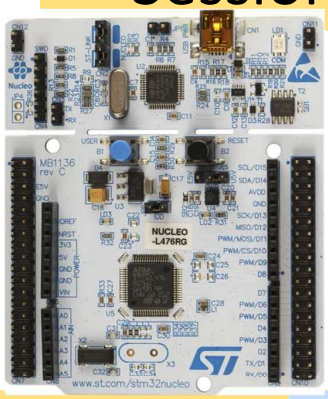
Version STM32L476

Session 1/4

MICROPROCESSORS II Course
Session 1: microcontroller architecture



(and microcontrollers)

2024-v1

ENSEA S7-September 2024

Laurent MONCHAL

C1

First Session Menu

- About « Microprocessors II » course
- History (of microprocessors)
- The Cortex M4 or M7 (microprocessor)
  - RISC Architecture
  - Pipeline
- STM32 (microcontroller)
  - Architecture
  - Communication with peripherals: the « registers »
- Peripherals programming
  - From “Bare metal coding”
  - To HAL functions

Course Themes & Keywords

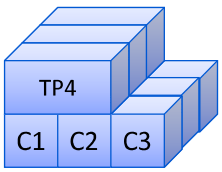
- DITN_2501
- Microprocessors (µP)
  - RISC Architecture
  - Example: Cortex M4, M7
- Microcontrollers (µC)
  - µC=µP + peripherals
  - Example: STM32
  - Peripherals: ADC, DAC, TIMER, UART, I2C...
- Interrupts
  - Principles
  - STM32 Interrupt Control (NVIC)
  - Priorities
- Microprocessor and C language
  - Memory management

The teaching team

- Christian JOSSE
- Lounis KESSAL
- Christian LAROCHE
- Laurent MONCHAL
- Nicolas SIMOND
- Antoine TAUVEL

The "DITN_2501" course

- Microprocessors season II
- Minor or Major Course "Computer Science S7".
  - « Computer Science » Minor with JAVA courses
  - « Computer Science » Major with JAVA courses, networks and system programming
- 4 x 2 hours = **8 hours** of **classes**
  - Essential to know the vocabulary
  - Necessary to discover the principles
  - Explanations about the LAB
- 3 x 2h = **6h hours for exercises (TD)**
  - Interrupts, Timer and ADC.
- 4 x 4h = **16 hours** of LAB **work** (or mini-project)
  - Programming an application on a STM32 development kit



Educational documents and resources

- Paper documents are distributed during the course sessions:
  - **Slides** from each session distributed at ?(acts as a poly)
  - **Text of the Exercises**
  - **Text of the LAB** (to be read imperatively before coming to a LAB session!)
  - **Development Guides: "Developper's Guides"**, distributed in public works.
    - STM32CubeIDE (software tool)
    - Basics (the classics)
    - STM32F4 or F7 or H7A3 Architecture
    - Nucleo (and Discovery) Card
- They could be **indispensable during the final exam**
- The Moodle page of this course **INFORMATIQUE> Microprocesseur II** is: https://moodle.ensea.fr/course/view.php?id=127
  - Quizzes to do after or during each lecture session
  - Links to videos on ubicast: Previous years' exams with elements of corrections
- A page is dedicated to store **the STM32 documentation** (required for the LAB sessions): https://moodle.ensea.fr/course/view.php?id=652
- For the classes: https://ensea.ubicast.tv/channels/#microprocesseurs_40546

The (other) sources

- STMicroelectronics website: www.st.com
  - For the technical documentation "Reference Manual", "Data Sheet" and "Application Notes".
- Website of Tilen Majerle: https://stm32f4-discovery.net
  - Lots of practical information to implement an STM32
- **The definitive guide to ARM Cortex-M3 and Cortex-M4 processors** by Joseph YIU (815 pages) – Newnes (ISBN13=978-0124080829)
- **The Designer's guide to the Cortex-M Processor family** by Trevor Martin – Newnes (ISBN13=978-0080982960) (newer version: 978-0081006290)

The next 3 books deal with embedded systems built around Cortex M3 and M4 based microcontrollers (but do not use STM32):

- **Embedded Systems: Introduction to ARM Cortex-M Microcontrollers** by Jonathan VALVANO – (ISBN=1477508996)
- **Embedded Systems: Real-Time Interfacing to ARM Cortex-M Microcontrollers** by Jonathan VALVANO – (ASIN: B00CQOJM84)
- **Embedded Systems: Real-Time Operating Systems for ARM Cortex-M** by Jonathan VALVANO – (1466468866)

For the pleasure of history and story:

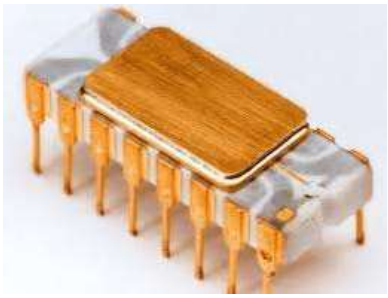
- **"The Innovators**" by Walter Isaacson - Simon and Schuster- (ISBN13=978-1471138805)

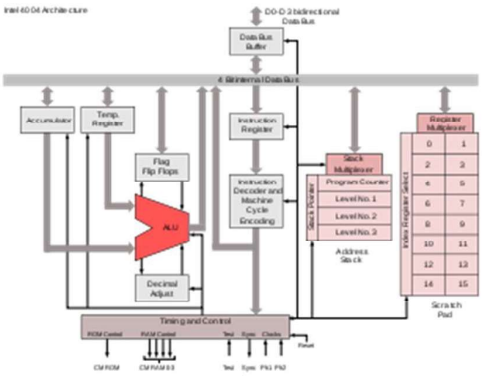
Great (historical) moments

- The precursors
  - Babbage and his analytical machine (1847)
  - The first programmer was a woman: Ada Lovelace.
- The first computer
  - ENIAC (1945)
- The inventions that changes the 20th century
  - The transistor was invented in 1947 by John Bardeen, Walter Brattain and William Schockley.
  - Integrated circuits by Kilby and Noyce
- Read: " The Innovators " by Walter Isaacson

In the seventies

- INTEL responds to a control of electronic circuits by placing the essential logic functions that make up a computer on a single integrated circuit: INTEL's 4004.
- It's the first microprocessor

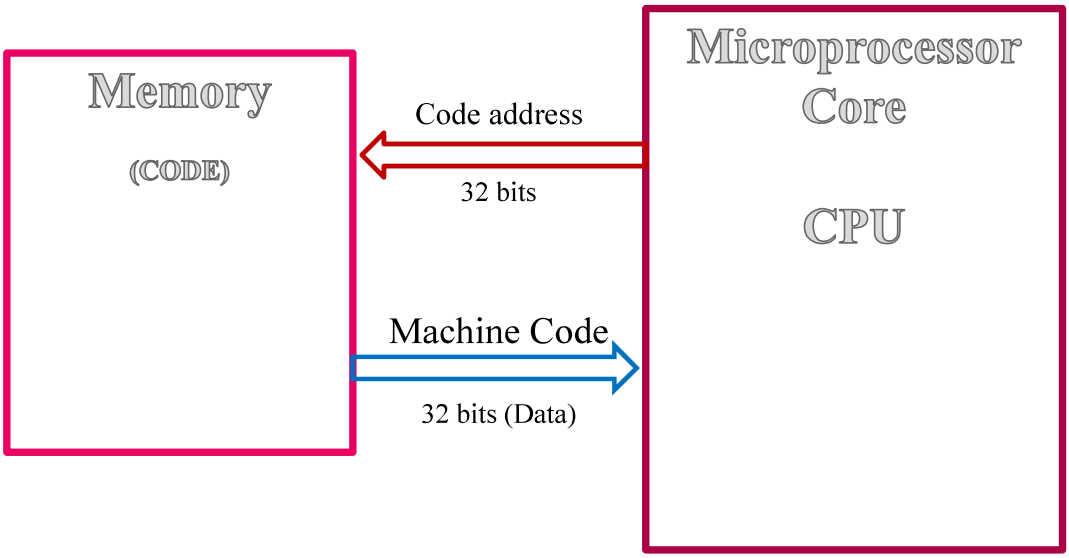




In the following years

- More and more transistors have been placed on the same integrated circuit surface (Moore's Law).
- The microprocessors were mass-produced at relatively low cost.
- They invaded our lives
- For the best and the worst!

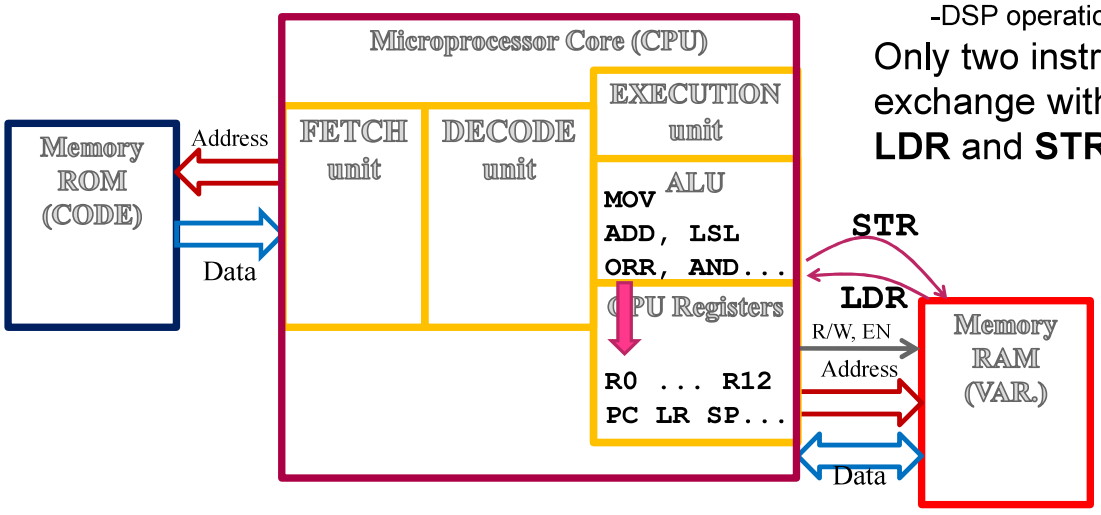
Principle of a microprocessor



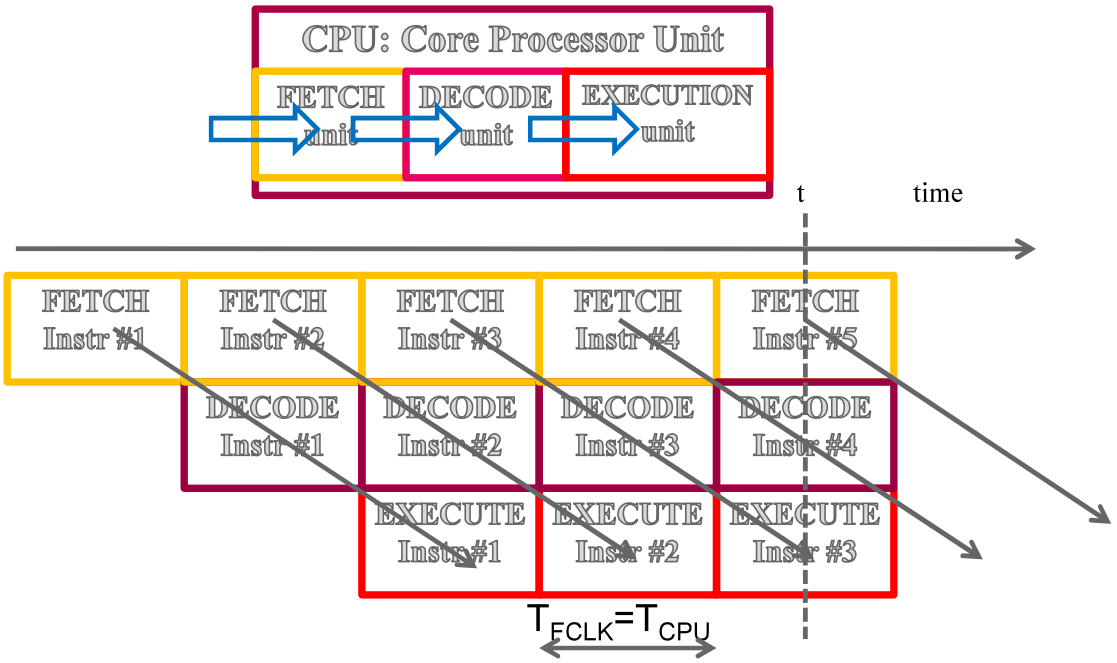
Cortex M Architecture

Execution Unit ALU:
- -Integer operations (+-*/) on R0, R1...
- -Floating **point** operation (+-*/)
- -DSP operations (MAC)

Only two instructions to exchange with memory: **LDR** and **STR**

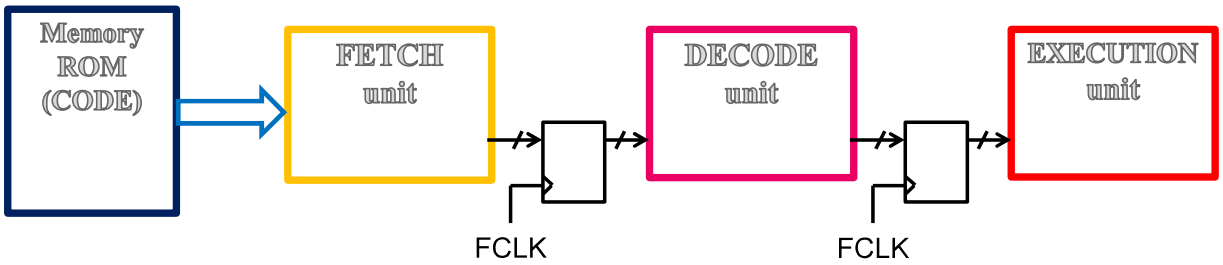


Pipeline in Cortex-M RISC architecture



Pipeline Details

- A pipeline consists of purely combinatorial blocks between which D flip-flops are inserted to synchronize them:

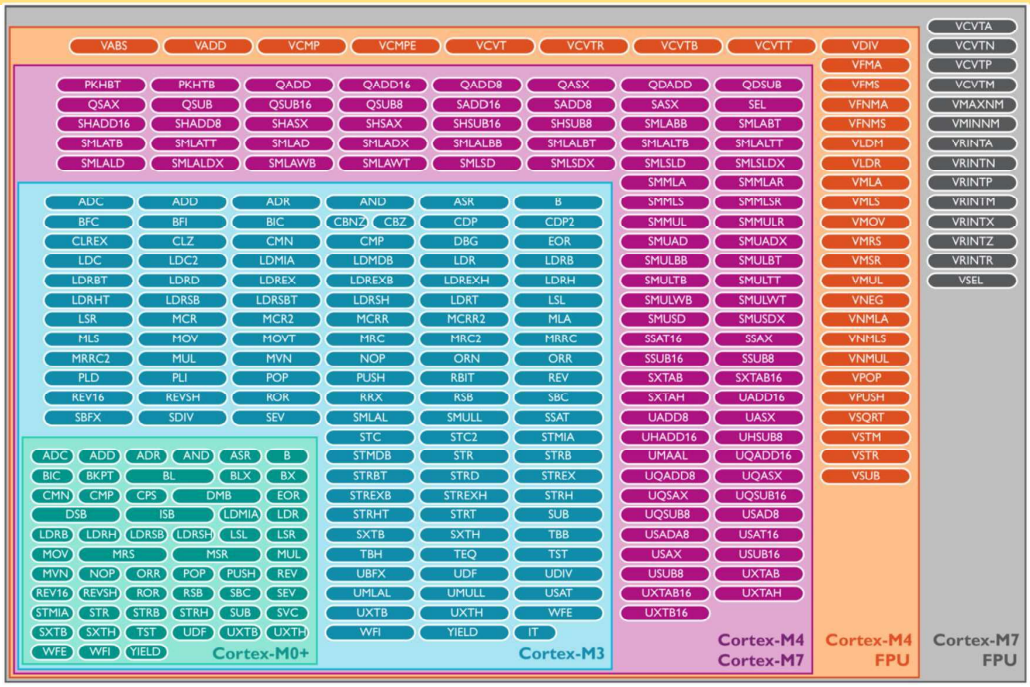


- HCLK is the synchronization clock of the pipeline for the Cortex.
- It takes 3 FCLK cycles T_{SYSCLK} to process (FETCH, DECODE, EXECUTE) the same instruction (latency).
- At each end of cycle, a new instruction is executed (flow rate).
- FCLK or HCLK

Working frequency of a microprocessor

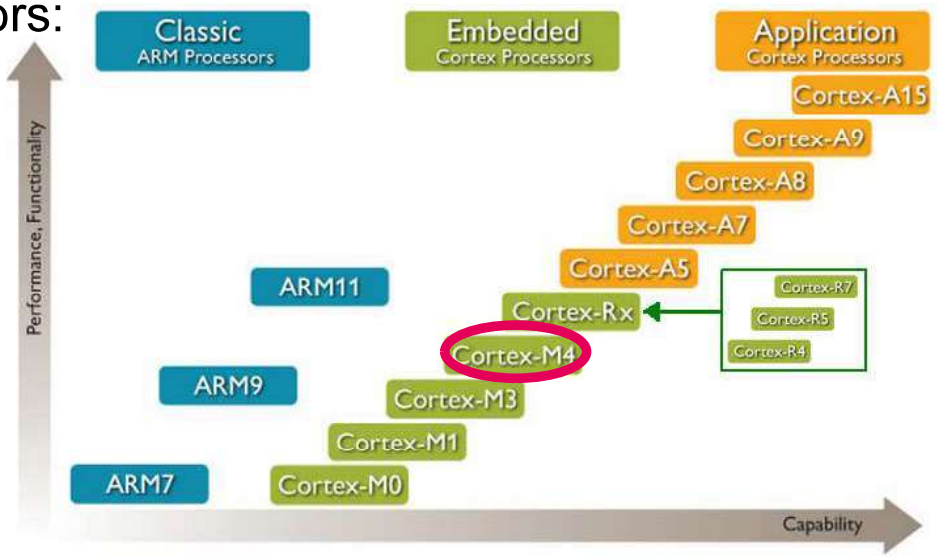
- The pipeline is synchronized by a clock: HCLK
  - Period: $T_{HCLK}=1/F_{HCLK}$
- F_{HCLK} is the « frequency » of the microprocessor
- How long does it take to execute an instruction?
  - Frequency : 200 MHz (for example)
  - A period: 1/200E6 = 5.0E-9 = 5 ns =T_{HCLK}

The ARM-Cortex M4/7 Instruction Set



ARM processors

- The Cortex M4/M7 is one of a large family of ARM processors:



Cortex-M7, M4 & L4

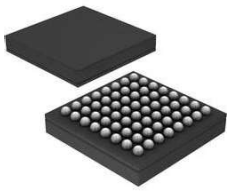
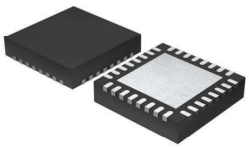
- Same "Cortex M" core (previously seen)
  - Faster for M7 (and more instructions)
  - Cortex M4 CPU in STM32F4xx and STM32L4xx families
  - Cortex M7 CPU in STM32H7Ax family
  - RISC processor architecture
  - "Harvard" architecture: data bus different from instruction bus
  - 2 instruction sets:
    - ARM: 32-bit instructions
    - Thumb2: 16-bit instructions
    - ALU for arithmetic with integers (not floating numbers)
  - A FPU => Floating Point Unit Instructions
  - A DSP => Digital Signal Processing Instructions

Advantages of the Cortex M4 and M7

- Low cost
- Excellent performance/consumption ratio
- Dedicated to embedded applications
- DSP and floating computing power
- STM32L4xx: ultra low power

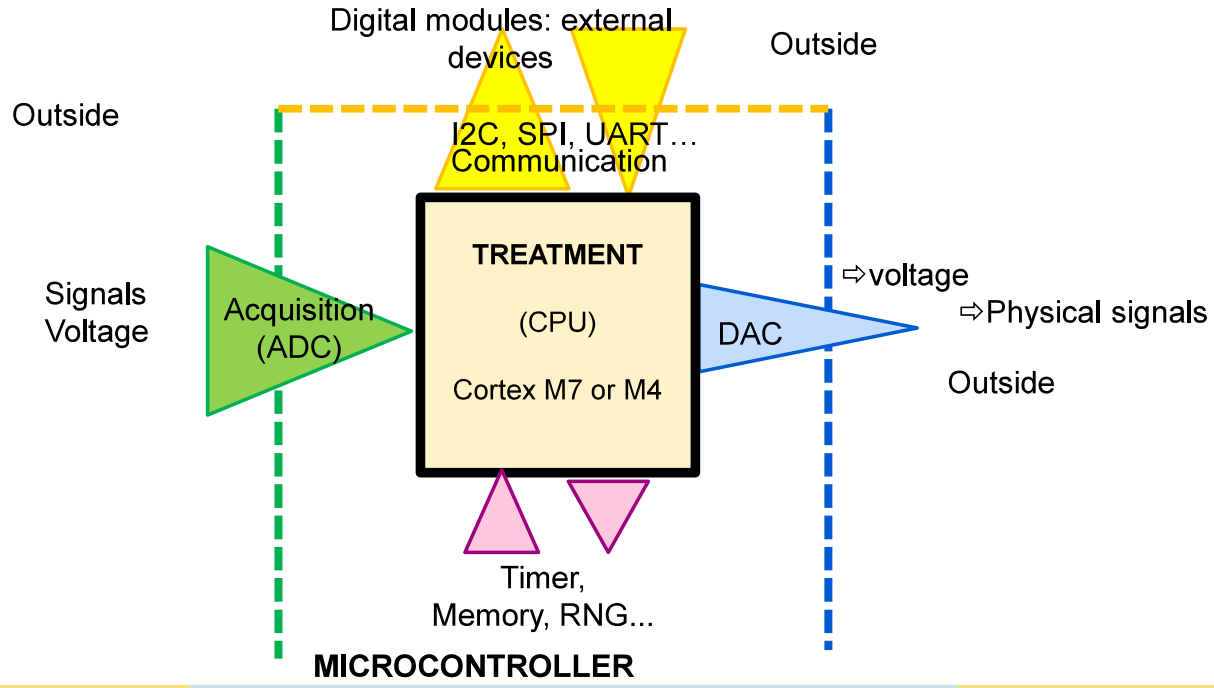
Characteristics of a µC

- System on Chip (SoC)

- Logic circuit for ***interacting with the environment***
- Equipped with a microprocessor

Data flow in a microcontroller (µC)



Functionnalities of a µC

- Acquisition of external physical signals
  - Analog-to-digital converter (ADC)
- Sending analog voltages to the outside world
  - Digital to Analog Converter (DAC)
- Digital communications with the outside world
  - Digital input/output ports (GPIO)
  - Data exchange modules: LCD driver (LTDC)
  - Bus modules: USART, SPI, I2C, USB...
- Processing of acquired data
  - CPU and/or DSP
- Internal Features
  - Counting time: counters called "Timers".
  - Memories (RAM, ROM).
  - Random Number Generator

The microcontroller chosen for the labs

- We chose the **STM32L4xx** microcontroller from STMicroelectronics (ST).
- The architecture of the **CPU** core: the **Cortex M4,** was designed by **ARM**
- All the rest (**peripherals**) of the STM32 is designed by **ST Microelectronics** (assisted by ARM)
  - 16 advanced Research and Development centers,
  - 1.1 billion investment in 2017
  - July 2022: 5.7 billion euros investment to produce better semiconductors
- Other companies use the ARM Cortex M7 or M4 core to build their own microcontrollers: NXP (LPC), Texas instrument (Stellaris), Fujitsu (FM3), etc.

Global view of a microcontroller (STM32L4xx)

Microcontroller (STM32 manufactured by STMicroelectronics)

MCU (Microprocessor Core Unit)
(designed by ARM Holdings)

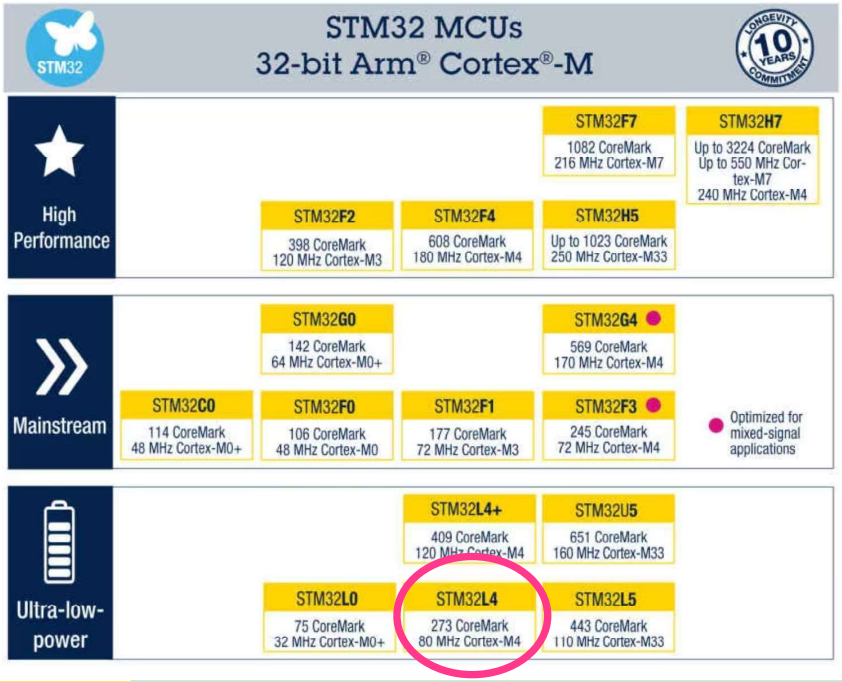
CORTEX M4

Peripherals
(designed and manufactured by STMicroelectronics in collaboration with ARM)
TIMERS, ADC, DAC, SPI Bus...
ROM (Flash)
RAM

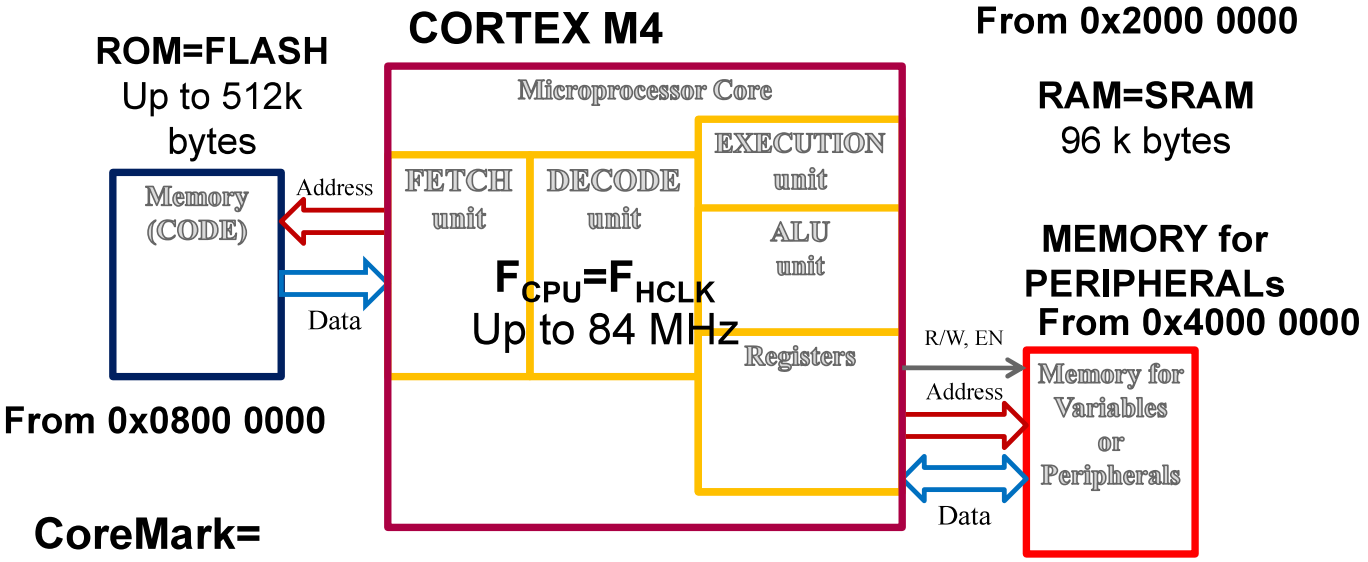
The arguments of STM32L476

- ARM Cortex-M4 Core inside
- Ultra Low Power
- Excellent behavior for real-time applications
- Very high performance in relation to power consumption
- Numerous peripherals
  - IOs, Timer, ADC, DAC, USB, UART, SPI, I2C
- Great integration
- Low cost
- Fast development
  - Numerous development tools: µVision, MBED, STM32CubeIDE,…

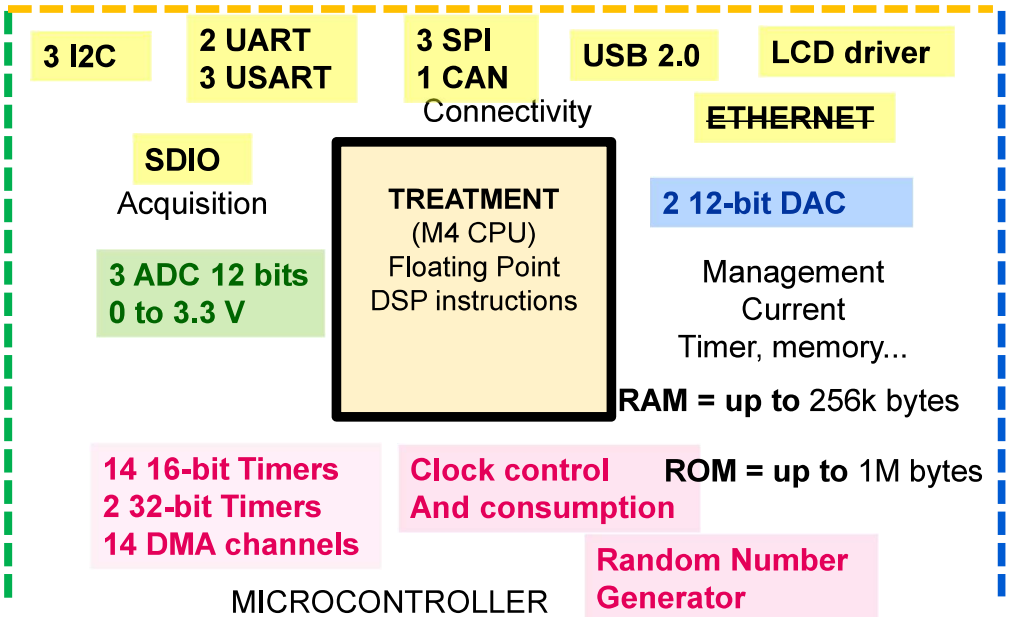
The big STM32 family



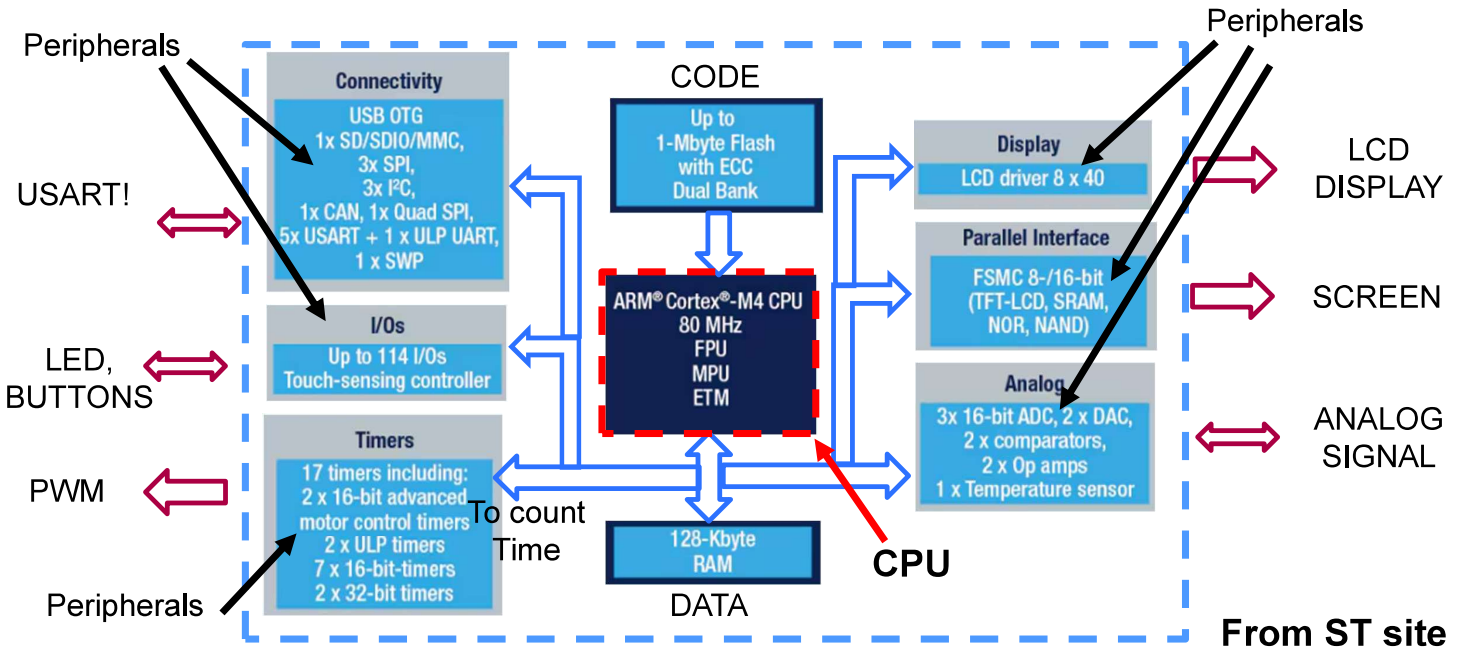
STM32F401 features: memory and clock



Peripherals of STM32L476

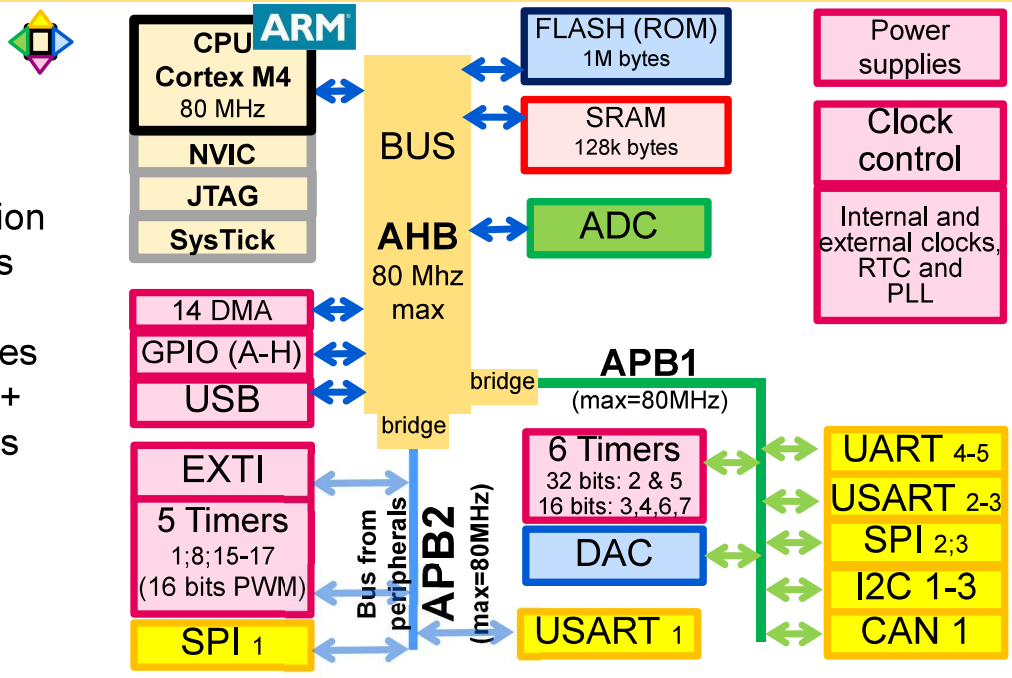


Peripherals of STM32L476

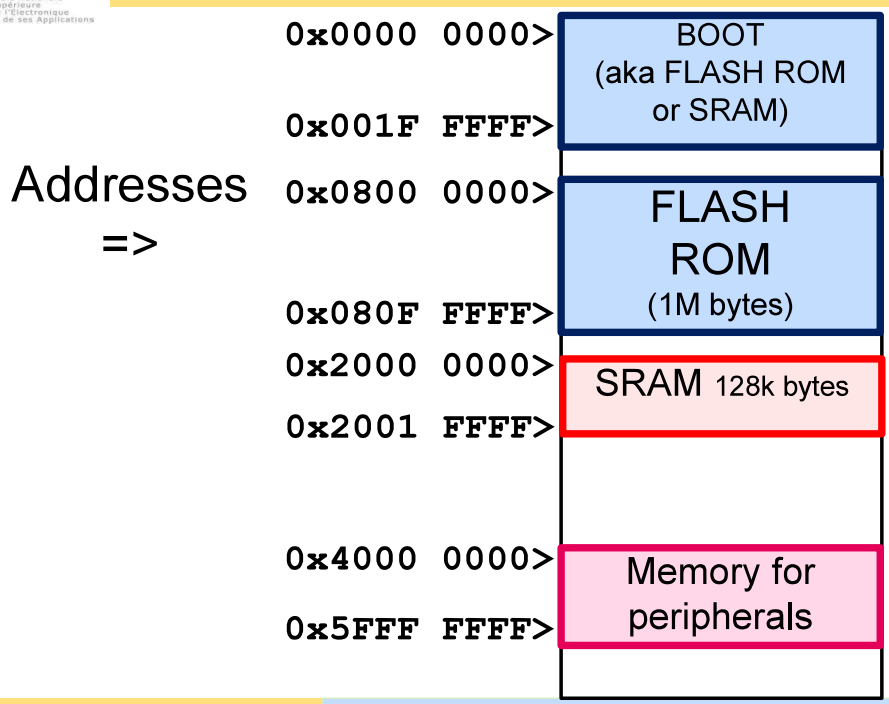


Detailed architecture of a STM32L476

BUS= Association of wires Here: Addresses + Data + Controls (RW)



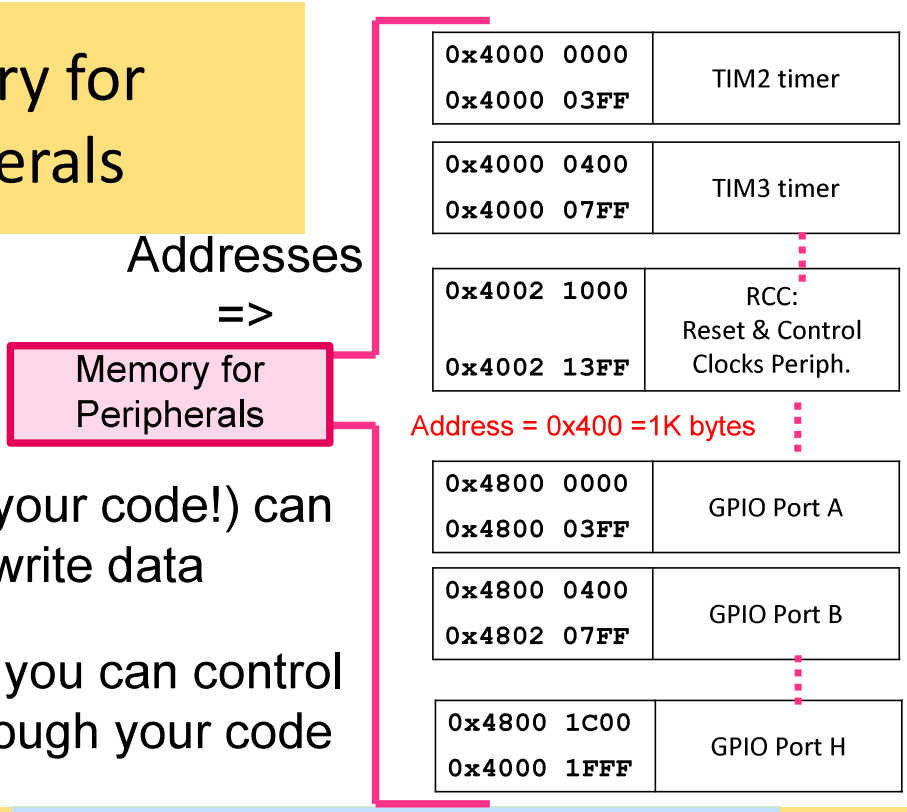
STM32L476 Memory Map



=> Data associated with a peripheral

Memory for peripherals

From Datasheet STM32L476 DS10198



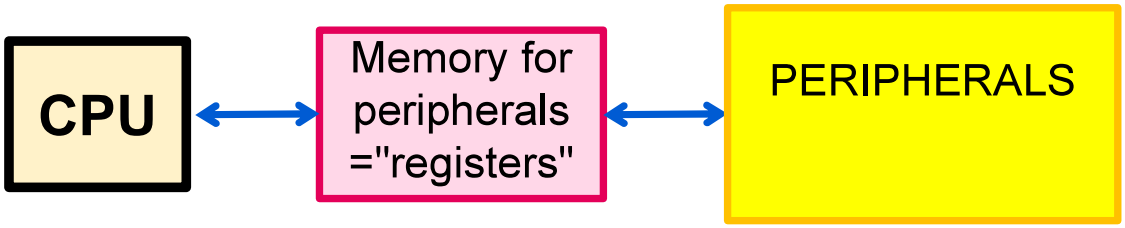
=> Data associated with a peripheral

The CPU (so your code!) can read or write data

In other words, you can control peripherals through your code

Peripheral control by CPU

- Each peripheral is controlled by bits that are in the "peripheral" memory area.
  - The operation of a peripheral is achieved by **writing** the memory associated with the peripheral.
  - The status of a peripheral is obtained by **reading the** memory associated with the peripheral.
  - Peripheral memory is the **link between the** CPU and the peripherals.
  - The operation of each peripheral and its links to the peripheral memory are **described in the documentation.**

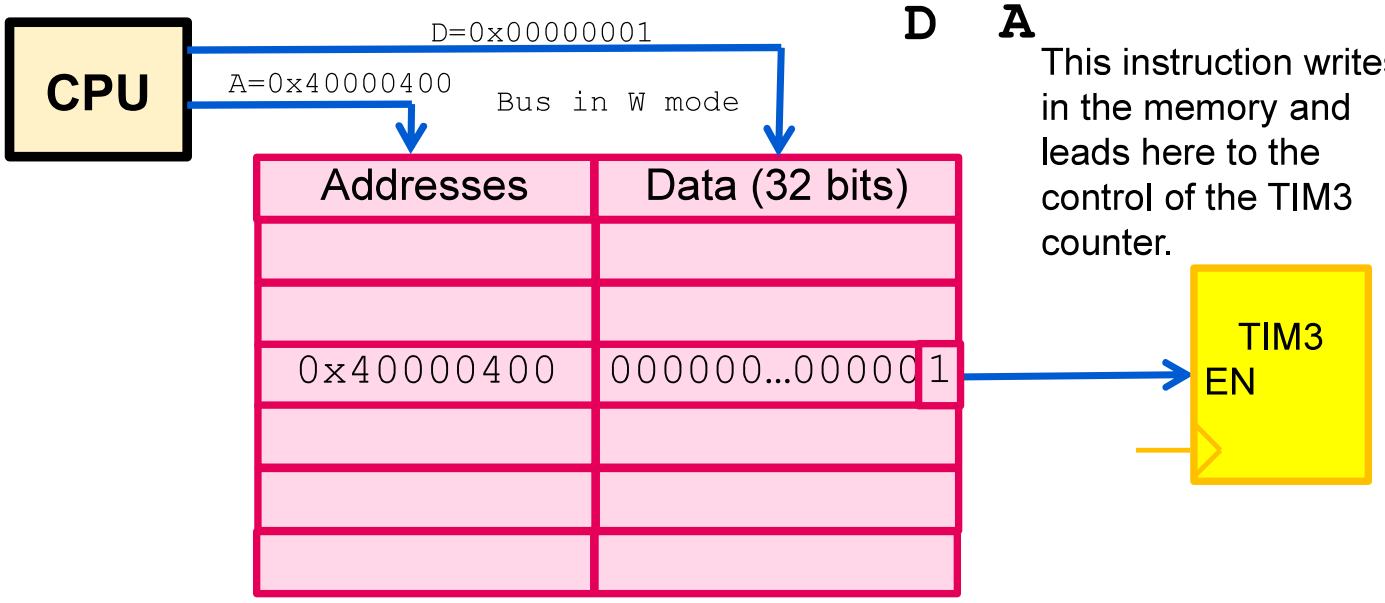


Registers

- Every Peripheral:
  - Has a 1024 byte memory
  - So 1024/4=256 32-bit words
- Each 32-bit word of peripheral memory
  - The peripheral is called the "**register**" **of** the peripheral
- Each peripheral
  - Can therefore have up to 256 peripheral "registers".
  - Does not necessarily use all 256 registers

Illustration of CPU peripheral control

At low level, the CPU writes data at an address: **STR Rn,[Rm]**...



This instruction writes in the memory and leads here to the control of the TIM3 counter.

The (big) secret

- **Getting a microcontroller to work the way you want means writing the right "0" and the right "1" at the right places.**
- Program memory:
  - It contains your code to be executed
- Peripherals memory:
  - It contains the bits that describe the behavior of the various peripherals.
  - These bits are often written during initialization.
  - These bits are used by the code you write to communicate with the peripherals.
  - ST has assigned to each peripheral a block marked by its start address.
- To control a peripheral, you have to **write in the memory** associated with it in the **32-bit word** called "**register**" (by abuse of language!).
- The STM32L47xxx … Reference Manual (RM0351) describes the **links** between the **behavior of each peripheral** and the **contents of its registers.**
- To control a peripheral, it is necessary to know its address in memory.

Warning on the « REGISTER » word

- **Do not confuse:**
  - registers (of peripherals)
  - with registers (of CPU): R0, R1... LR, PC, SP...
- The "registers" of the Peripherals are:
  - 4 Byte Words (32 bits) data
    - Not all 32 bits are necessarily used
    - In some registers (of 32 bits) only one bit is used.
  - At Addresses in memory
    - Located in memory from the address 0x40000000
    - Each Peripheral has 1024 bytes or 256 registers.
    - They are addressable from 4 to 4 starting from a base address.
    - A Peripheral typically uses 10 to 20 registers.

Identification of a « register »

- A (peripheral) register is associated with
  - A peripheral (you would have bet on it!)
  - And to one or more functionalities or uses
- Its name is typically of the form: PERIPH_function
- It is the documentation that sets the names associated with the (peripheral) registers.
- Example:
  - The status register of ADC1 is called:
  - ADC1_SR (SR= State Register)

Address of a "register"

- **A register is associated with**
  - **A PERIPHERAL**
  - **And a FUNCTIONALITY (or many) of the peripheral**
- Many registers for one peripheral are necessary because:
  - One register contains 32 bits
  - And hundreds of bits are necessary to control and set functionalities
- **The address of a « register »** is computed by **adding**
  - The address of the beginning of the memory associated with the PERIPHERAL: base address or **peripheral address**
    - You find it in "3.3 Memory Map" of "reference manual" (and in next slides).
  - The **Offset** (Address Offset)
    - Distance of the register from the beginning of the memory
  - **REGISTER_ADDRESS =**
    PERIPHERAL_ADDRESS
    + ADDRESS_OFFSET

PERIPHERAL Addresses for STM32L476 (1)

- The memory for peripherals starts at 0x40000000
- Examples:
  - 0x 4000 0000: TIM2 Timer
  - 0x 4000 0400: TIM3 Timer
  - 0x 4000 0800: TIM4 Timer
  - 0x 4000 0C00: TIM5 Timer
  - 0x 4000 1000: TIM6 Timer
  - 0x 4000 1400: TIM7 Timer
  - 0x 4000 3800: SPI2
  - 0x 4000 3C00: SPI3
  - 0x 4000 4400: USART2
  - 0x 4000 4800: USART3
  - 0x 4000 4C00: UART4
  - 0x 4000 5000: UART5

From the Datasheet.

For more details, download the datasheet there:
https://www.st.com/resource/en/datasheet/stm32l476rg.pdf

Peripheral addresses (2)

- 0x 4000 5400: I2C1
- 0x 4000 5800: I2C2
- 0x 4000 5C00: I2C3
- 0x 4000 7000: PWR
- 0x 4000 7400: DAC1
- 0x 4001 0000: SYSCFG
- 0x 4001 2C00: EXTI
- 0x 4001 2C00: TIM1
- 0x 4001 3000: SPI1
- 0x 4001 3400: TIM8
- 0x 4001 3800: USART1
- 0x 4001 4000: TIM15
- 0x 4001 4400: TIM16
- 0x 4001 4800: TIM17
- 0x 4002 0000: DMA1
- 0x 4002 0400: DMA2
- 0x 4002 1000: RCC
- 0x 4800 0000: GPIOA
- 0x 4800 0400: GPIOB
- 0x 4800 0800: GPIOC
- 0x 4800 0C00: GPIOD
- 0x 4800 1000: GPIOE
- 0x 4800 1400: GPIOF
- 0x 4800 1800: GPIOG
- 0x 4800 1C00: GPIOH
- 0x 5000 0000: USB
- 0x 5004 0000: ADC (ADC1 first)
- 0x 5006 0800: RNG
- 0x 5004 1000: ADC

The register to ENABLE ADC1

- My goal: to enable the ADC1
- I read the Reference Manual

The functionality is "Control Register 2" = CR2

15.13.3 ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

This is bit 31 (useless here)

The peripheral is an **ADC**
Here it is #1: ADC1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	SWSTART	EXTEN		EXTSEL[3:0]				Res.	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ALIGN	EOCS	DDS	DMA	Res.	Res.	Res.	Res.	Res.	Res.	CONT	ADON
				rw	rw	rw	rw							rw	rw

Bit 0 is here

Bit 0 **ADON:** A/D Converter ON / OFF
This bit is set and cleared by software.
Note: 0: Disable ADC conversion and go to power down mode
1: Enable ADC

Description of an action carried out by writing in bit 0 of ADC1_CR2

Calculation of the address of ADC1_CR2

- Documentation refers to the ADC CR2 Register
- We want to calculate the address where control register 2 (CR2) of ADC1 is located.
  - Start address of ADC (given previously): `0x50040000`
  - The Reference Manual documentation provides the following information for ADC_CR2: "Address offset: `0x08`".
  - We therefore conclude that register ADC1_CR2 is located at the address: `0x50040000 + 0x8 =` **`0x50040008`**

Writing in a register with a pointer (bare metal programming version)

- Pointer or « Bare Metal programming » version
  - Declaration of a pointer to an integer (32 bits)
  - Initializing the pointer with the address
  - Writing or reading using the previously defined pointer
- Example
  - `int * ADC1_CR2;`
  - `ADC1_CR2 =(int*) 0x50040008;`
  - `*ADC1_CR2 = (*ADC1_CR2) |1 ;`
- Advantage
  - Compact Code
- Disadvantages
  - Need to retrieve addresses from the documentation
  - Laborious and therefore frequent risk of errors

Writing in a register using provided data structures

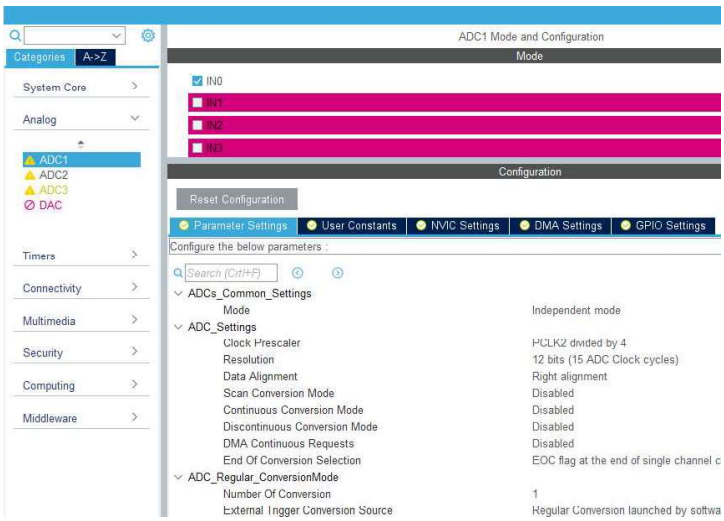
- Version " data structure " " already in library".
  - Inclusion of the "stm32f4xx.h" library (see when initializing the project)
  - Write using already defined data structures
- Example
  - `ADC1->CR2|=1;`
- Advantage
  - Easier to use (no?)
  - The address of the register cannot be mistaken: it is defined through the structure
- Disadvantages
  - Maybe a little less compact.

Use of a HAL function

- Goal: to enable ADC1
- Inclusion of the ADC library ("stm32f746hal_adc.h")
  - Read: UM1905 « Description of STM32F7 HAL and Low-layer drivers"
- Function:`HAL_ADC_Start(ADC_HandleTypeDef*)`
- The code
  - `ADC_HandleTypeDef hadc1;`
  - `hadc1.Instance = ADC1;`
  - `HAL_ADC_Start(&hadc1);`
- Advantage
  - More reliable
  - More effective for fast development
- Disadvantages
  - Much less compact
  - Slower execution time
  - Not easy to fill in the hadc data structure the first time
  - Sometimes not easy to find the right function

Use of STM32CubeIDE (bonus)

- Goal: Enable ADC1
- High level" version
  - Define configuration in ioc view of CubeSTM32IDE
  - `HAL_ADC_Start(ADC_HandleTypeDef* hadc)`
- Advantage
  - Easy use
  - More effective for fast development
- Disadvantages
  - Much less compact
  - Slower execution time
  - Sometimes not easy to find the right function (not so easy)



One basic knowledge of session 1

- **How to locate a PERIPHERAL register in memory**
- Exercise: give the addresses of the following registers
  - DAC1_CR

    DAC control register (DAC_CR)

    Address offset: 0x00

    Reset value: 0x0000 0000
  - GPIOB_IDR

    GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

    Address offset: 0x10

    Reset value: 0x0000 XXXX (where X means undefined)
  - RCC_APB1ENR

    APB1 peripheral clock enable register 1 (RCC_APB1ENR1)

    Address: 0x58

    Reset value:
    0x0000 0000 (for STM32L47x/L48x devices)
  - THE_END

For the session Next!