/2023 16:27	File PTH	169 KB
 model.onnx	05/12/2023 16:27	File ONNX	270 KB
 model_prep.onnx	05/12/2023 16:27	File ONNX	271 KB
 model_fp32.pth	05/12/2023 16:26	File PTH	274 KB

QUANTIZATION RESULTS

- From these results we can see that the quantized model is still pretty good in approximating the Sin function
- To obtain a better quantized model we could increase the number of neurons (or just spend more time on testing different hyperparameters)
- But, consider that this model fits on a cheap STM32F401RE



LOADING THE MODEL ON AN STM32

- Create a new project, scroll down to reach “Artificial Intelligence”
- On “model” select ONNX
- Load the model’s weights (in this case model_int8.onnx)
- Try not to apply further compression
- Finally, click on “Analyze”

STM32 Project

Target Selection

STM32 target or STM32Cube example selection is required

MCU/MPU Selector Board Selector Example Selector Cross Selector

MCU/MPU Filters

Commercial Part Number

I/O

Connectivity

SMPS

MIDDLEWARE

Artificial Intelligence

Enable

Model Please select

Type Please select

Structure B1TFLite

Weight ONNX

Compression None

Analyze

PHYSICAL

Voltage Min From 0.0 to 2.4 (V)

Voltage Max From 0.0 to 3.6 (V)

Features Block Diagram Docs & Resources CAD Resources Datasheet Buy

STM32WL3 New Sub-1GHz SoC Low-Power & Flexibility

AI Summary

Minimum Flash: undefined

Minimum Ram: undefined

MCUs/MPUs List: 3166 items

Commercial P...	Part No	Reference	M...	Unit Price fo...	Board	Package	Flash	RAM	IO	Freq...
☆ STM32C011D6...	STM32C011D6Yx	Co...	NA			WLCSOP 12 1.7x1.42...	32 kB	6 kBytes	10	48 MHz
☆ STM32C011D6...	STM32C011D6Yx	Ac...	0.3213			WLCSOP 12 1.7x1.42...	32 kB	6 kBytes	10	48 MHz
☆ STM32C011F4P3	STM32C011F4Px	Co...	NA			TSSOP-20	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4P6	STM32C011F4Px	Ac...	0.3116			TSSOP-20	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4P7	STM32C011F4Px	Ac...	0.3335			TSSOP-20	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4P...	STM32C011F4Px	Ac...	0.3335			TSSOP-20	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4U3	STM32C011F4Ux	Co...	NA			UFQFPN 20 3x3x0.6...	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4U...	STM32C011F4Ux	Ac...	0.3583			UFQFPN 20 3x3x0.6...	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F4U...	STM32C011F4Ux	Ac...	0.3116			UFQFPN 20 3x3x0.6...	16 kB	6 kBytes	18	48 MHz
☆ STM32C011F6P3	STM32C011F6Px	Ac...	0.4144			TSSOP-20	32 kB	6 kBytes	18	48 MHz
☆ STM32C011F6P...	STM32C011F6Px	Ac...	0.4144			TSSOP-20	32 kB	6 kBytes	18	48 MHz
☆ STM32C011F6P6	STM32C011F6Px	Ac...	0.3604			TSSOP-20	32 kB	6 kBytes	18	48 MHz
☆ STM32C011F6P...	STM32C011F6Px	Ac...	0.3604			TSSOP-20	32 kB	6 kBytes	18	48 MHz

CHOOSE YOUR MICROCONTROLLER

- Select the board you want to use and check if it supports the model
- If it is not supported:
 - Change microcontroller
 - Try to reduce the model's size

The screenshot displays the STM32CubeMX software interface. On the left, the 'MCU/MPU Selector' panel shows filters for 'Commercial Part Number' (STM32F401RET6), 'I/O', 'Connectivity', 'SMPS', 'MIDDLEWARE', 'Artificial Intelligence' (with 'Enable' checked), and 'PHYSICAL' (with 'Voltage Min' at 1.7V and 'Voltage Max' at 3.6V). The main panel shows the 'STM32F4 Series' and the 'STM32F401RET6' product details, including its unit price (\$2.7654), board (NUCLEO-F401RE), and package (LQFP 64 10x10x1.4 mm). Below this, the 'AI Summary' section shows the minimum flash and ram requirements. At the bottom, the 'MCUs/MPUs List' table shows two items:

Commercial P...	Part No	Reference	Unit Price fo...	Board	Package	Flash	RAM	IO	Freq...
☆ STM32F401RET6	STM3...	STM32F401RETx	Ac... 2.7654	NUCLEO-F401RE	LQFP 64 10x10x1.4	512 k...	96 kB...	50	84 MHz
☆ STM32F401RET...	STM3...	STM32F401RETx	Ac... 2.7654		LQFP 64 10x10x1.4	512 k...	96 kB...	50	84 MHz

CUBE AI PROJECT SETUP

To import the Neural Network go to “middleware”, look for “X-CUBE AI” and in the meanwhile click on “Software Packs” -> “Select Components and enable “Core” flag under STMicroelectronics-X-CUBE-AI

The screenshot displays the STM32CubeIDE interface. The 'Pinout & Configuration' window is open, showing the 'Mode' section with 'Artificial Intelligence X-CUBE-AI' selected. Below this, the 'Configuration' section shows a table with columns 'Name', 'Used Ram', 'Used Flash', and 'Complexity'. The table lists 'sinnet' and 'Library' with their respective values, and a 'Total (1)' row.

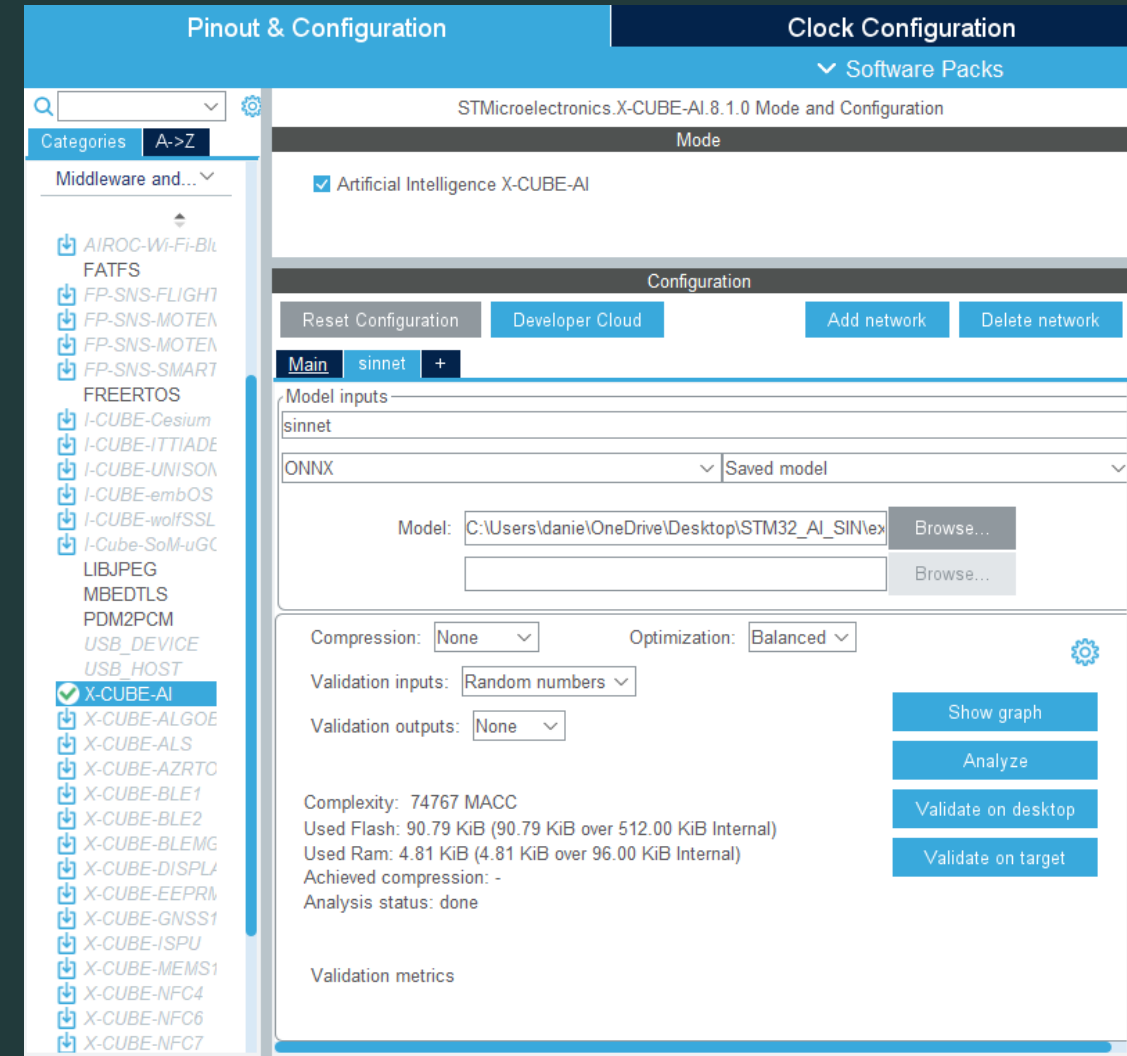
Name	Used Ram	Used Flash	Complexity
sinnet	1024.00 B	70.50 KiB	74767 MACC
Library	3.81 KiB	20.29 KiB	
Total (1)	4.81 KiB	90.79 KiB	74767 MACC

The 'Software Packs Component Selector' dialog is also open, showing a list of packs. The 'Artificial Intelligence X-CUBE-AI' pack is selected, and the 'Core' flag is enabled under the 'Device Application' section.

Pack / Bundle / Component	Status	Version	Selection
ITIA_DB I-CUBE-ITTIADB		8.8.0	Install
Infineon AIROC-Wi-Fi-Bluetooth-STM32		1.5.1	Install
RoweBots I-CUBE-UNISONRTOS		5.5.0.4	Install
SEgger I-CUBE-embOS		1.3.0	Install
STMicroelectronics FP-ATR-ASTRA1		2.0.0	Install
STMicroelectronics FP-ATR-SIGFOX1		3.2.0	Install
STMicroelectronics FP-SNS-FLIGHT1		5.0.2	Install
STMicroelectronics FP-SNS-MOTENV1		4.3.2	Install
STMicroelectronics FP-SNS-MOTENVWB1		1.3.1	Install
STMicroelectronics FP-SNS-SMARTAG2		1.2.0	Install
STMicroelectronics X-CUBE-AI		8.1.0	
Artificial Intelligence X-CUBE-AI		8.1.0	
Core		8.1.0	<input checked="" type="checkbox"/>
Device Application		8.1.0	
Application			Not selected
STMicroelectronics X-CUBE-ALGOBUILD		1.3.0	Install
STMicroelectronics X-CUBE-ALS		1.0.1	Install
STMicroelectronics X-CUBE-AZRTOS-F4		1.1.0	Install
STMicroelectronics X-CUBE-AZRTOS-F7		1.1.0	Install
STMicroelectronics X-CUBE-AZRTOS-G0		1.1.0	Install
STMicroelectronics X-CUBE-AZRTOS-G4		2.0.0	Install
STMicroelectronics X-CUBE-AZRTOS-H7		3.2.0	Install
STMicroelectronics X-CUBE-AZRTOS-L4		2.0.0	Install
STMicroelectronics X-CUBE-AZRTOS-L5		2.0.0	Install
STMicroelectronics X-CUBE-AZRTOS-WB		2.0.0	Install
STMicroelectronics X-CUBE-AZRTOS-WL		2.0.0	Install
STMicroelectronics X-CUBE-BLE1		7.0.0	Install
STMicroelectronics X-CUBE-BLE2		3.3.0	Install

X-CUBE AI MENU

- In this menu you can add, remove neural networks, edit and validate them
- Under “Main” you can check memory and RAM consumption
- If you choose to further compress the network, it is recommended to run “Validate on desktop”



RUNNING THE MODEL

- Getting the model to run is not straightforward since there are many expedients to do before
- The neural network files are generated under "X-CUBE_AI/App"

```
// Chunk of memory used to store the intermediate values of the network
AI_ALIGNED(4) ai_u8 activations[AI_SINNET_DATA_ACTIVATIONS_SIZE];

// Chunk of memory used to store input and output data
AI_ALIGNED(4) ai_i8 input_data[AI_SINNET_IN_1_SIZE_BYTES];
AI_ALIGNED(4) ai_i8 output_data[AI_SINNET_OUT_1_SIZE_BYTES];

// pointer which points at the model
ai_handle model = AI_HANDLE_NULL;

// structs which points to input and output data
ai_buffer model_input[AI_SINNET_IN_NUM]; // = AI_SINNET_IN;
ai_buffer model_output[AI_SINNET_OUT_NUM]; // = AI_SINNET_OUT;

// get weights and activations of the model
ai_network_params model_params = {
    AI_SINNET_DATA_WEIGHTS(ai_sinnet_data_weights_get()),
    AI_SINNET_DATA_ACTIVATIONS(activations)
};

// let model input and output point towards input_data and output_data memories
model_input[0].size = 1;
model_input[0].data = AI_HANDLE_PTR(input_data);
model_output[0].size = 1; // theoretically number of batches
model_output[0].data = AI_HANDLE_PTR(output_data);
```

USEFUL RESOURCES

- https://www.st.com/resource/en/data_brief/x-cube-ai.pdf
- https://www.st.com/resource/en/user_manual/um2526-getting-started-with-xcubeai-expansion-package-for-artificial-intelligence-ai-stmicroelectronics.pdf



THE END

- A deeper explanation of STM32, AI for embedded systems and AI on STM32 Microcontrollers would take an entire course
- My personal recommendation is to exploit the mandatory IoT Project for the exam to dive deeper into these topics or even base your master thesis on a research project or possible application in the field of AI for embedded systems

For any kind of support you can contact me [here](#)