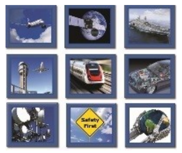
Introduction to Nucleo-64 platform



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.

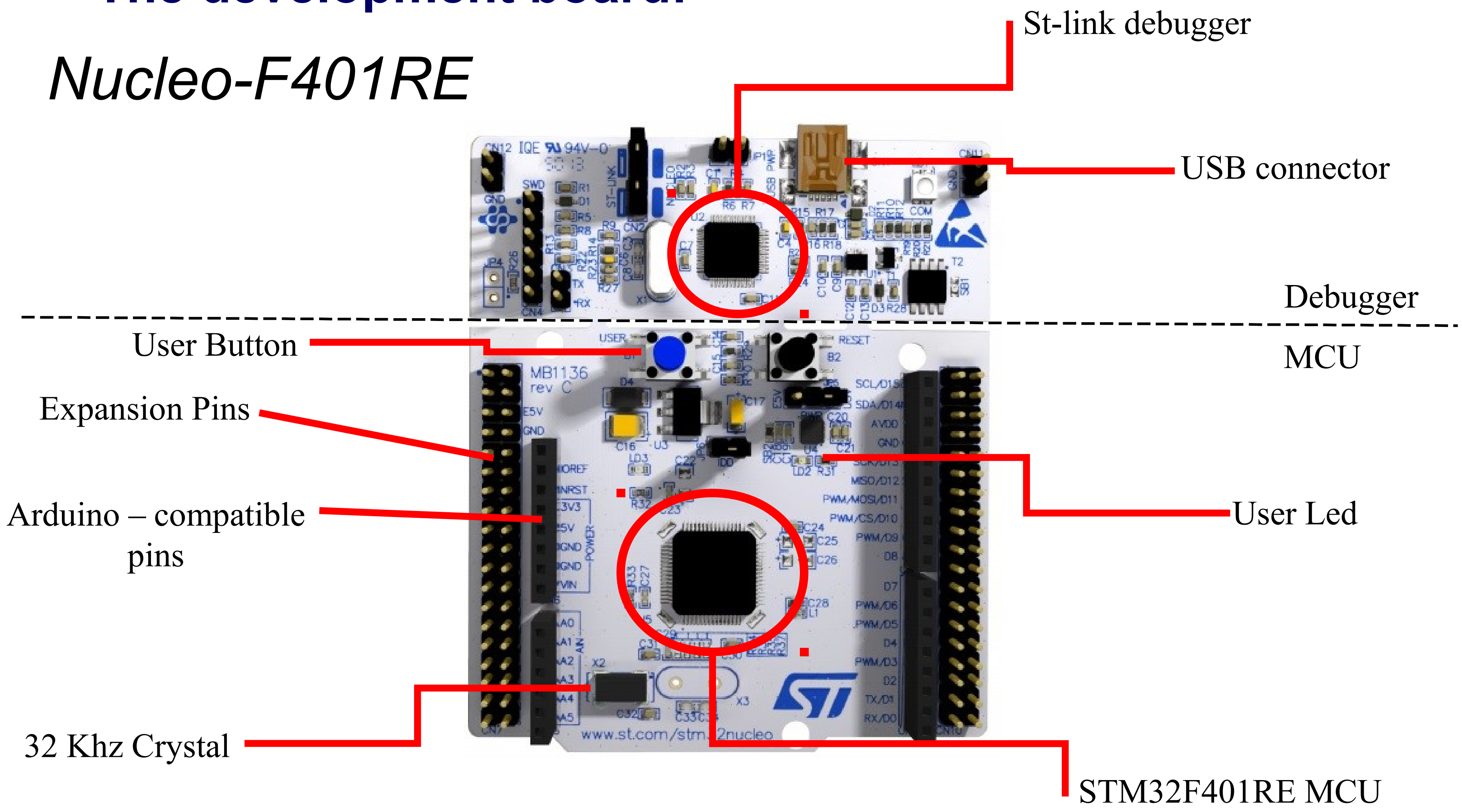
Introducing myself

- 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:

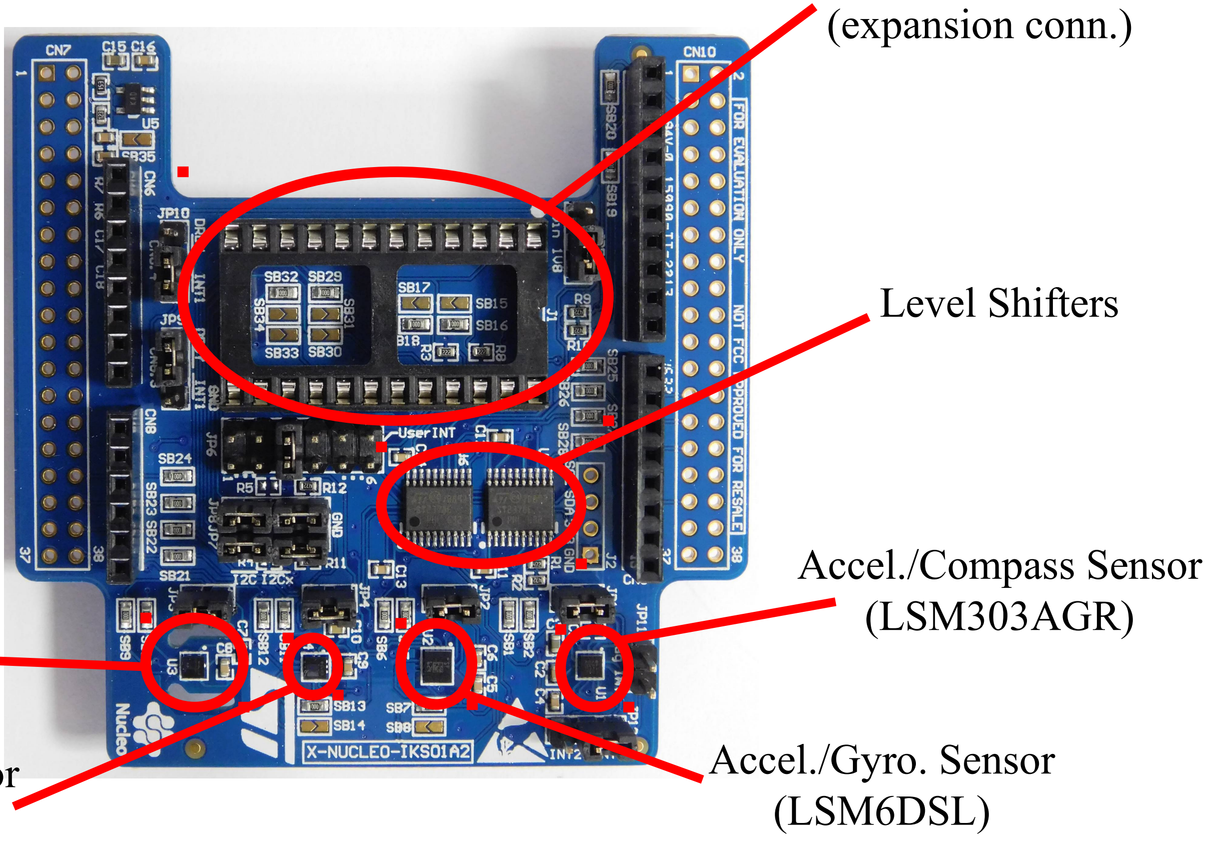
Nucleo-F401RE



Tools used in this class (2/4)

Sensor expansion board:

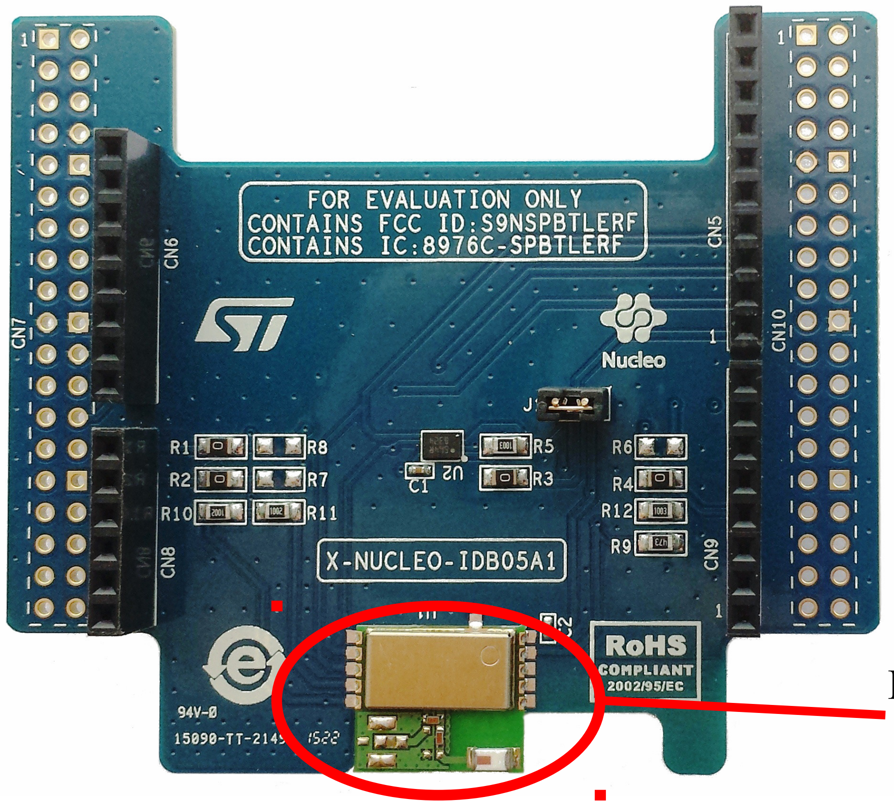
X-Nucleo-IKS01A2



Tools used in this class (3/4)

Bluetooth expansion board:

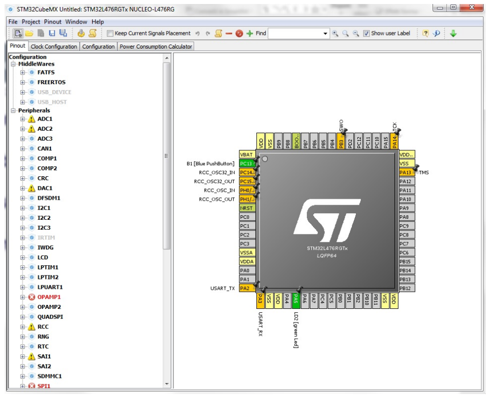
X-Nucleo-IDB05A1


FOR EVALUATION ONLY
CONTAINS FCC ID:S9NSPBTLERF
CONTAINS IC:8976C-SPBTLERF
Nucleo
X-NUCLEO-IDB05A1
RoHS COMPLIANT 2002/95/EC

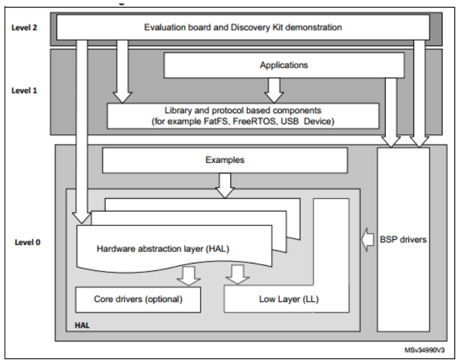
Bluetooth module (SPBTLE-RF)

Tools used in this class (4/4)

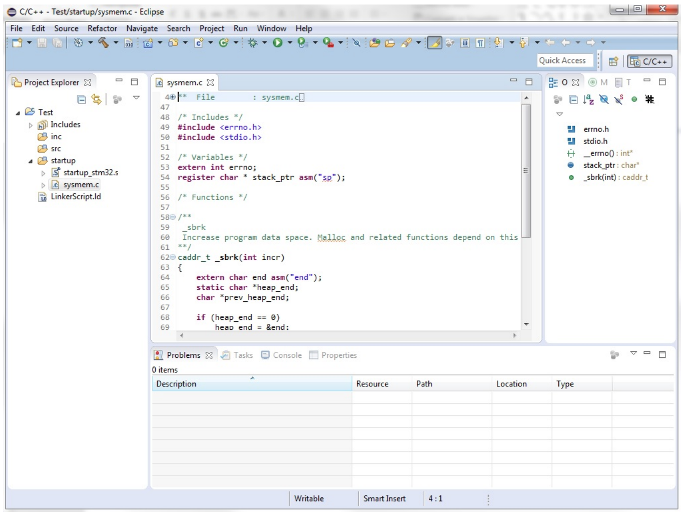
Framework, IDE & tools



STM CubeMX


Level 2
Evaluation board and Discovery Kit demonstration
Applications
Level 1
Library and protocol based components
(for example FatFS, FreeRTOS, USB Device)
Examples
Level 0
Hardware abstraction layer (HAL)
BSP drivers
Core drivers (optional)
Low Layer (LL)
HAL
MSv34990V3

Stm32CubeF4



```c
/**  File        : sysmem.c

/* Includes */
#include <errno.h>
#include <stdio.h>

/* Variables */
extern int errno;
register char * stack_ptr asm("sp");

/* Functions */

/**
 _sbrk
 Increase program data space. Malloc and related functions depend on this
**/
caddr_t _sbrk(int incr)
{
    extern char end asm("end");
    static char *heap_end;
    char *prev_heap_end;

    if (heap_end == 0)
        heap_end = &end;
```

System Workbench 4

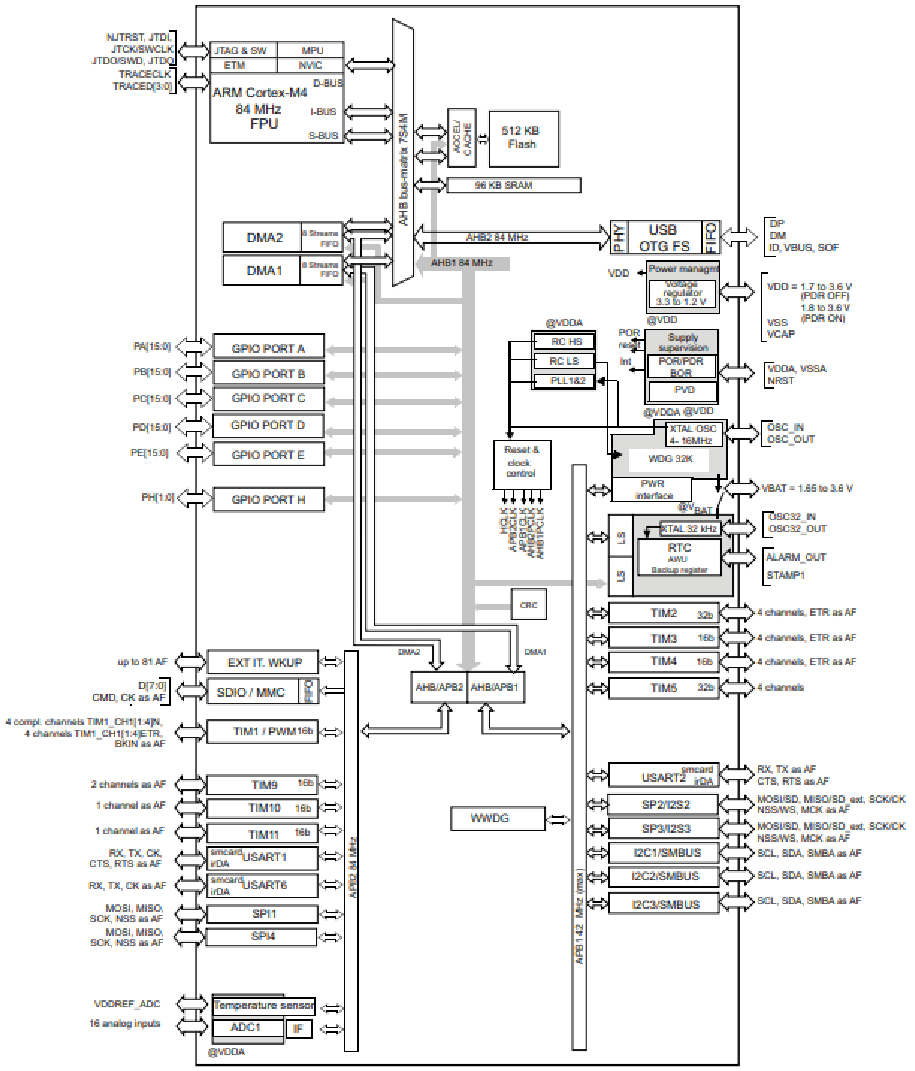
The Microcontroller

Stm32F401 Microcontroller:

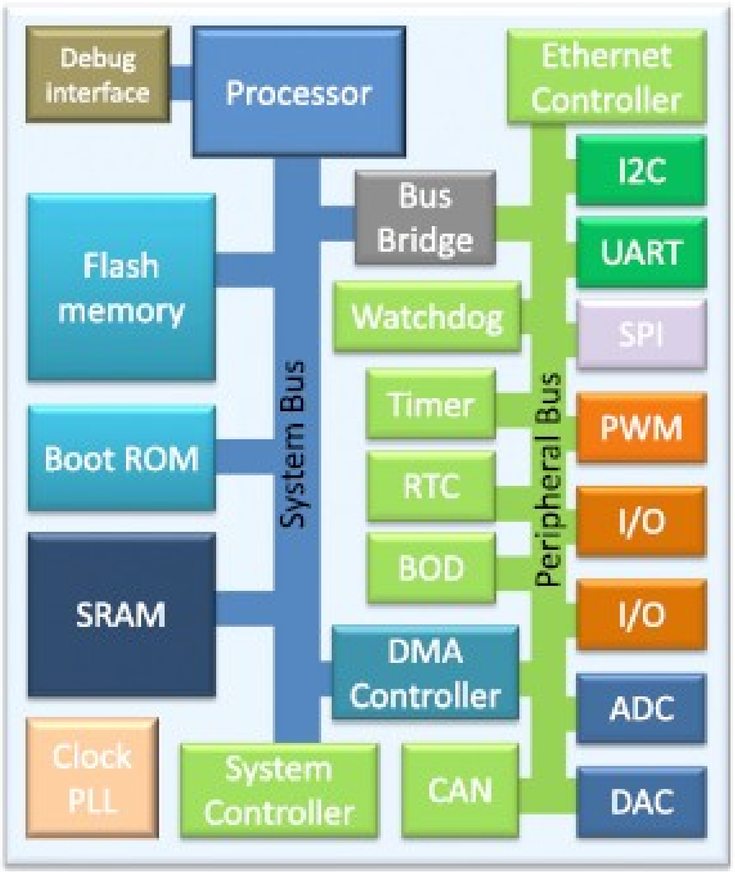
- Based on Cortex M4 processor
- 512KB ROM Flash memory
- 96KB SRAM data memory
- Up to 84Mhz operating frequency
- 42uA in sleep (stop mode) w/RTC

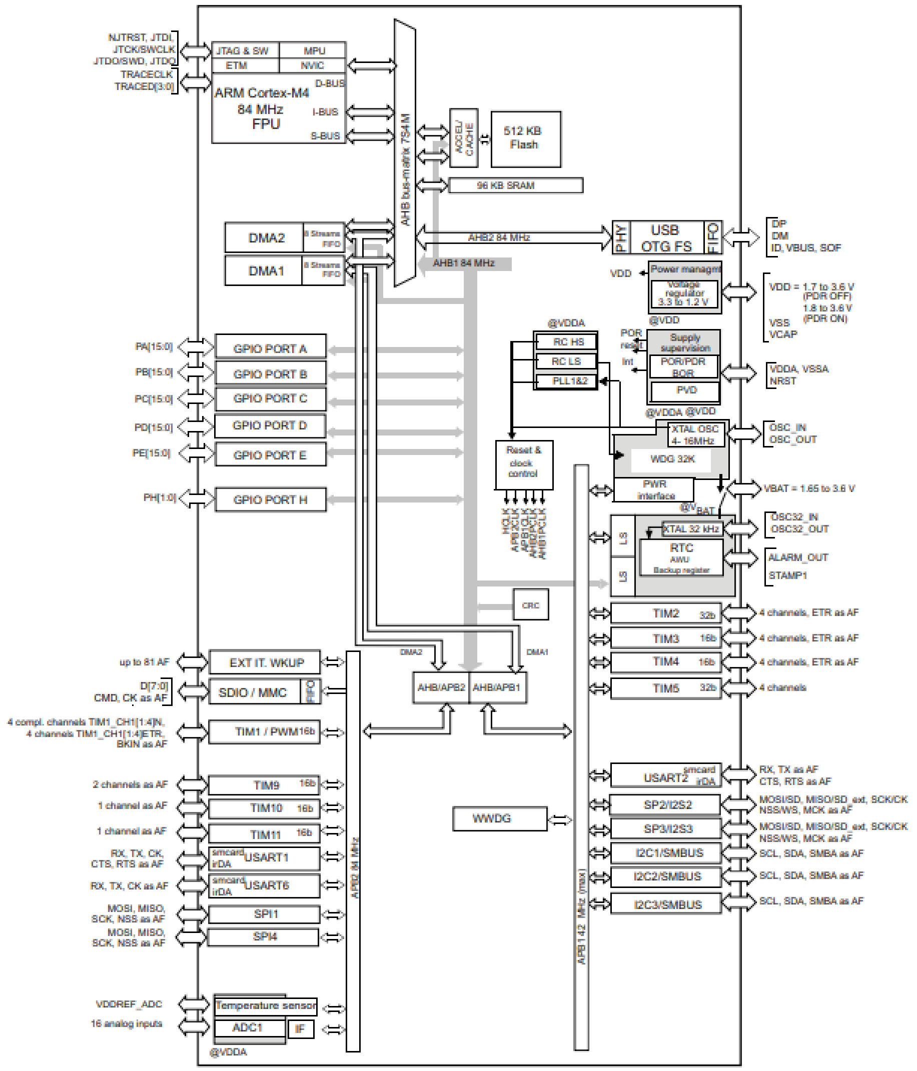
WLCSP49 (3.06 x 3.06 mm) LQFP100 (14 × 14 mm LQFP64 (10 × 10 mm) UFQFPN48 (7 × 7 mm) UFBGA100 (7 × 7 mm)

Available packages

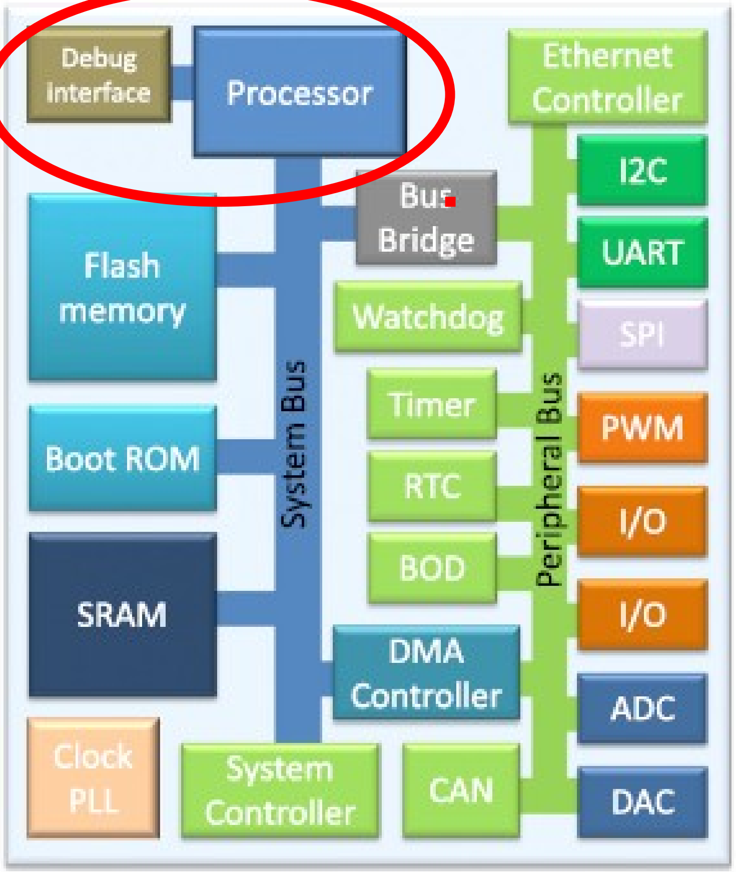


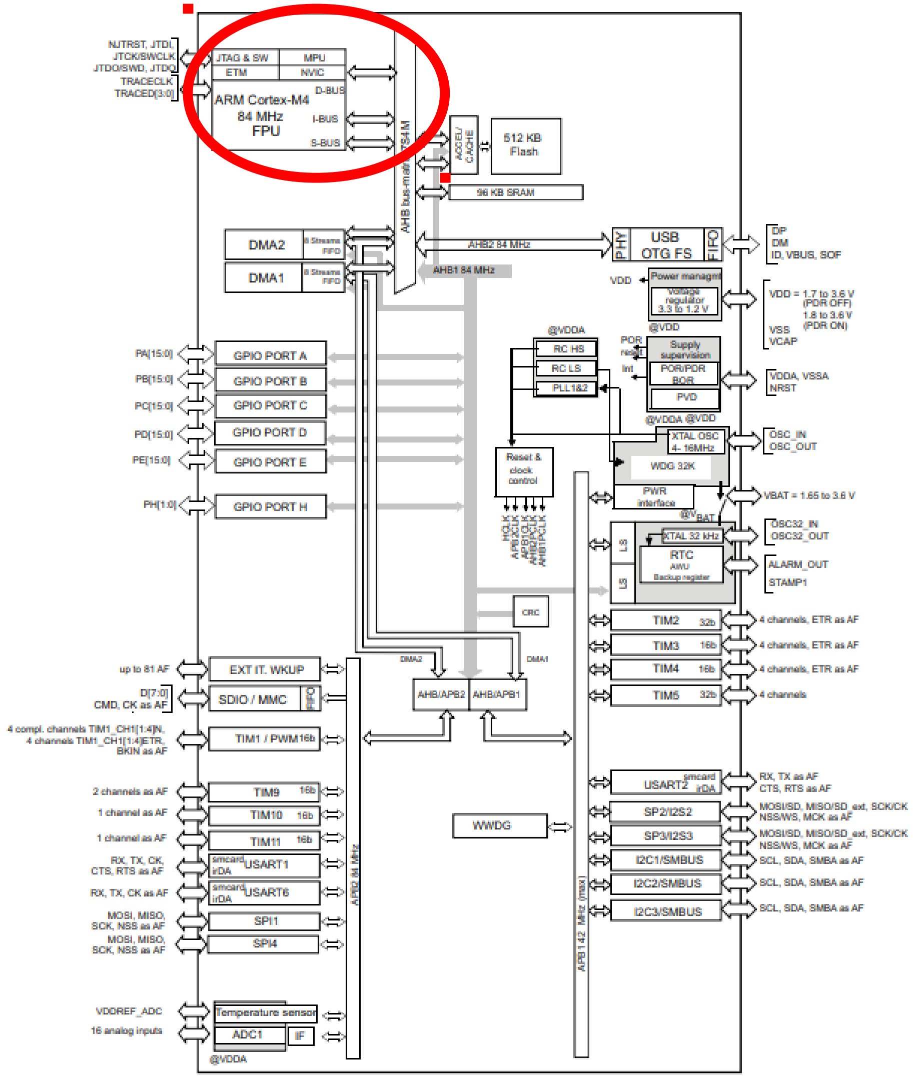
Architecture



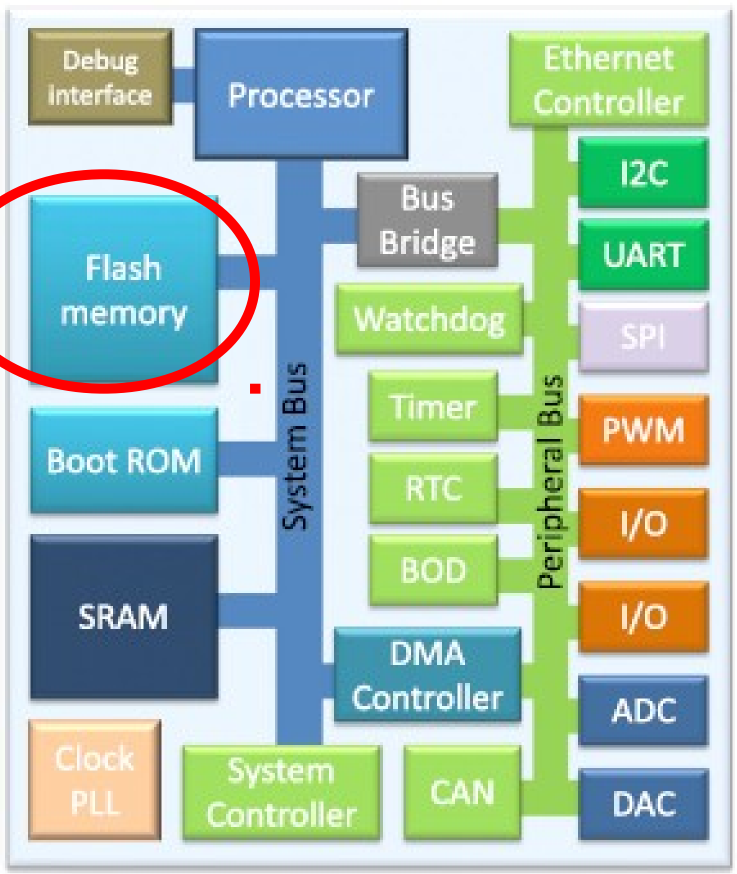


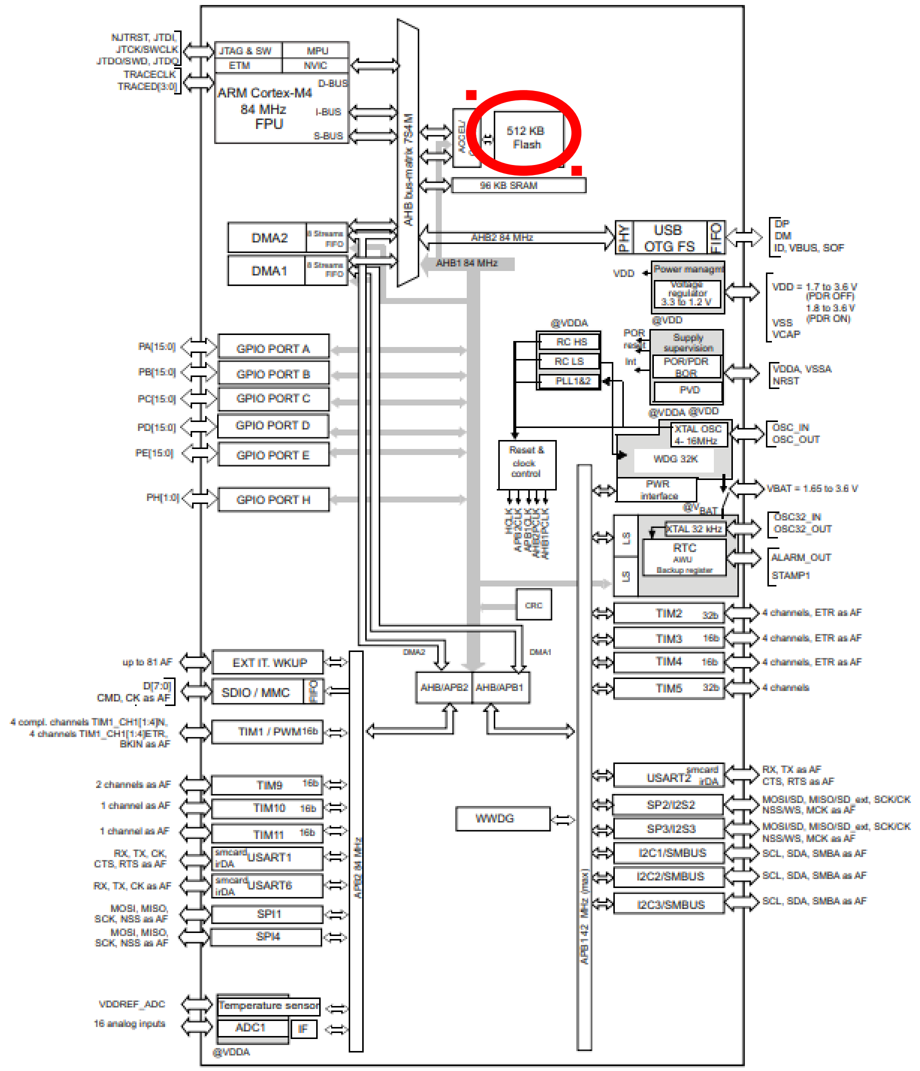
Architecture



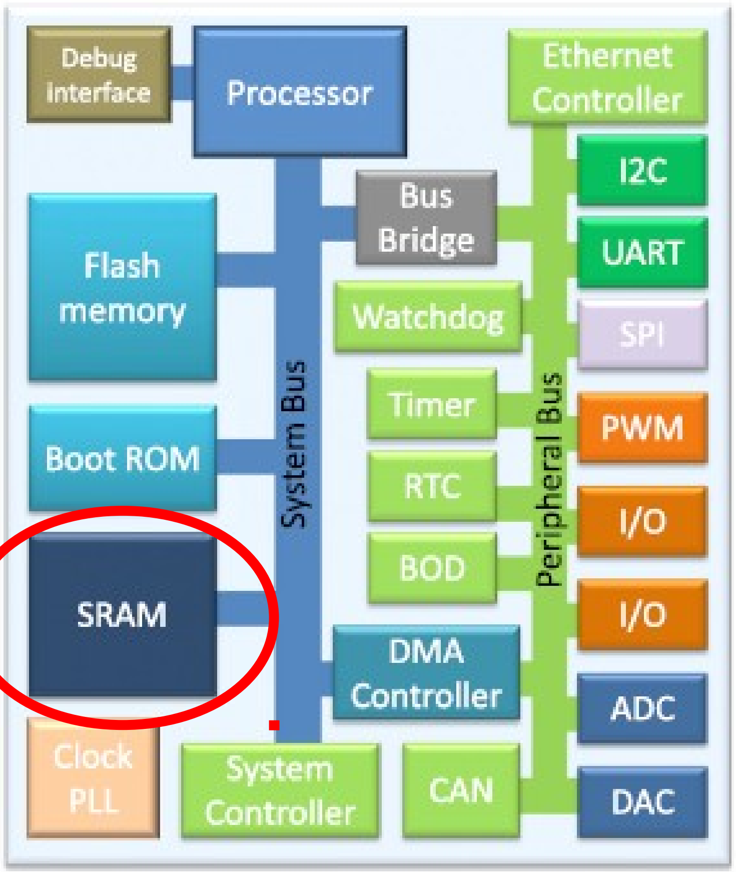


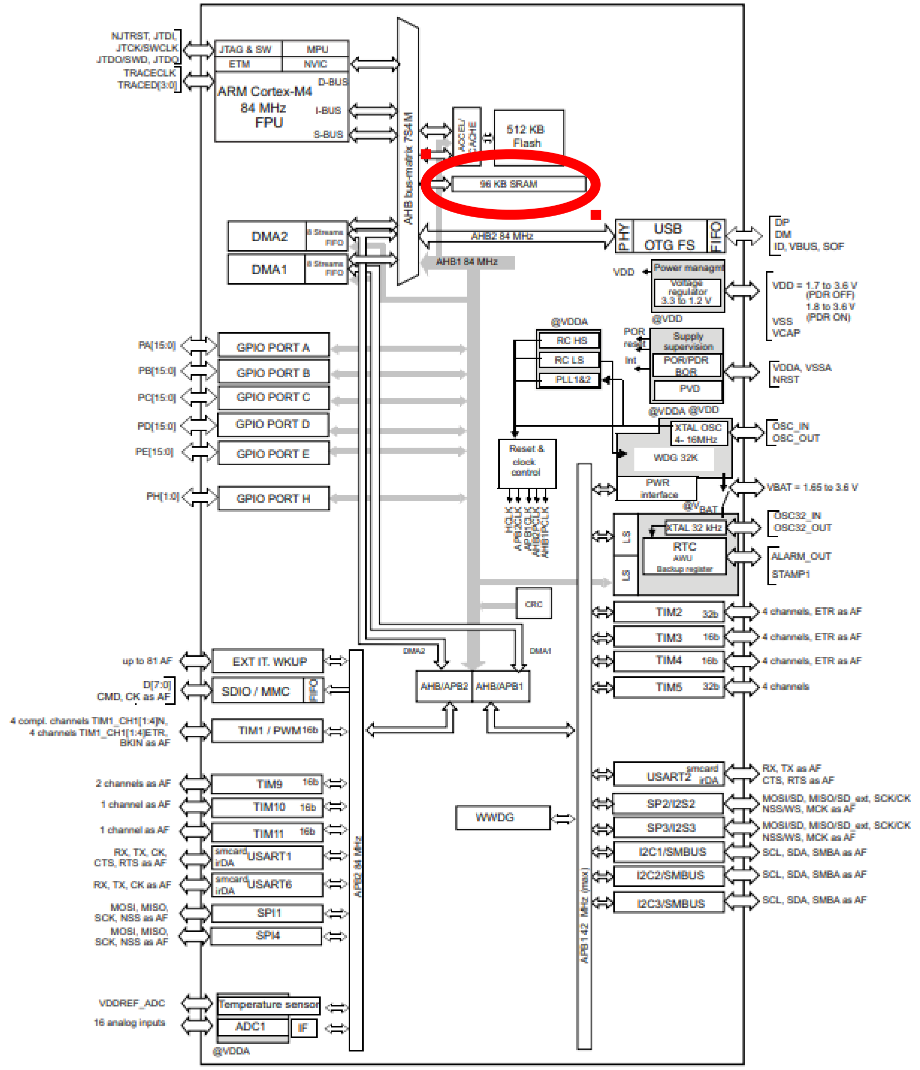
Architecture



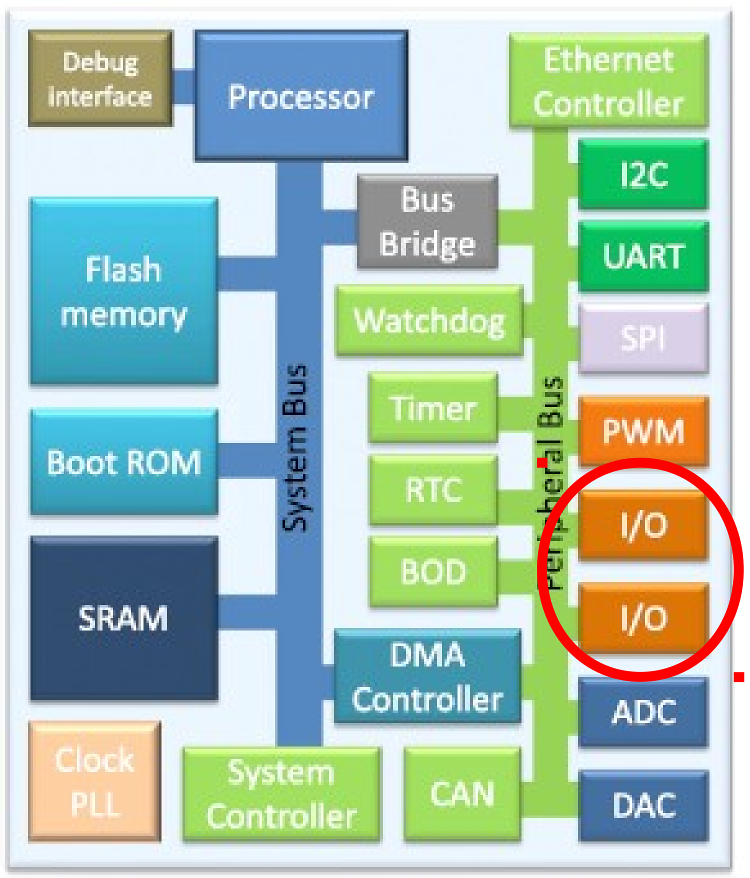


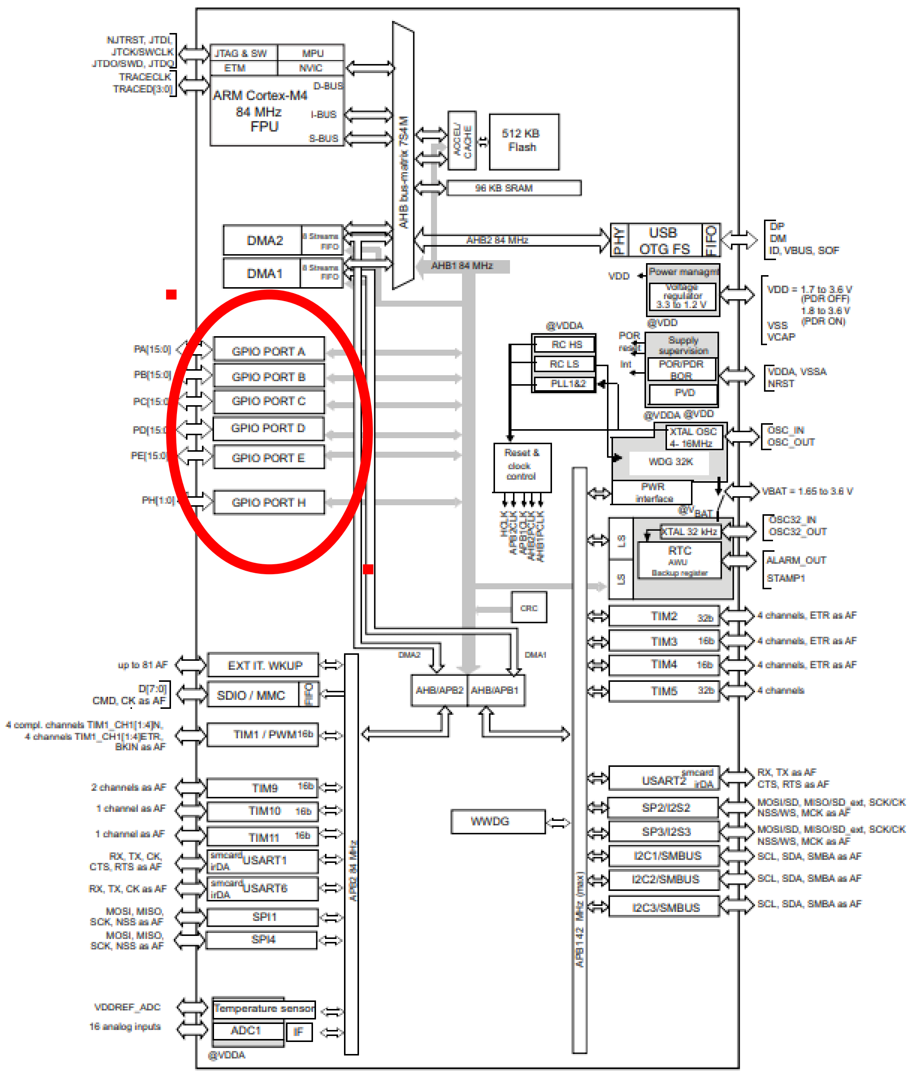
Architecture





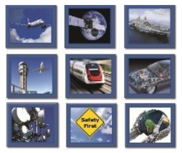
Architecture

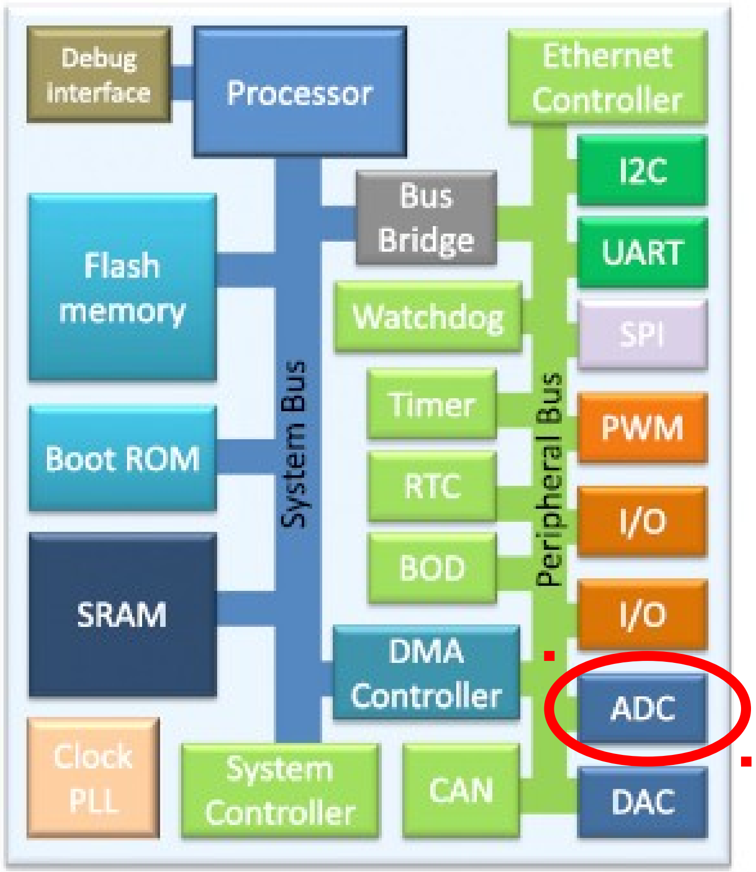


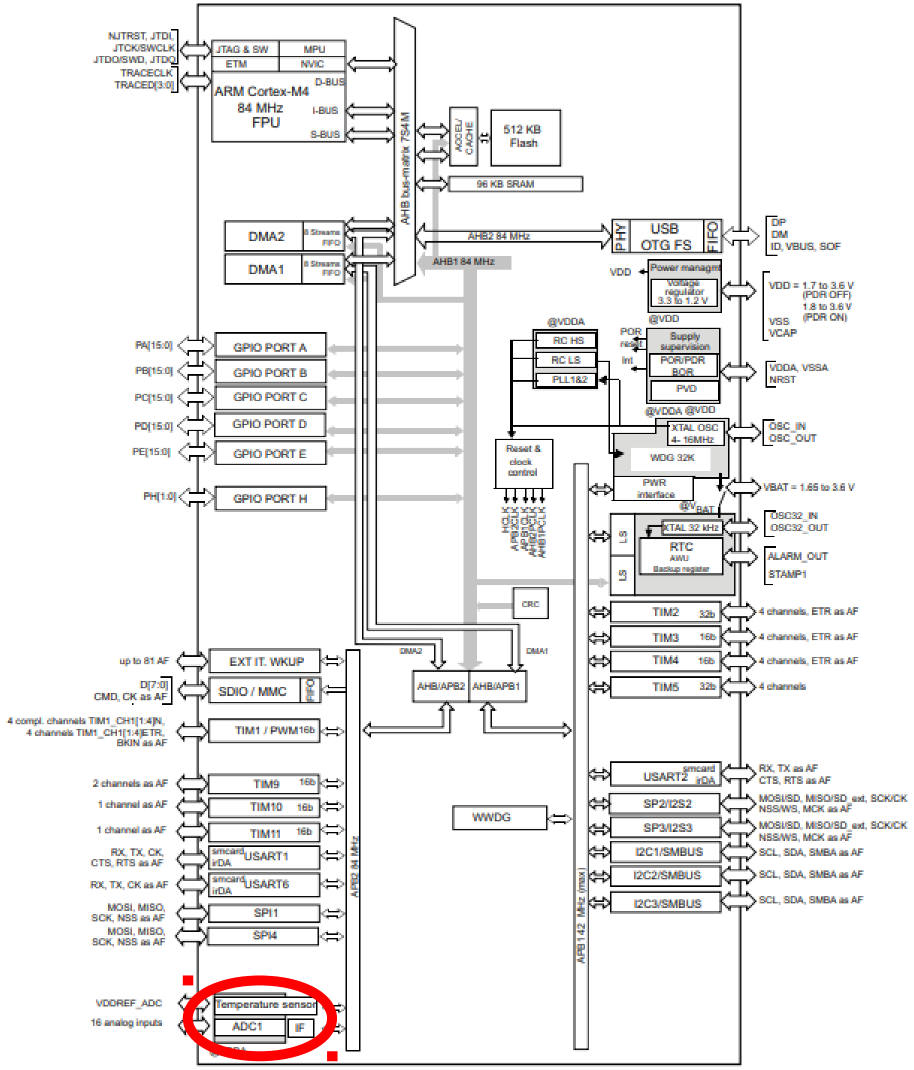


intecs Solutions
the Brainware company

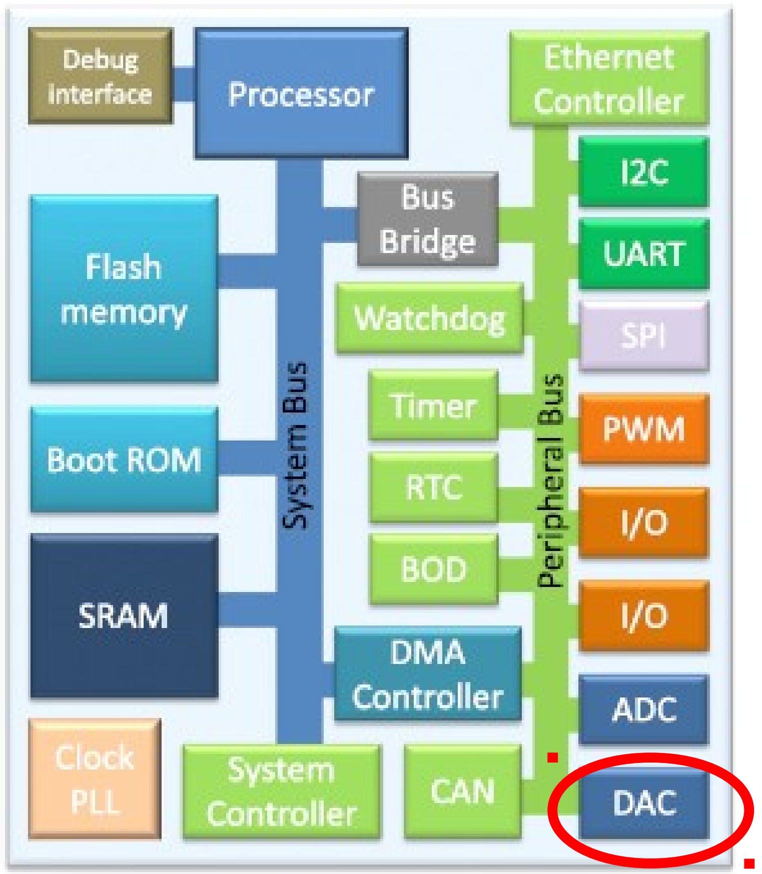
Architecture

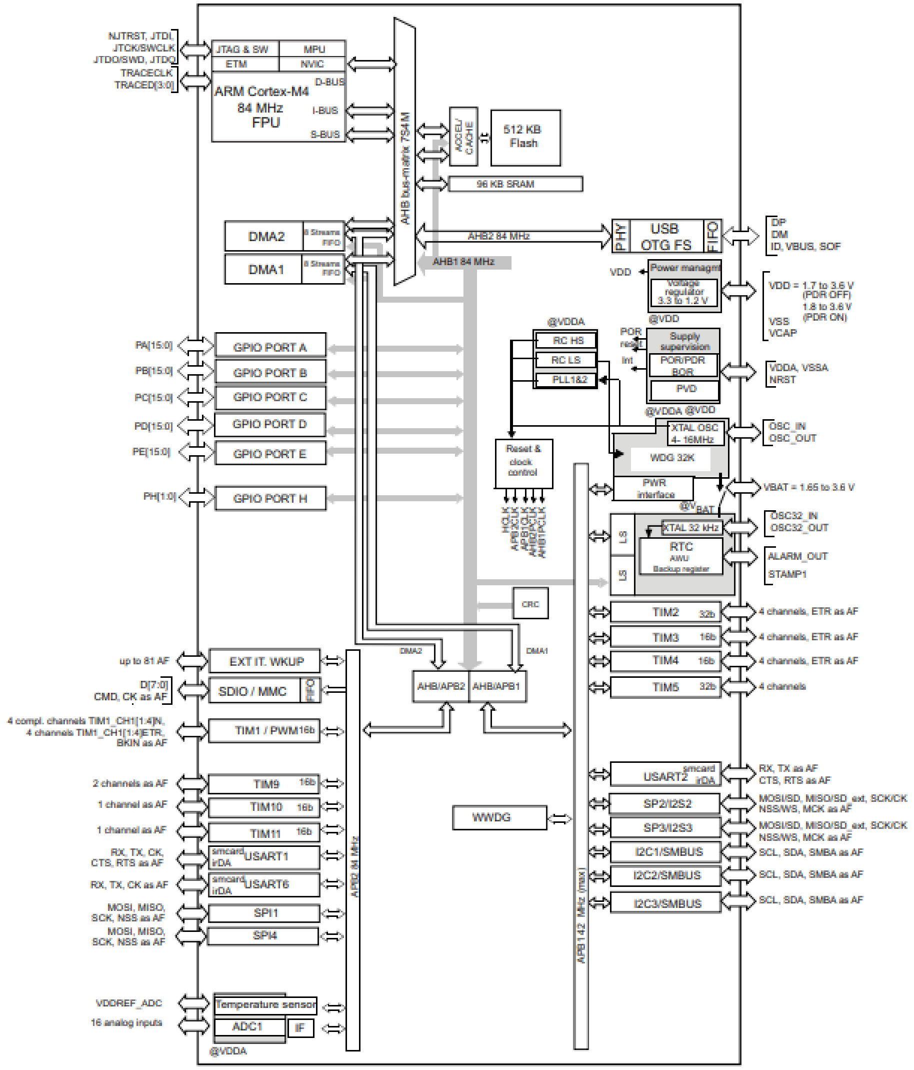






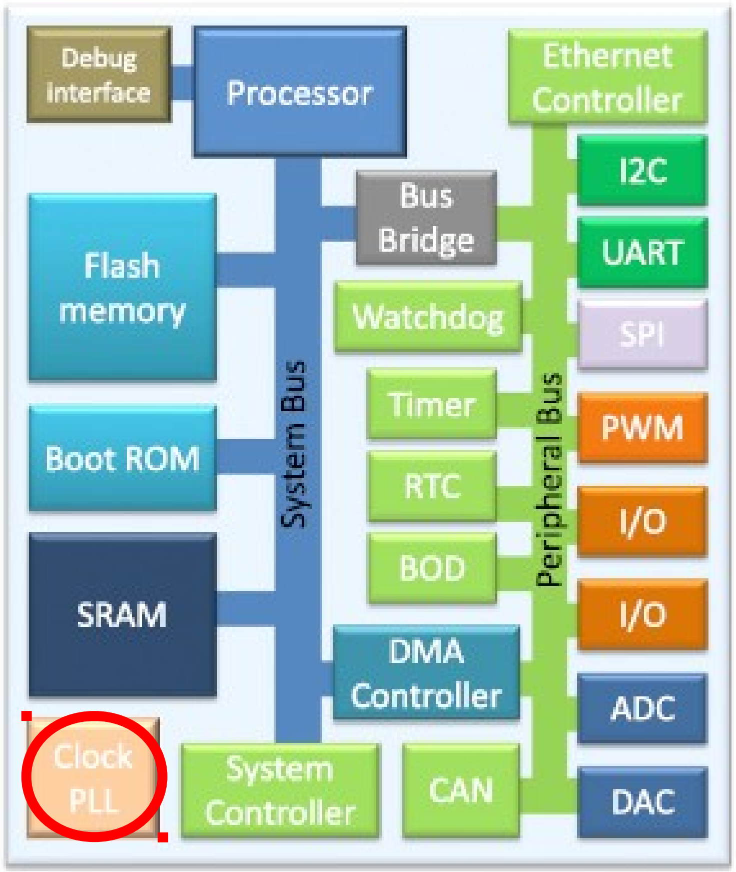
Architecture

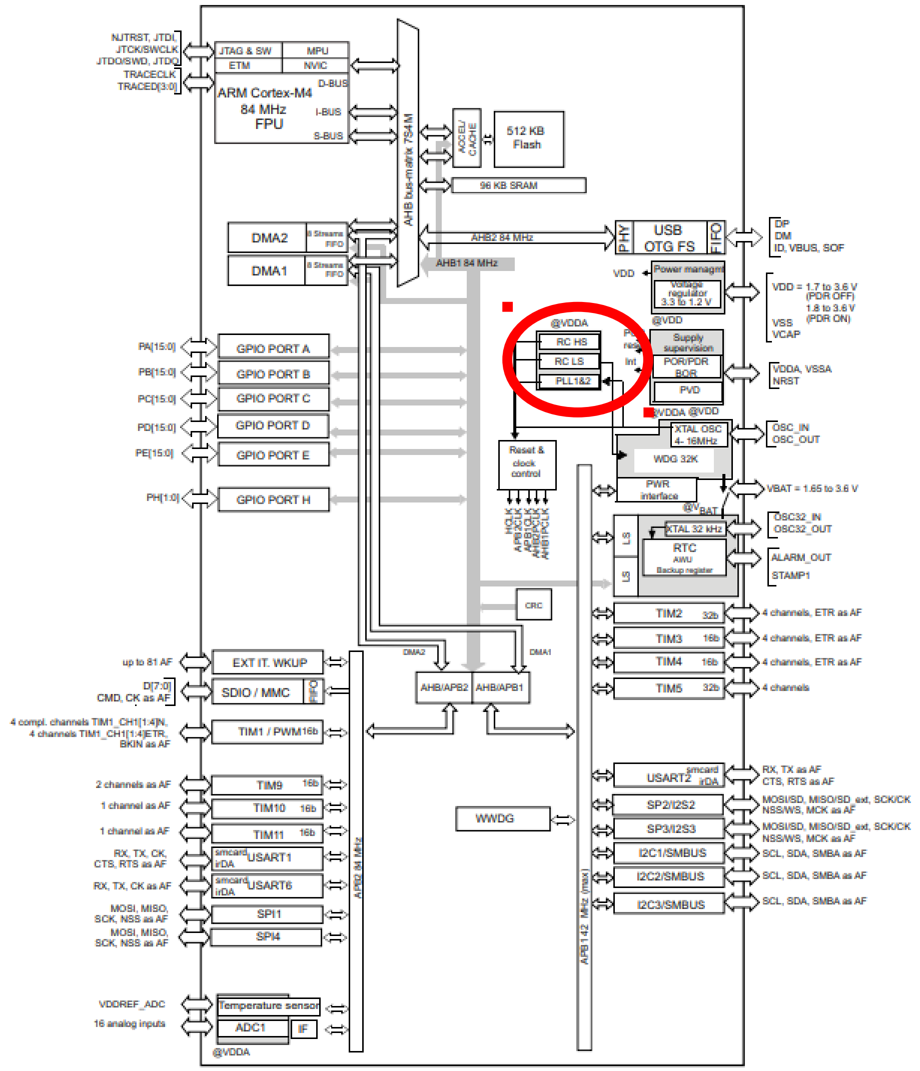




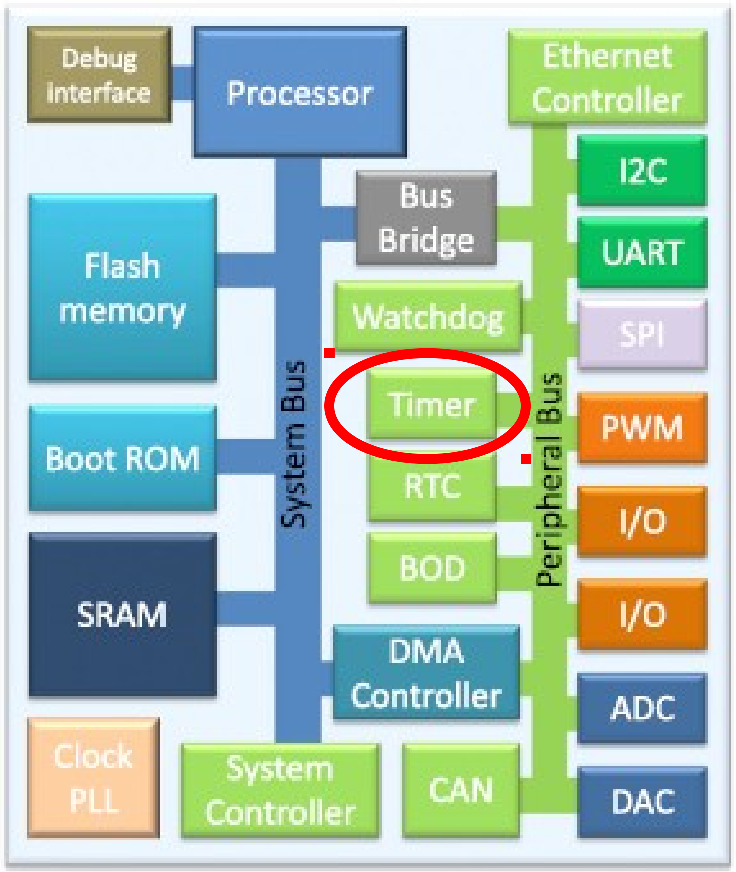
intecs Solutions
the Brainware company

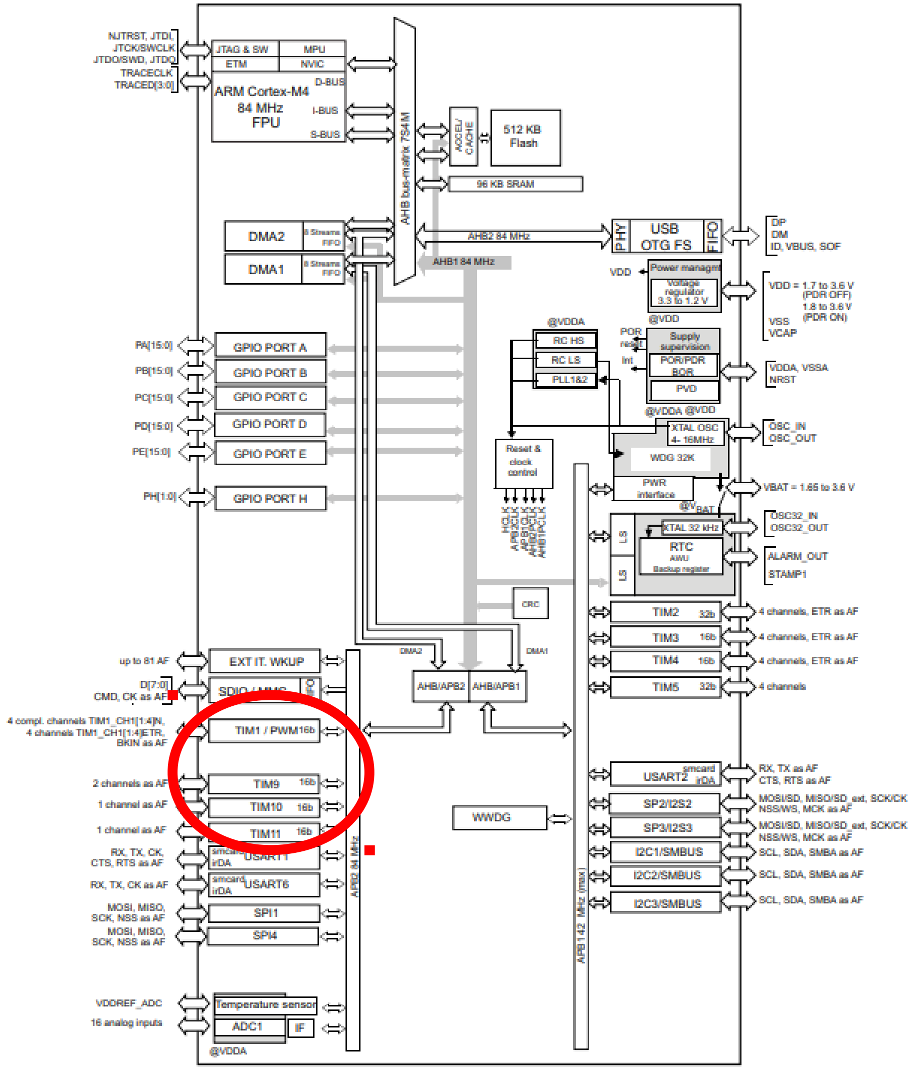
Architecture



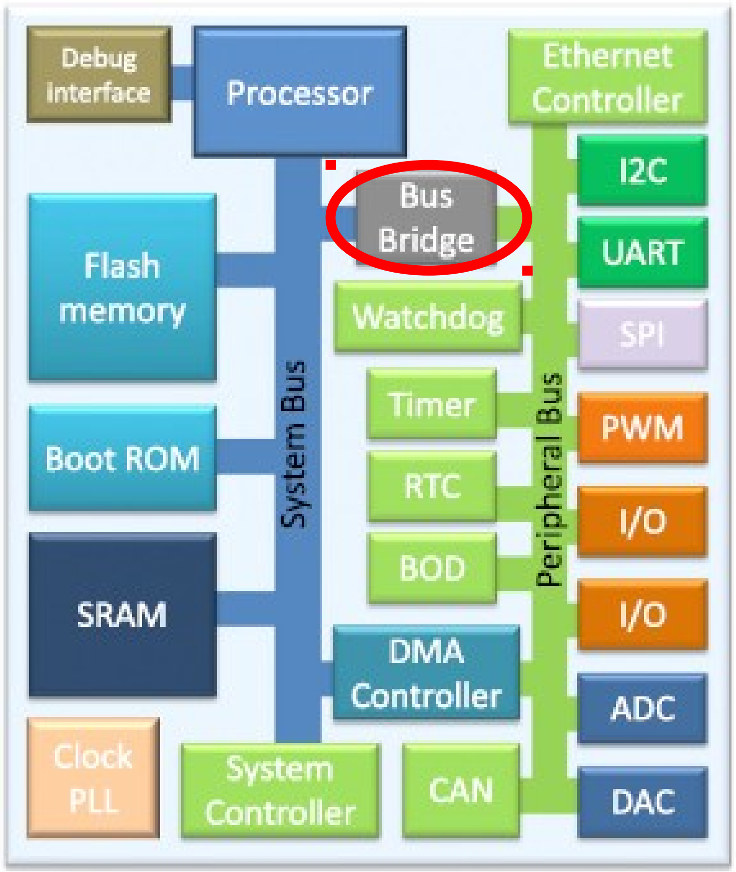


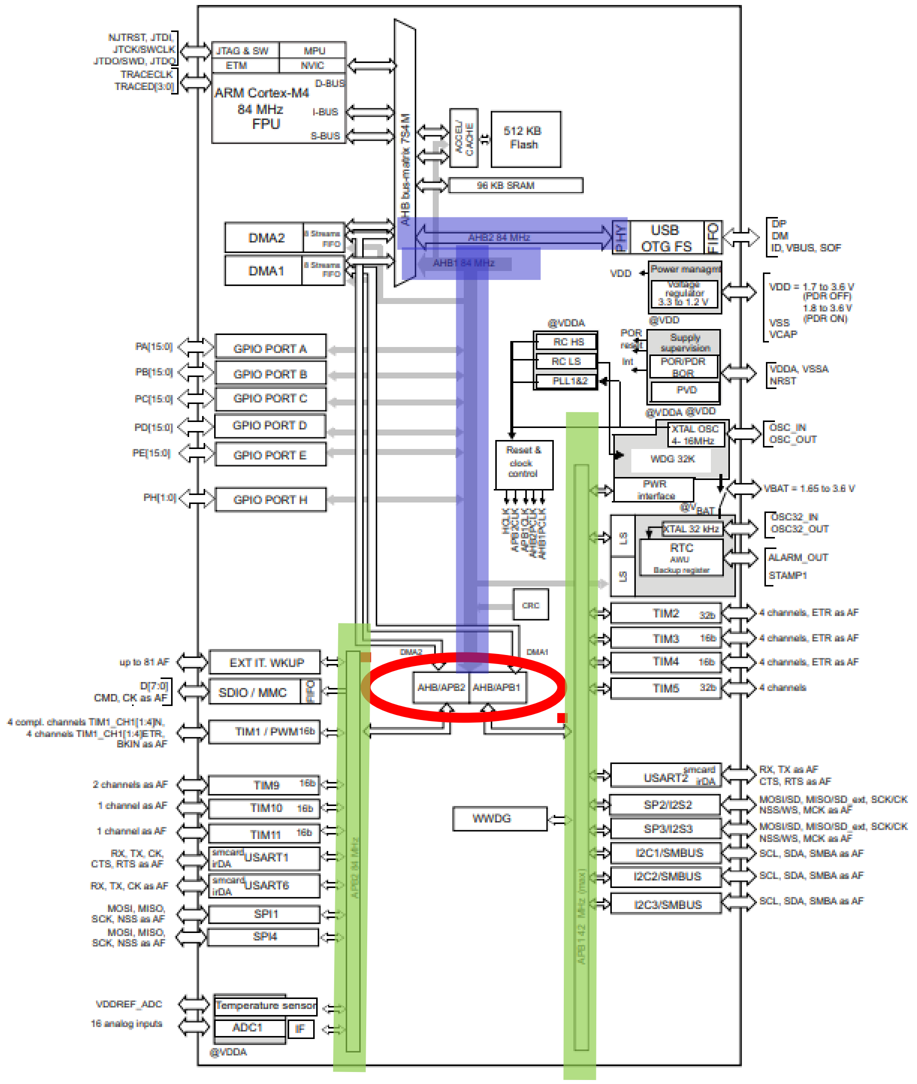
Architecture





Architecture



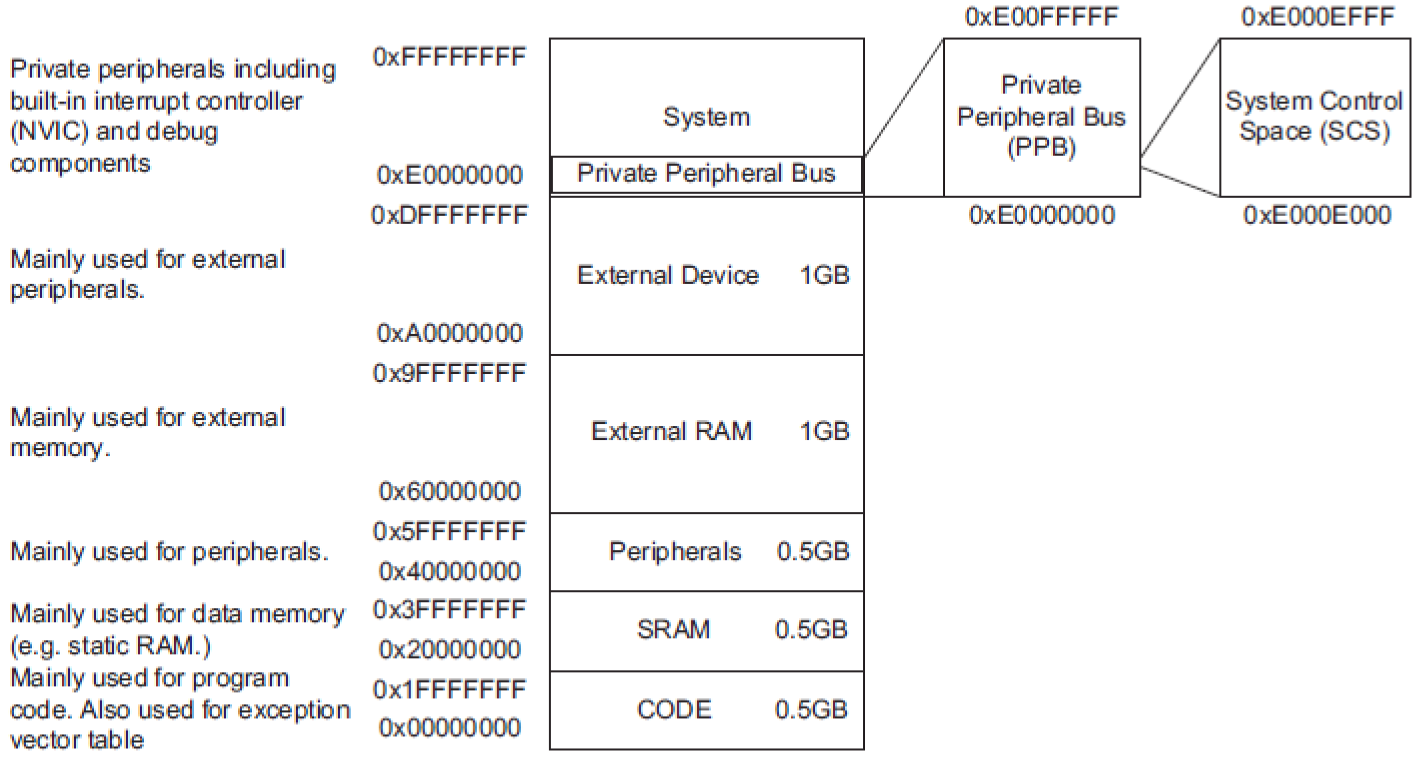


Cortex M-4

- 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


Private peripherals including built-in interrupt controller (NVIC) and debug components
Mainly used for external peripherals.
Mainly used for external memory.
Mainly used for peripherals.
Mainly used for data memory (e.g. static RAM.)
Mainly used for program code. Also used for exception vector table
0xFFFFFFFF
0xE0000000
0xDFFFFFFF
0xA0000000
0x9FFFFFFF
0x60000000
0x5FFFFFFF
0x40000000
0x3FFFFFFF
0x20000000
0x1FFFFFFF
0x00000000
System
Private Peripheral Bus
External Device 1GB
External RAM 1GB
Peripherals 0.5GB
SRAM 0.5GB
CODE 0.5GB
0xE00FFFFF
Private Peripheral Bus (PPB)
0xE0000000
0xE000EFFF
System Control Space (SCS)
0xE000E000

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

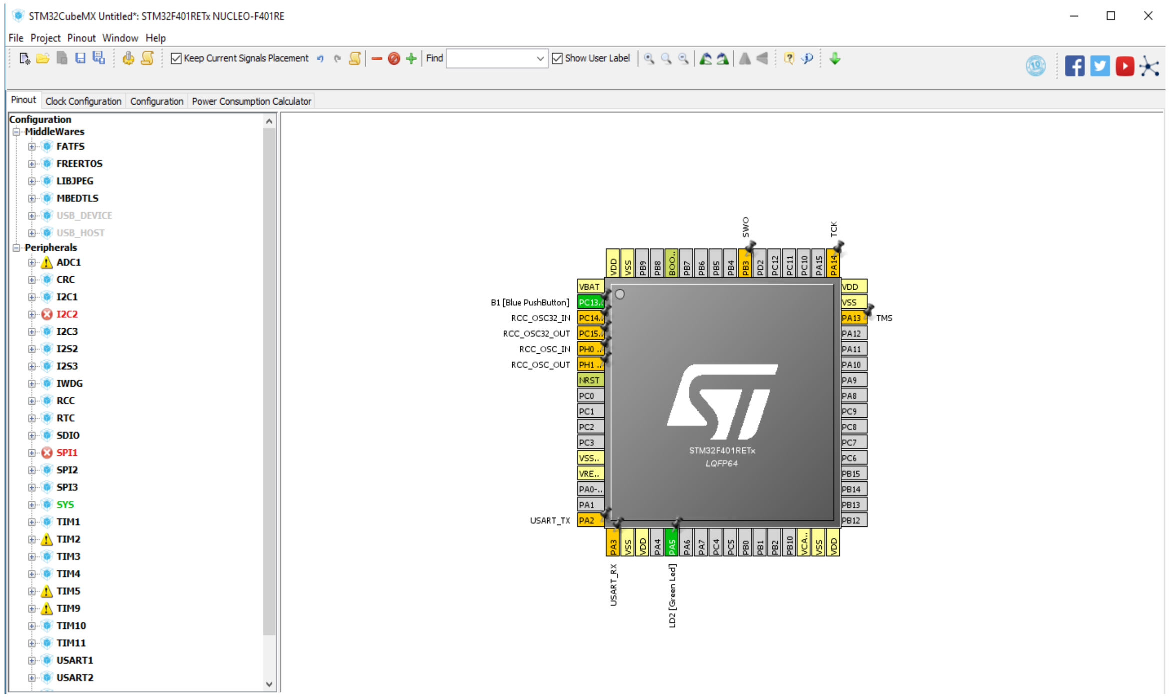
Stm32L476 Lookup

- 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

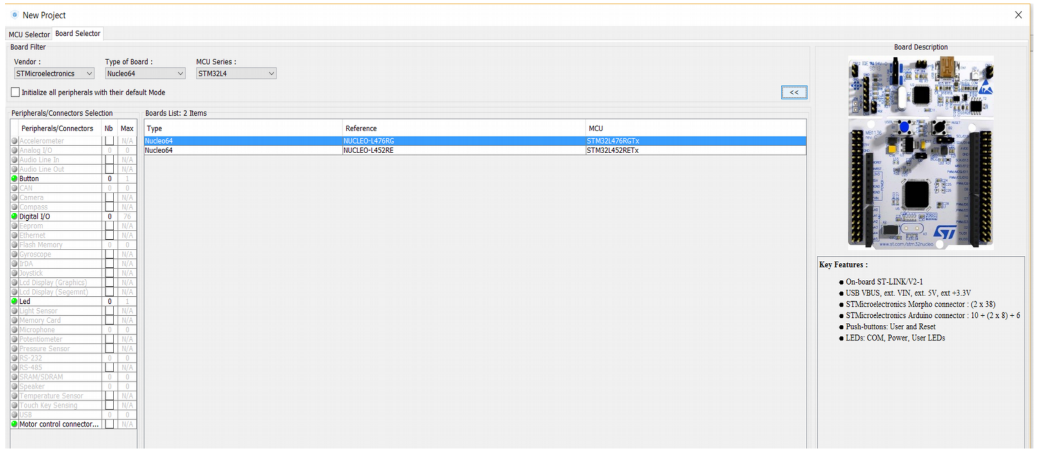
- Configuration tool:
  - Clock sources
  - Peripherals
  - Pinout
  - Middlewares
- Code generation:
  - IDE support



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



Vendor : STMicroelectronics
Type of Board : Nucleo64
MCU Series : STM32L4

☐ Initialize all peripherals with their default Mode

Peripherals/Connectors Selection

	Peripherals/Connectors	Nb	Max
●	Accelerometer	☐	N/A
●	Analog I/O	0	0
●	Audio Line In	☐	N/A
●	Audio Line Out	☐	N/A
🟢	Button	0	1
●	CAN	0	0
●	Camera	☐	N/A
●	Compass	☐	N/A
🟢	Digital I/O	0	76
●	Eeprom	☐	N/A
●	Ethernet	☐	N/A
●	Flash Memory	0	0
●	Gyroscope	☐	N/A
●	IrDA	☐	N/A
●	Joystick	☐	N/A
●	Lcd Display (Graphics)	☐	N/A
●	Lcd Display (Segemnt)	☐	N/A
🟢	Led	0	1
●	Light Sensor	☐	N/A
●	Memory Card	☐	N/A
●	Microphone	0	0
●	Potentiometer	☐	N/A
●	Pressure Sensor	☐	N/A
●	RS-232	0	0
●	RS-485	☐	N/A
●	SRAM/SDRAM	0	0
●	Speaker	0	0
●	Temperature Sensor	☐	N/A
●	Touch Key Sensing	☐	N/A
●	USB	0	0
🟢	Motor control connector...	☐	N/A

Boards List: 2 Items

Type	Reference	MCU
Nucleo64	NUCLEO-L476RG	STM32L476RGTx
Nucleo64	NUCLEO-L452RE	STM32L452RETx

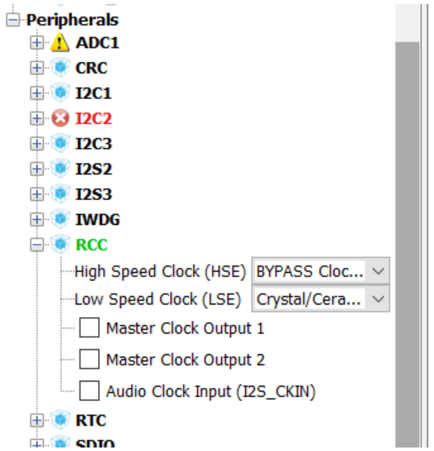
Key Features :

- On-board ST-LINK/V2-1
- USB VBUS, ext. VIN, ext. 5V, ext +3.3V
- STMicroelectronics Morpho connector : (2 x 38)
- STMicroelectronics Arduino connector : 10 + (2 x 8) + 6
- Push-buttons: User and Reset
- LEDs: COM, Power, User LEDs

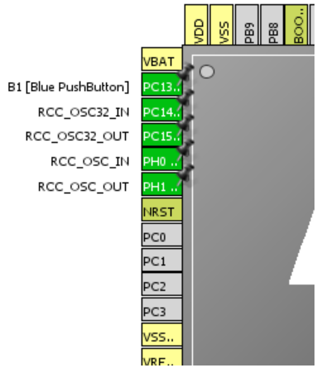
STM CubeMX

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



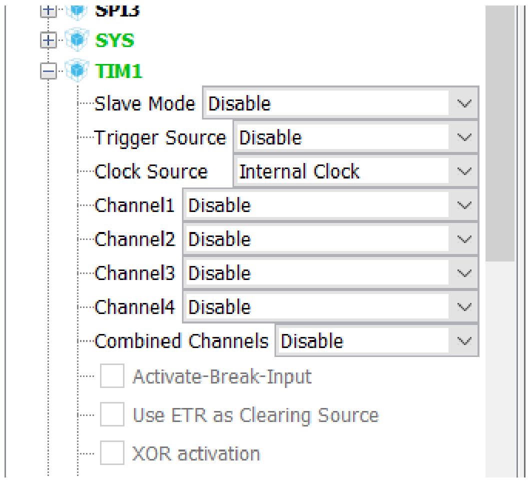
Peripherals
- ADC1
- CRC
- I2C1
- I2C2
- I2C3
- I2S2
- I2S3
- IWDG
- RCC
  - High Speed Clock (HSE): BYPASS Cloc...
  - Low Speed Clock (LSE): Crystal/Cera...
  - ☐ Master Clock Output 1
  - ☐ Master Clock Output 2
  - ☐ Audio Clock Input (I2S_CKIN)
- RTC
- SDIO



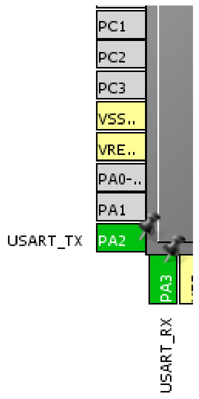
STM CubeMX

Usage Example: Clock and Timer 1 configuration

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



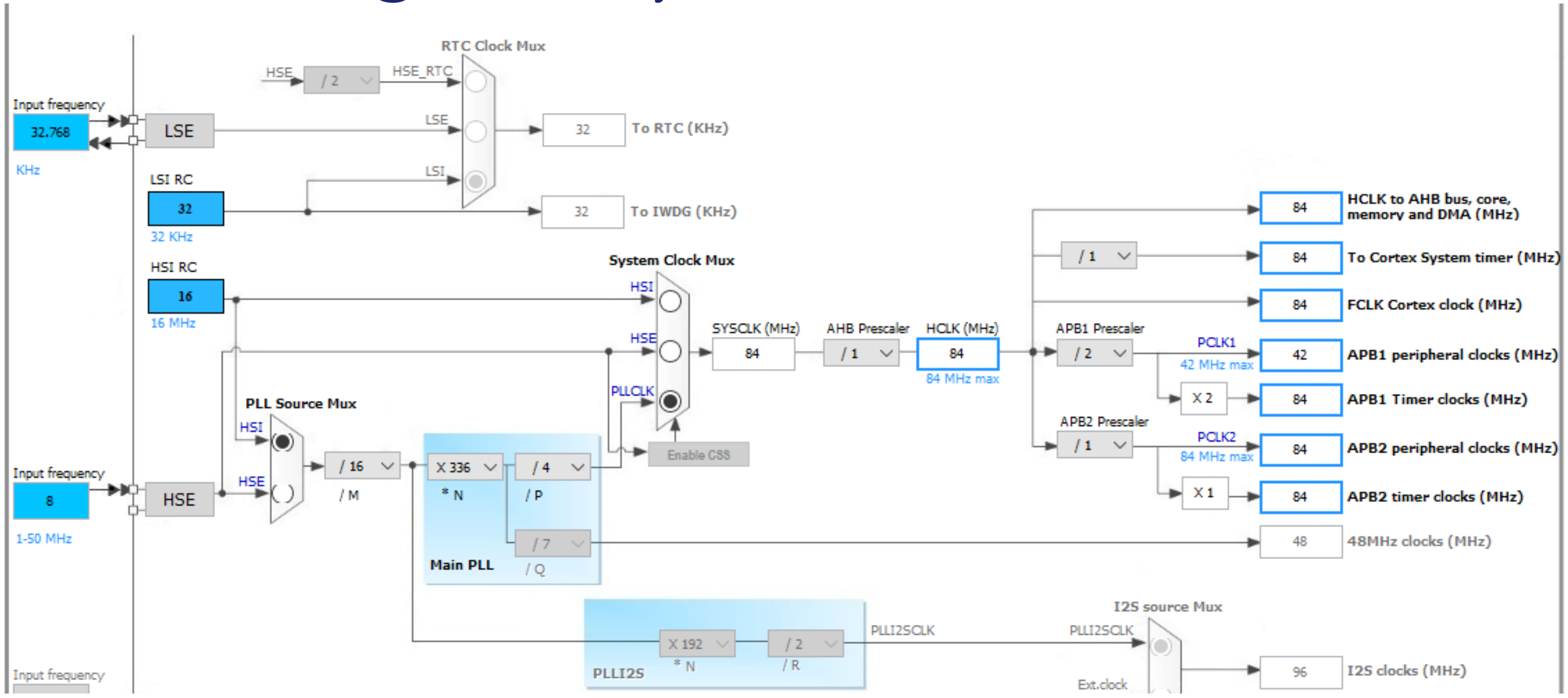
SPI3
SYS
TIM1
Slave Mode: Disable
Trigger Source: Disable
Clock Source: Internal Clock
Channel1: Disable
Channel2: Disable
Channel3: Disable
Channel4: Disable
Combined Channels: Disable
☐ Activate-Break-Input
☐ Use ETR as Clearing Source
☐ XOR activation



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

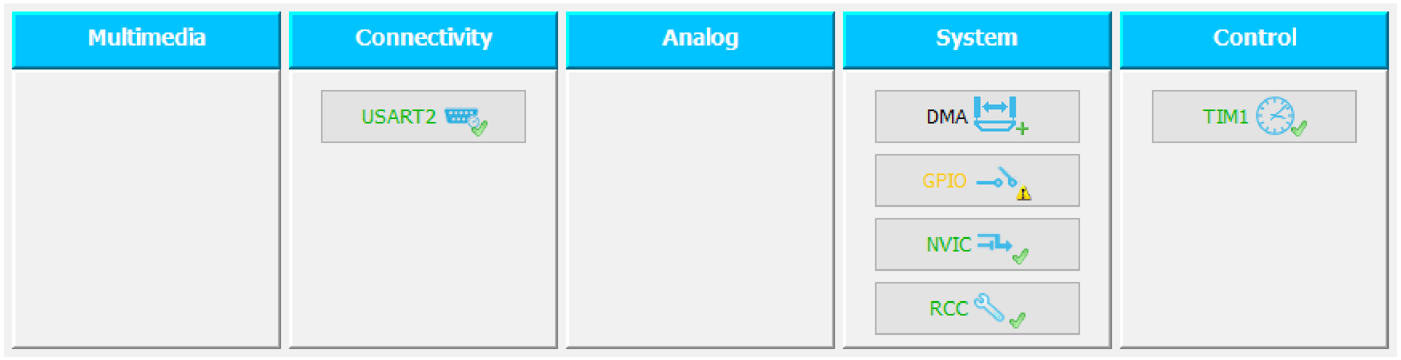
- Step 5:
  - Q: Which bus is connected to TIM1?
  - Annotate its frequency

STM CubeMX

Usage Example: Clock and Timer 1 configuration

Step 7:

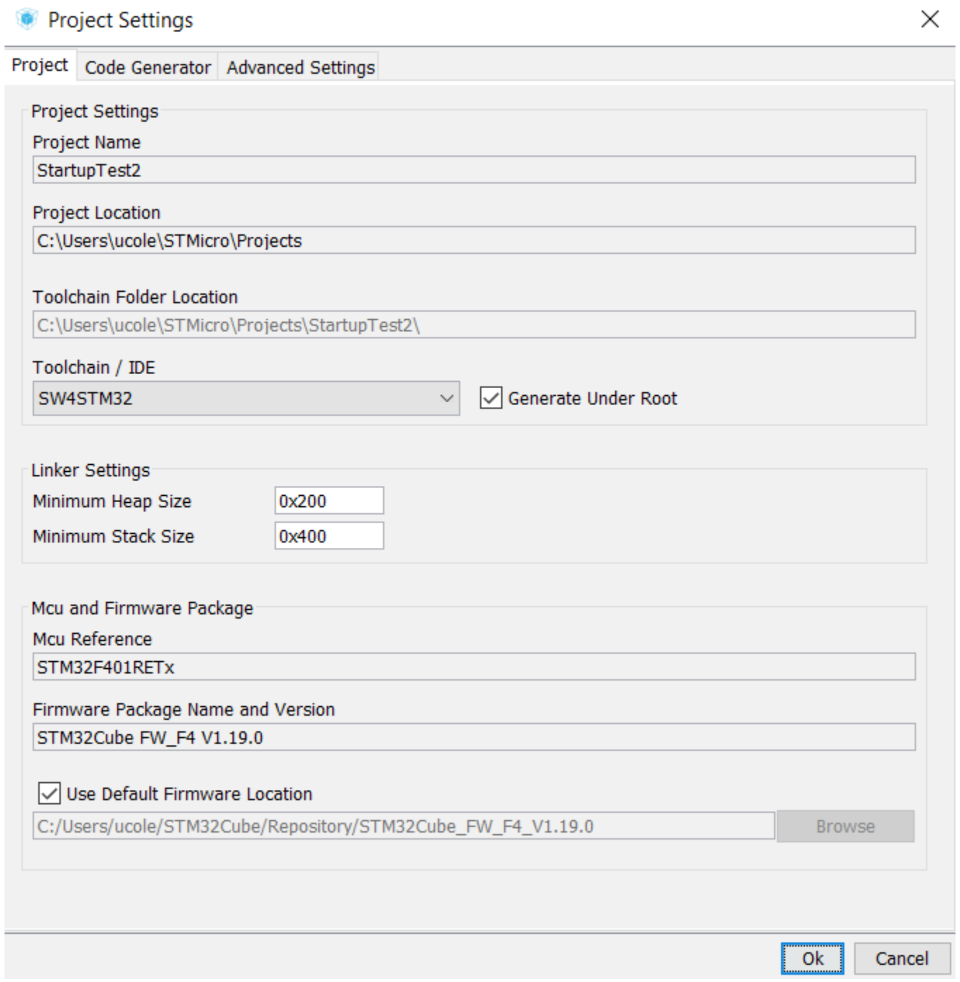
•From “Configuration” tab
•Check that peripherals and clocks are set correctly
•Double click on TIM1, select counter period to be 65535
•Q: What prescaler and division should we set for 1ms tick timer?
•NVIC settings enable TIM1 update interrupt



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


Project Settings

Project Settings

Project Name: StartupTest2

Project Location: C:\Users\ucole\STMicro\Projects

Toolchain Folder Location: C:\Users\ucole\STMicro\Projects\StartupTest2\

Toolchain / IDE: SW4STM32 ☑ Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

Mcu and Firmware Package

Mcu Reference: STM32F401RETx

Firmware Package Name and Version: STM32Cube FW_F4 V1.19.0

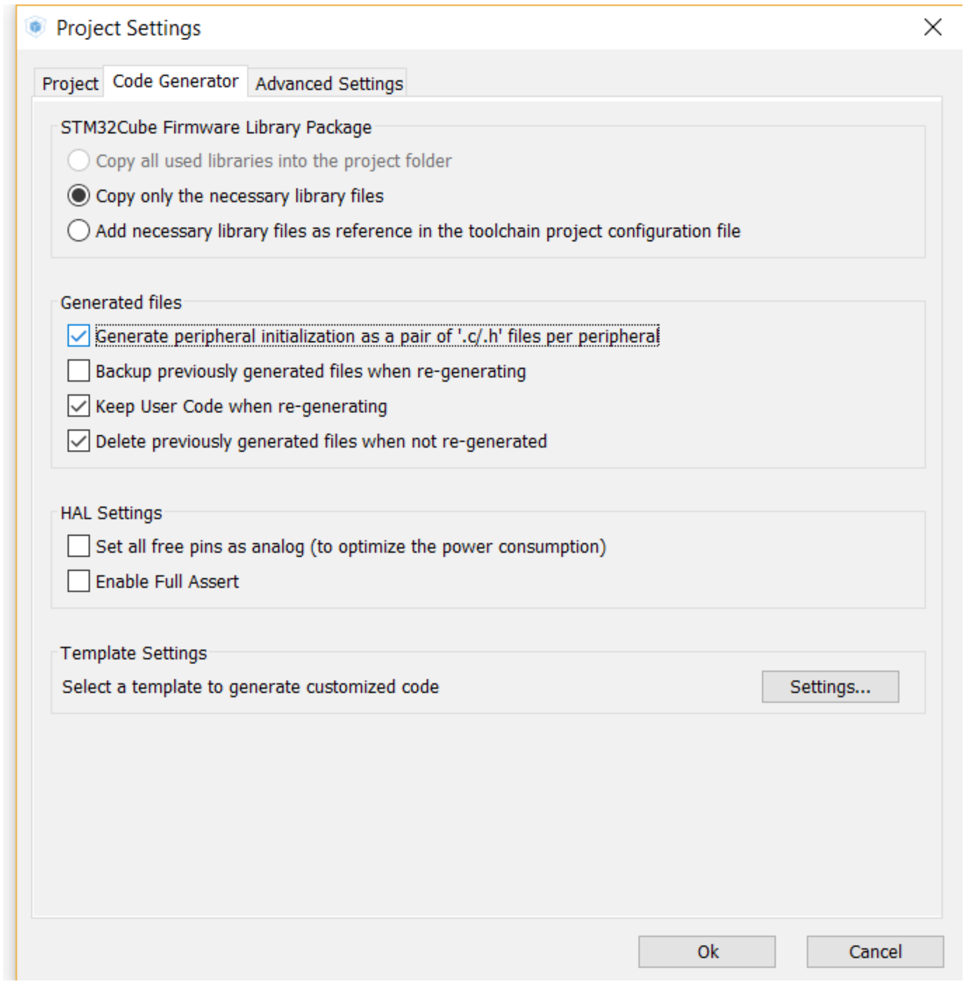
☑ Use Default Firmware Location

C:/Users/ucole/STM32Cube/Repository/STM32Cube_FW_F4_V1.19.0

STM CubeMX

Usage Example: Code Generation

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



STM32Cube Firmware Library Package

- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files
- ○ Add necessary library files as reference in the toolchain project configuration file

Generated files

- ☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- ☐ Backup previously generated files when re-generating
- ☑ Keep User Code when re-generating
- ☑ Delete previously generated files when not re-generated

HAL Settings

- ☐ Set all free pins as analog (to optimize the power consumption)
- ☐ Enable Full Assert

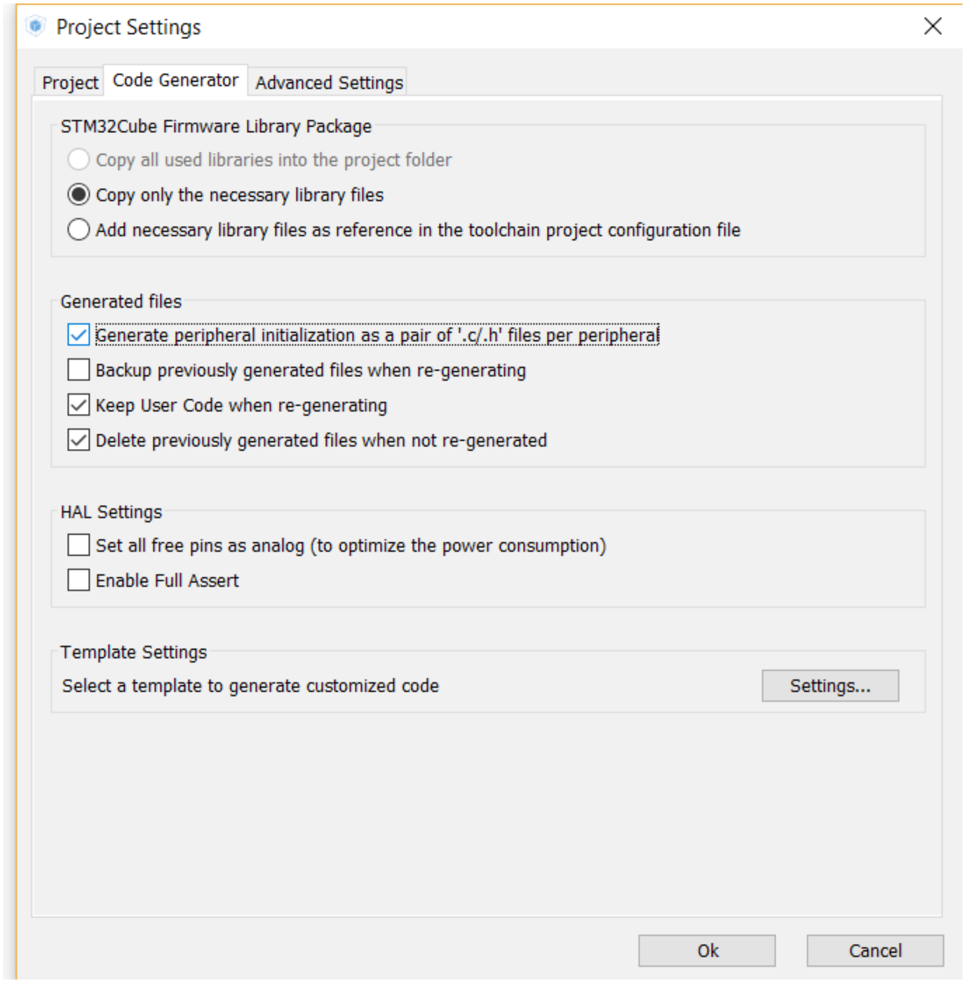
Template Settings

Select a template to generate customized code

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



Project Settings

Project | Code Generator | Advanced Settings

STM32Cube Firmware Library Package

- ○ Copy all used libraries into the project folder
- ● Copy only the necessary library files
- ○ Add necessary library files as reference in the toolchain project configuration file

Generated files

- ☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
- ☐ Backup previously generated files when re-generating
- ☑ Keep User Code when re-generating
- ☑ Delete previously generated files when not re-generated

HAL Settings

- ☐ Set all free pins as analog (to optimize the power consumption)
- ☐ Enable Full Assert

Template Settings

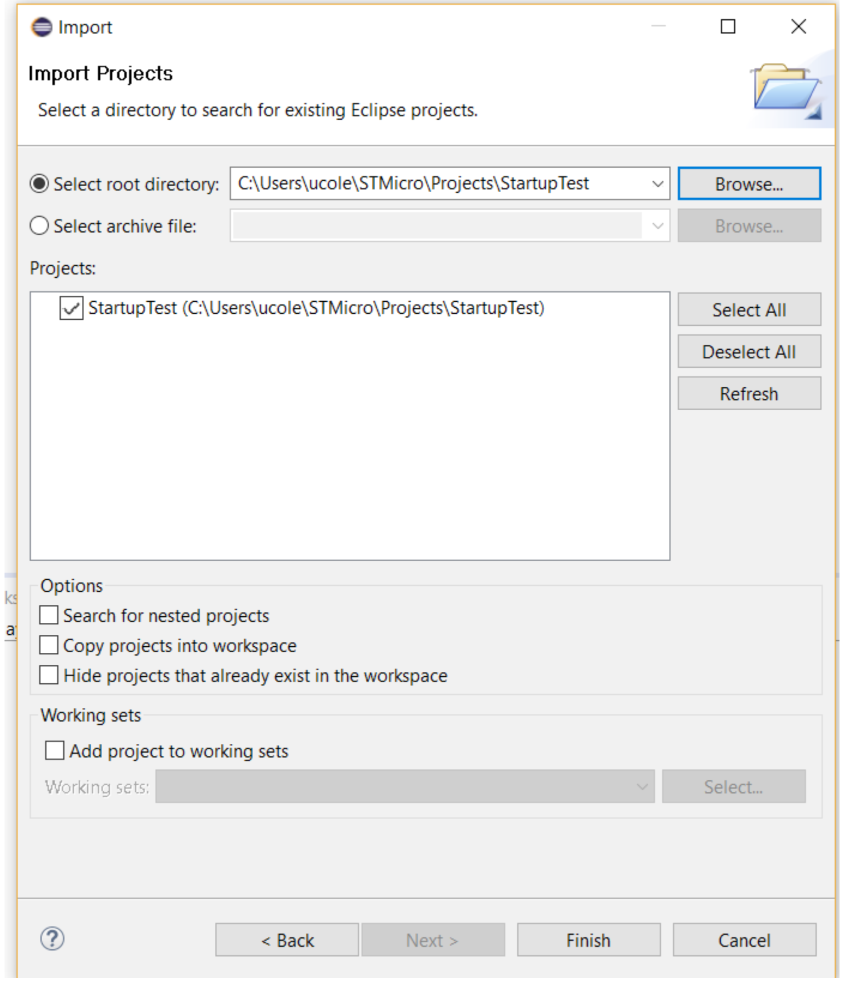
Select a template to generate customized code

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



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: C:\Users\ucole\STMicro\Projects\StartupTest

○ Select archive file:

Projects:

✓ StartupTest (C:\Users\ucole\STMicro\Projects\StartupTest)

Options
- ☐ Search for nested projects
- ☐ Copy projects into workspace
- ☐ Hide projects that already exist in the workspace

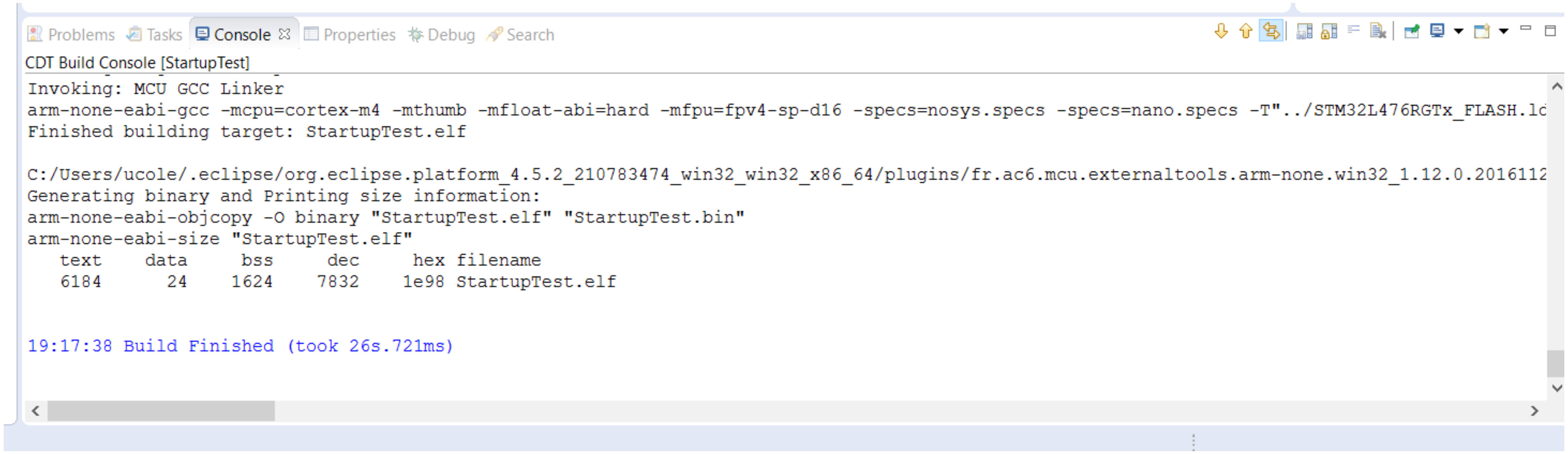
Working sets
- ☐ Add project to working sets

Working sets:

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



Problems Tasks Console Properties Debug Search

CDT Build Console [StartupTest]

```
Invoking: MCU GCC Linker
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -mfloat-abi=hard -mfpu=fpv4-sp-d16 -specs=nosys.specs -specs=nano.specs -T"../STM32L476RGTx_FLASH.ld
Finished building target: StartupTest.elf

C:/Users/ucole/.eclipse/org.eclipse.platform_4.5.2_210783474_win32_win32_x86_64/plugins/fr.ac6.mcu.externaltools.arm-none.win32_1.12.0.2016112
Generating binary and Printing size information:
arm-none-eabi-objcopy -O binary "StartupTest.elf" "StartupTest.bin"
arm-none-eabi-size "StartupTest.elf"
   text    data     bss     dec     hex filename
   6184      24    1624    7832    1e98 StartupTest.elf


19:17:38 Build Finished (took 26s.721ms)
```

System Workbench 4

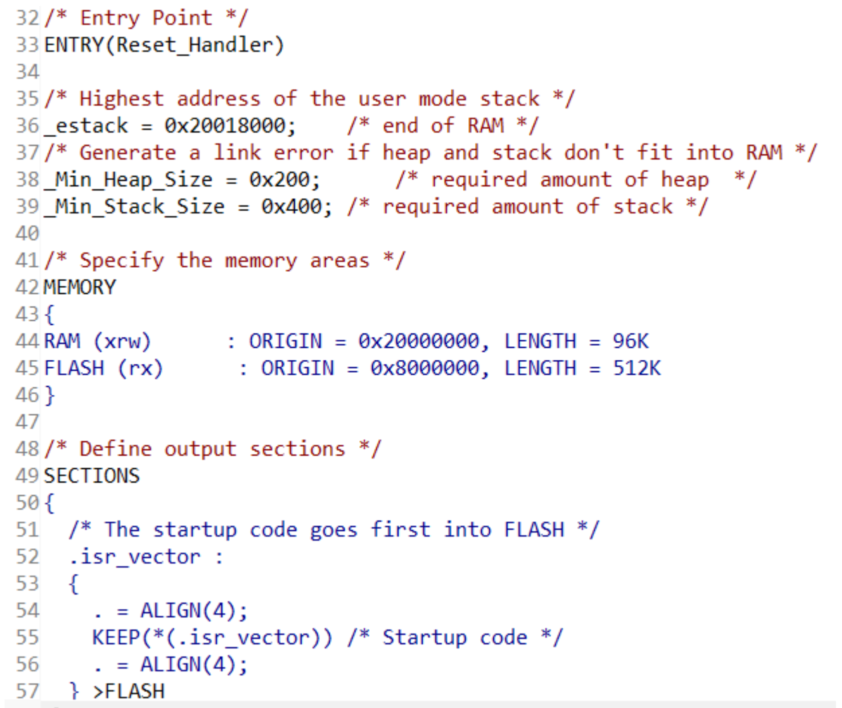
Importing project generated with CubeMX

- 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

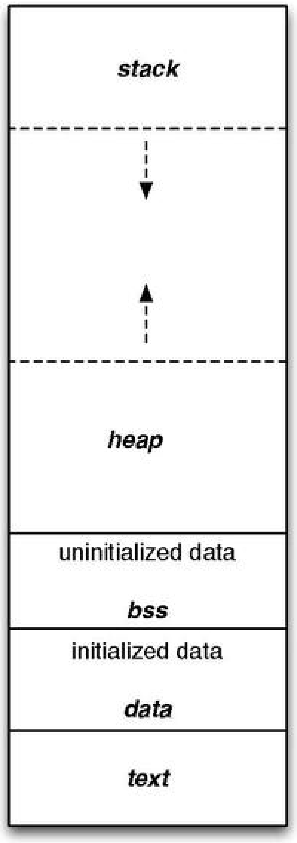
- In project explorer: STM32F401RETx_FLASH.ld
- Where to find program and data memory (RAM,FLASH) w.r.t. the linear memory map of the MCU
- What to put inside each area (e.g., .isr_vector, .text and constant data in flash, .data and .bss in ram etc…)



```
/* 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

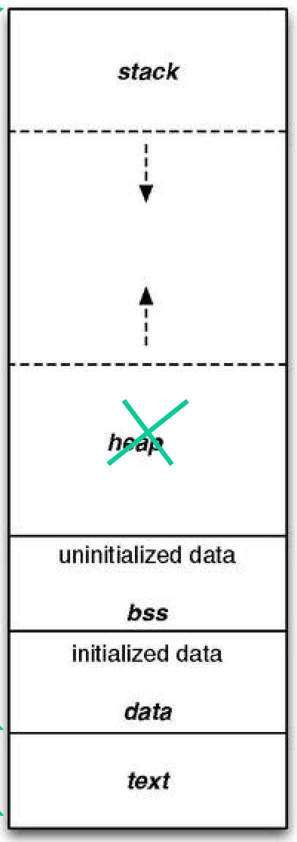
Read only data (e.g., code)

System Workbench 4

Important files: the linker script

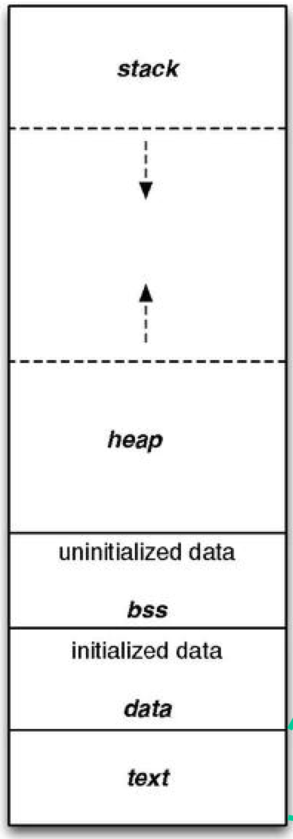
- Quick recall on memory segments:

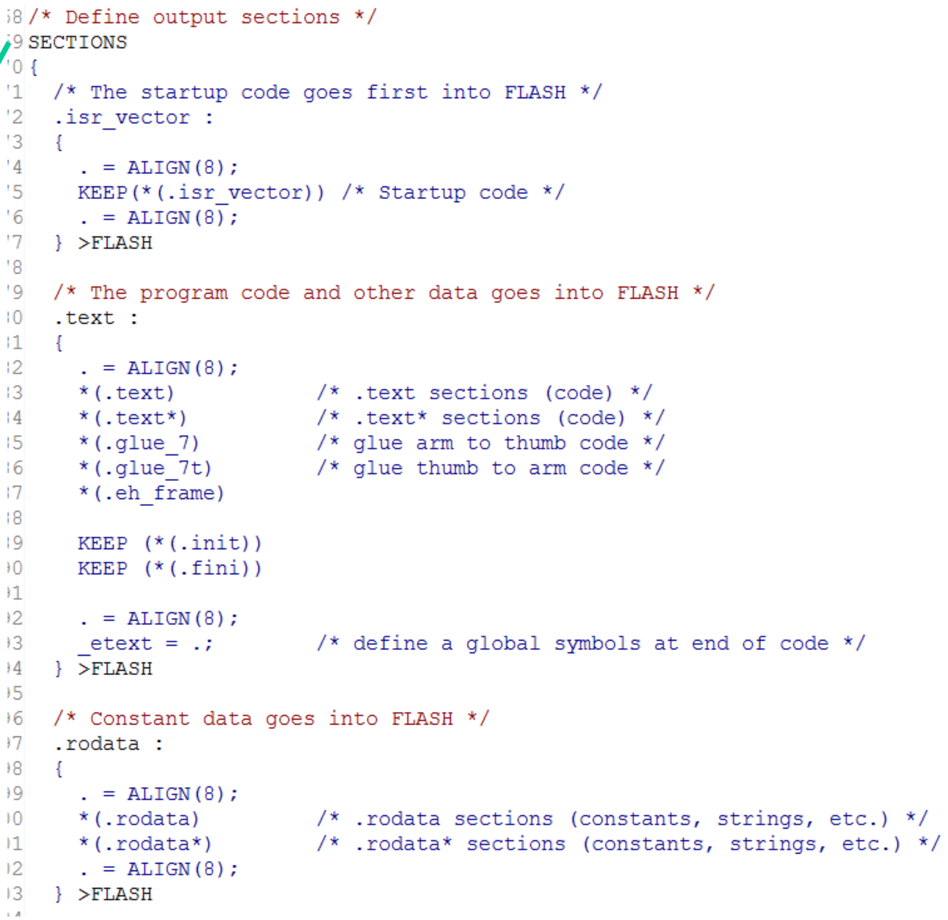
```
/* Highest address of the user mode stack */
_estack = 0x20018000;    /* end of RAM */
/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

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



Important files: the linker script

- Quick recall on memory segments:

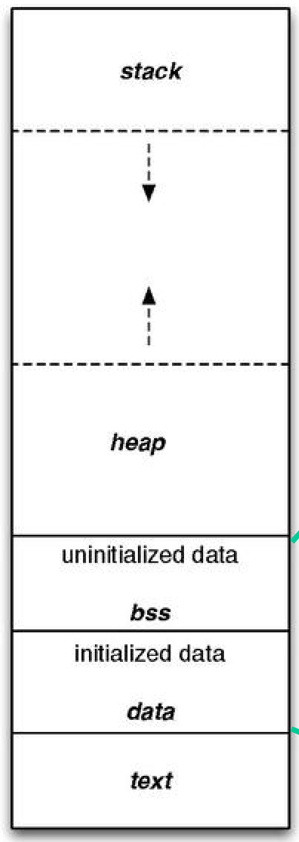




```
/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(8);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(8);
  } >FLASH

  /* 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

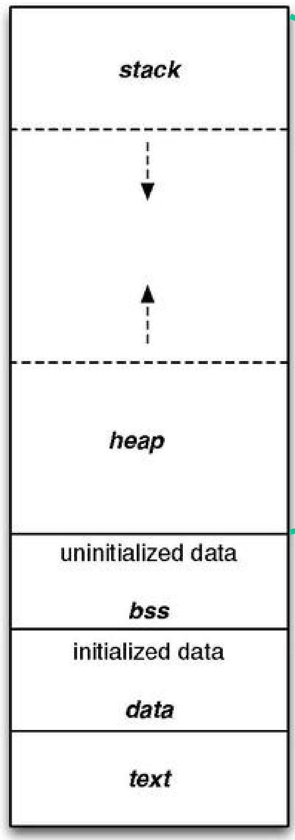
- Quick recall on memory segments:



```
/* 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

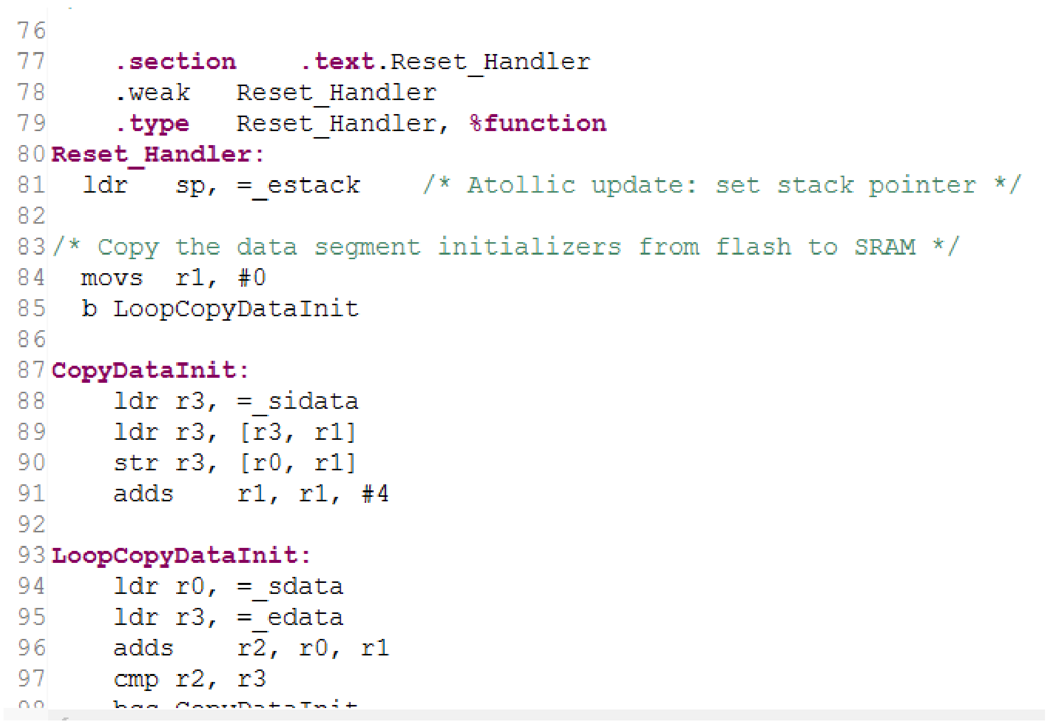
- Quick recall on memory segments:



```
/* 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
```

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

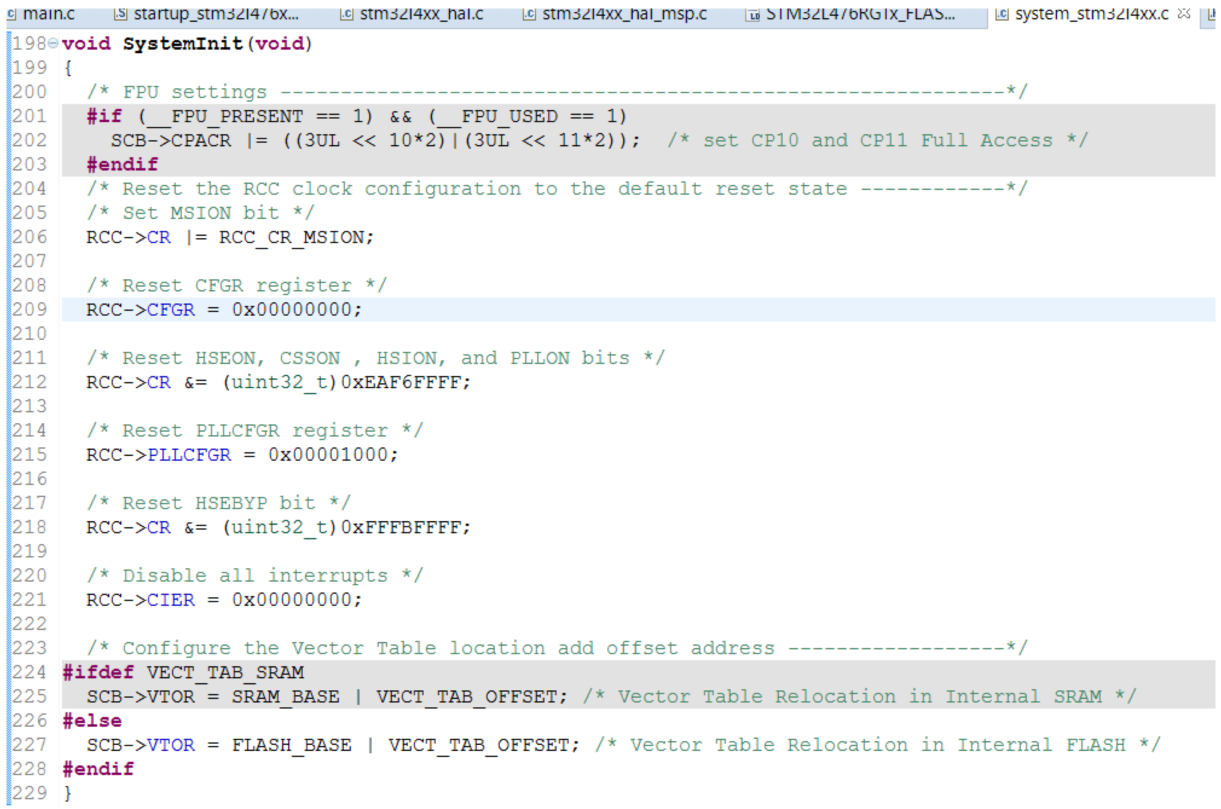


```
    .section    .text.Reset_Handler
    .weak   Reset_Handler
    .type   Reset_Handler, %function
Reset_Handler:
  ldr   sp, =_estack    /* Atollic update: set stack pointer */

/* Copy the data segment initializers from flash to SRAM */
  movs  r1, #0
  b LoopCopyDataInit

CopyDataInit:
    ldr r3, =_sidata
    ldr r3, [r3, r1]
    str r3, [r0, r1]
    adds    r1, r1, #4

LoopCopyDataInit:
    ldr r0, =_sdata
    ldr r3, =_edata
    adds    r2, r0, r1
    cmp r2, r3
```

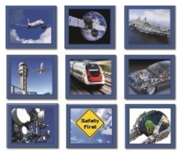
System Workbench 4

Important files: the system file

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



```
void SystemInit(void)
{
  /* FPU settings ------------------------------------------------------------*/
  #if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
    SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full Access */
  #endif
  /* Reset the RCC clock configuration to the default reset state ------------*/
  /* Set MSION bit */
  RCC->CR |= RCC_CR_MSION;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON , HSION, and PLLON bits */
  RCC->CR &= (uint32_t)0xEAF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x00001000;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* Disable all interrupts */
  RCC->CIER = 0x00000000;

  /* Configure the Vector Table location add offset address ------------------*/
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal SRAM */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* Vector Table Relocation in Internal FLASH */
#endif
}
```



Thank You!

Master thesis: ugomaria.colesanti@intecs.it

www.intecs.it