intecs Solutions

the Brainware company

Introduction to Nucleo-64 platform

The company

- Intecs - Italian company with activities in:
  - Defense
  - Railway
  - Aerospace
  - Traffic Control & Surveillance
  - Automotive
  - Telecom
- Approx. 500 employees over 6 cities in Italy (not only)
- Purpose of these classes: getting familiar with the world of embedded systems and microcontrollers.

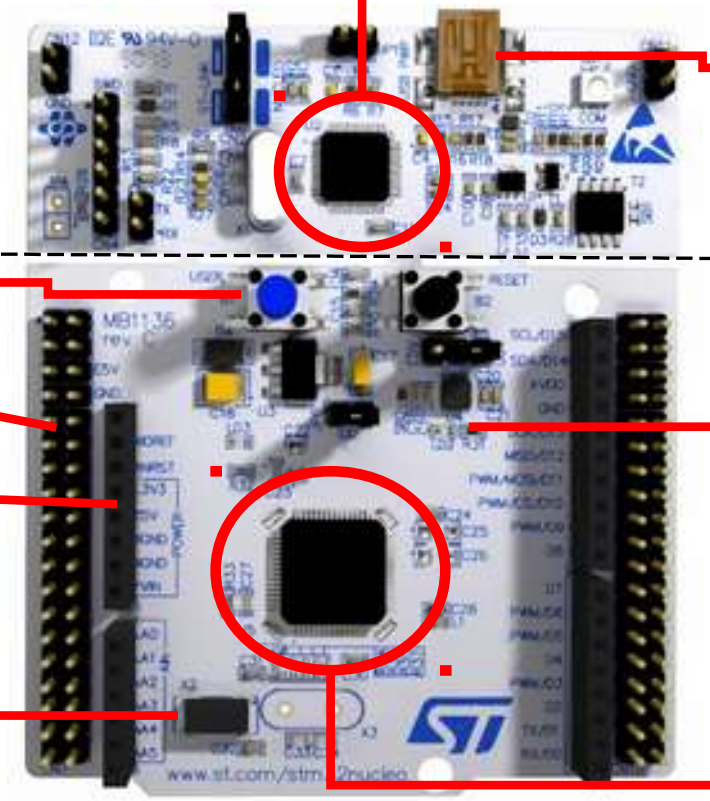
Introducing myself

- PhD in computer engineering @diag
- Focus on wireless sensor networks and low power devices.
- Since 2012 partner of Wsense (university spin-off): hw + microcontroller software development.
- In Intecs since October 2016: head of HW Lab in Rome, embedded sw developer/hw designer.

Tools used in this class (1/4)

The development board:

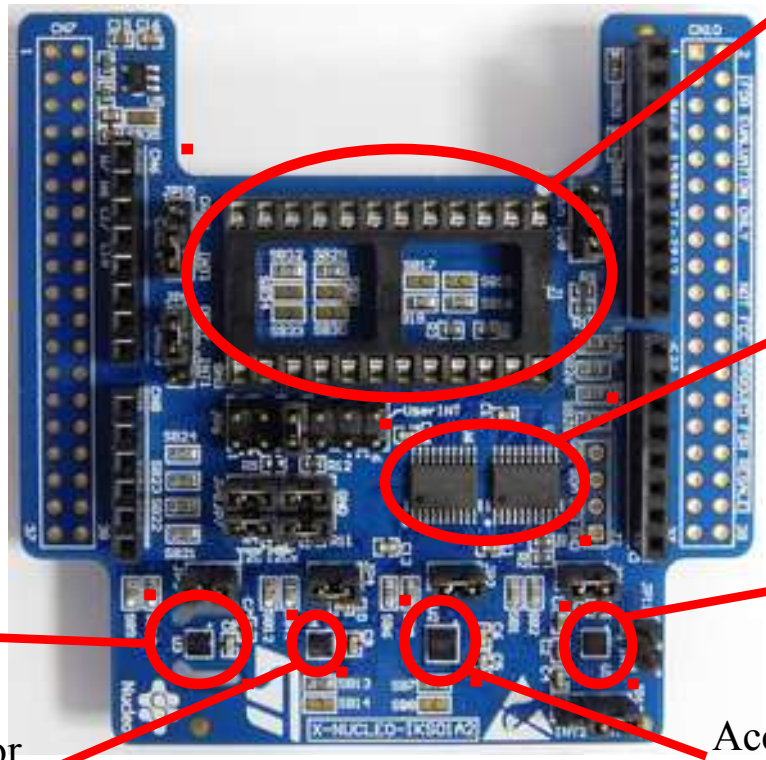
Nucleo-F401RE



Tools used in this class (2/4)

Sensor expansion board:

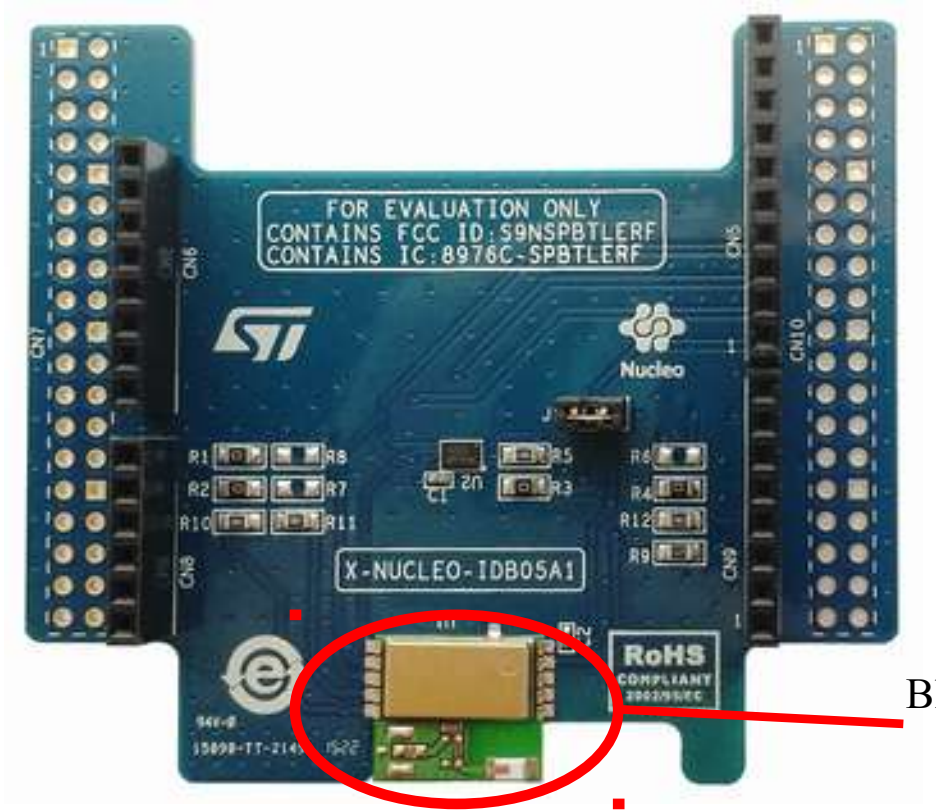
X-Nucleo-IKS01A2



Tools used in this class (3/4)

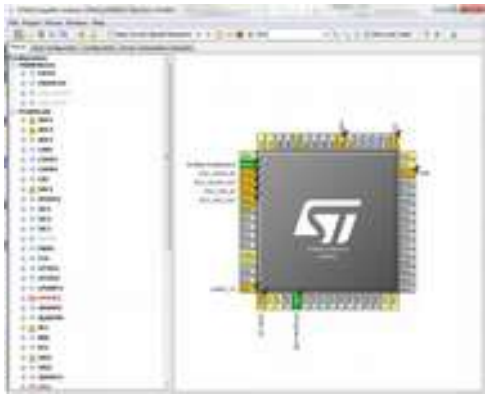
Bluetooth expansion board:

X-Nucleo-IDB05A1

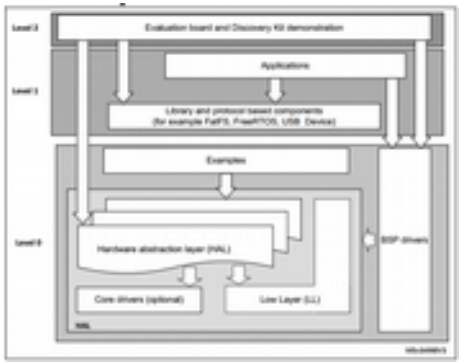


Tools used in this class (4/4)

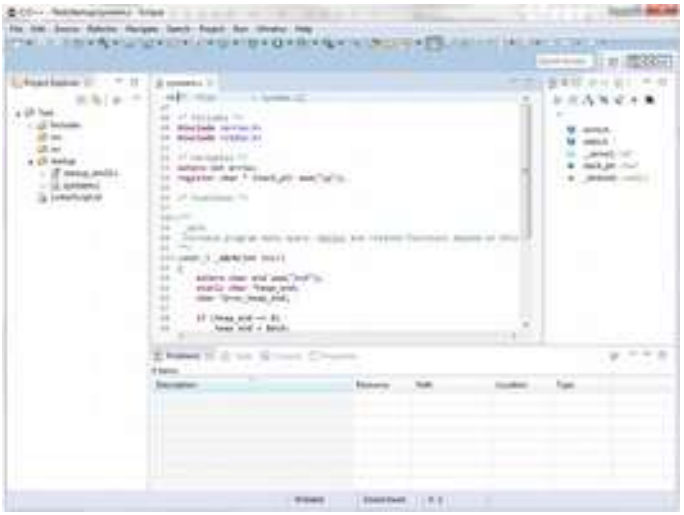
Framework, IDE & tools



STM CubeMX



Stm32CubeF4



System Workbench 4

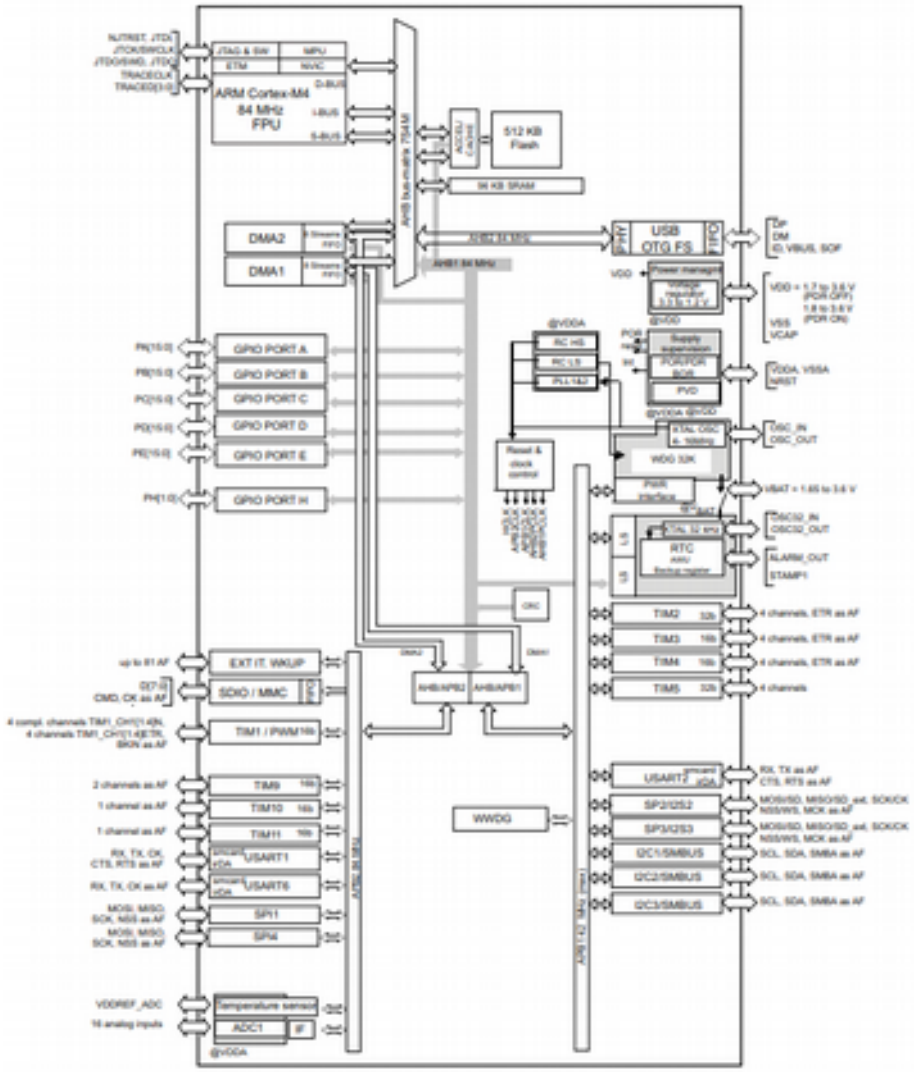
The Microcontroller

Stm32F401 Microcontroller:

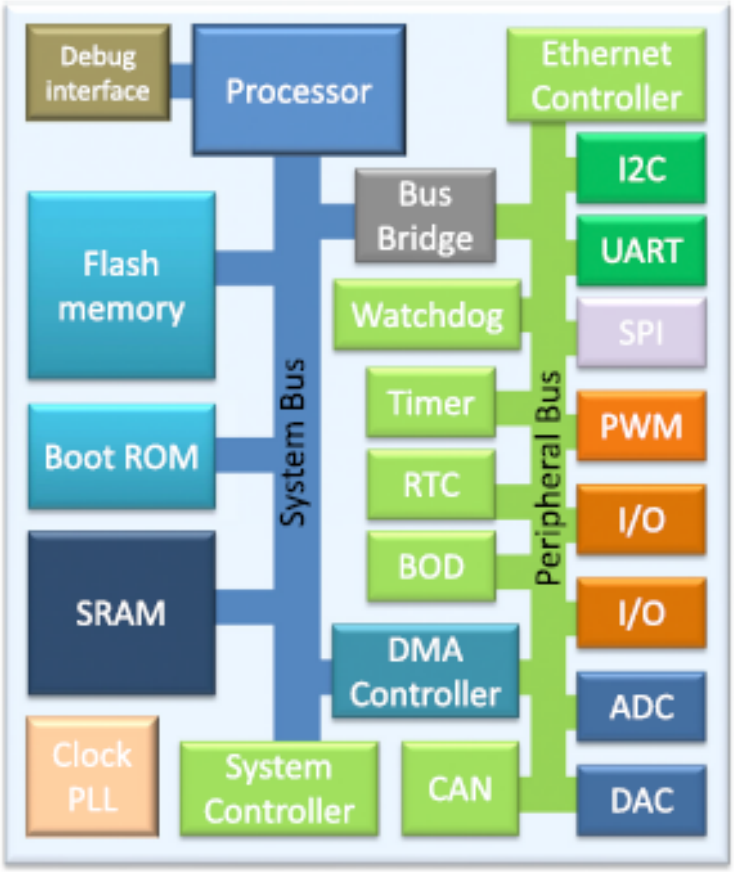
- Based on Cortex M4 processor
- 512KB ROM Flash memory
- 96KB SRAM data memory
- Up to 84Mhz operating frequency
- 42uA in sleep (stop mode) w/RTC

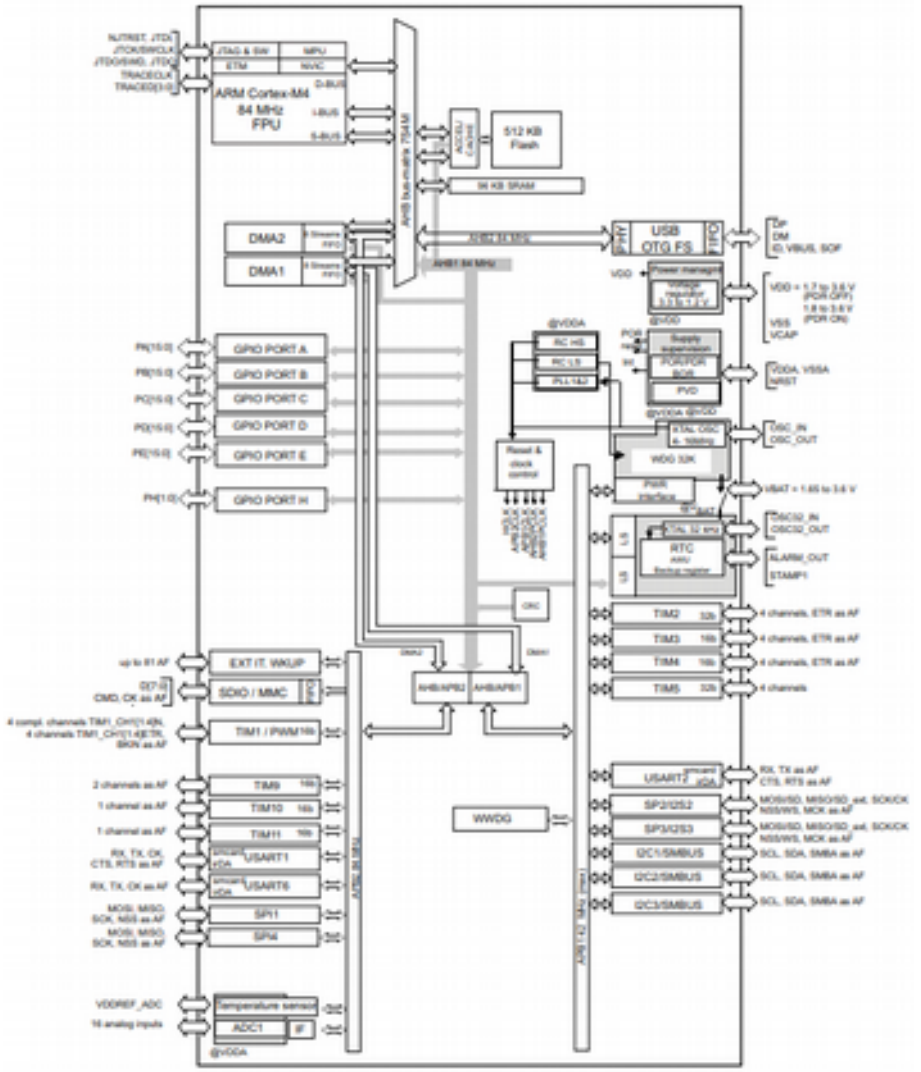
WLCSP49 (3.06 x 3.06 mm) LQFP100 (14 × 14 mm LQFP64 (10 × 10 mm) UFQFPN48 (7 × 7 mm) UFBGA100 (7 × 7 mm)

Available packages

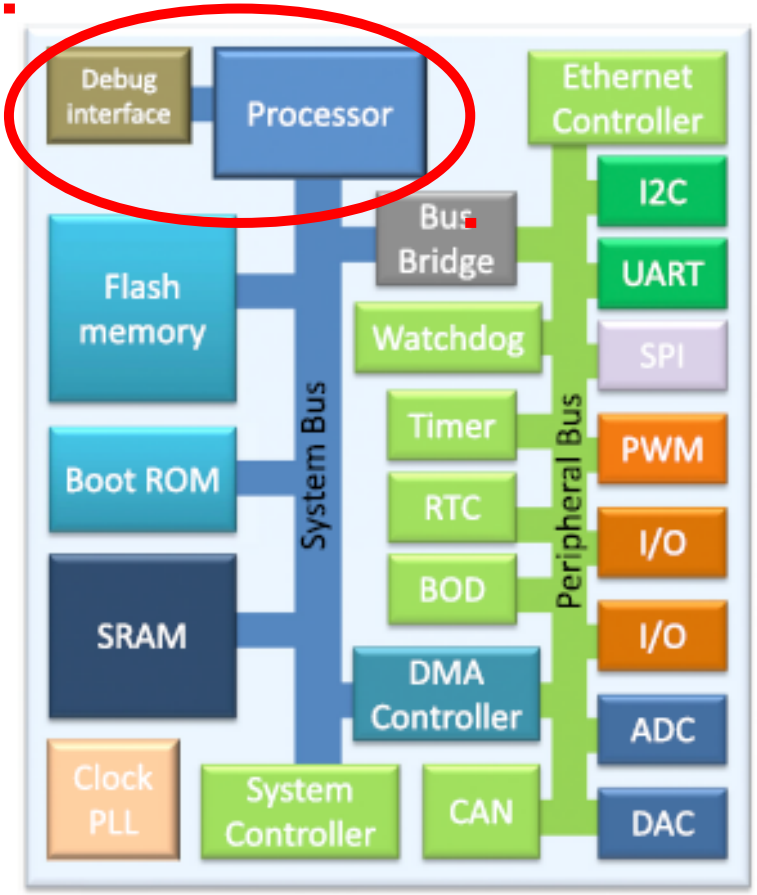


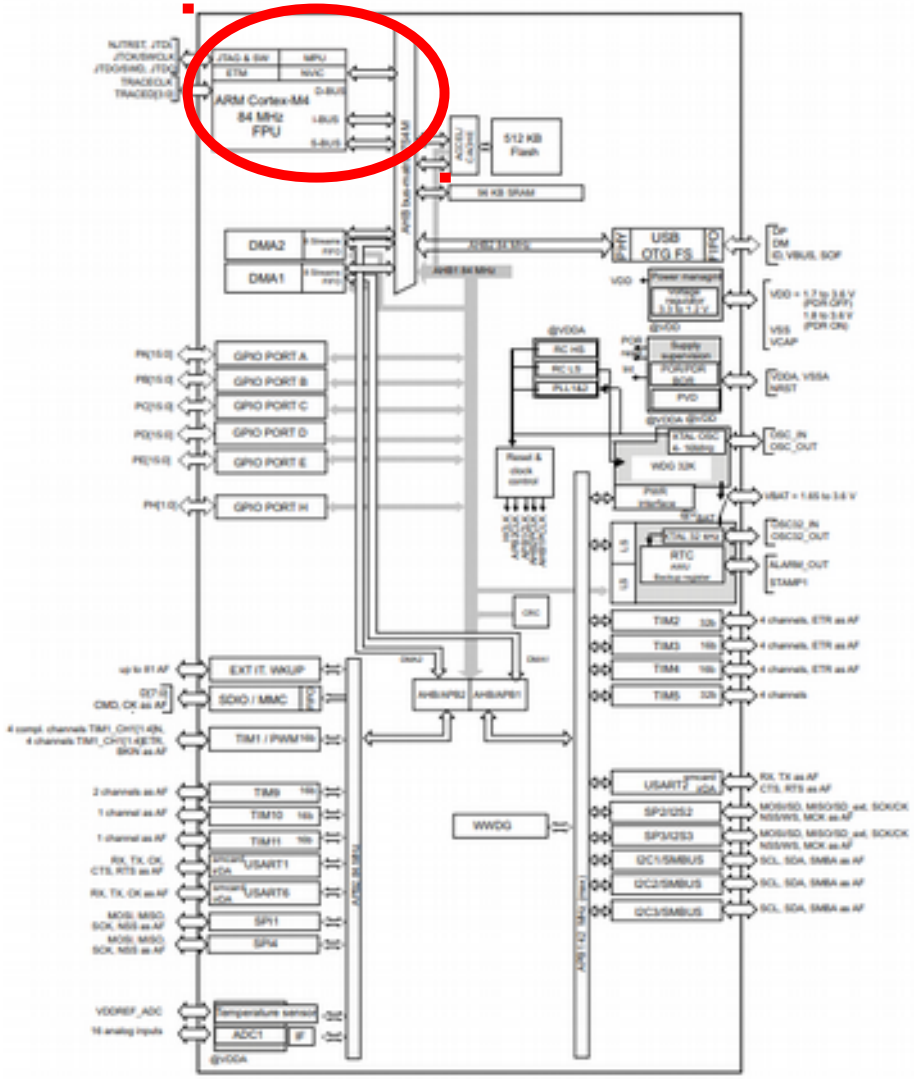
Architecture



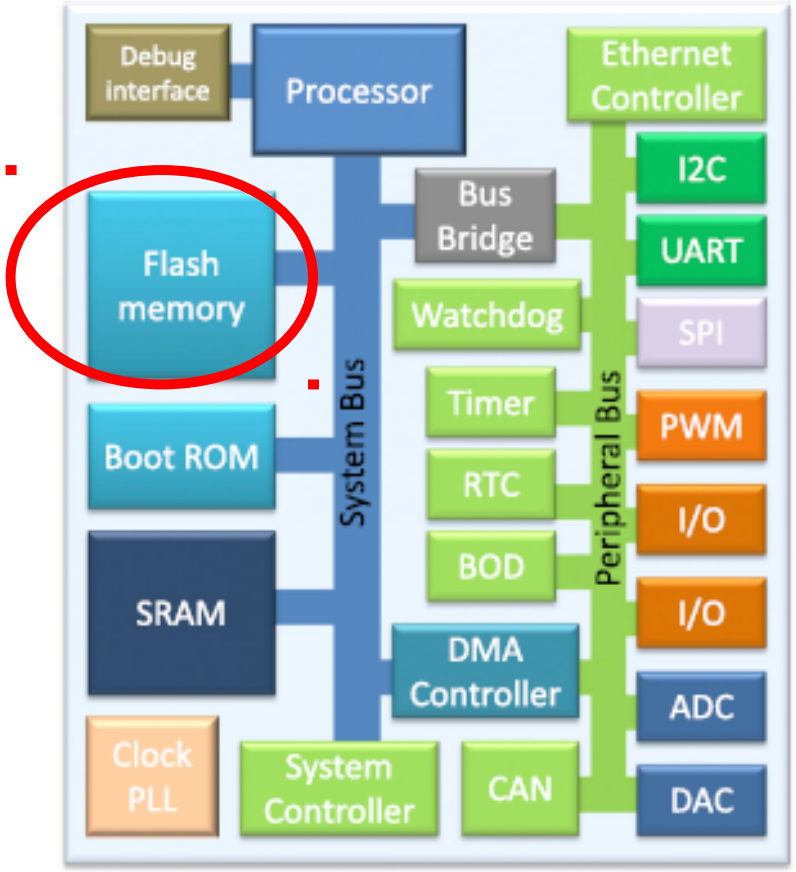


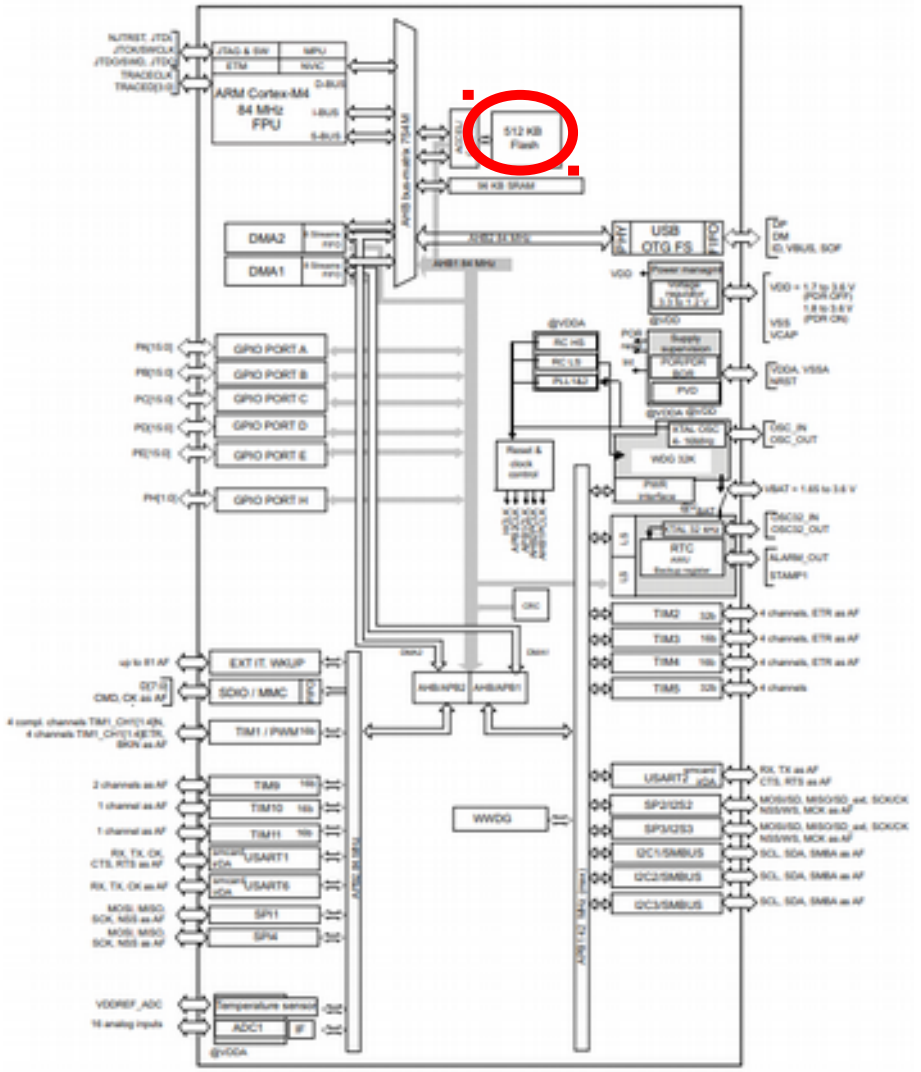
Architecture



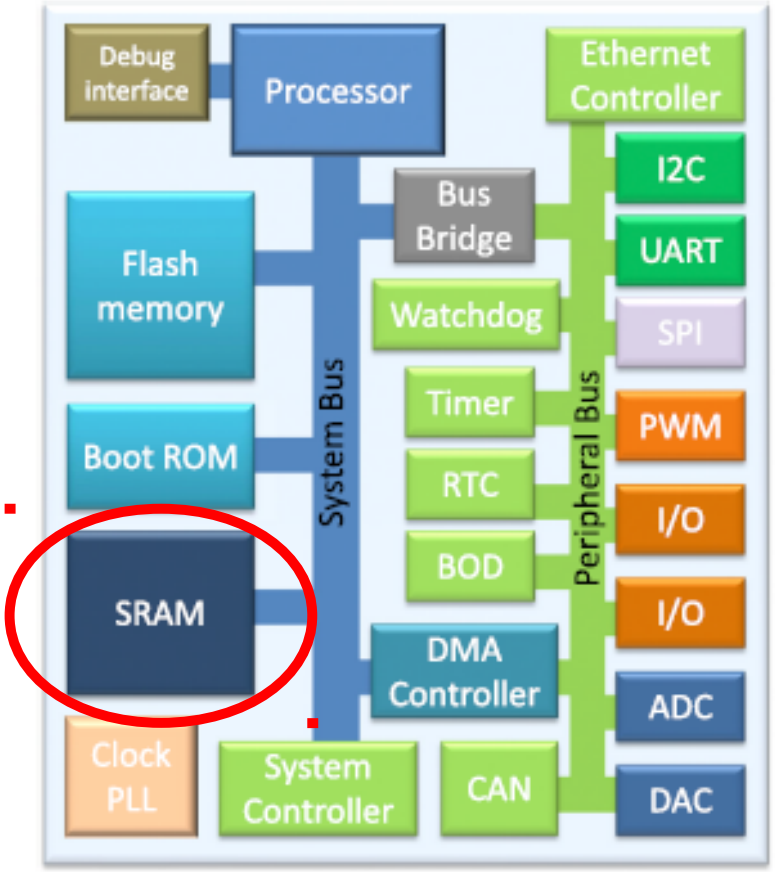


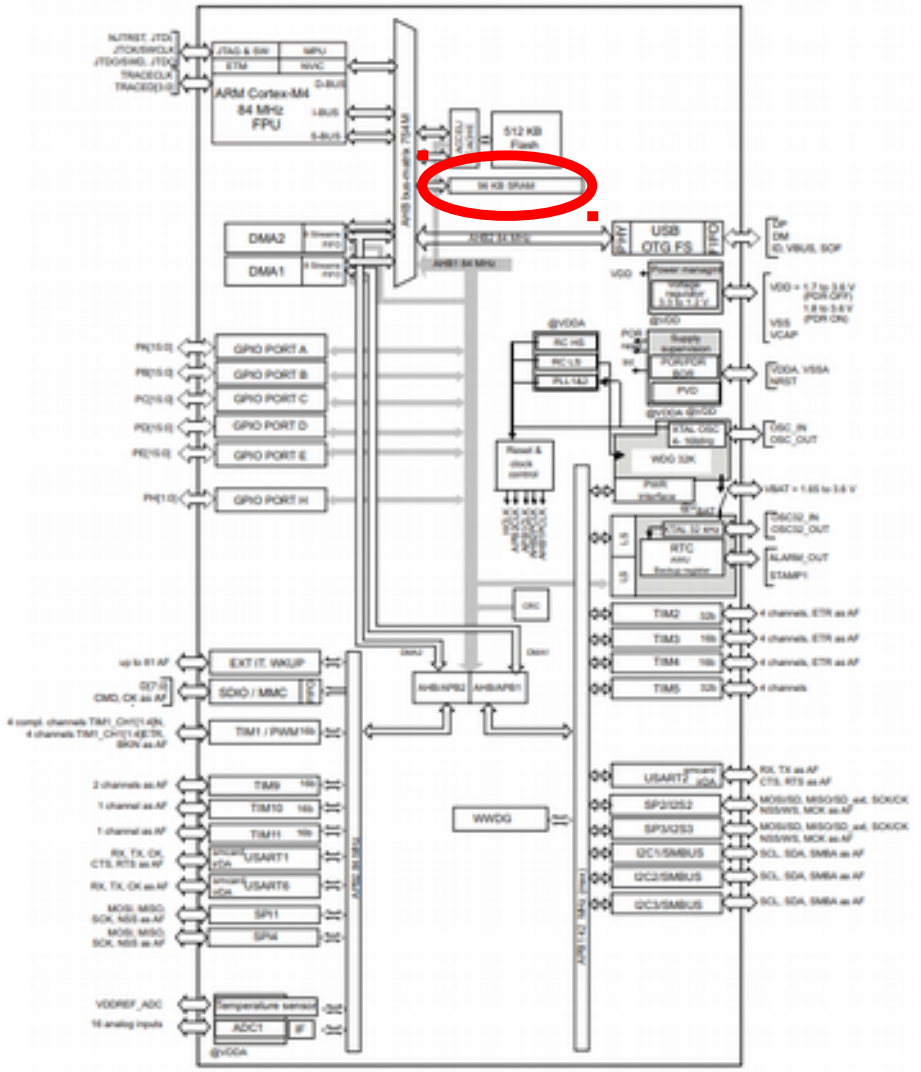
Architecture



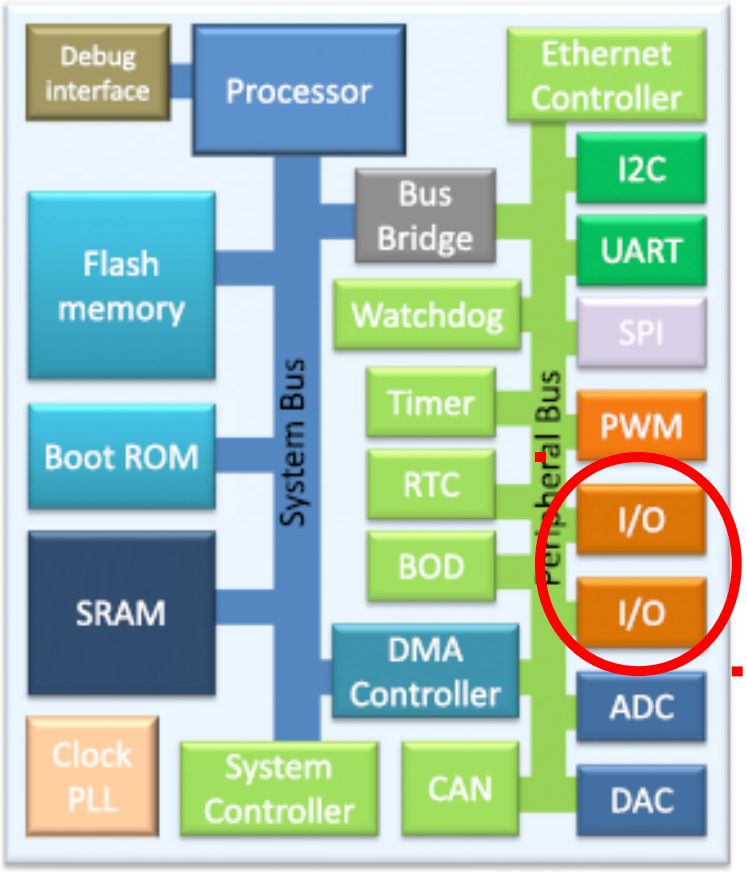


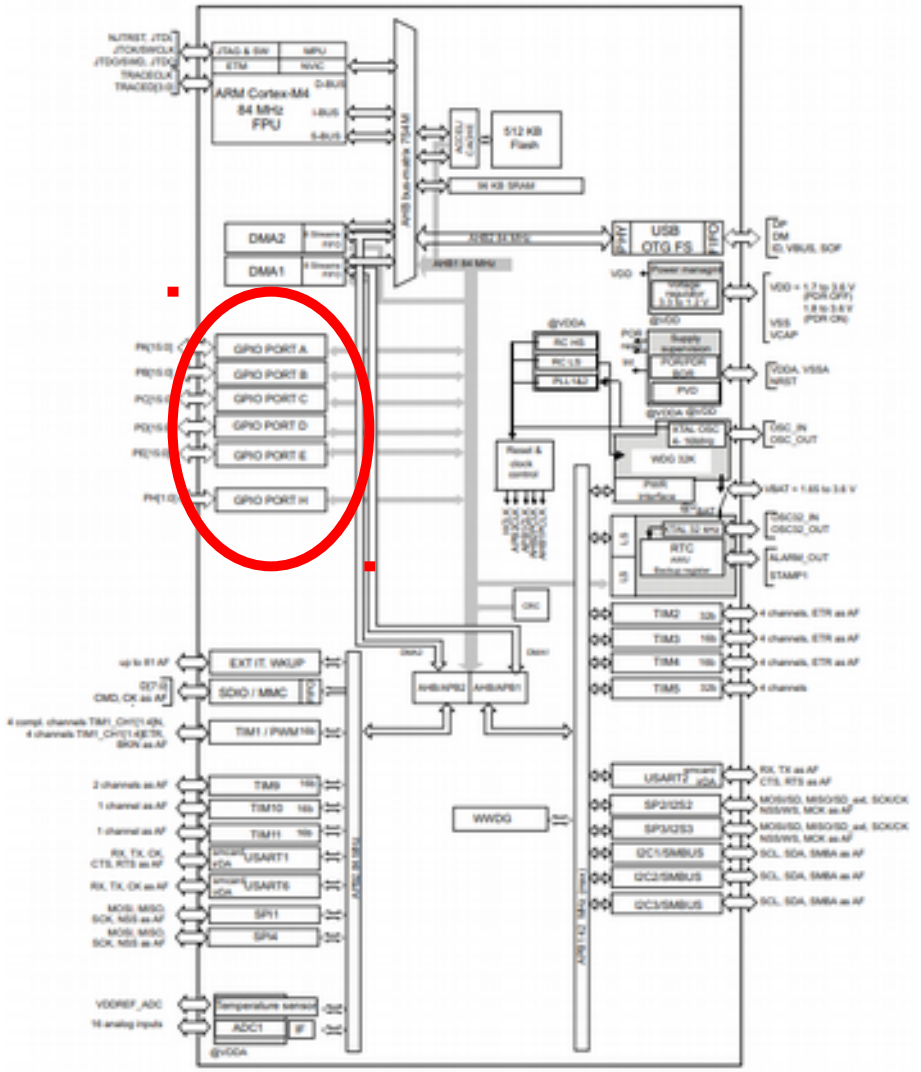
Architecture



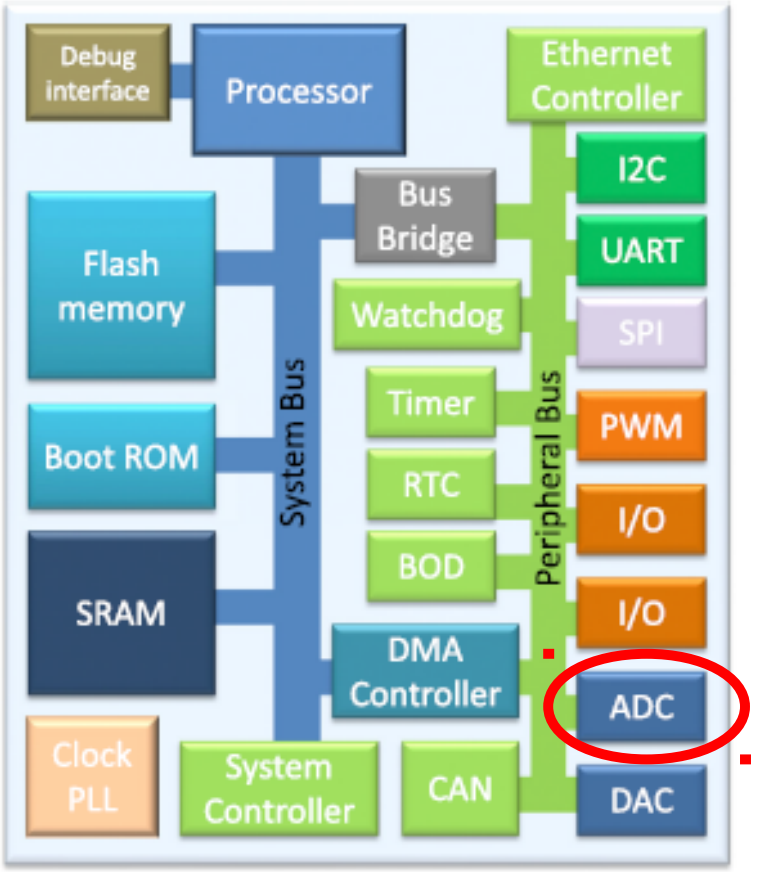


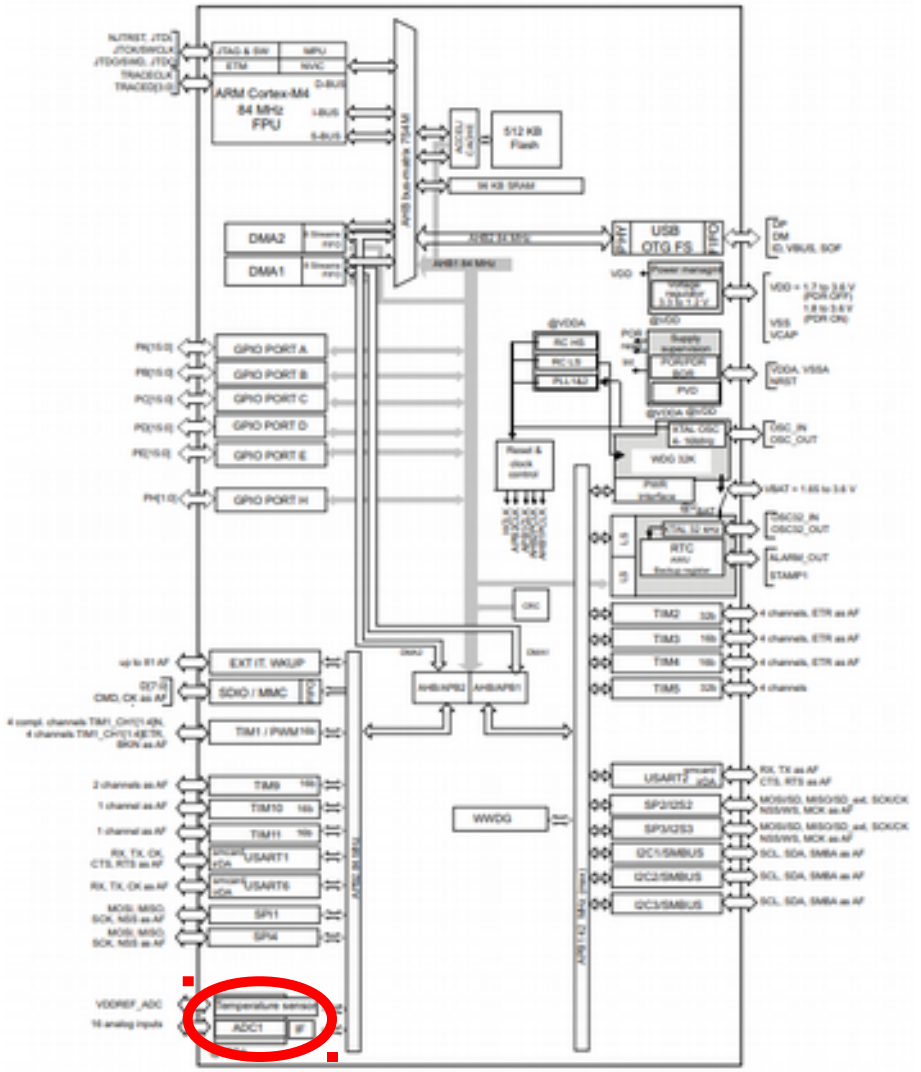
Architecture



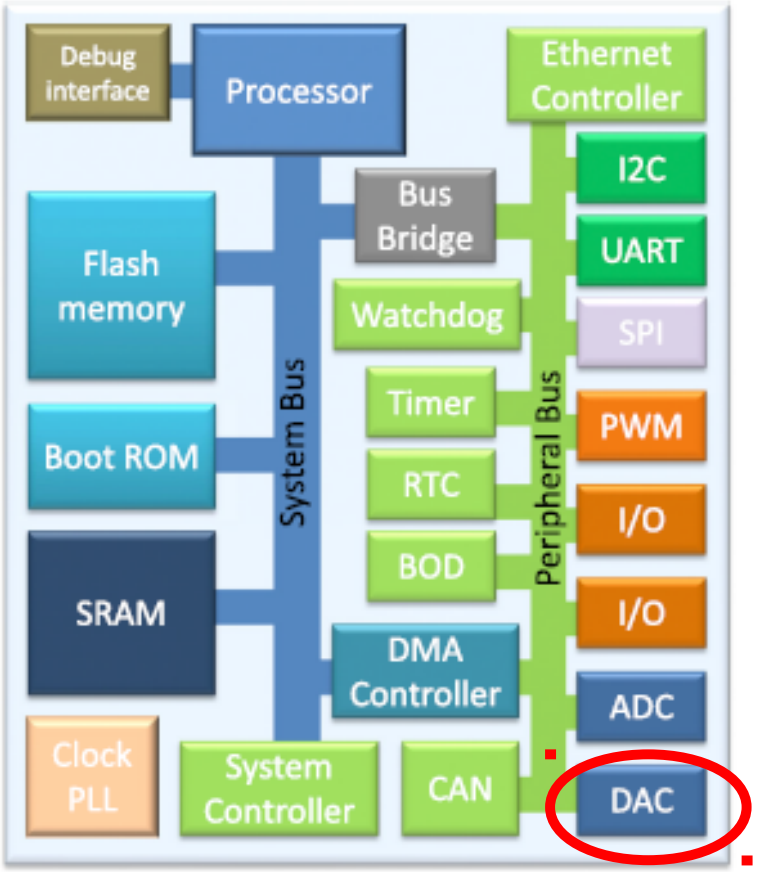


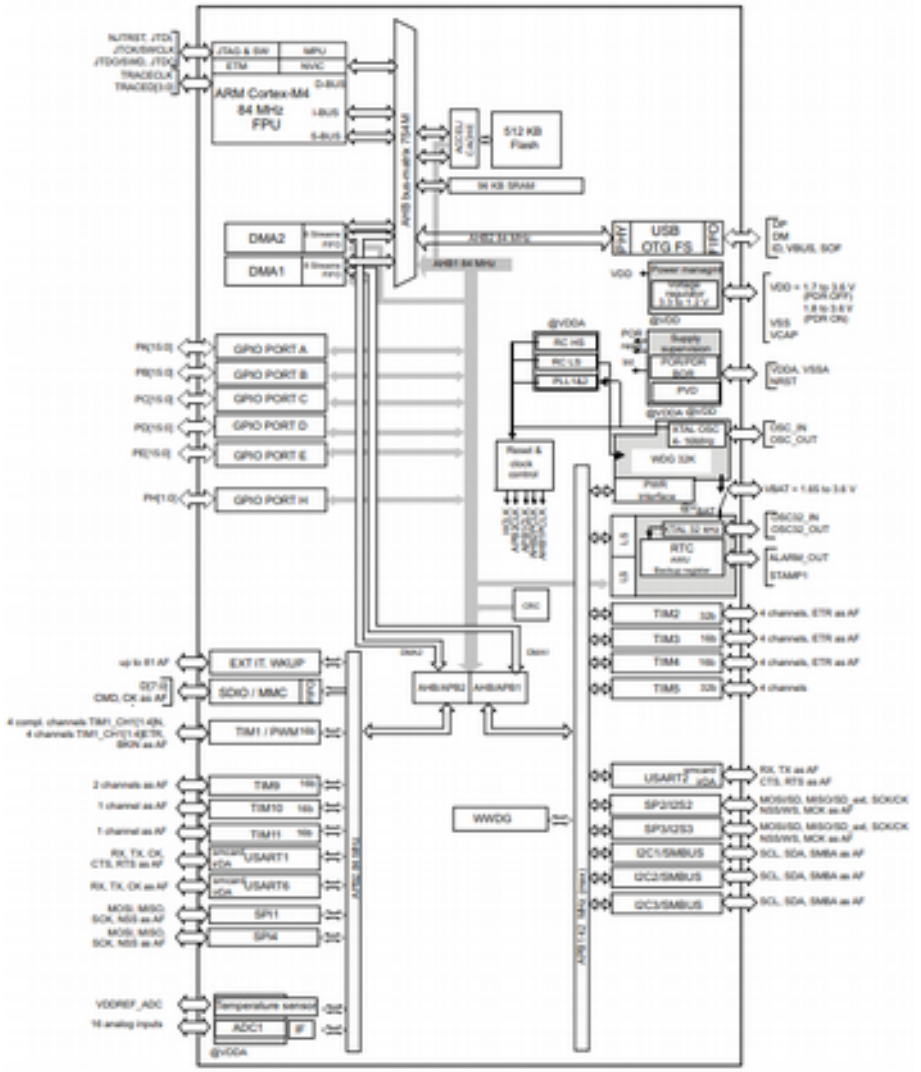
Architecture



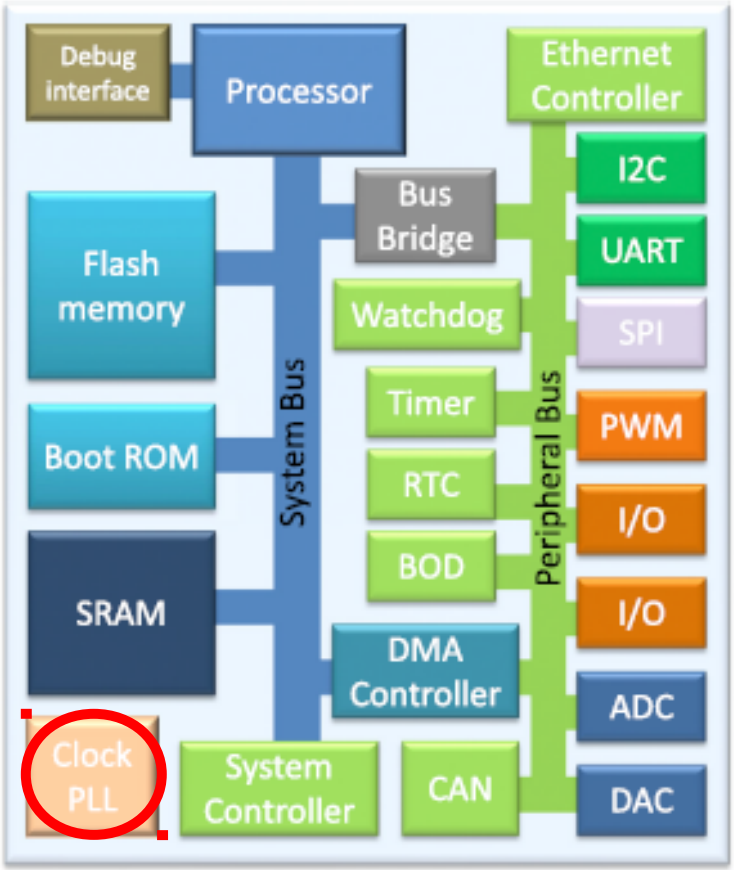


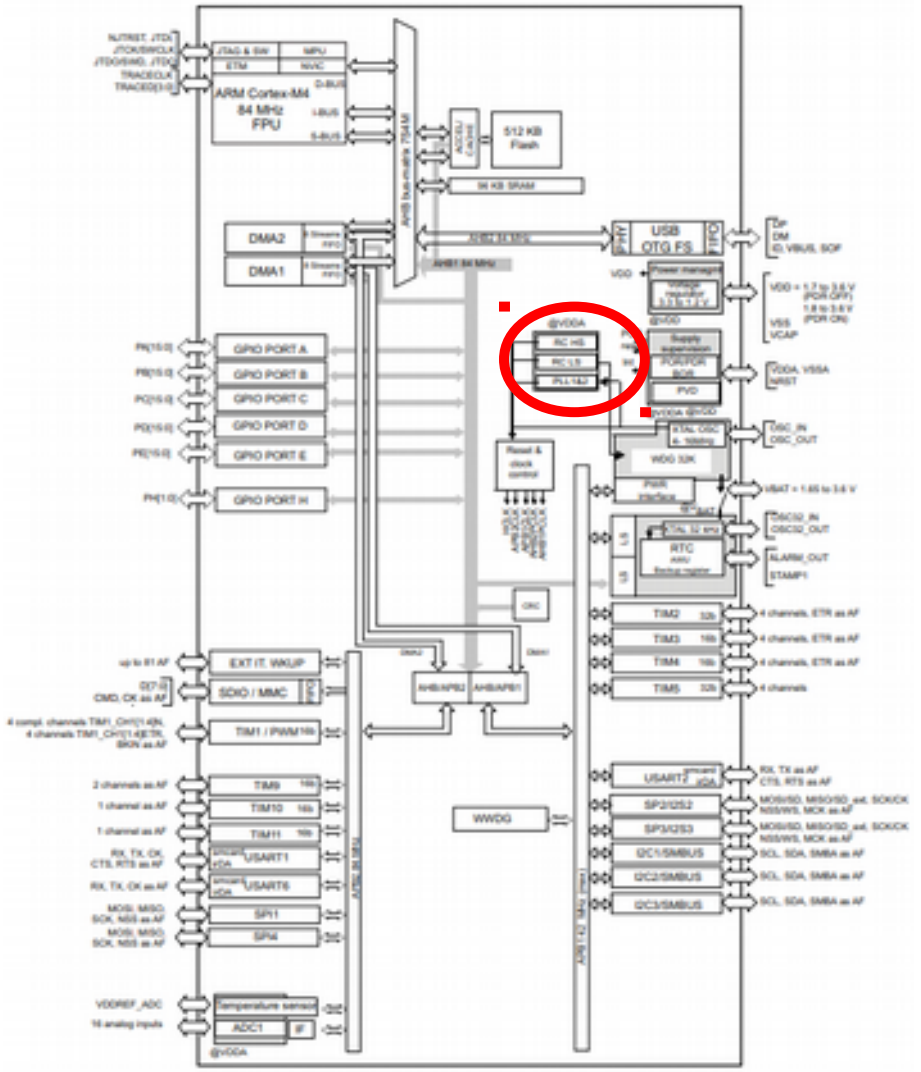
Architecture



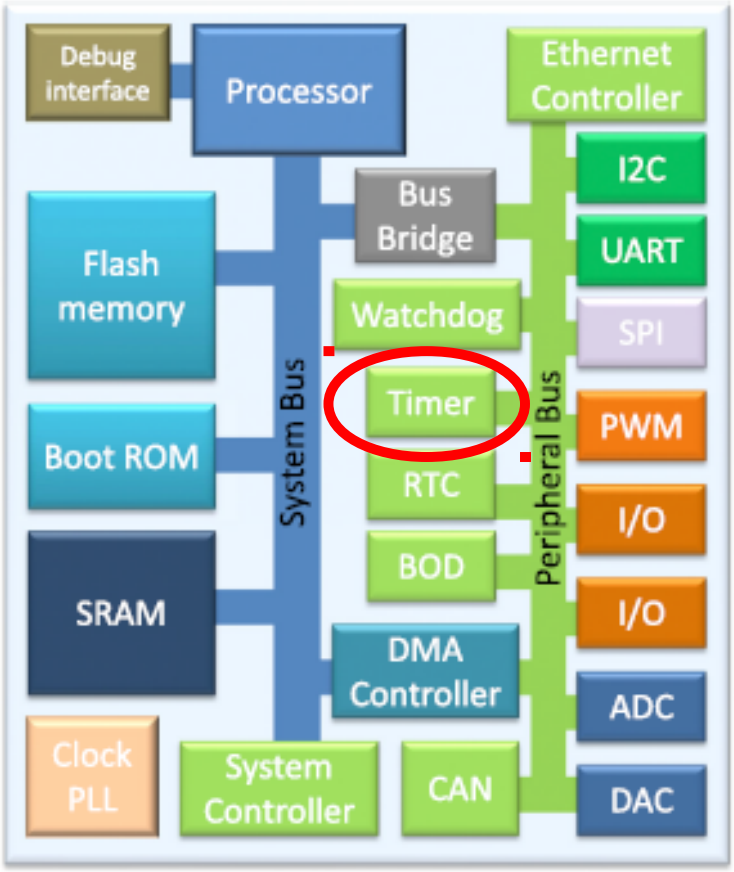


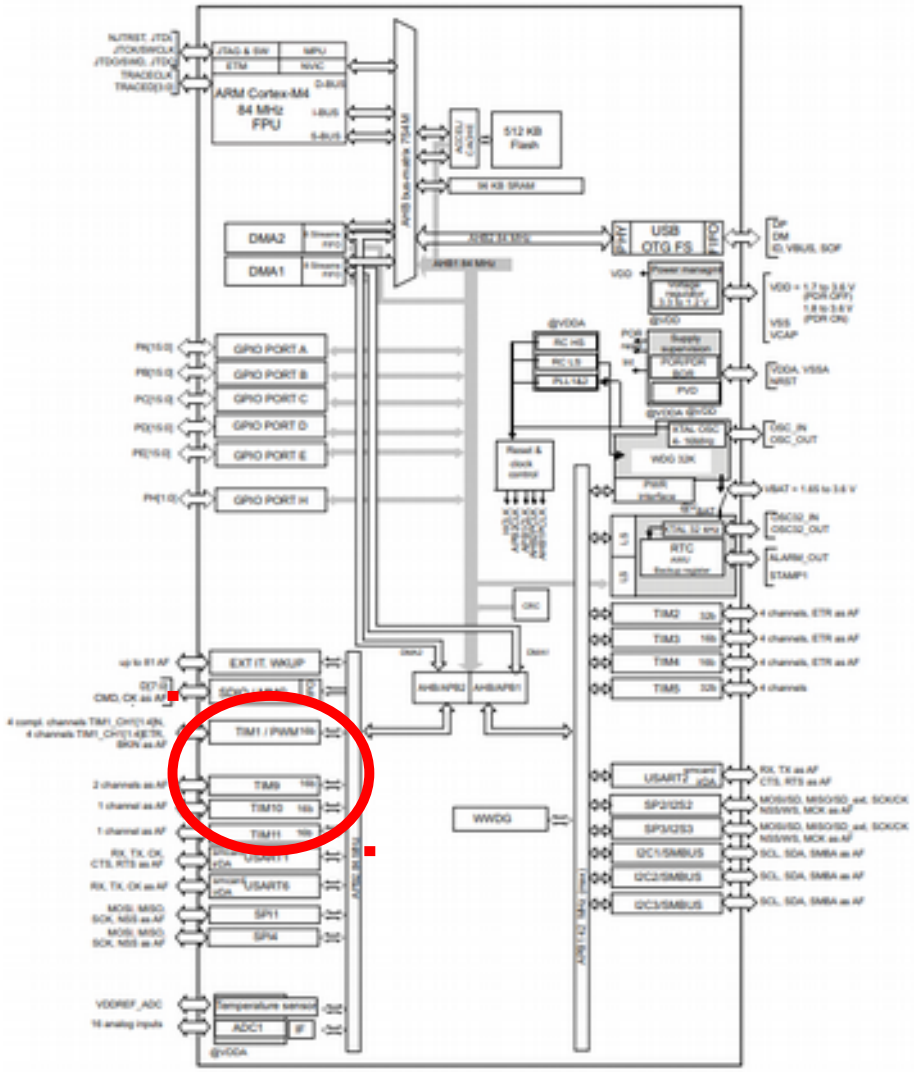
Architecture



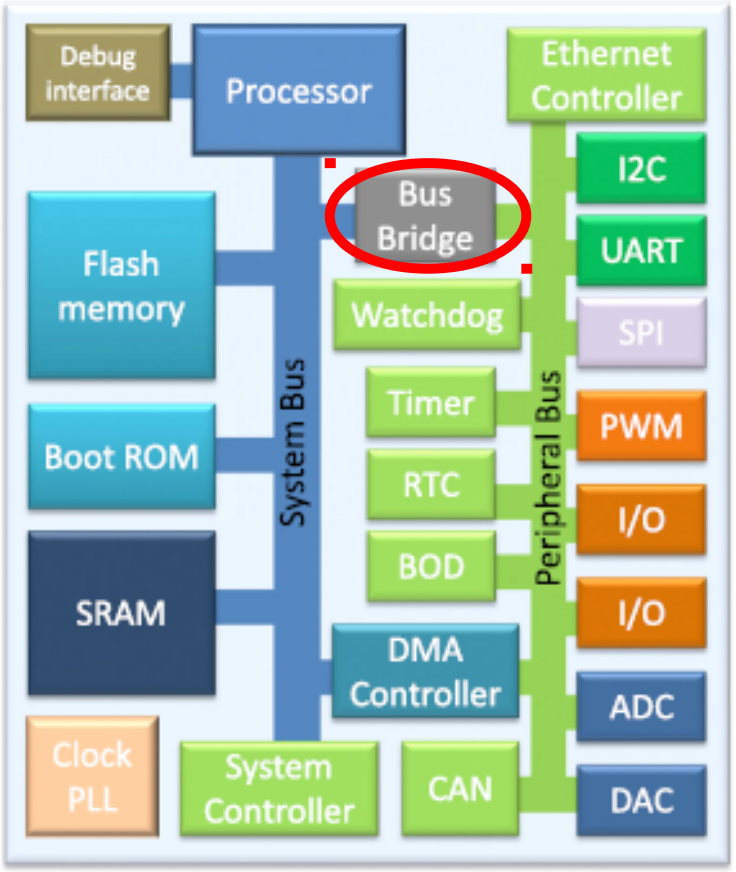


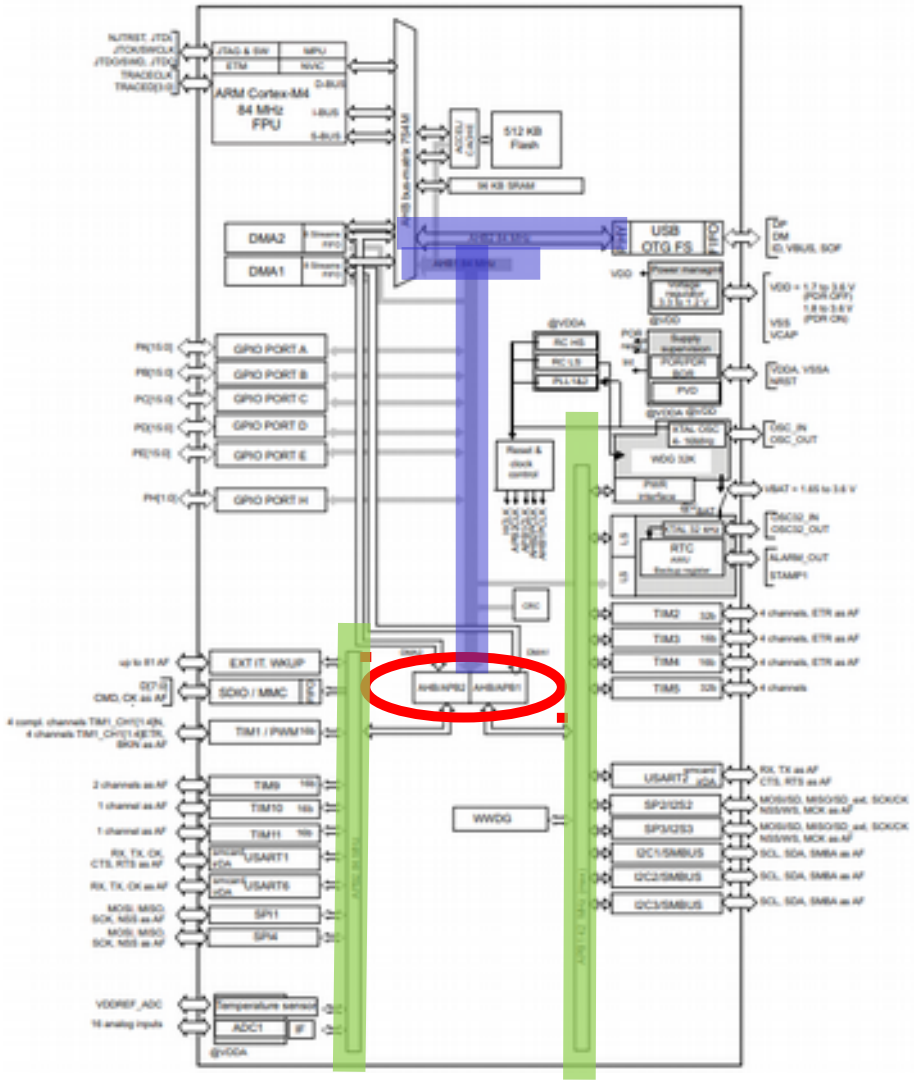
Architecture





Architecture



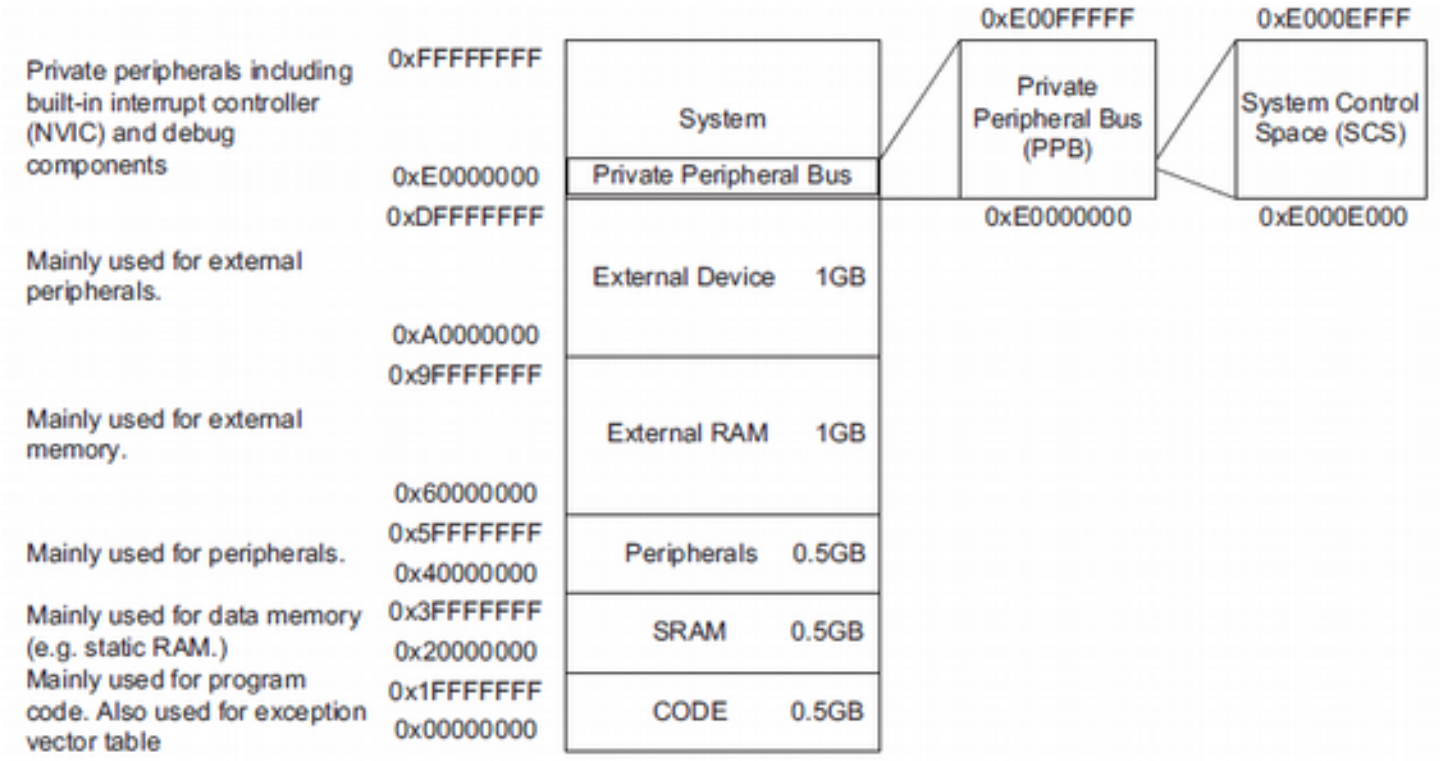


Cortex M-4

- Cortex M-4
  - Armv7-m architecture: Harvard architecture, 32-bit architecture (internal registers, data path, bus interface)
  - Thumb-2 instruction set (16/32 instructions)
  - Unified memory space 4GB
  - On-chip bus interfaces based on ARM AMBA
  - NVIC controller with priority levels (12 clock cycles)
  - Systick timer
  - Optimized for power consumption (alternatives: Cortex R or Cortex A)
  - Optional advanced debug features and MPU

Cortex M-4

Address space: 4GB, little/big endian



Cortex M-4

- Systick Timer
  - Part of the NVIC, 24-bit decrement timer
  - Sourced from a reference clock source (typ. on-chip)
  - Has its own exception hanlder
  - Can be used as system clock for an OS (task management, context switch)
  - Used for portability

Cortex M-4

- Power consumption:
  - Various sleep modes available
  - Commands: Wait For Event (WFE) / Wait For Interrupt (WFI)
  - Code stops running
  - Based on the sleep mode, clock signals can selectively be turned off:
    - Deeper sleep mode -> less peripherals running
    - Deeper sleep mode -> higher wakeup time
    - Deeper sleep mode -> less wake-up sources

Stm32L476 Lookup

- Clock Sources:
  - External 4-26 Mhz crystal osc. (HSE)
  - Internal 16Mhz factory-trimmed RC (HSI16)
  - Internal 32 Khz low power RC (LSI)
  - External 32 Khz crystal for RTC (LSE)
  - System PLL (uses HSE,HSI16) up to 84Mhz
- At startup, the MCU uses HSI at 16Mhz
- Clock sources managed by Reset and Clock Control (RCC) module

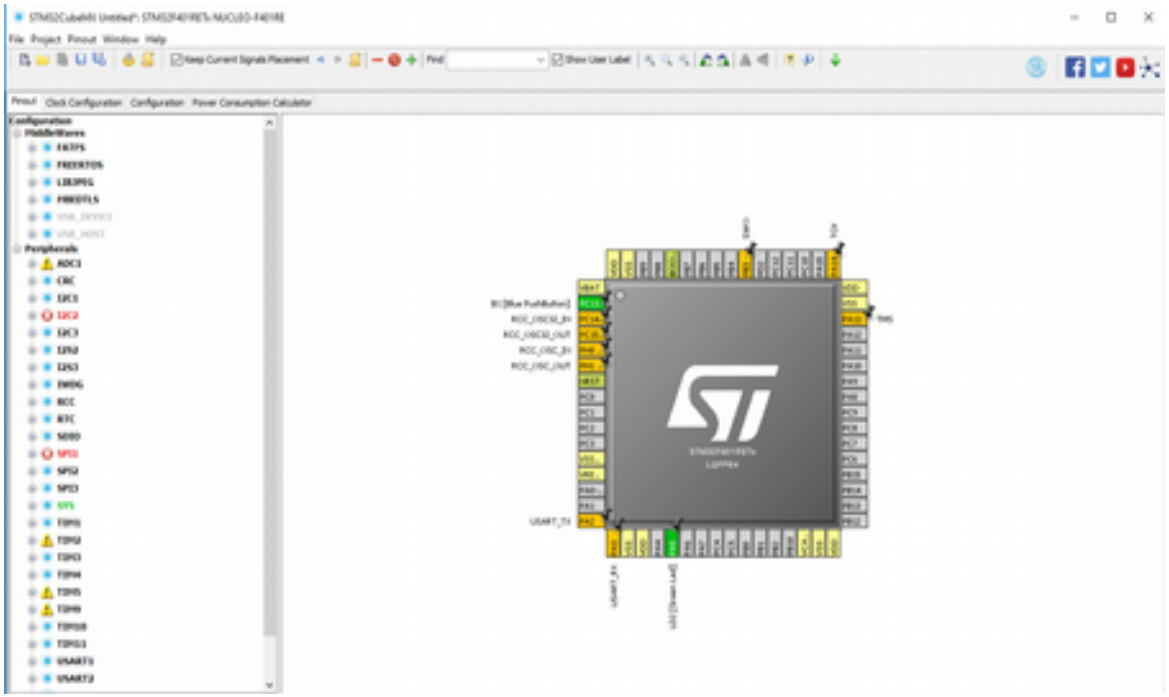
Stm32L476 Lookup

- Peripherals:
  - 11 x Timers
    - 6 x 16bit low power
    - 2 x 32bit
    - 2 x Watchdogs
    - 1 x Systick timer
  - 1 x RTC
  - 1 x ADC 12 bit
  - 2 x SAI Interfaces
  - 3 x I2C
  - 3 x USART
  - 4 x SPI (+ I2S)
  - 1 x DMA 16 ch.
  - 1 x SDIO
  - 1 x USB OTG FS
  - 81 x GPIO

Getting Started with CubeMX

STM CubeMX

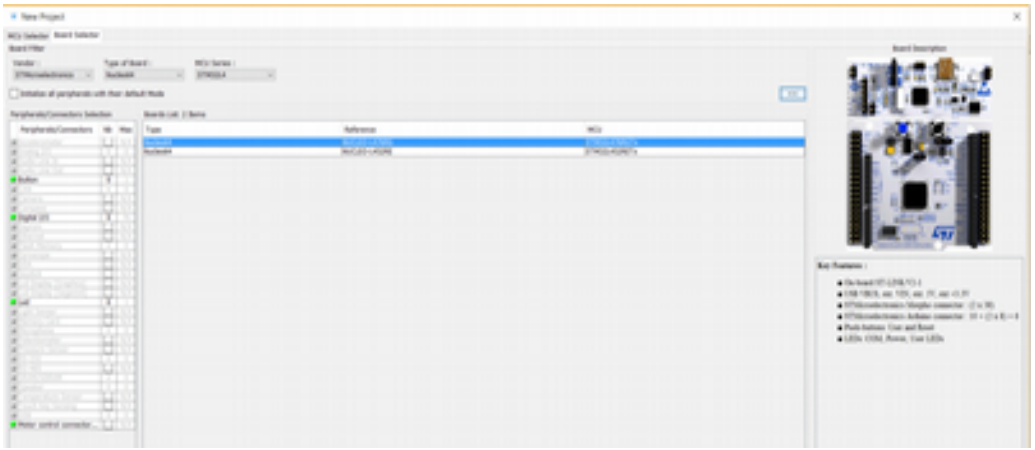
- Configuration tool:
  - Clock sources
  - Peripherals
  - Pinout
  - Middlewares
- Code generation:
  - IDE support



STM CubeMX

Usage Example: Clock and Timer 1 configuration

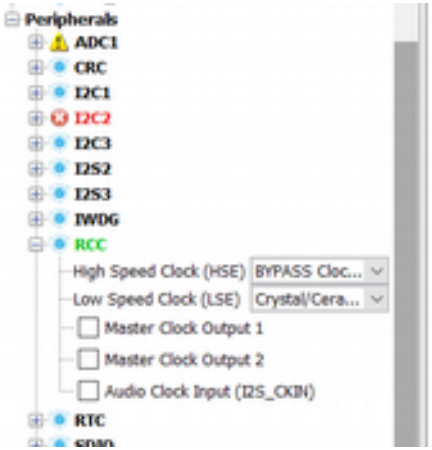
- Step 1:
  - Launch CubeMX
  - Select “New Project”
  - Choose “Board Selector”
  - Vendor “ST Microelectronics”
  - Type of Board “Nucleo 64”
  - MCU Series “Stm32F4”
  - Select “Nucleo-F401RE”
  - Double click on it

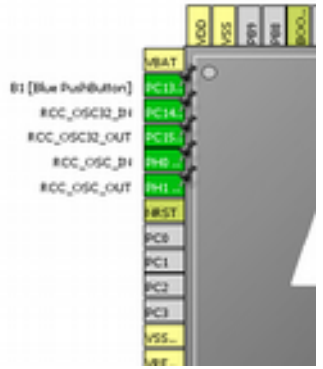


STM CubeMX

Usage Example: Clock and LPTimer 1 configuration

- Step 2:
  - From “Pinout” tab
  - Expand “RCC”
  - Select “Crystal/Ceramic resonator” in Low Speed Clock (LSE)
  - This will enable external 32Khz crystal of the Nucleo Board

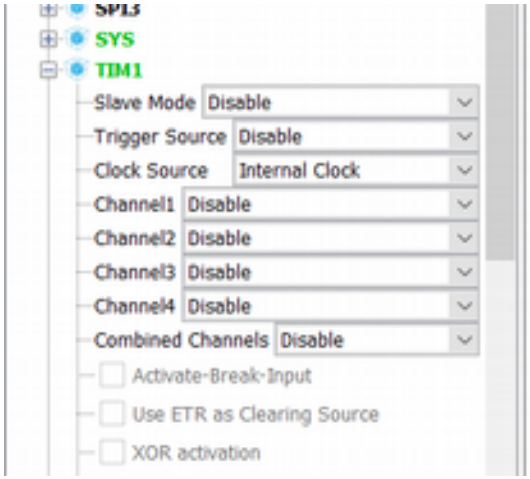


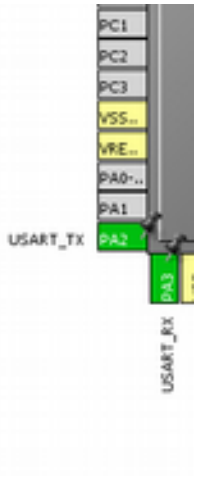


STM CubeMX

Usage Example: Clock and Timer 1 configuration

- Step 3:
  - From “Pinout” tab
  - Expand “TIM1”
  - Select “Internal Clock” as clock source

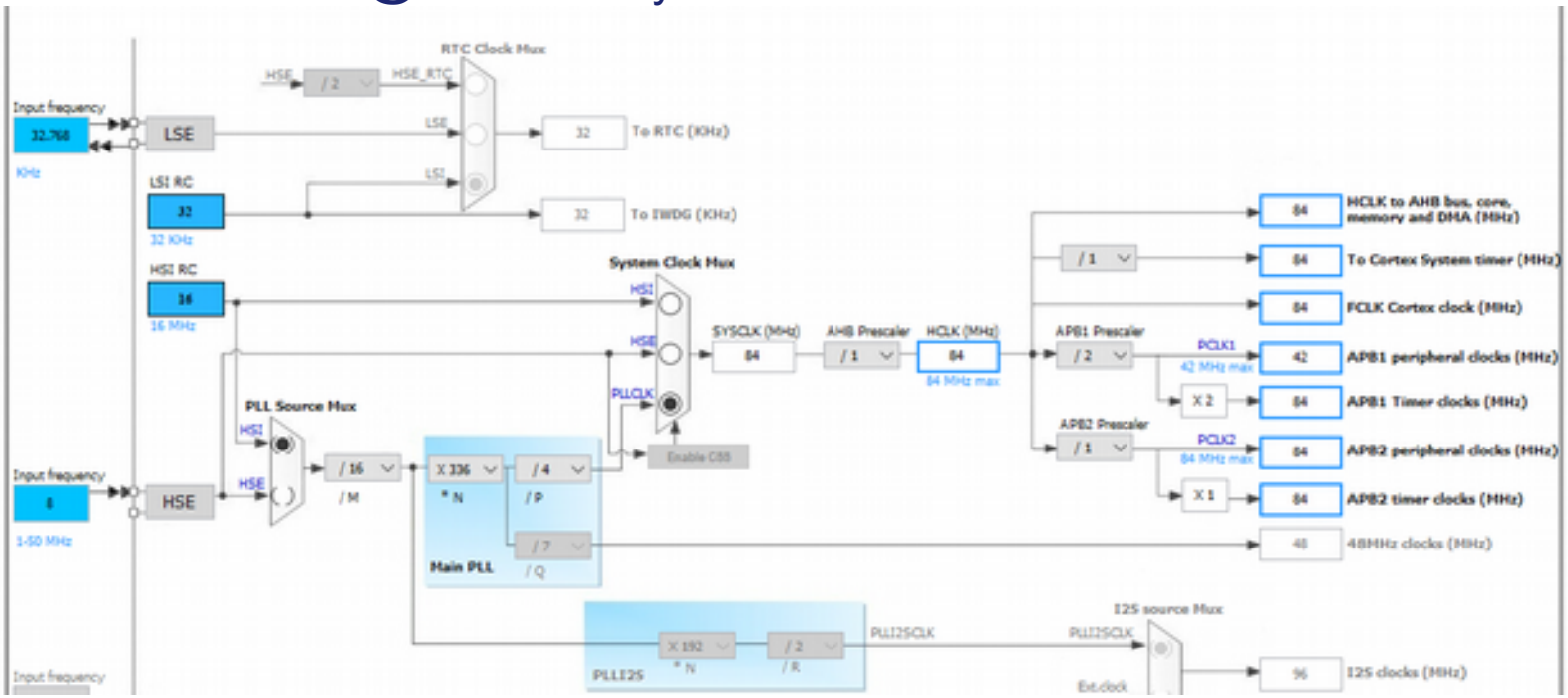




STM CubeMX

Usage Example: Clock and Timer 1 configuration

- Step 4:
  - From “Clock Configuration” tab
  - Leave HSI@84Mhz in System Clock Mux



STM CubeMX

Usage Example: Clock and Timer 1 configuration

- Step 5:
  - Q: Which bus is connected to TIM1?
  - Annotate its frequency

STM CubeMX

Usage Example: Clock and Timer 1 configuration

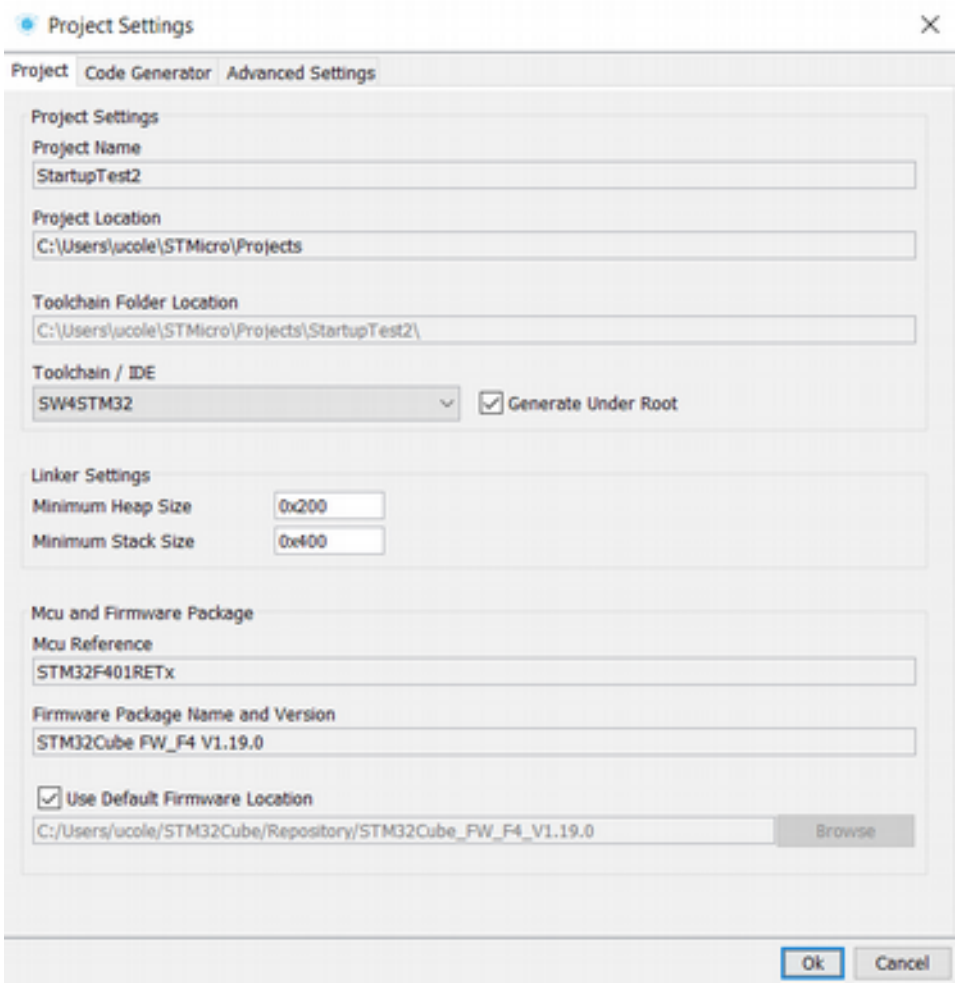
Step 7:

- From “Configuration” tab
- Check that peripherals and clocks are set correctly
- Double click on TIM1, select counter period to be 65535
- Q: What prescaler and division should we set for 1ms tick timer?
- NVIC settings enable TIM1 update interrupt

STM CubeMX

Usage Example: Code Generation

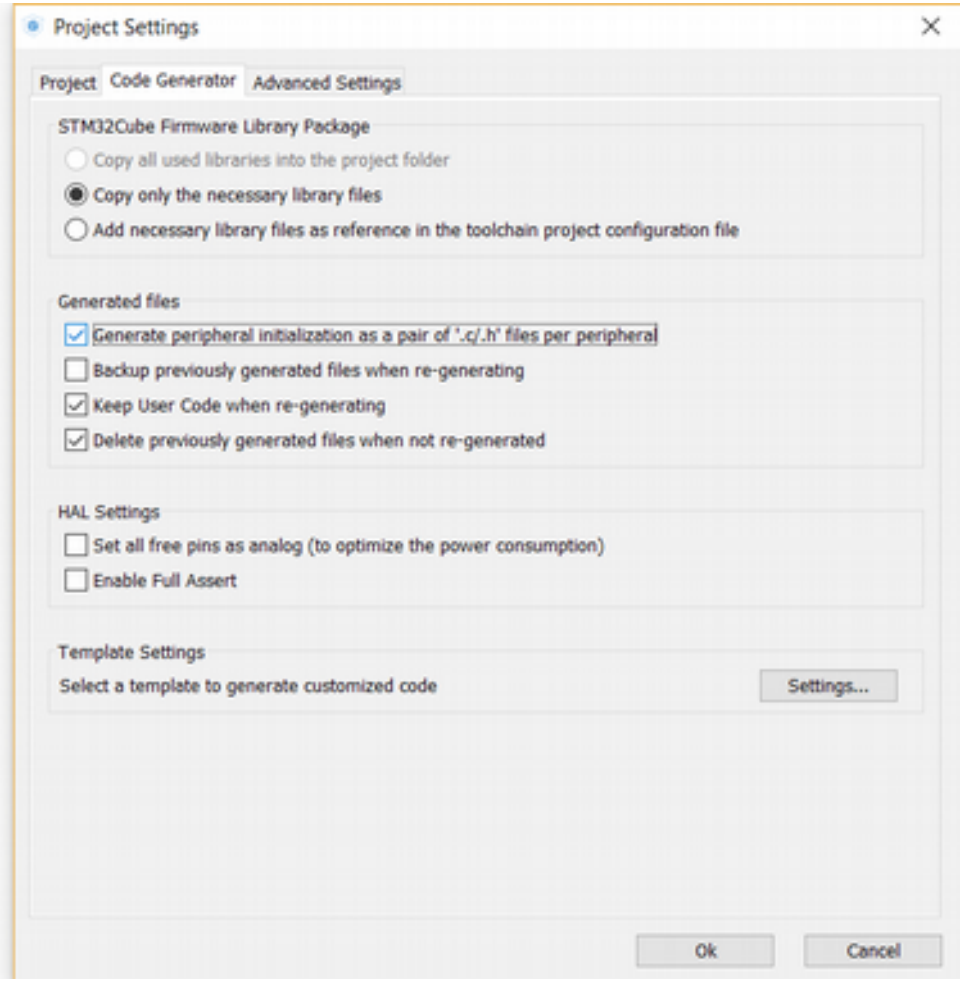
- Step 1:
  - Click on “Project” -> “Settings”
  - In “Project” tab
  - Set a project name
  - Select SW4STM32 IDE
  - Check that MCU and Firmware package are correct



STM CubeMX

Usage Example: Code Generation

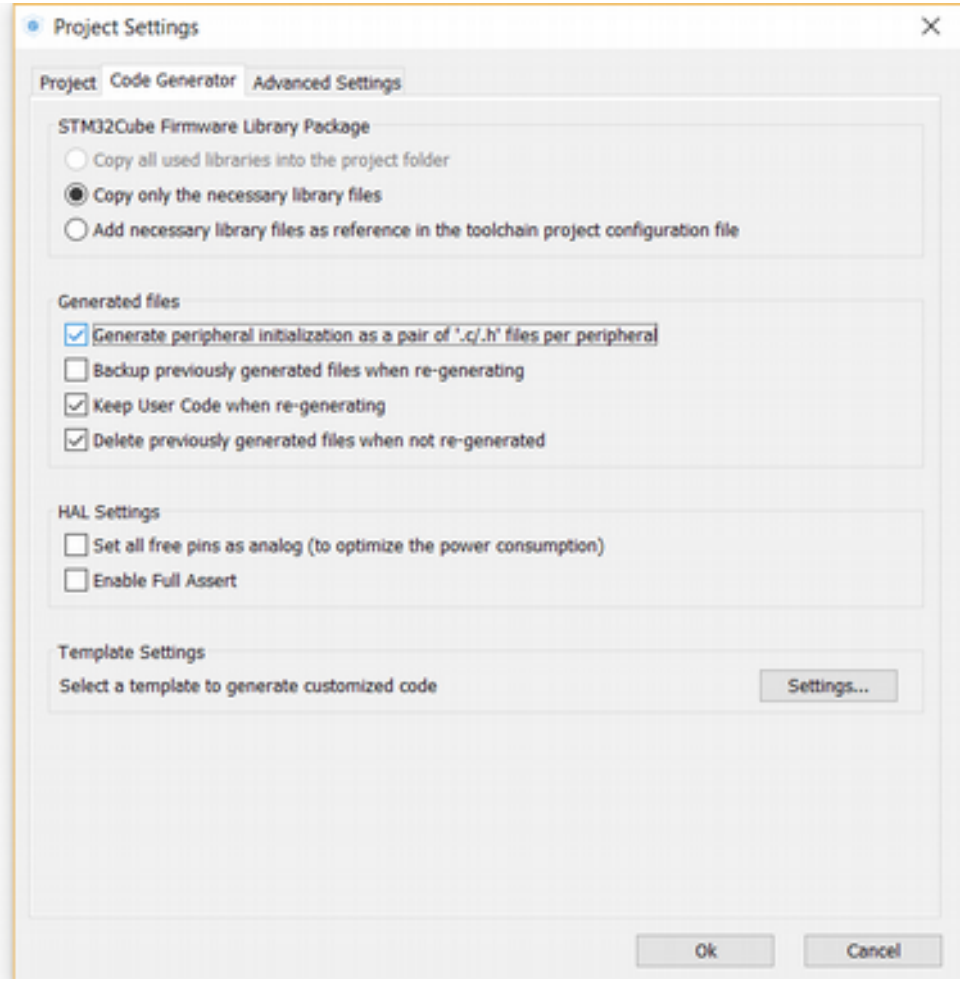
- Step 2:
  - In “Code Generator” tab
  - Select “Generate peripheral initialization…”
  - Keep other options unchanged
  - Click on “OK”



STM CubeMX

Usage Example: Code Generation

- Step 3:
  - Click on “Project” -> “Generate Code”
  - Wait the end of the execution
  - You can now import the project on System Workbench 4

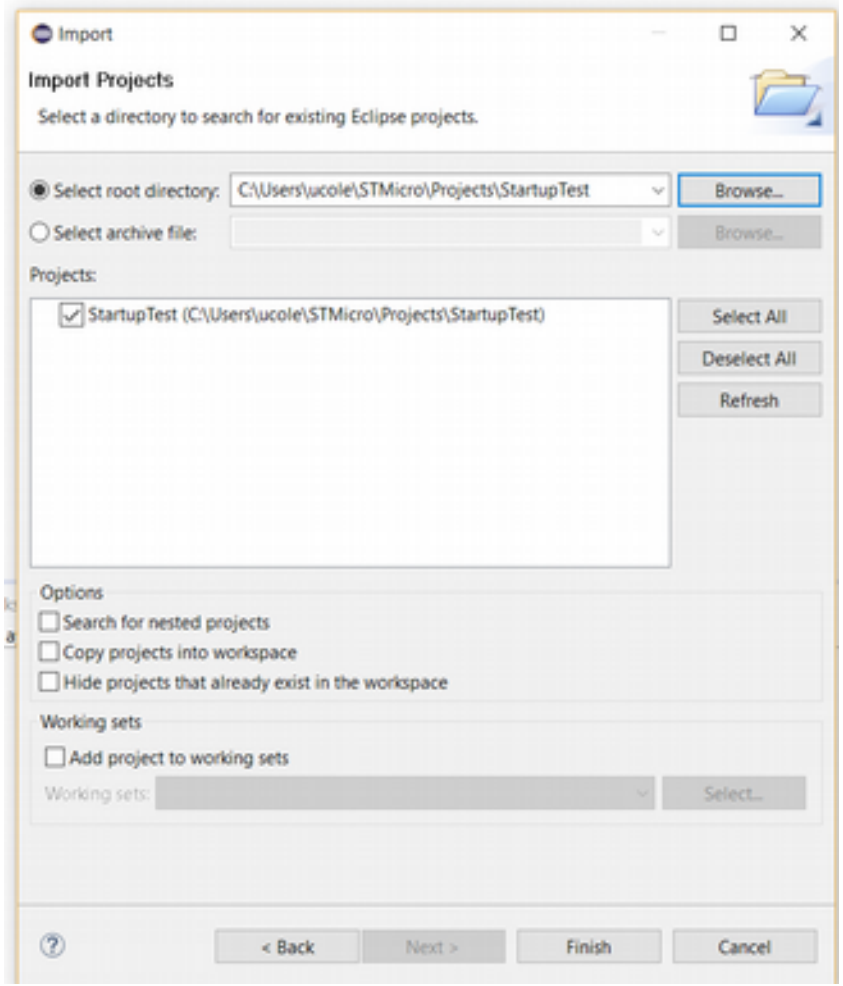


System Workbench 4

Importing project and debugging

Importing project generated with CubeMX

- Step 1:
  - Launch SW4STM32
  - In “File” menu click on “import…”
  - In “General”, select “Existing Project into Workspace”
  - Select the root folder generated with CubeMx
  - Keep default options and click finish



System Workbench 4

Importing project generated with CubeMX

- Step 2:
  - Right click on the project and select “Build Project”
  - Wait for compilation to finish and check that no errors were generated

Importing project generated with CubeMX

- Step 3:
  - Plug the nucleo
  - Right click on the project and select “Debug as”
  - When prompted to switch in debug view click yes (check the “keep option” if you don’t want to repeat this step each time)
  - The code will halt on HAL_Init()
  - Click on “step over” or “step into” to get familiar with the IDE in debugging mode
  - You can click on “Resume” if you want your code to freely run (but it won’t do anything since it’s empty ☺)

System Workbench 4

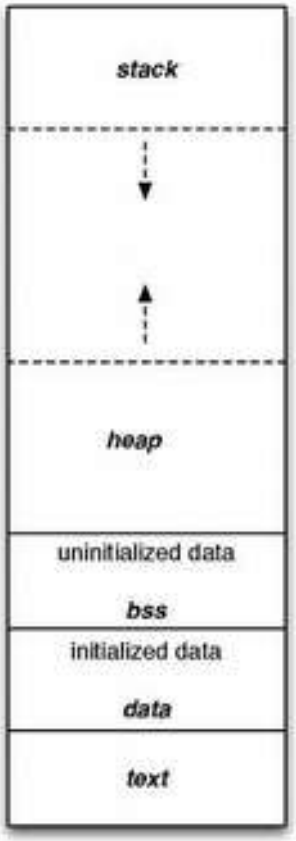
Important files: the linker script

- In project explorer: STM32F401RETx_FLASH.ld
- Where to find program and data memory (RAM,FLASH) w.r.t. the linear memory map of the MCU
- What to put inside each area (e.g., .isr_vector, .text and constant data in flash, .data and .bss in ram etc…)

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20018000;    /* end of RAM */
/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 96K
FLASH (rx)      : ORIGIN = 0x8000000, LENGTH = 512K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH
```

Important files: the linker script

- Quick recall on memory segments:



Automatic variables, returned address …

```
int dummy_func(int arg1){
        int arg2;
        if(arg2 > arg1) return arg2;
        else return arg1;
}
```

Dynamic memory allocation (e.g., malloc,…)

Global or static variables initialized to 0 or not explicitly initialized

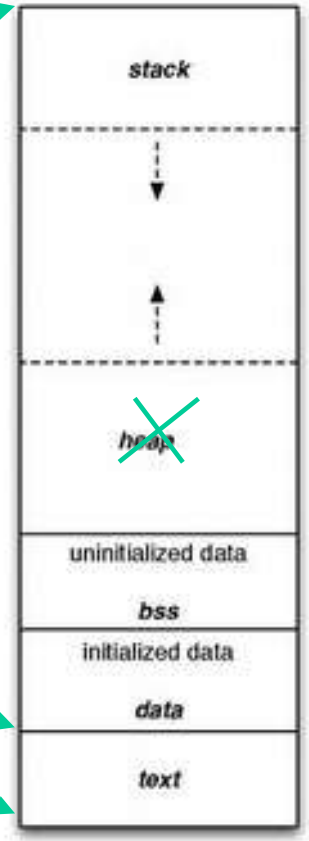
Global or static variable with pre-defined value and that can be modified

Read only data (e.g., code)

Important files: the linker script

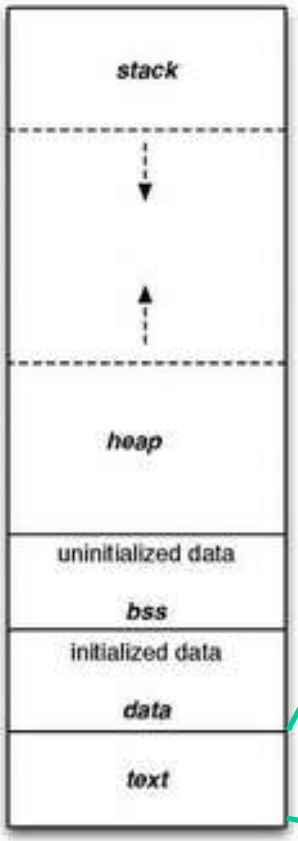
- Quick recall on memory segments:

```
35 /* Highest address of the user mode stack */
36 _estack = 0x20018000;    /* end of RAM */
37 /* Generate a link error if heap and stack don't fit into RAM */
38 _Min_Heap_Size = 0x200;      /* required amount of heap  */
39 _Min_Stack_Size = 0x400; /* required amount of stack */
40
41 /* Specify the memory areas */
42 MEMORY
43 {
44 RAM (xrw)      : ORIGIN = 0x20000000, LENGTH = 96K
45 FLASH (rx)      : ORIGIN = 0x8000000, LENGTH = 512K
46 }
```



Important files: the linker script

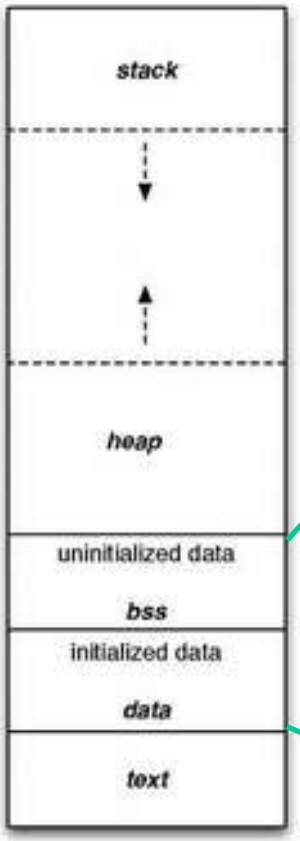
- Quick recall on memory segments:




```
/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(8);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(8);
  } >FLASH

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(8);
    *(.text)           /* .text sections (code) */
    *(.text*)          /* .text* sections (code) */
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(8);
    _etext = .;        /* define a global symbols at end of code */
  } >FLASH

  /* Constant data goes into FLASH */
  .rodata :
  {
    . = ALIGN(8);
    *(.rodata)         /* .rodata sections (constants, strings, etc.) */
    *(.rodata*)        /* .rodata* sections (constants, strings, etc.) */
    . = ALIGN(8);
  } >FLASH
```

Important files: the linker script

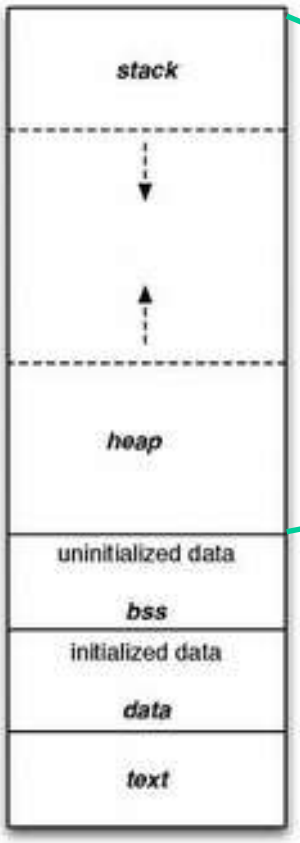
- Quick recall on memory segments:




```
/* used by the startup to initialize data */
_sidata = LOADADDR(.data);

/* Initialized data sections goes into RAM, load LMA copy after code */
.data :
{
  . = ALIGN(8);
  _sdata = .;        /* create a global symbol at data start */
  *(.data)           /* .data sections */
  *(.data*)          /* .data* sections */

  . = ALIGN(8);
  _edata = .;        /* define a global symbol at data end */
} >RAM AT> FLASH


/* Uninitialized data section */
. = ALIGN(4);
.bss :
{
  /* This is used by the startup in order to initialize the .bss secion */
  _sbss = .;         /* define a global symbol at bss start */
  __bss_start__ = _sbss;
  *(.bss)
  *(.bss*)
  *(COMMON)

  . = ALIGN(4);
  _ebss = .;         /* define a global symbol at bss end */
  __bss_end__ = _ebss;
} >RAM
```

Important files: the linker script

- Quick recall on memory segments:




```
/* User_heap_stack section, used to check that there is enough RAM left */
._user_heap_stack :
{
  . = ALIGN(8);
  PROVIDE ( end = . );
  PROVIDE ( _end = . );
  . = . + _Min_Heap_Size;
  . = . + _Min_Stack_Size;
  . = ALIGN(8);
} >RAM
```

System Workbench 4

Important files: the startup file

- startup/startup_stm32f401xe.s
- Written in assembly, it holds the reset handler (first code to be executed) and the vector table

```
76
77      .section    .text.Reset_Handler
78      .weak   Reset_Handler
79      .type   Reset_Handler, %function
80  Reset_Handler:
81    ldr   sp, =_estack    /* Atollic update: set stack pointer */
82
83  /* Copy the data segment initializers from flash to SRAM */
84    movs  r1, #0
85    b LoopCopyDataInit
86
87  CopyDataInit:
88      ldr r3, =_sidata
89      ldr r3, [r3, r1]
90      str r3, [r0, r1]
91      adds    r1, r1, #4
92
93  LoopCopyDataInit:
94      ldr r0, =_sdata
95      ldr r3, =_edata
96      adds    r2, r0, r1
97      cmp r2, r3
```

System Workbench 4

Important files: the system file

- Src/system_stm32f4xx.c
- SystemInit function for clock and vector table initialization
- Other clock utilities…

```
198 void SystemInit(void)
199 {
200   /* FPU settings ------------------------------------------------------------*/
201   #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
202     SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
203   #endif
204   /* Reset the RCC clock configuration to the default reset state ------------*/
205   /* Set MSION bit */
206   RCC->CR |= RCC_CR_MSION;
207
208   /* Reset CFGR register */
209   RCC->CFGR = 0x00000000;
210
211   /* Reset HSEON, CSSON , HSION, and PLLON bits */
212   RCC->CR &= (uint32_t)0xEAF6FFFF;
213
214   /* Reset PLLCFGR register */
215   RCC->PLLCFGR = 0x00001000;
216
217   /* Reset HSEBYP bit */
218   RCC->CR &= (uint32_t)0xFFFBFFFF;
219
220   /* Disable all interrupts */
221   RCC->CIER = 0x00000000;
222
223   /* Configure the Vector Table location add offset address ------------------*/
224 #ifdef VECT_TAB_SRAM
225   SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
226 #else
227   SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
228 #endif
229 }
```

System Workbench 4

Important files: peripherals initialization

- Src/tim.c
- Src/gpio.c

- MX_[peripheral]_init: high-level init

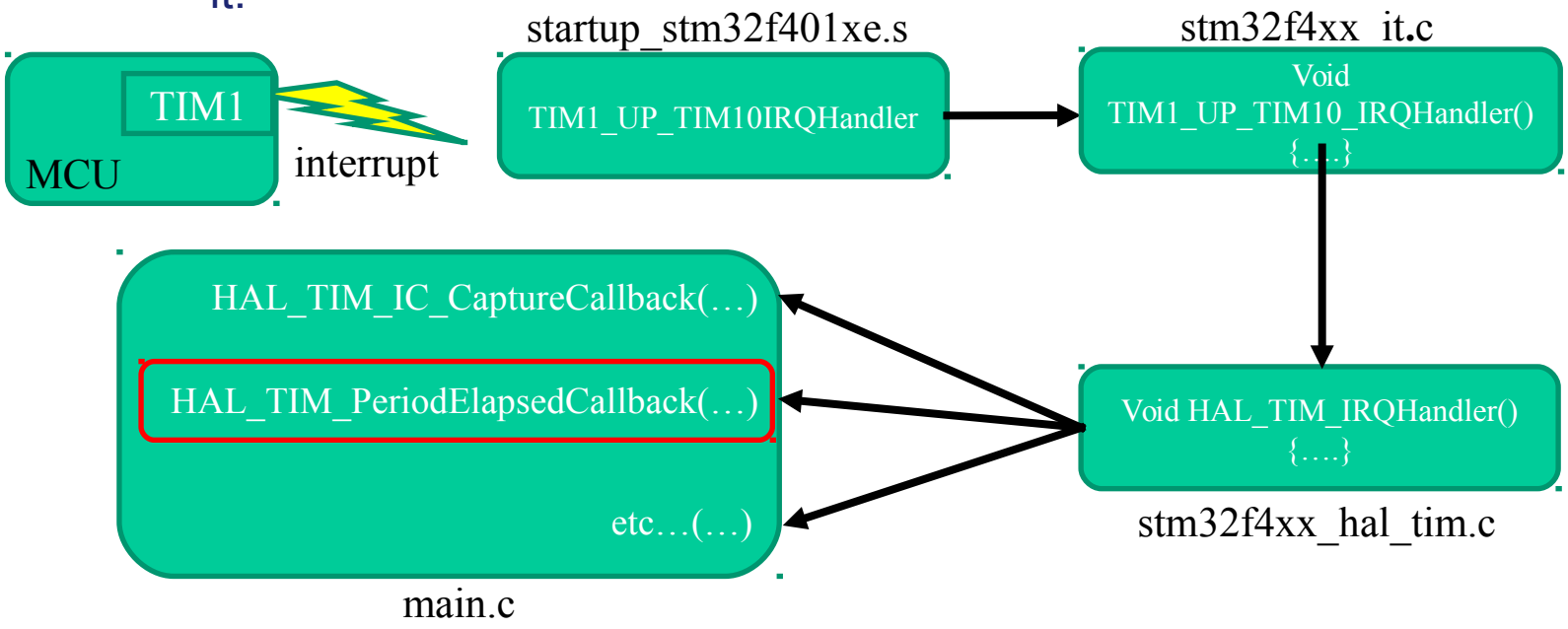
- HAL_[peripheral]_init: low-level init

Important files: Hardware Abstraction Layer drivers

- Drivers/STM32L4xx_HAL_Driver/stm32l4xx_hal_[peripheral].c

Important files: interrupt management

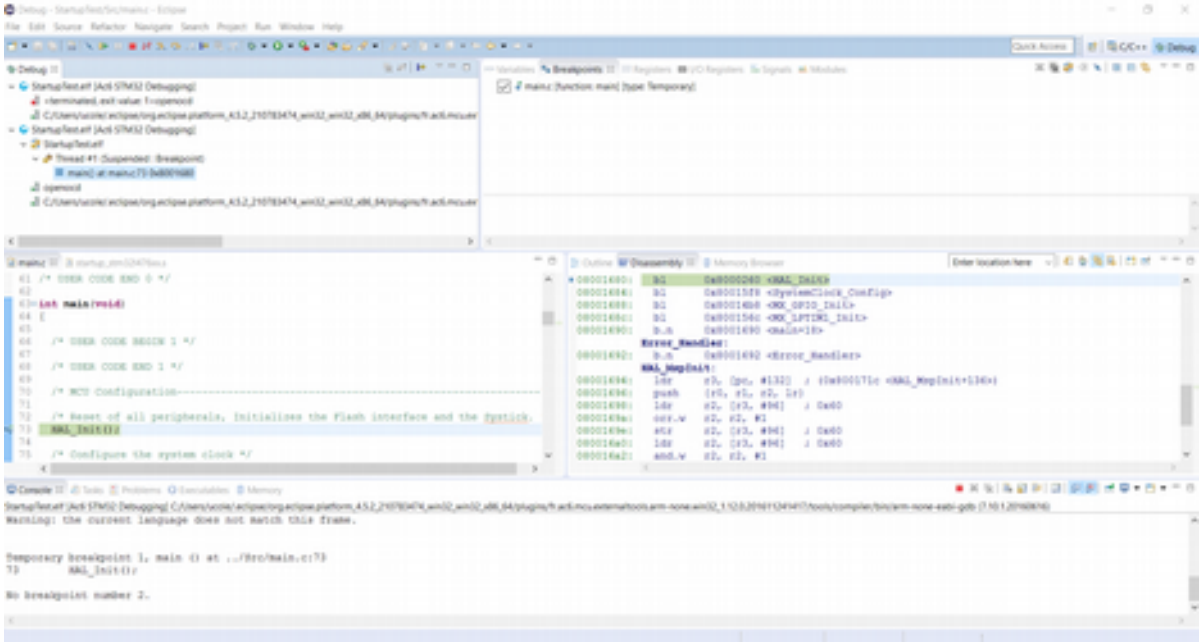
- Src/stm32f4xx_it.c
- Glue code between HAL peripheral and your code:
  - Interrupt handlers in the vector table are not directly defined in the HAL layer.
  - User can define them in the stm32f4xx_it.c file and call the HAL_Handler in it.



System Workbench 4

Launching and debugging

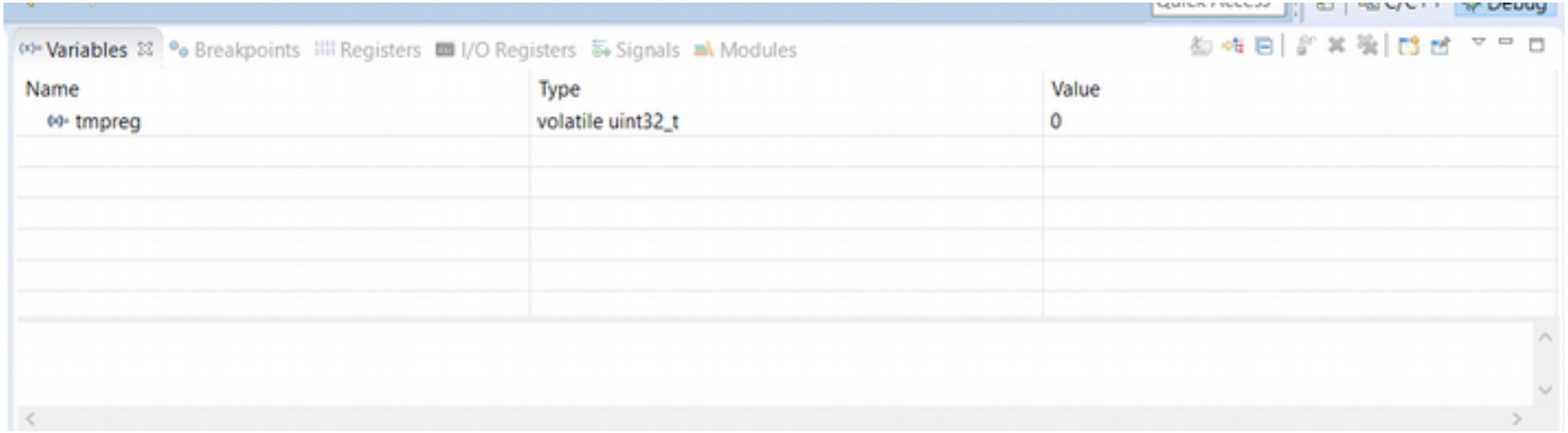
- Plug the Stm32F401 board
- Right click on the project and select “Debug as…” and select “Ac6 Stm32 C/C++ Application”
- This will trigger a recompilation (but it’s already done)
- Then the board will be programmed and the mcu reset
- The program will halt by default at the beginning of the main function



System Workbench 4

Useful views during a breakpoint

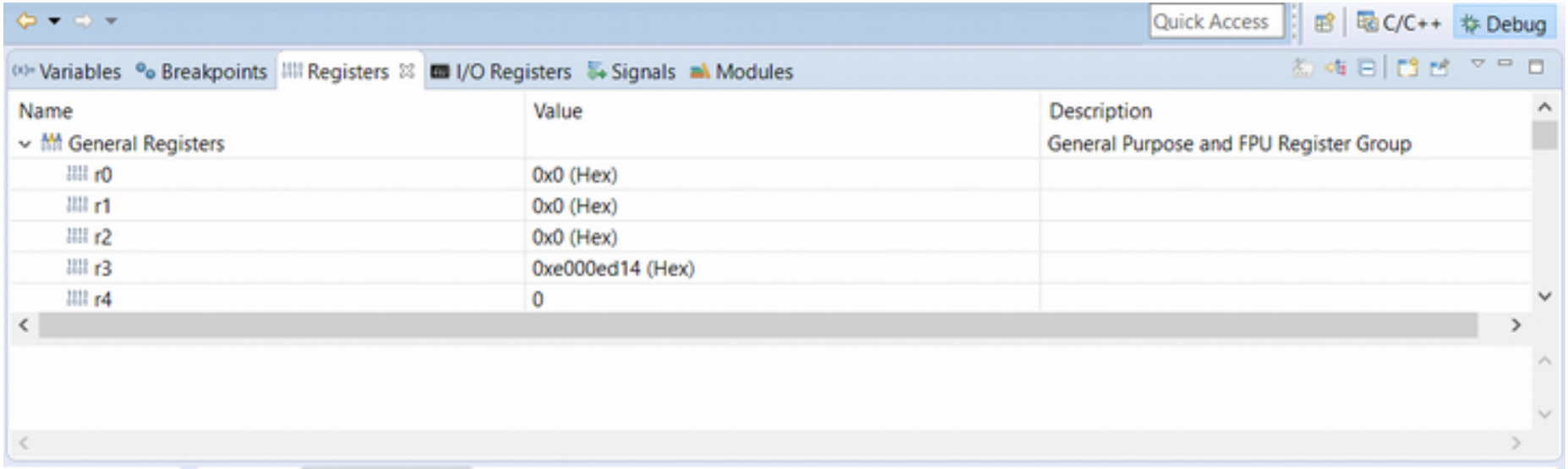
•Variables: shows (some?) global and automatic variables with their values



System Workbench 4

Useful views during a breakpoint

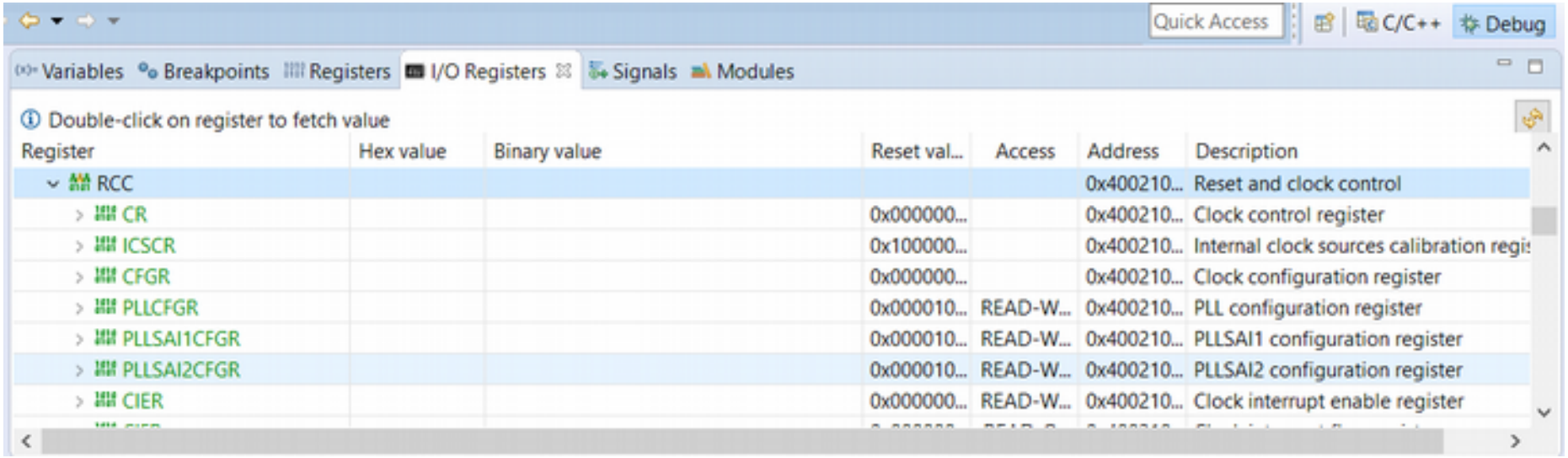
•Registers: shows the values of all the MCU registers



System Workbench 4

Useful views during a breakpoint

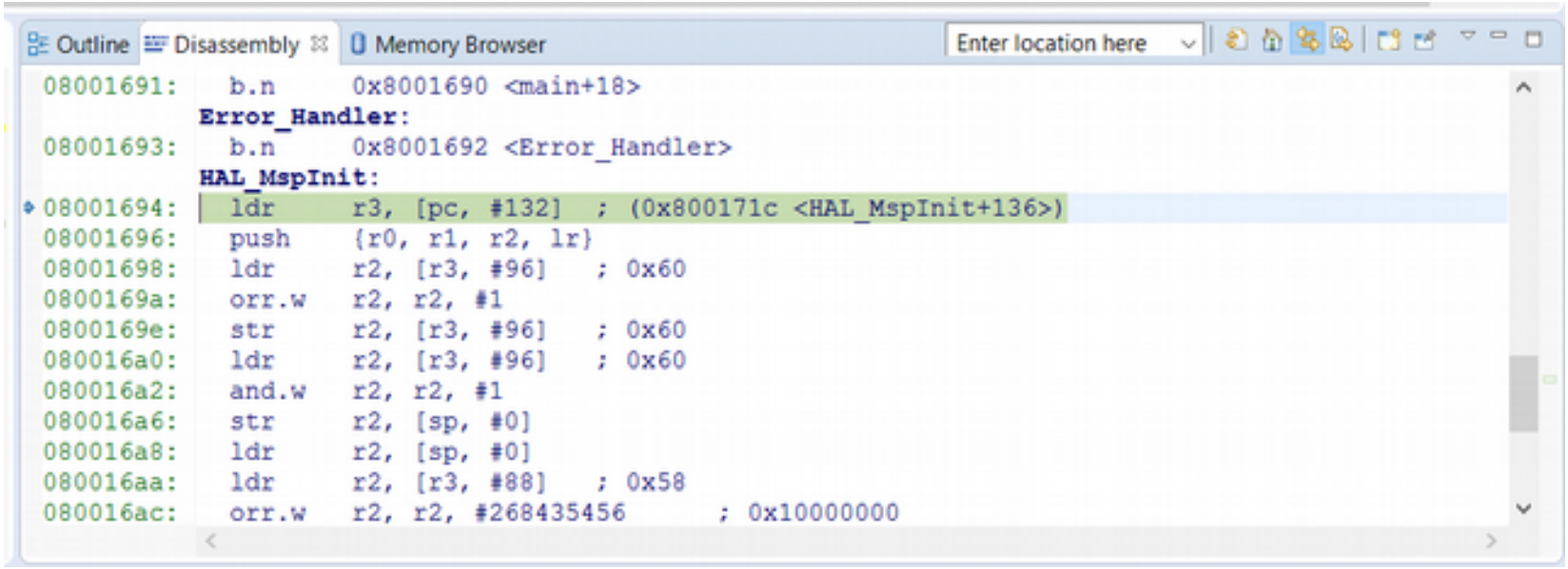
- I/O Registers: detailed view of all peripheral registers with their offsets and values



System Workbench 4

Useful views during a breakpoint

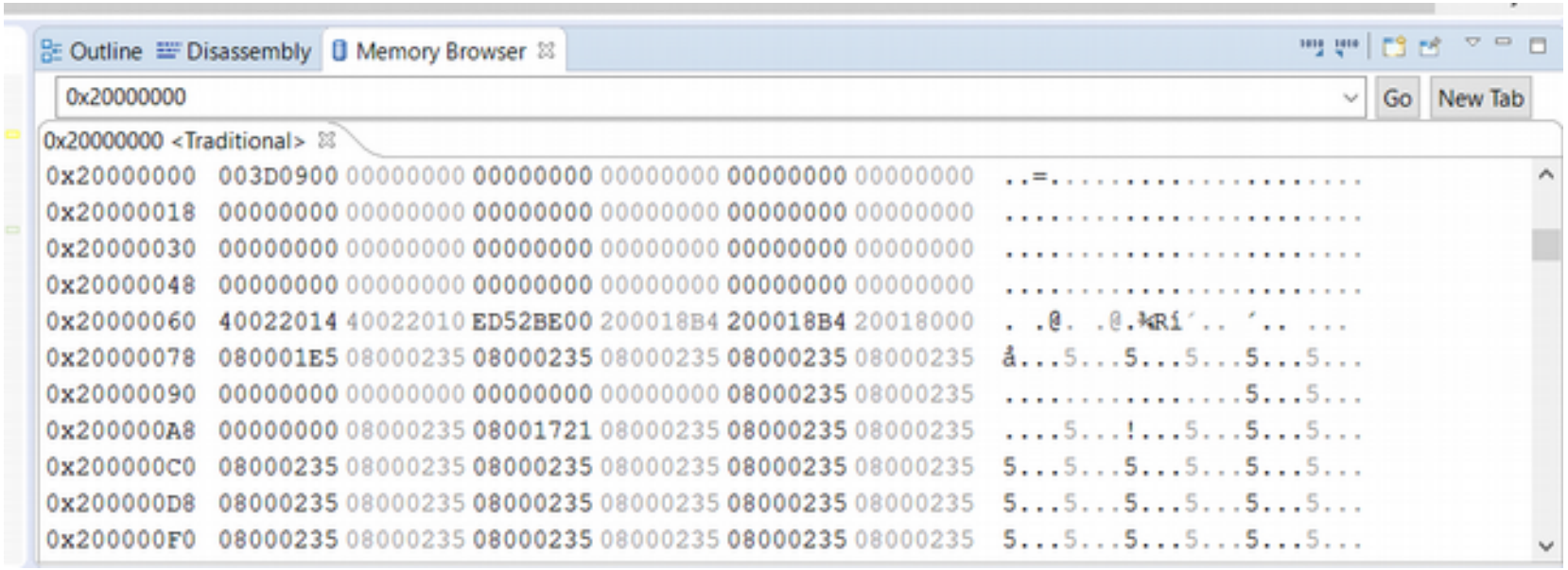
- Disassembly: disassembled code on the go (not always reliable)



System Workbench 4

Useful views during a breakpoint

- Memory Browser: allows to browse over the entire linear memory of the MCU



System Workbench 4

Hands-on: Le'ts put a breakpoint on the Reset Handler…

- The first steps of the cortex-m during power on are:
  - Load address 0x0 (address of the stack pointer) in the stack pointer register (MSP)
  - Load address 0x4 (address of the reset handler) in the program counter (PC)

- After re-running "debug as":
  - Follow the boot sequence up to the main

Hands on: let's switch on the led (LD2)

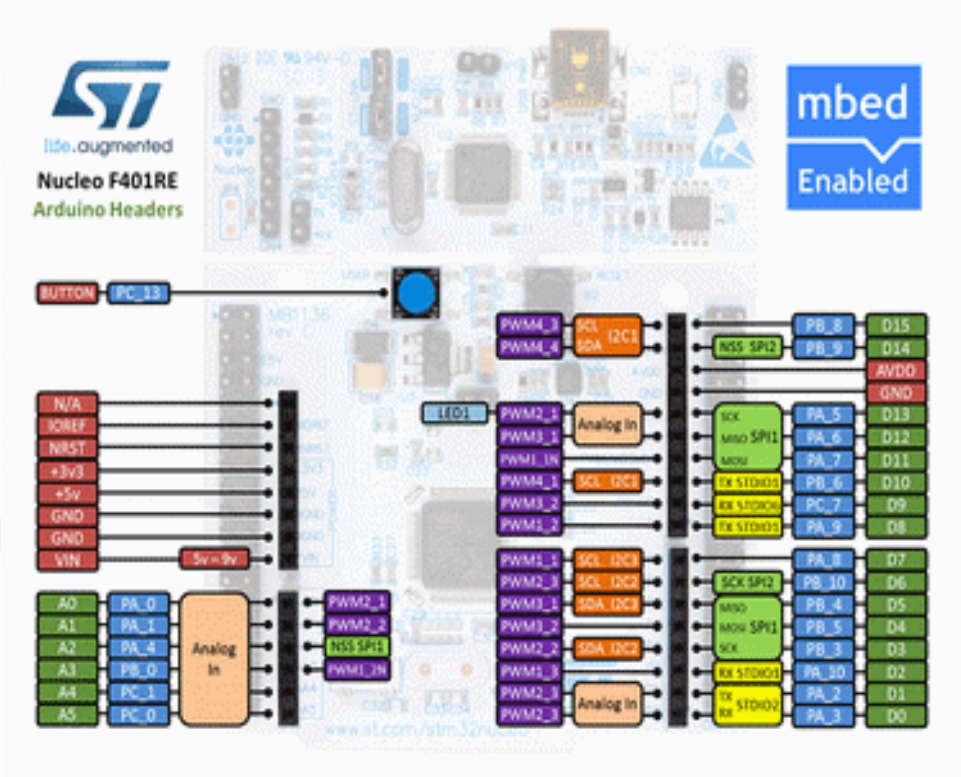
- Just one line of code in the main.c…

User LD2: the green LED is a user LED connected to Arduino signal D13 corresponding to STM32 I/O PA5 (pin 21) or PB13 (pin 34) depending on the STM32 target. Refer to *Table 11* to *Table 23* when:

- the I/O is HIGH value, the LED is on
- the I/O is LOW, the LED is off

LD3 PWR: the red LED indicates that the STM32 part is powered and +5V power is available.

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

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_LPTIM1_Init();

/* USER CODE BEGIN 2 */
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
/* USER CODE END 2 */
```



Hands on: let's switch on the led (LD2)

- Let's have a closer look to what it is happening through the debugger…

System Workbench 4

Hands on: let's toggle the led (LD2) each 500ms

- Using the HAL library and systick timer:

```
  /* USER CODE BEGIN 2 */
  HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
  /* USER CODE END 2 */

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

  /* USER CODE BEGIN 3 */
      HAL_Delay(500);
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
  }
  /* USER CODE END 3 */

}
```

main.c

```
/**
 * @brief This function handles System tick timer.
 */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
```

stm32l4xx_it.c

Polls uwTick which increases by one at each systick interrupt (check with the debugger)

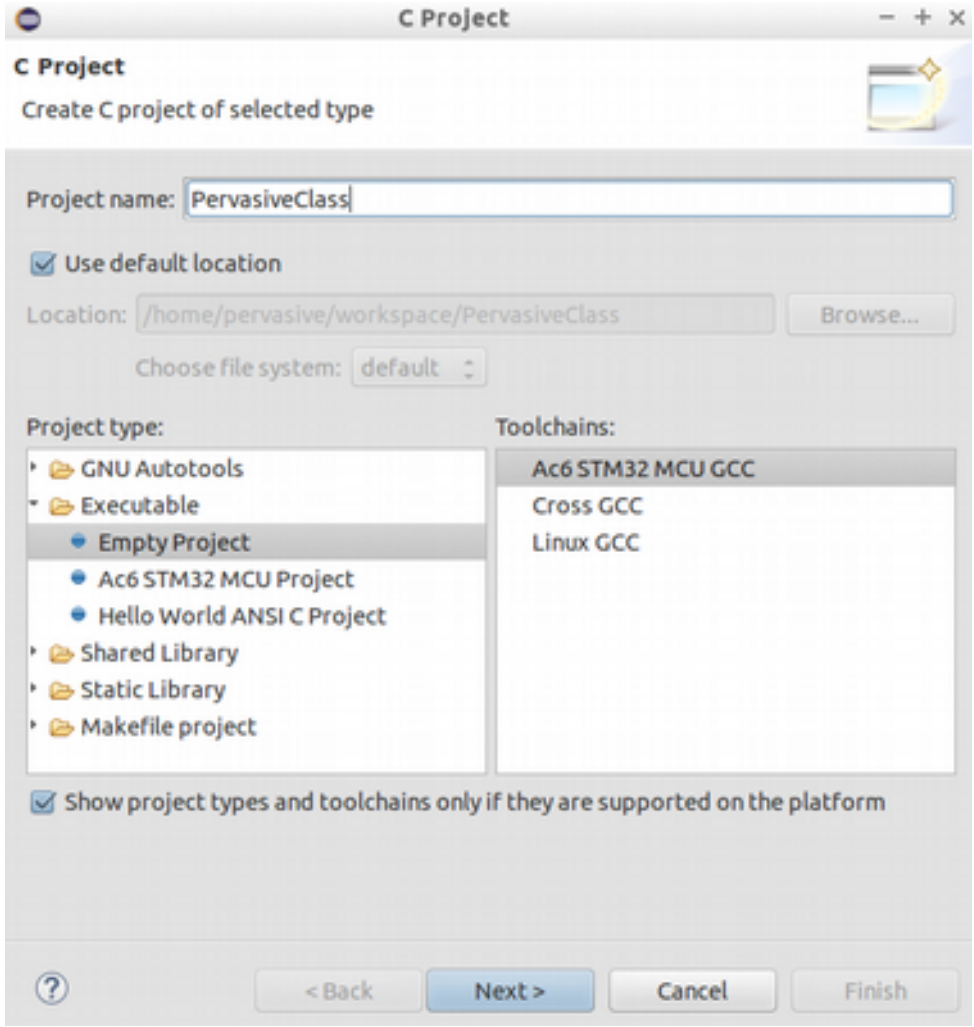
System Workbench 4

SW4 – CubeMX integration

SW4 – CubeMX Integration

Create a plain project

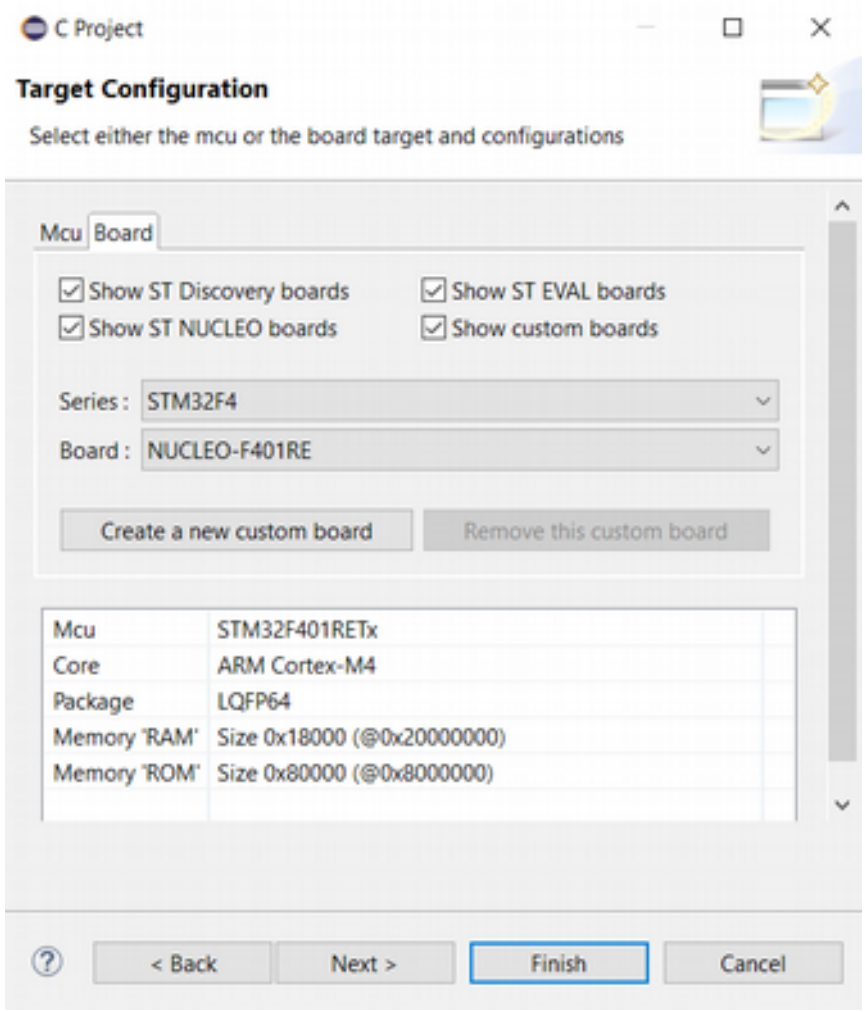
- Step 1:
  - Launch SW4STM32
  - In “File” menu click on “new”
  - Select “C project”
  - Enter the project name
  - Make sure you select “Empty Project” with “AC6” toolchain
  - Click on “Next”



SW4 – CubeMX Integration

Create a plain project

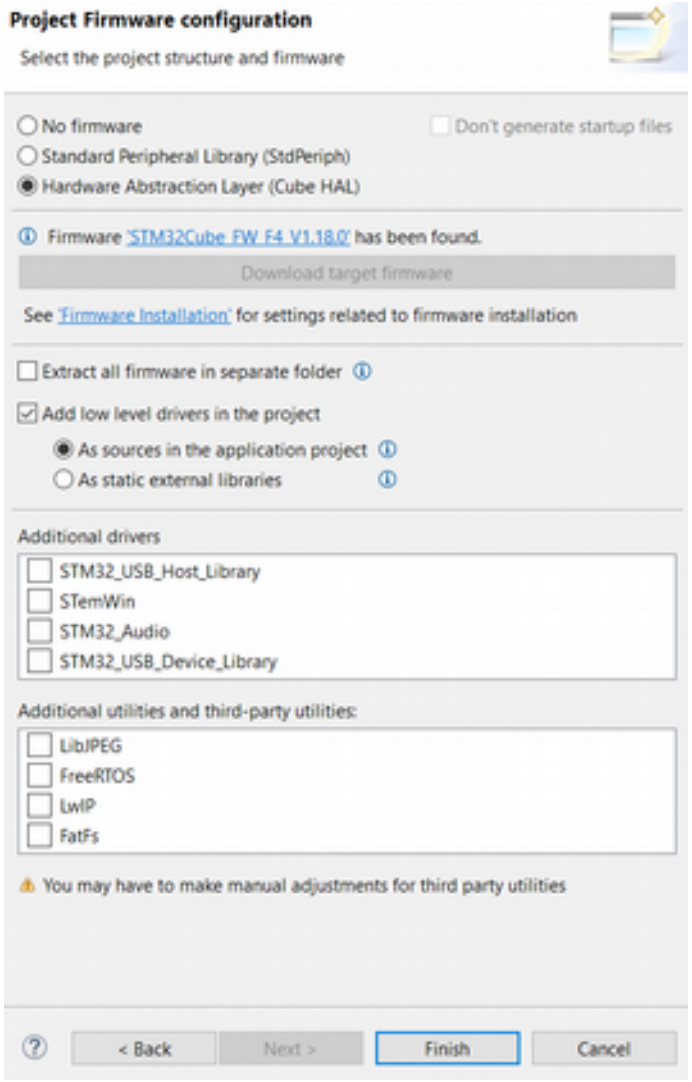
- Step 2:
  - Select “Debug” and “Release”
  - Click on “Next”
  - Select STM32F4 as “Series”
  - Select Nucleo-F401RE as “Board”
  - Click on “Next”



SW4 – CubeMX Integration

Create a plain project

- Step 3:
  - Select “Hardware Abstraction Layer” and keep the rest unchanged
  - Download the framework if you do not have it
  - Click on “Finish”



SW4 – CubeMX Integration

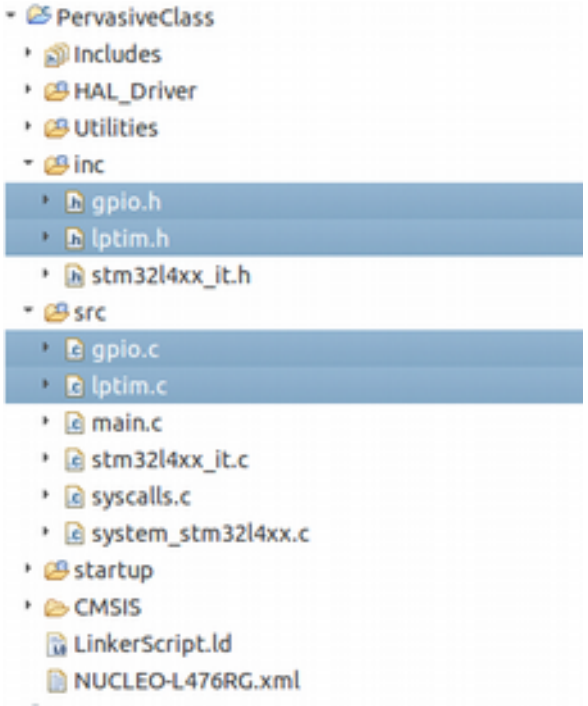
Create a plain project

- Pros:
  - All HAL drivers available
  - Board Support Packages (BSP) drivers for external peripherals (button,led on nucleo-64)
- Cons:
  - Missing peripheral and HAL setup
  - Compiles all the files (even HAL drivers that are not used)

SW4 – CubeMX Integration

Merging with CubeMX

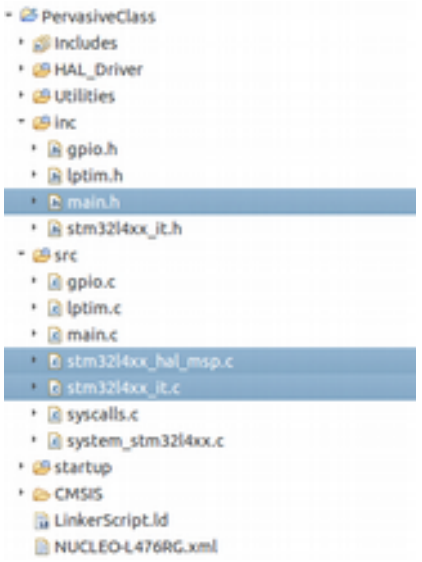
- Step 1:
  - Create a plain project in SW4
  - Create a project in CubeMX with Clock and peripheral initialized
  - Copy the peripheral configuration files (e.g. gpio.c) from src in CubeMX to src in SW4
  - Do the same with header files (e.g. gpio.h) in the inc folder

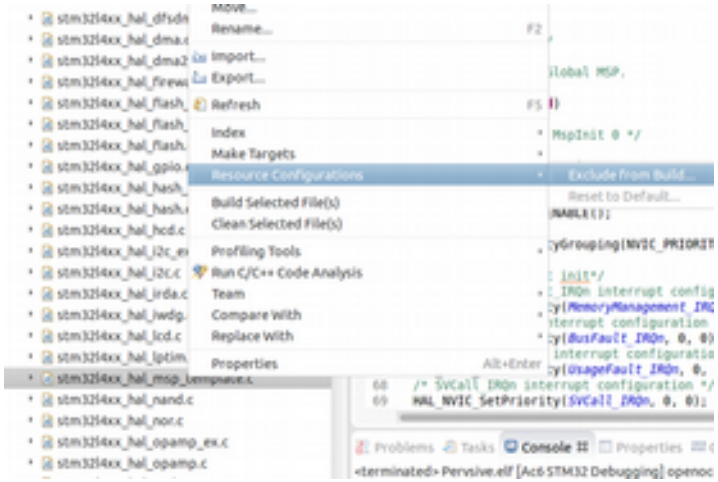


SW4 – CubeMX Integration

Merging with CubeMX

- Step 2:
  - Copy the main.h header file from inc folder of CubeMX to inc folder of SW4
  - Replace the stm32f4xx_it.c (and .h) file in SW4 with the one in CubeMX
  - Copy the stm32f4xx_hal_msp.c file from CubeMX to SW4
  - Remove the stm32f4xx_hal_msp_template.c file from HAL_Driver/Src (or right-click on it and select Resoure Configurations->exclude from build and select Debug and Release)





SW4 – CubeMX Integration

Merging with CubeMX

- Step 3:
  - Edit main.c of SW4 by including the imported header files (main.h, gpio.h etc…)
  - If a peripheral configuration file has a handle variable (e.g. TIM1_HandleTypeDef), declare it as extern in the main file

main.c | usart.c | main.c | stm32l4xx_hal_msp.c

```
 3*   * @file    main.c
10 #include "main.h"
11 #include "lptim.h"
12 #include "gpio.h"
13
14 #include "stm32l4xx.h"
15 #include "stm32l4xx_nucleo.h"
16
17 extern LPTIM_HandleTypeDef hlptim1;
18
19
20 int main(void)
21 {
22
23     for(;;);
24 }
```

SW4 – CubeMX Integration

Merging with CubeMX

- Step 4:
  - Copy the SystemClock_Config and the ErrorHandler functions from the main.c of CubeMX to the main.c of SW4
  - Forward declare the copied functions
  - Copy the inner code of the main function from CubeMX to SW4

main.c ⊠ | main.c

```
18
19 void SystemClock_Config(void);
20 void _Error_Handler(char * file, int line);
21
22 int main(void)
23 {
24       /* MCU Configuration--------------------------------
25
26       /* Reset of all peripherals, Initializes the Flash in
27       HAL_Init();
28
29       /* Configure the system clock */
30       SystemClock_Config();
31
32      /* Initialize all configured peripherals */
33       MX_GPIO_Init();
34       MX_LPTIM1_Init();
35
36       /* USER CODE BEGIN 2 */
37       HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
38       /* USER CODE END 2 */
39
40       /* Infinite loop */
41       /* USER CODE BEGIN WHILE */
42       while (1)
43       {
44       /* USER CODE END WHILE */
45           HAL_Delay(500);
46           HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);
47       /* USER CODE BEGIN 3 */
```

SW4 – CubeMX Integration

Using the BSP:

- Check the file “stm32f4xx_nucleo.c” in Utilities/STM32F4XX_Nucleo
- Modify the main as follows and run the code (right click->run as->Ac6):

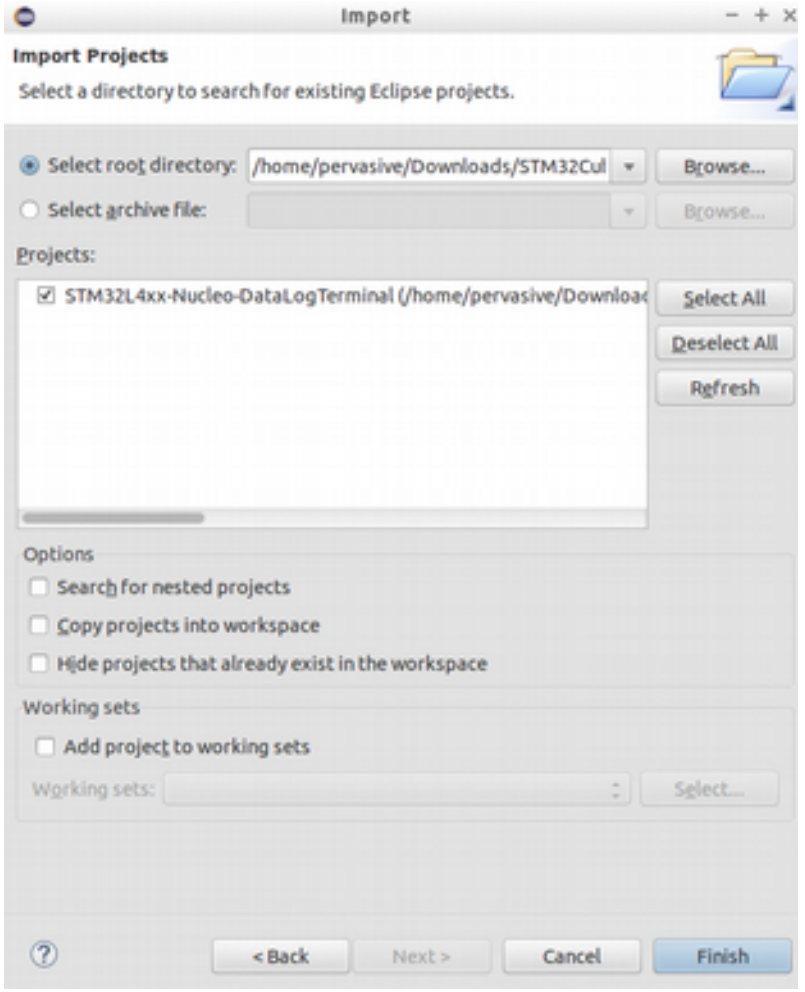
```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_TIM1_Init();
/* USER CODE BEGIN 2 */
//HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);
BSP_PB_Init(BUTTON_USER,BUTTON_MODE_GPIO);
BSP_LED_Init(LED2);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    if(BSP_PB_GetState(BUTTON_USER) == 1){
        BSP_LED_On(LED2);
    }
    else{
        BSP_LED_Off(LED2);
    }

    //HAL_Delay(500);
    //HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_5);

/* USER CODE END WHILE */
```

Using the Sensor Expansion Board

Sensor Expansion Board

Importing an existing SW4 project

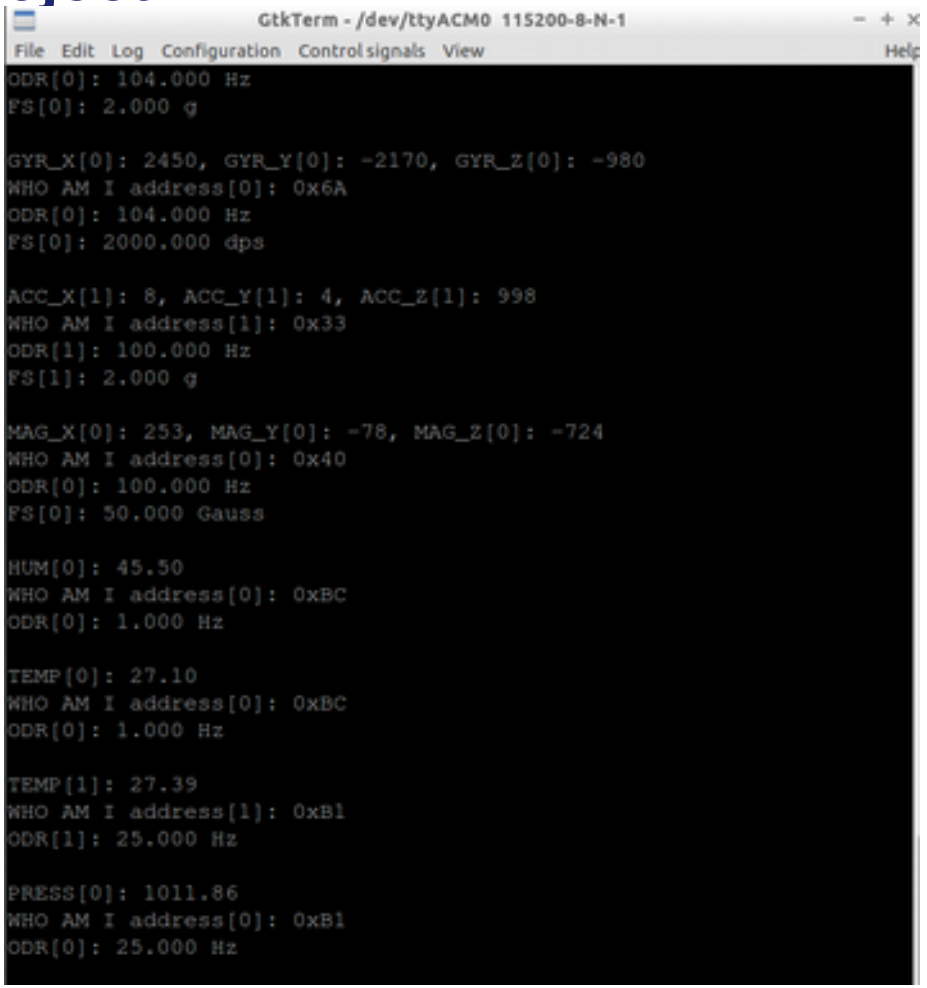
- Step 1:
  - In SW4 click on File->import-> existing projects into workspace
  - Click “Next”
  - Browse to the MEMS1_V4.3.0 folder and select Projects->Multi->Examples->IKS01A2->DataLogTerminal->SW4STM32->STM32F401RE-Nucleo
  - Click “OK”
  - Keep everything unchanged
  - Click “Finish”



Sensor Expansion Board

Importing an existing SW4 project

- Step 2:
  - Compile and run
  - Open a serial port terminal
  - Configure ttyACM0 with 8n1 and 115200bps
  - You should see the sensor output



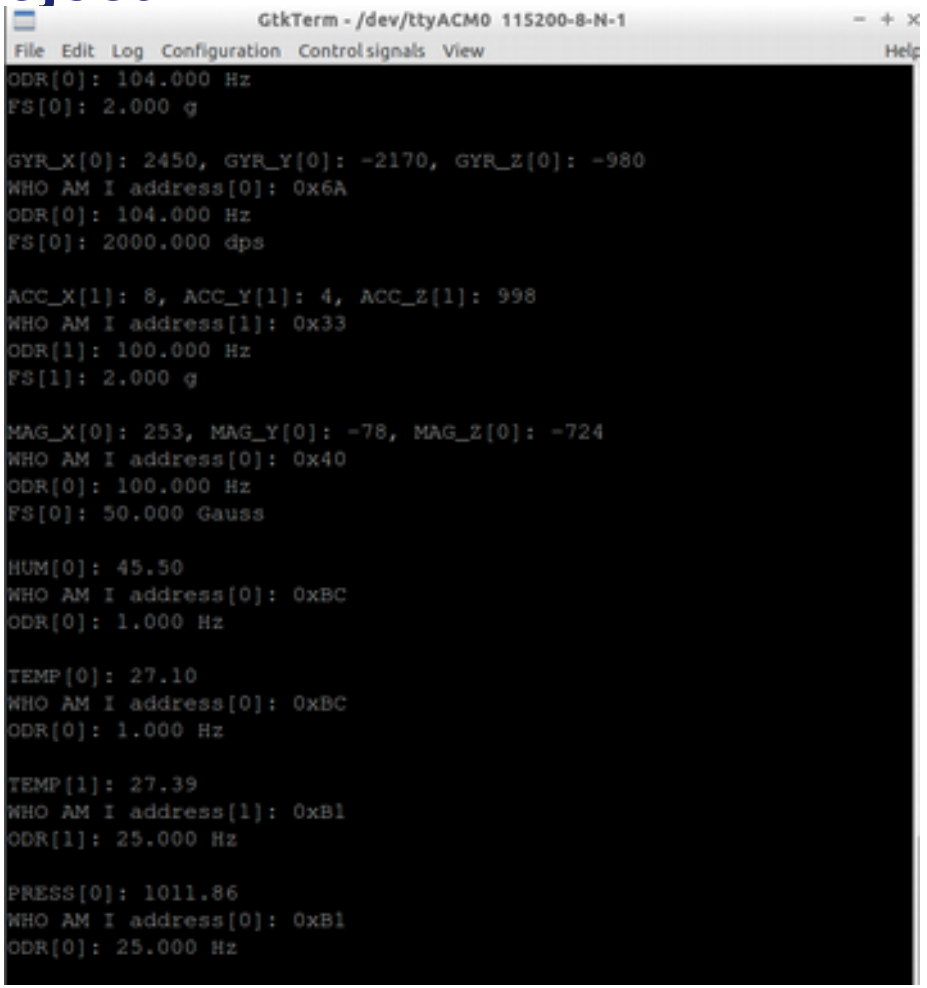
Sensor Expansion Board

- Step 1:
  - Import the DataLogTerminal example from MEMS framework home/ps/Desktop/resources/STM32CubeExpansion_MEMS1_V4.3.0/Projects/Examples/IKS01A1/DataLogTerminal/SW4STM32/STM32F401RE-Nucleo)
  - Build the project
  - Unplug the Nucleo and install the extension board
  - Plug the Nucleo and program it

Sensor Expansion Board

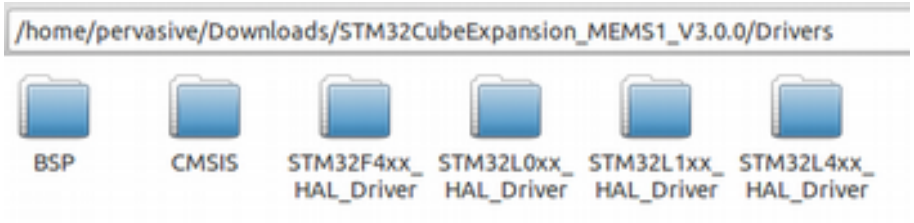
Importing an existing SW4 project

- Step 2:
  - Compile and run
  - Open a serial port terminal
  - Configure ttyACM0 with 8n1 and 115200bps
  - You should see the sensor output



Sensor Expansion Board

- Unplug the board and attach the sensor board on it
- Open the STM32CubeExpansion_MEMS1_V4.3.0
  - Drivers
    - BSP -> You need mems drivers (Components folder), Board adaptation files (X_NUCLEO_IKS01A2 folder). Don't need the generic L476 BSP file (you already have it)
    - CMSIS -> Don't need: already have your CMSIS library
    - STMXXX -> Don't need: already have your HAL driver
  - Projects -> several examples
  - Utilities -> GUI program for the PC (we won't use it)



Sensor Expansion Board

WARNING!!!

- The imported project has several files linked in the framework folder
- It is highly unrecommended to change this project as the changes might affect other projects based on the same linked files
- Import the necessary file in a plain SW4 project instead.

Sensor Expansion Board

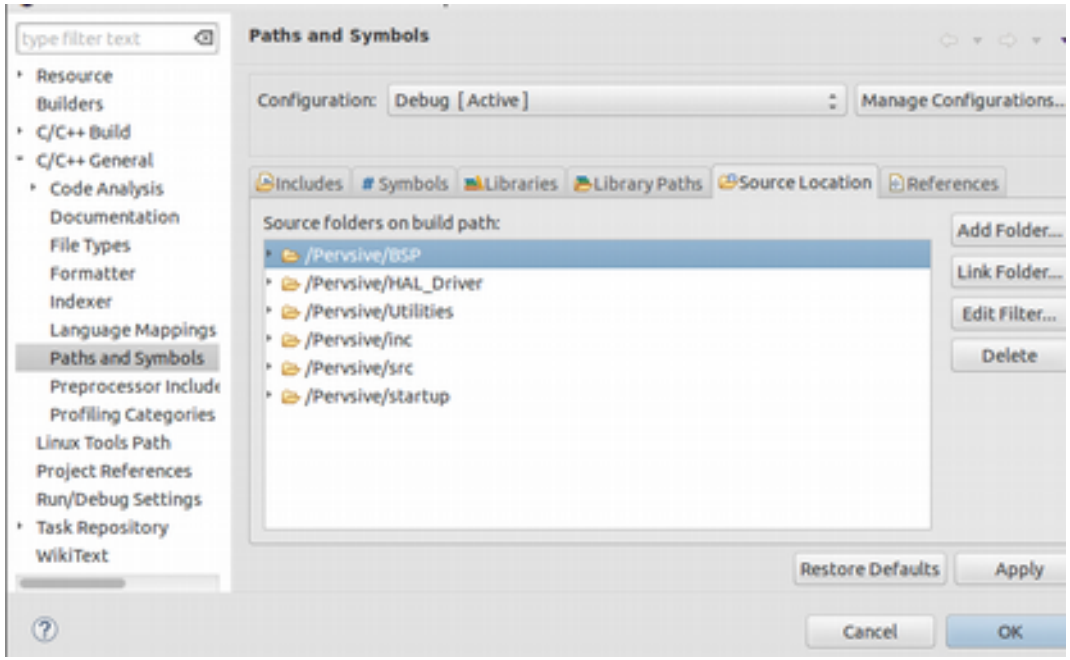
Merging an existing linked project in a plain one

- Step 1:
  - Import an existing project in SW4
  - Create a plain SW4 project with CubeMX settings merged in it

Sensor Expansion Board

Merging an existing linked project in a plain one

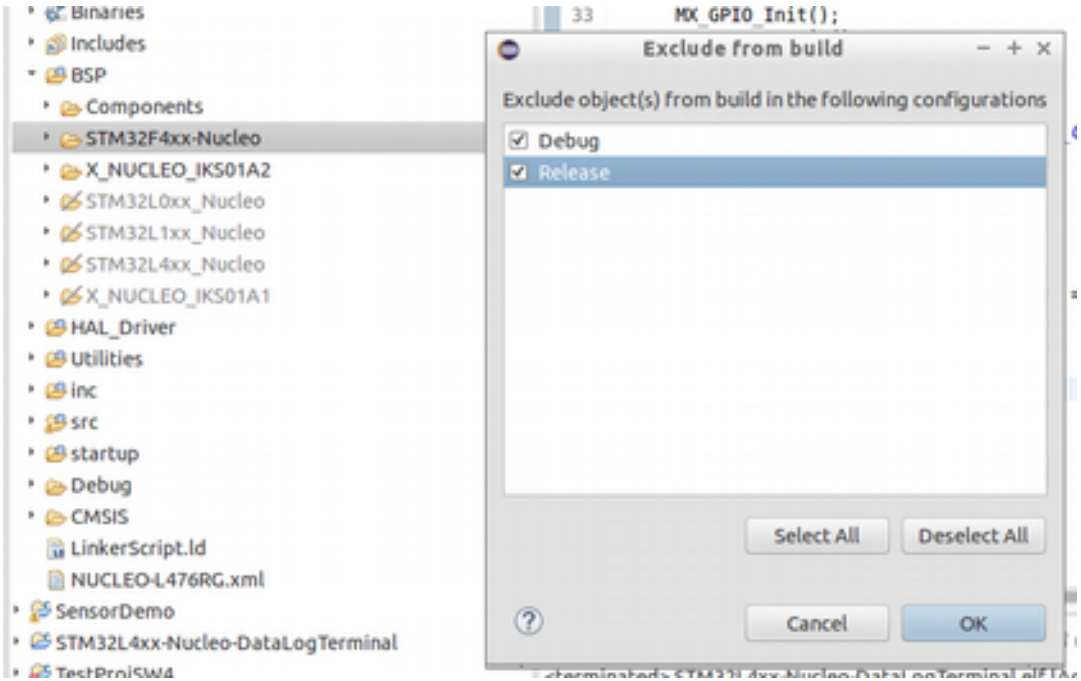
- Step 2:
  - Copy the BSP folder from MEMS_V3_0_0 to your main project folder
  - Include the BSP folder as source folder by right-clicking on the project->properties->C/C++ General->Paths and Symbols->Source Location->Add Folder



Sensor Expansion Board

Merging an existing linked project in a plain one

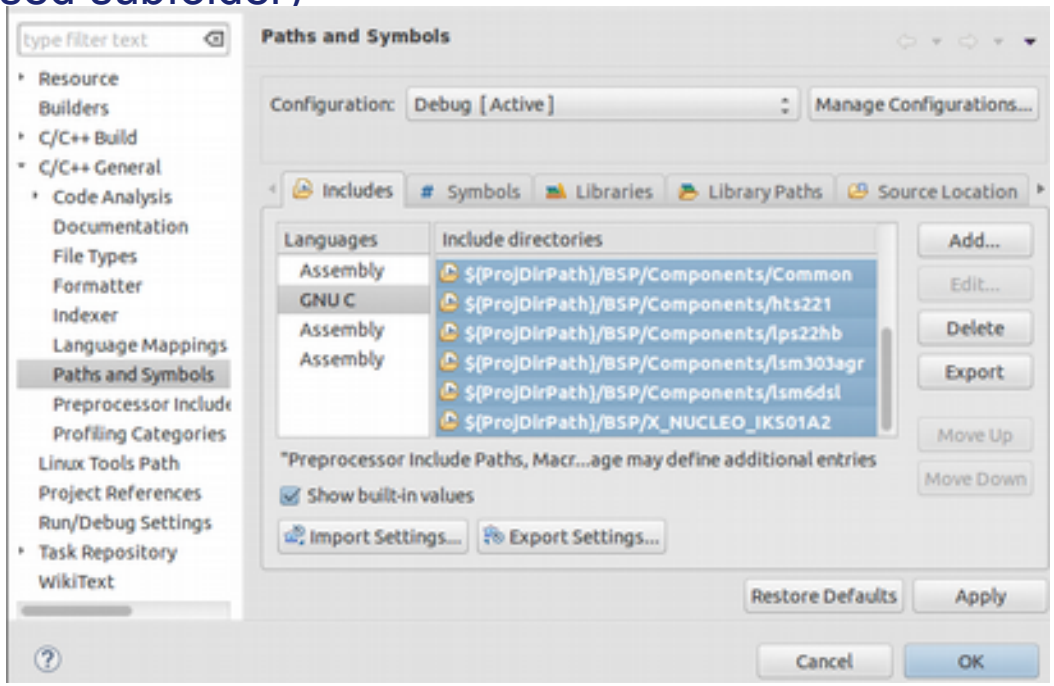
- Step 3:
  - Filter out every c file in this folder that it is not used in the imported project: right click on a file -> resource configure->exclude from build (check the files that are used by navigating the imported project first)



Sensor Expansion Board

Merging an existing linked project in a plain one

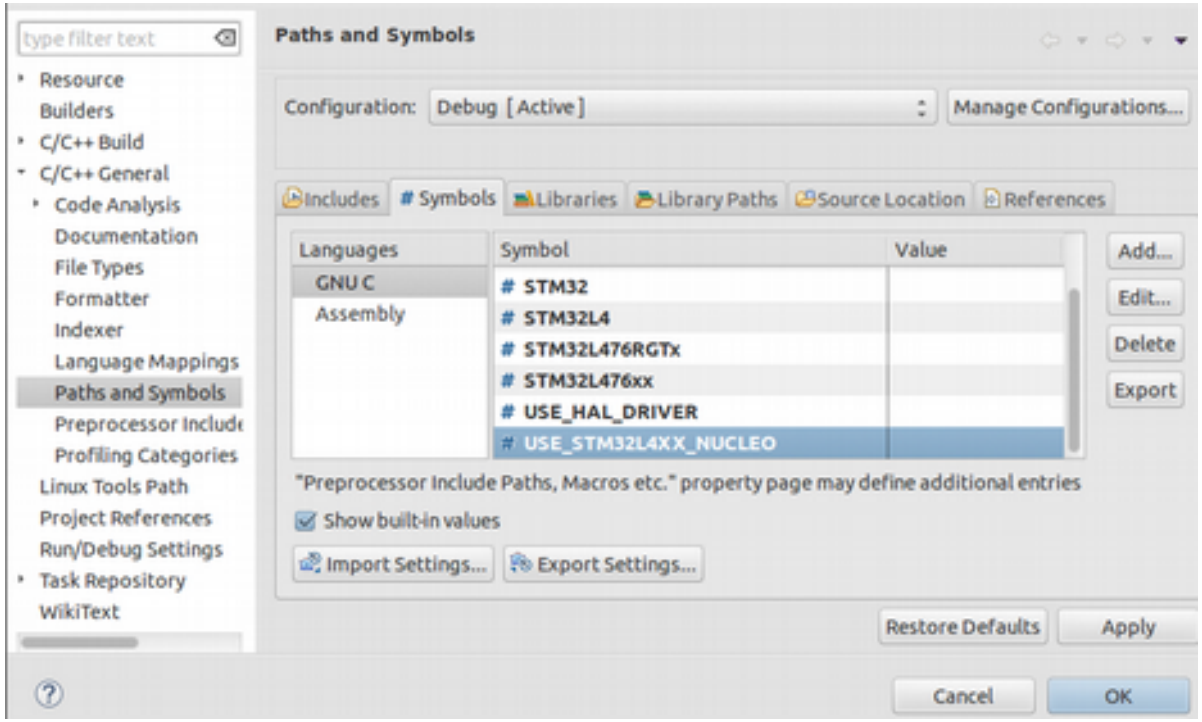
- Step 4:
  - Righ-click on the project->properties->C/C++ General->Paths and Symbols->includes and selecting GNU C as “Languages”
  - Add all the relevant folders that uses headers related to the new BSP folder (every used subfolder)



Sensor Expansion Board

Merging an existing linked project in a plain one

- Step 5:
  - Righ-click on the project->properties->C/C++ General->Paths and Symbols->symbols and add existing symbols that are present in the imported project



Sensor Expansion Board

Merging an existing linked project in a plain one

- Step 6:
  - Import interrupt handlers from the stm32l4xx_it.c file
  - Import MCU initialization functions from the stm32l4xx_hal_msp.c file
  - Import BSP headers in your main.h and in any other file where it is required

main.c | main.h

```
 3  * File Name          : main.h
38 /* Define to prevent recursive inclusion ---
39 #ifndef __MAIN_H
40 #define __MAIN_H
41   /* Includes ------------------------------
42
43 /* USER CODE BEGIN Includes */
44 #include "stm32l4xx_hal.h"
45 #include "stm32l4xx_nucleo.h"
46
47 #include "x_nucleo_iks01a2.h"
48 #include "x_nucleo_iks01a2_accelero.h"
49 #include "x_nucleo_iks01a2_gyro.h"
50 #include "x_nucleo_iks01a2_magneto.h"
51 #include "x_nucleo_iks01a2_pressure.h"
52 #include "x_nucleo_iks01a2_humidity.h"
53 #include "x_nucleo_iks01a2_temperature.h"
54 /* USER CODE END Includes */
55
56 /* Private define ------------------------
57
58 #define B1_Pin GPIO_PIN_13
59 #define B1_GPIO_Port GPIOC
60 #define USART_TX_Pin GPIO_PIN_2
61 #define USART_TX_GPIO_Port GPIOA
62 #define USART_RX_Pin GPIO_PIN_3
63 #define USART_RX_GPIO_Port GPIOA
64 #define LD2_Pin GPIO_PIN_5
65 #define LD2_GPIO_Port GPIOA
66 #define TMS_Pin GPIO_PIN_13
67 #define TMS_GPIO_Port GPIOA
68 #define TCK_Pin GPIO_PIN_14
```

Sensor Expansion Board

Merging an existing linked project in a plain one

- Step 7:
  - Copy whatever you want from the main.c file
  - Fix the remaining minor issues (check for errors, check missing files/functions, optionally initialize peripherals with CubeMX etc…)

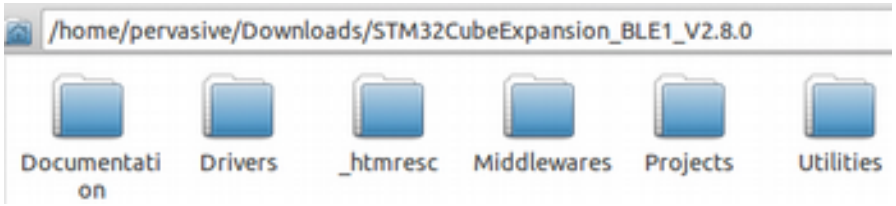
main.c | main.h

```
17 #include <stdio.h>  /* sprintf */
18 #include <math.h>   /* trunc */
19
20 #include "stm32l4xx.h"
21 #include "stm32l4xx_nucleo.h"
22
23 extern LPTIM_HandleTypeDef hlptim1;
24 extern UART_HandleTypeDef huart2;
25
26
27 void SystemClock_Config(void);
28 void _Error_Handler(char * file, int line);
29
30
31 extern int use_LSI;
32 int RTC_SYNCH_PREDIV;
33
34 /* Private typedef -----------------------------------------
35 /* Private define ------------------------------------------
36 /* Private macro -------------------------------------------
37 /* Private variables ---------------------------------------
38 static volatile uint8_t acquire_data_enable_request  = 1;
39 static volatile uint8_t acquire_data_disable_request = 0;
40
41 static uint8_t acquire_data_enabled = 0;
42 static uint8_t verbose              = 1;  /* Verbose output
43 static RTC_HandleTypeDef RtcHandle;
44 static char dataOut[256];
45
46 static void *LSM6DSL_X_0_handle = NULL;
47 static void *LSM6DSL_G_0_handle = NULL;
48 static void *LSM303AGR_X_0_handle = NULL;
```

Bluetooth Expansion Board

Sensor Expansion Board

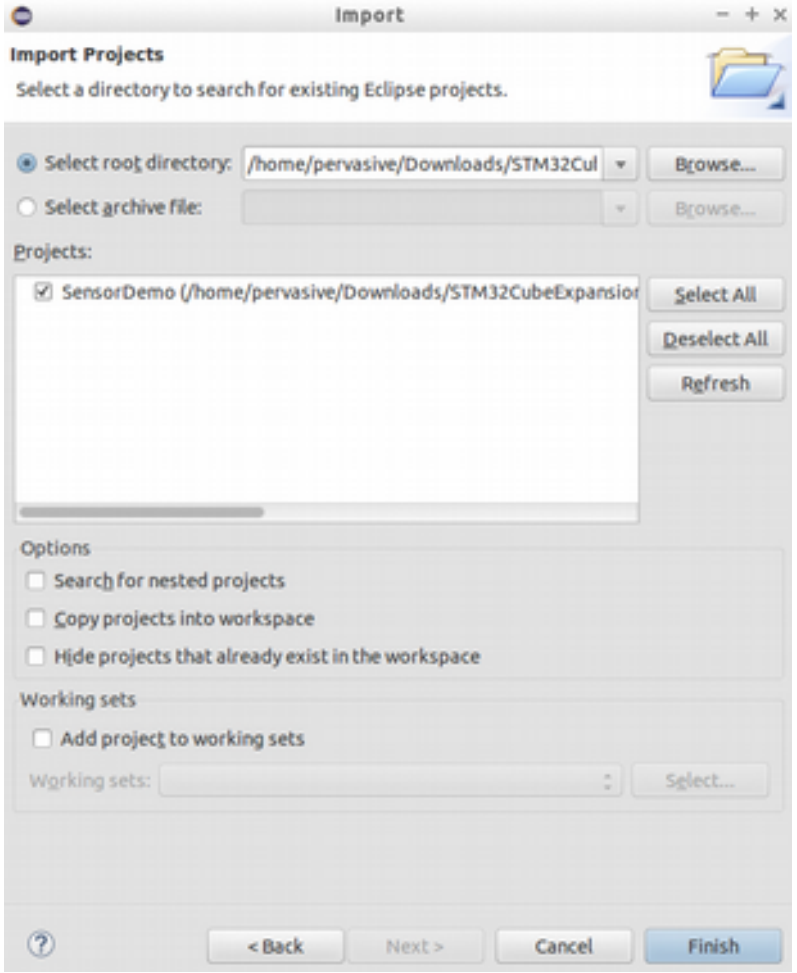
- Unplug the board and attach the Bluetooth board on top of the sensor board
- Open the STM32CubeExpansion_BLE1_V2.8.0
  - Drivers
    - BSP -> You need board adaptation files (X-NUCLEO-IDB0xA1 folder). Don't need the generic L476 BSP file (you already have it)
    - CMSIS -> Don't need: already have your CMSIS library
    - STMXXX -> Don't need: already have your HAL driver
  - Middlewares -> You need the whole folder
  - Projects -> several examples
  - Utilities -> GUI program for the PC (we won't use it)



Sensor Expansion Board

Importing an existing SW4 project

- Step 1:
  - In SW4 click on File->import-> existing projects into workspace
  - Click “Next”
  - Browse to the MEMS1_V3.0.0 folder and select Projects->Multi->Applications->SensorDemo->SW4STM32->STM32L476RG-Nucleo
  - Click “OK”
  - Keep everything unchanged
  - Click “Finish”



Sensor Expansion Board

Importing an existing SW4 project

- Step 2:
  - Compile and run the example application
  - On your Android or iOS smartphone install “blueNRG” from the store
  - Launch the phone application and bind the Bluetooth board
  - You should see a cube that rotates each time you press the user button on the board

Sensor Expansion Board

Assignment (if there is time)

- Merge the Bluetooth framework in your Accelerometer project
- Send actual data from the accelerometer to the smartphone

Thank You!

Master thesis: ugomaria.colesanti@intecs.it

www.intecs.it