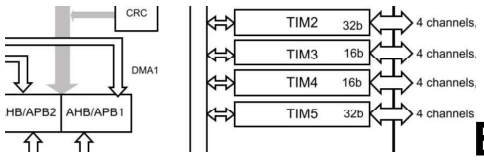
Session 2/4

Version STM32L476

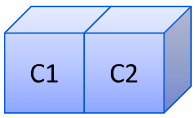
Microprocessors II course
Session 2: the **peripherals** of a microcontroller

(and microcontrollers)

2024-v1



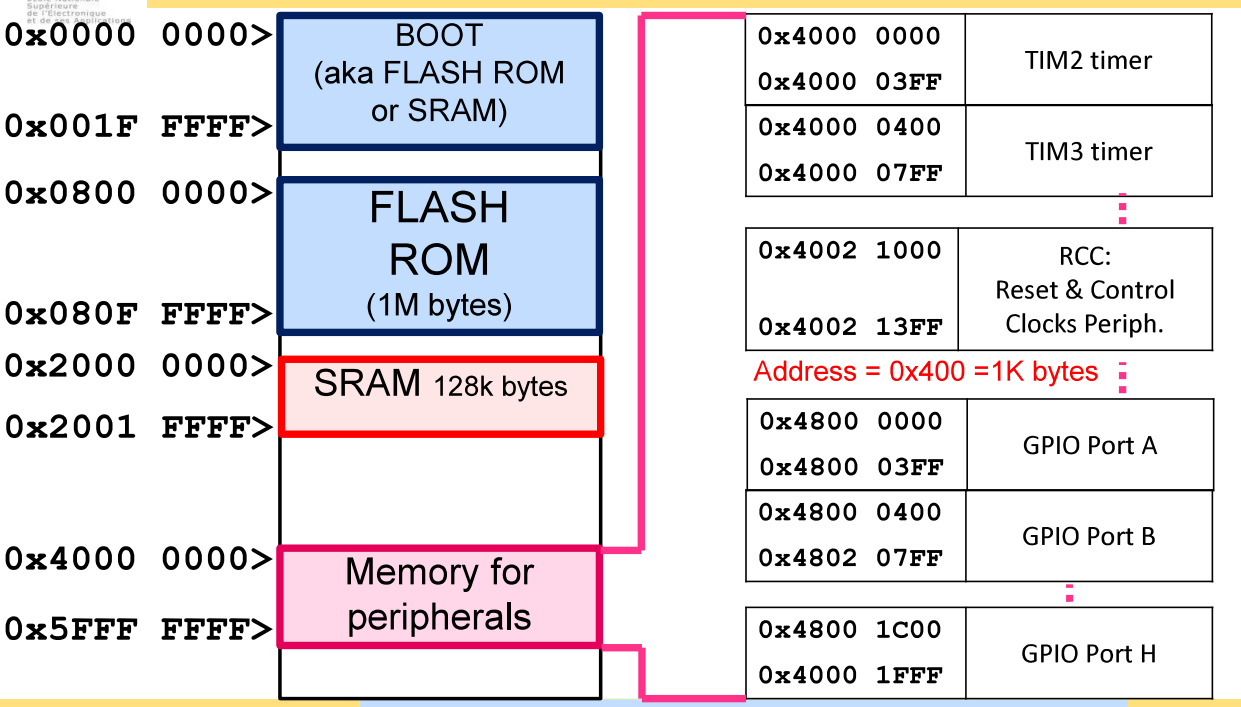
ENSEA S7-September 2024
Laurent MONCHAL
monchal@ensea.fr



Today's session:

- Previously seen: architecture of a STM32L4xx
- STM32 clocks of a STM32L4xx
  - Architecture: **clock tree**
  - Reset and Clock Control (RCC): **Enabling Peripherals**
  - **Clock Setting**
- Main STM32 **PERIPHERALS**
  - **GPIO**s to configure IOs (seen last year)
  - **Timers** to count time, interrupt the CPU periodically and generate PWM
  - The **DAC** to convert Digital data into analog signal
  - The **ADC** to convert analog signal into Digital data
  - The **DMA** to exchange data between peripherals

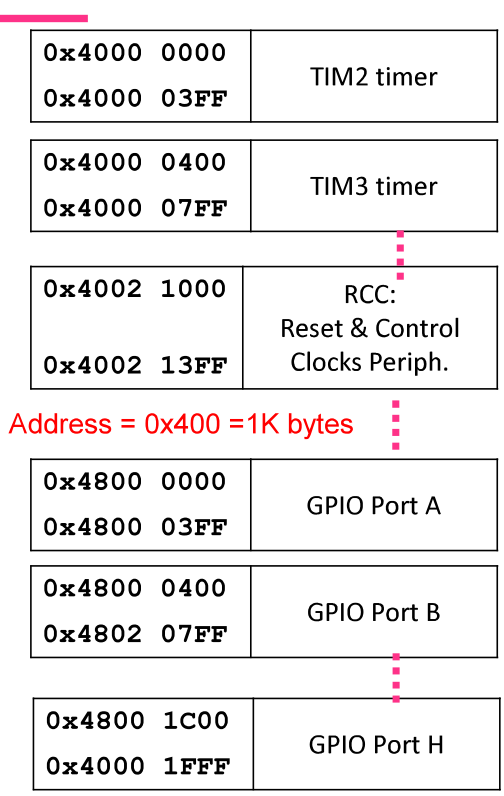
STM32L476 Memory Map



Memory for peripherals

From Datasheet
STM32L476
DS10198

Addresses
=>
Memory for
Peripherals

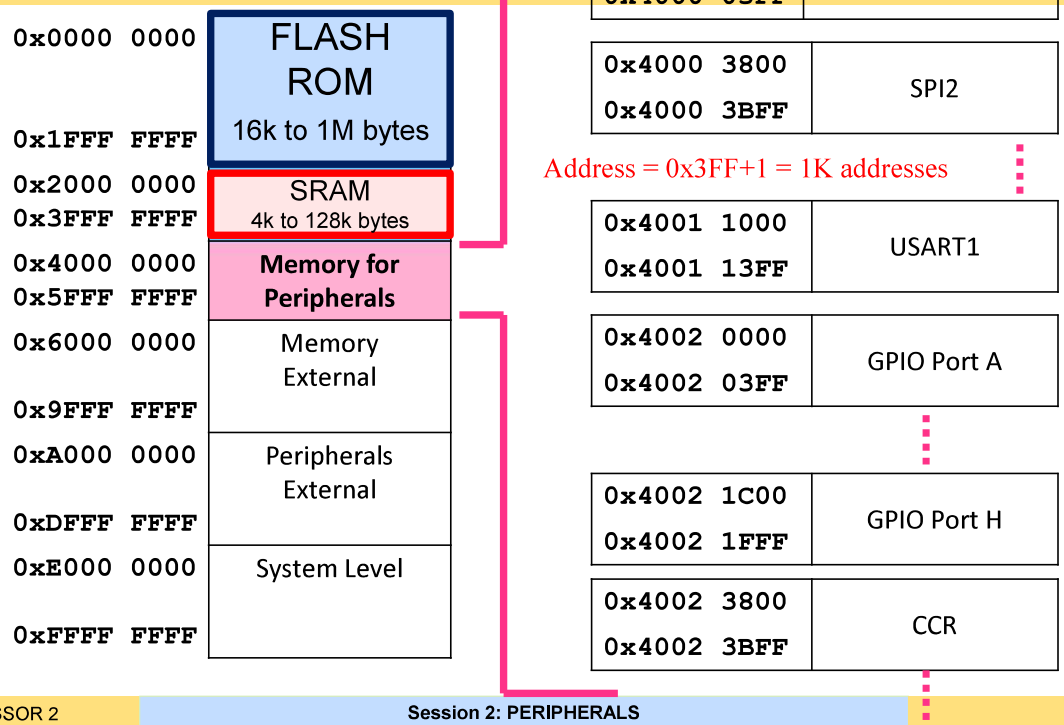


=> Data associated with a peripheral

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

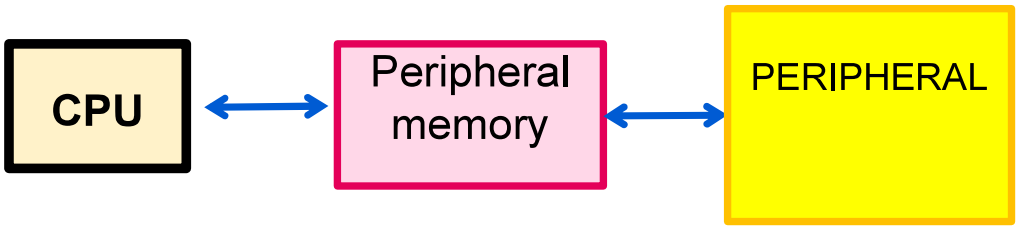
In other words, you can control peripherals through your code

Memories (reminder)



Control of internal peripherals

- An internal peripheral of the STM32
  - Is a digital circuit characterized by a functionality (counter, UART...)
  - Synchronized by a clock
- Each peripheral is controlled by bits from the "peripherals" memory area associated with it.
  - The operation of a peripheral is achieved by **writing to and reading from the** device's memory.
  - The operation of each peripheral is **described in the documentation** (Reference Manual).

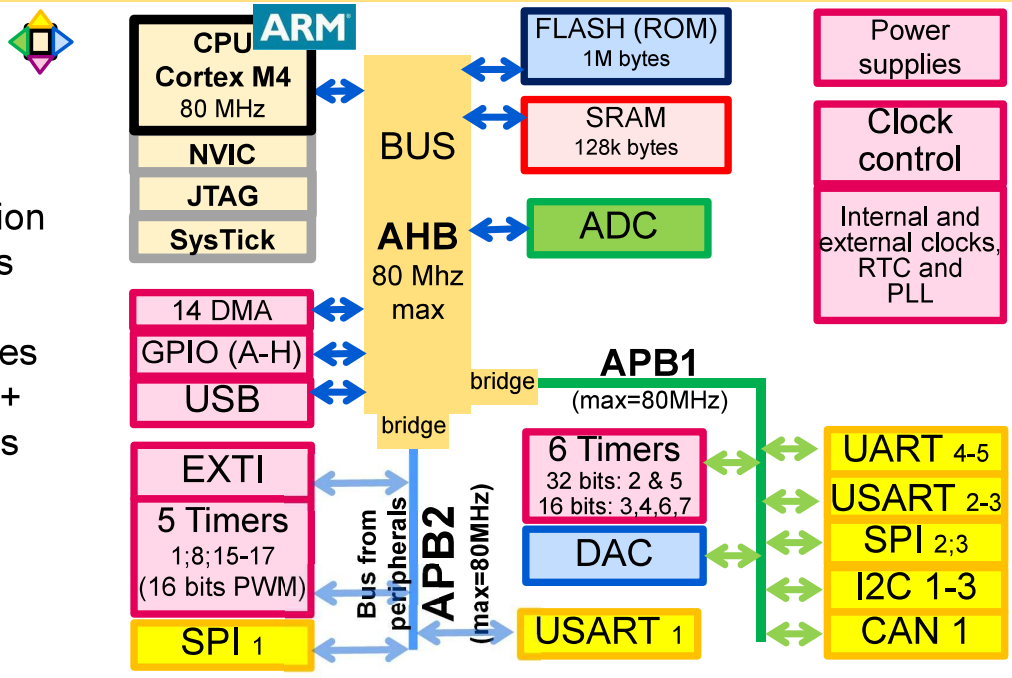


Correction of the little exercise from session 1

- Give the addresses of the following registers (STM32L476):
  - DAC_CR "Address Offset= 0x00".
    - 0x40007400
  - GPIOB_IDR "Address Offset= 0x10".
    - 0x48000410
  - RCC_APB1ENR "Address Offset= 0x40".
    - 0x40021040

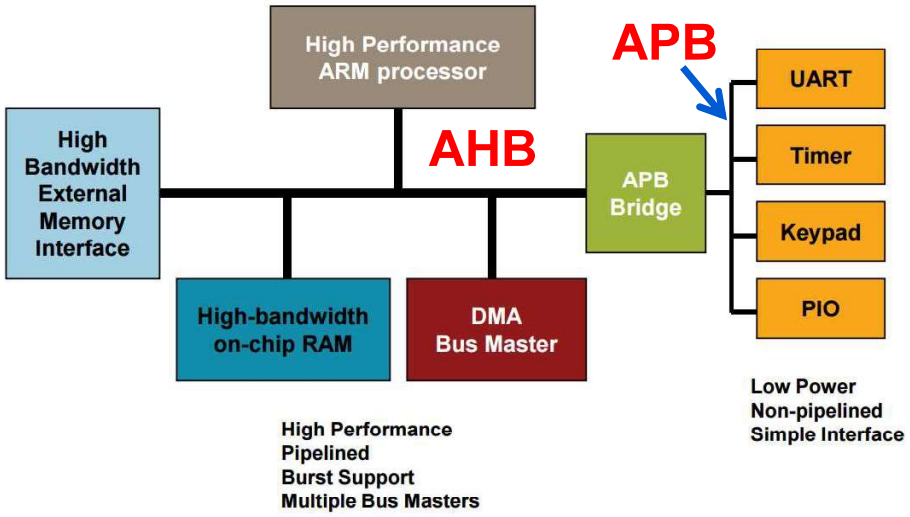
Detailed architecture of a STM32L476

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



The main clocks

An Example AMBA System

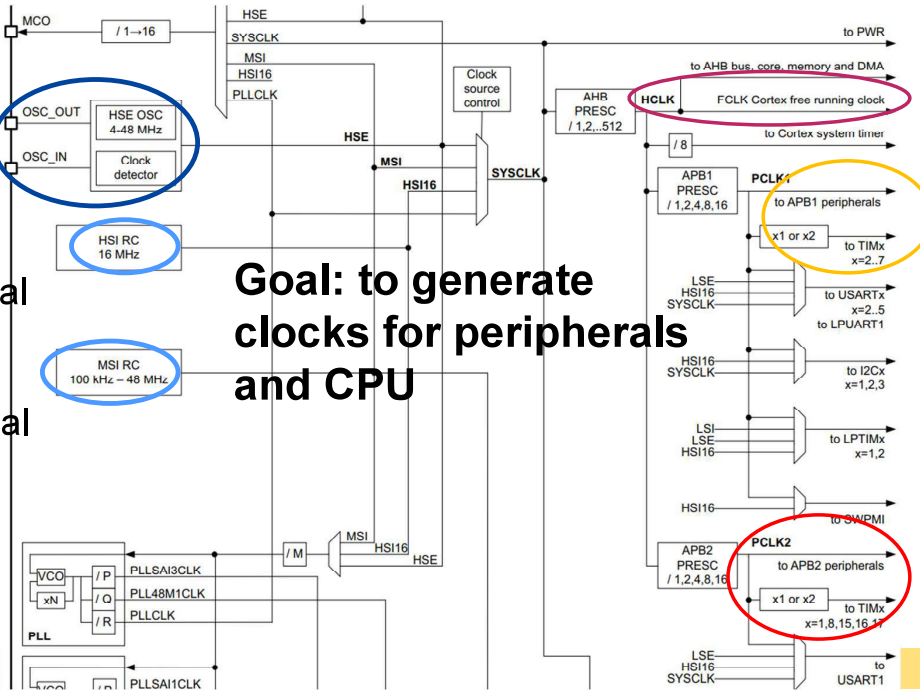


STM32L476 Clock Tree

HSE = High Speed External

HSI = High Speed Internal 16 MHz

MSI = Multi Speed Internal 16 MHz

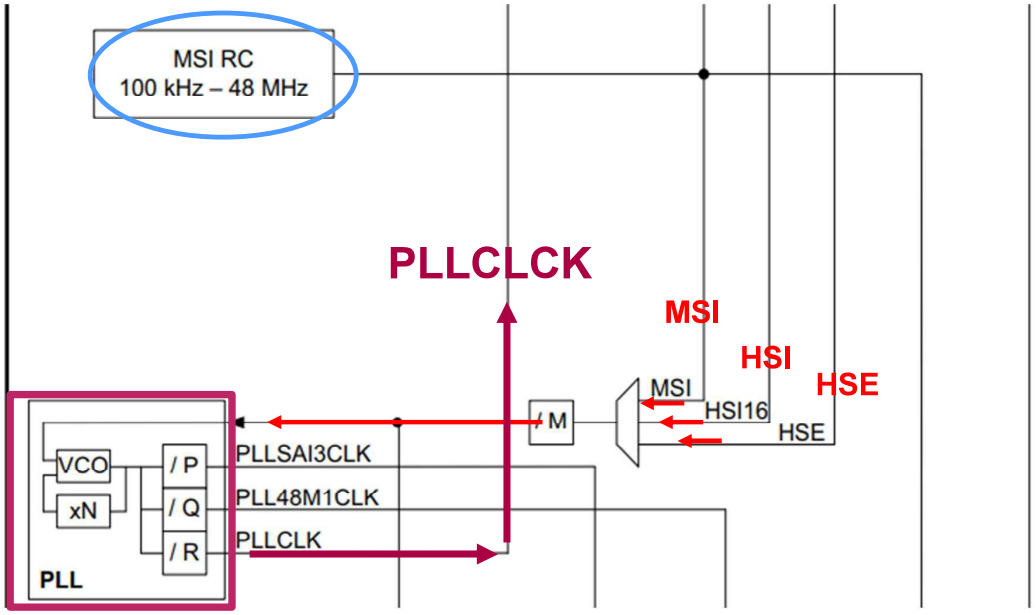


HCLK = FCLK processor clock

APB1 peripherals

APB2 peripherals

STM32L476 Clock Tree (PLL)



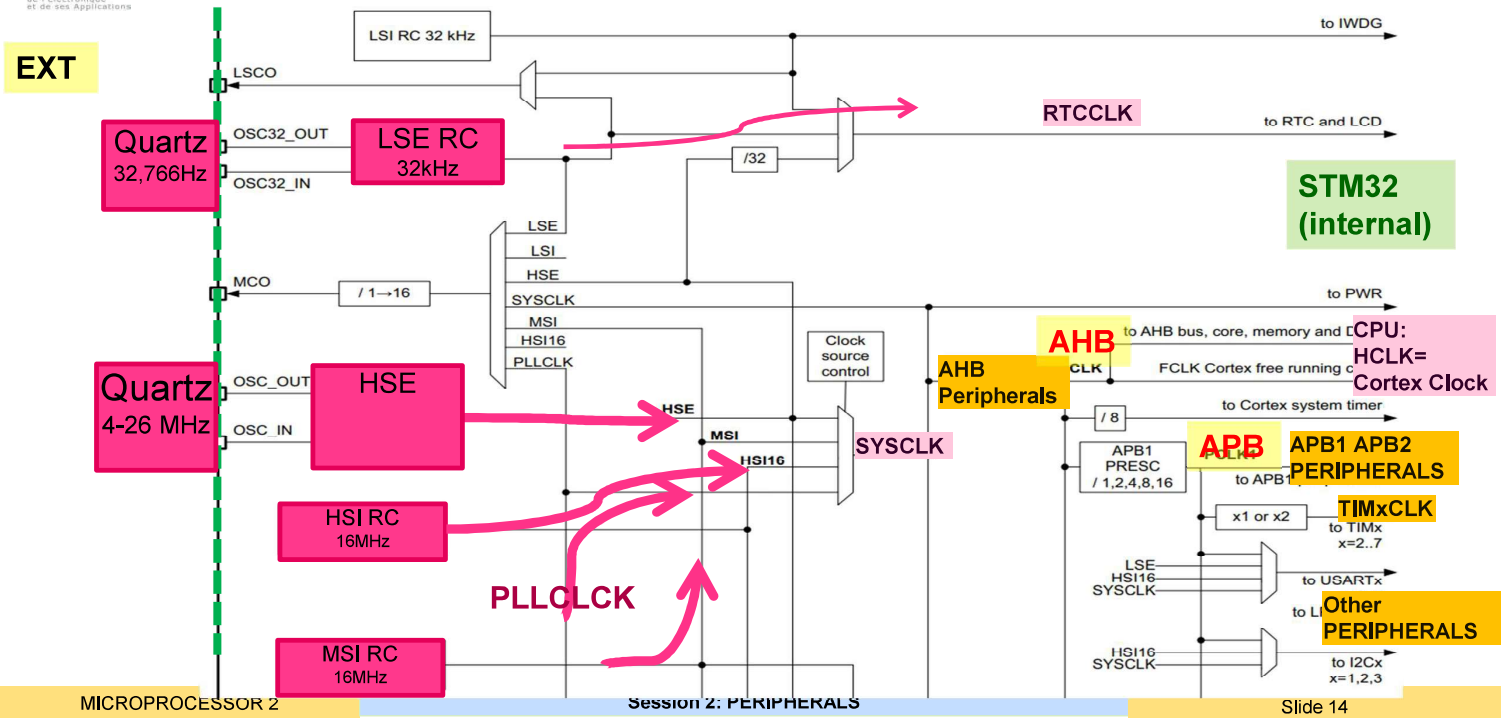
Source of clocks

- Peripherals are synchronous digital circuits
  - They are synchronized by different clocks
  - All of these clocks are made from the SYSCLK
  - The SYSCLK clock is derived from a fast internal or external clock.
- 4 primary clock sources
  - 3 internal
    - RC 16MHz (HSI)
    - RC 32kHz (LSI)
    - RC 400kHz-48MHz (MSI)
  - 2 external
    - Quartz 4 to 26 MHz (HSE)
    - Quartz 32 768 Hz (LSE)

Generated clocks

- A clock signal is obtained from another clock signal:
  - By simple **frequency division = PRESCALER**: easy to obtain (seen last year).
  - By **frequency multiplication** using a **PLL**: more complex (seen in analog course) (VCO)
- Many clocks are generated from the primary clocks:
  - **SYSCLK** (System clock)
  - RTCCLK (real time) (calendar function)
  - IWDGCLK (guard dog)
  - …
- Multitude of clocks generated for peripherals, from SYSCLK:
  - AHB for high-speed peripherals
  - APB1 and APB2 peripheral clocks
  - APB1 and ABP2 timer clocks (different from the previous ones!)
  - LCD-TFT clocks
  - …

Dependencies between clocks



RCC: Reset and Clock Control

- A peripheral is dedicated to the control of clocks: RCC (Reset and Clock Control)
  - It allows you to **adjust the speed of the** different clocks:
    - By signal selection
    - Frequency division
    - Frequency multiplication (PLL)
  - It allows to **enable** a peripheral (synchronized by the clock).
  - Remark 1: without clock, a device is inactive and mute; it cannot be read or written (so any initialization has no effect).
  - Note 2: RCC is one of the few devices that does not require validation to be active (and for good reason!).

RCC: Clock settings and reset (L476)

0x4002 1000 0x4002 13FF	Reset & Control Clocks Periph.

- Example: RCC_CFGR is located at address 0x40021000+0x08 = 0x40021008

Offset	Name Register	
0x00	Clock Control **RCC_CR**	Choice of clocks
0x04	PLL configuration **RCC_ICSCR**	Set the speed of clocks
0x08	Clock Configuration **RCC_CFGR**	
0x0C	PLL configuration **RCC_PLLCFGR**	
0x10	PLLSAI1 Configuration Register **RCC_PLLSAI1CFGR**	To set PLLSAI1 and PLLSAI2
0x14	PLLSAI2 Configuration Register **RCC_PLLSAI2CFGR**	
0x18	Clock Interrupt Enable Register **RCC_CIER**	To set Events from the clocks
0x1C	Clock Interrupt Flag Register **RCC_CIFR**	
0x20	Clock Interrupt Clear Register **RCC_CICR**	For more details: RM
0x24	…	

RCC: Activation of Peripheral Clocks (L476)

0x4002 1000 0x4002 13FF	Reset & Control Clocks Periph.

Enable activation of clocks peripherals

Without this activation, the peripherals are TOTALLY INACTIVE
They can't be used!

Offset	Name Register	
0x48	Peripheral clock enable register **RCC_AHB1ENR**	AHB
0x4C	Peripheral clock enable register **RCC_AHB2ENR**	
0x50	Peripheral clock enable register **RCC_AHB3ENR**	
0x54	- unused	
0x58	Peripheral clock enable register **RCC_APB1ENR1**	APB1
0x5C	Peripheral clock enable register **RCC_APB1ENR2**	
0x60	Peripheral clock enable register **RCC_APB2ENR**	APB2
0x64	- unused	
0x68	**RCC_AHB1SMENR**	
0x6C	**RCC_AHB2SMENR**	

PERIPHERAL activation

- Registers allow the activation of clocks to use ports (GPIO) and peripherals (internal):
  - RCC_AHB1ENR, RCC_AHB2ENR and RCC_AHB3ENR for peripherals on **AHB**
  - RCC_APB1ENR1 and RCC_APB1ENR2 for peripherals on **APB1**
  - RCC_APB2ENR for peripherals on **APB2**
  - The **principle is: setting a bit to 1 activates the peripheral associated with that bit.**
  - HAL functions are available to do it quickly:

```
__HAL_RCC_GPIOB_CLK_ENABLE();
```

Details of RCC_AHB1ENR (L476)

AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x48

Reset value: 0x0000 0100

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DMA2DEN	TSCEN
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	CRCEN	Res.	Res.	Res.	FLASHEN	Res.	Res.	Res.	Res.	Res.	Res.	DMA2EN	DMA1EN
			rw				rw							rw	rw

- We can enable these peripherals:
  - DMA1, DMA2, CRC, TSC, DMA2D
- FLASH is already enabled at Reset!

Details of RCC_AHB2ENR (L476)

AHB2 peripheral clock enable register (RCC_AHB2ENR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNGEN	HASHEN	AESEN
													rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	DCMIEN	ADCEN	OTGFSEN	Res.	Res.	Res.	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
	rw	rw	rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

- Useful for GPIOs and ADC

Details of RCC_AHB3ENR (L476)

AHB3 peripheral clock enable register(RCC_AHB3ENR)

Address offset: 0x50

Reset value: 0x00000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	QSPI EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	FMC EN
							rw								rw

- Only 2 peripherals there.

Details of RCC_APB1ENR1 (L476)

APB1 peripheral clock enable register 1 (RCC_APB1ENR1)

Address: 0x58

Reset value:
0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
LPTIM1 EN	OPAMP EN	DAC1 EN	PWR EN	Res.	CAN2 EN	CAN1 EN	CRSEN	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
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
SPI3 EN	SPI2 EN	Res.	Res.	WWD GEN	RTCA PBEN	LCD EN	Res.	Res.	Res.	TIM7 EN	TIM6EN	TIM5EN	TIM4EN	TIM3EN	TIM2 EN
rw	rw			rs	rw	rw				rw	rw	rw	rw	rw	rw

- Plenty of peripherals on APB1:
  - I2C, UART, SPI, TIM2 to TIM7

Details of RCC_APB1ENR2 (L476)

APB1 peripheral clock enable register 2 (RCC_APB1ENR2)

Address offset: 0x5C

Reset value: 0x00000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LPTIM2 EN	Res.	Res.	SWP MI1 EN	I2C4EN	LP UART1 EN
													rw	rw	rw

- Even more peripherals on APB1.

Details of RCC_APB2ENR (L476)

APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x60

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	DFSDM1 EN	Res.	SAI2 EN	SAI1 EN	Res.	Res.	TIM 17EN	TIM16 EN	TIM15 EN
							rw		rw	rw			rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1 EN	TIM8 EN	SPI1 EN	TIM1 EN	SD MMC1 EN	Res.	Res.	FW EN	Res.	Res.	Res.	Res.	Res.	Res.	SYS CFGEN
	rw	rw	rw	rw	rw			rs							rw

- More peripherals on APB2:
  - TIM1,TIM8,TIM15 to TIM17, SPI1, USART1

Setting bus frequencies

- The frequencies of the individual busses can be set by directly changing the registers: RCC_CFGR and RCC_PLLCFGR
  - This is tricky: order to be respected, documentation to be read...
- It is better to use HAL: functions like HAL_RCC_ClockConfig
- Simpler with STM32CubeIDE
  - Use the STM32CubeMX interface that generates the code using HAL functions to set the bus frequencies.
  - Do not lose the link with the low level!

RCC registers for frequency control (L476)

Clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MCOPRE[2:0]			MCOSEL[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	rw	rw	rw	rw	rw	rw	rw								
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP WUCK	Res.	PPRE2[2:0]			PPRE1[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

PLL configuration register (RCC_PLLCFGR)

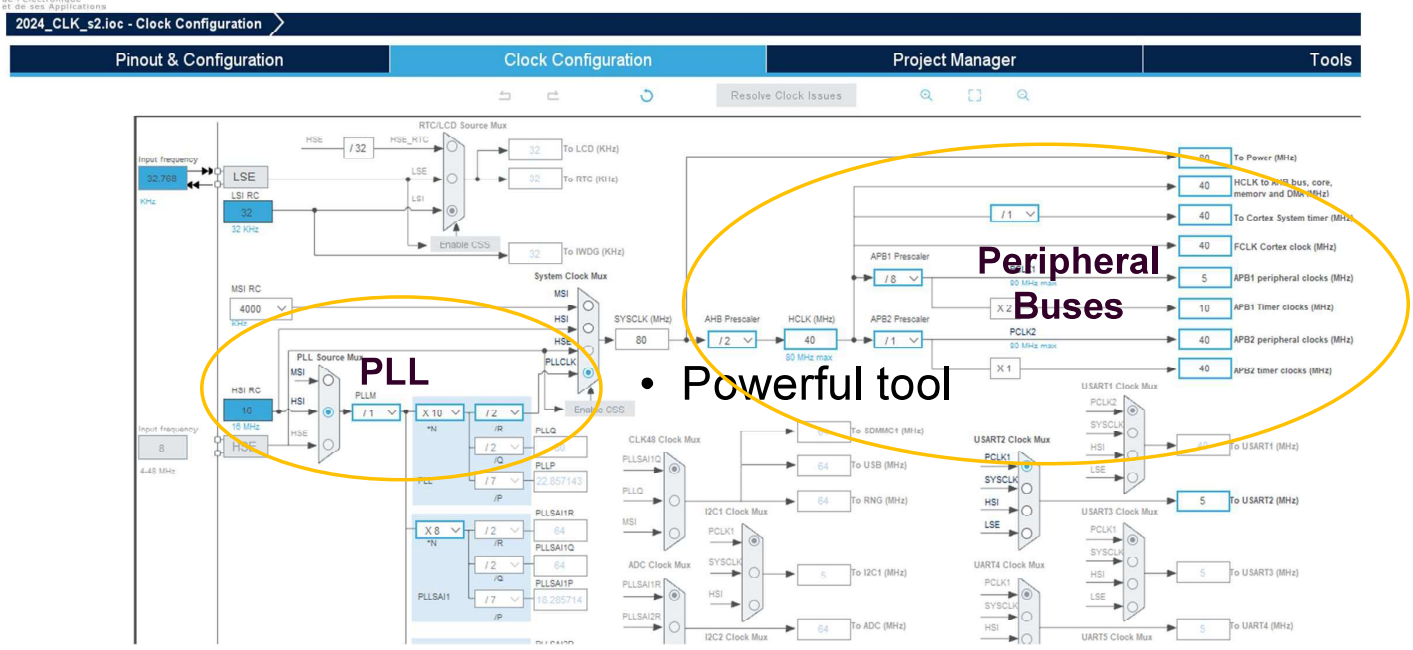
Address offset: 0x0C

Reset value: 0x0000 1000

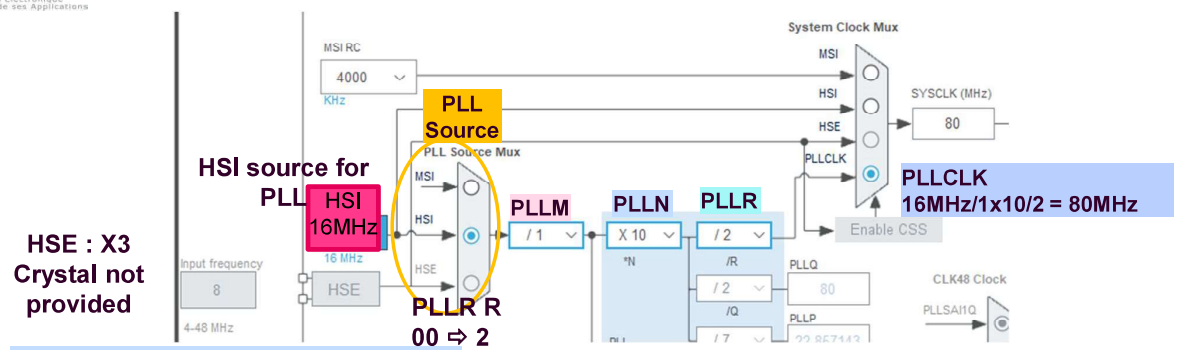
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLLPDIV[4:0]					PLLR[1:0]		PLL REN	Res.	PLLQ[1:0]		PLL QEN	Res.	Res.	PLLP	PLL PEN
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLN[7:0]							Res.	PLLM[2:0]			Res.	Res.	PLLSRC[1:0]	
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw			rw	rw

- Note: " Reset value " is the value in the register after it has been powered on (or reset).

Setting clock with STM32CubeIDE



PLL frequency setting (L476)



Fields from RCC_PLLCFGR

PLLR R
00 ⇨ 2
01 ⇨ 4
10 ⇨ 6
11 ⇨ 8

⇨R=2 ⇨PLL ENABLED

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLLPDIV[4:0]					PLLR		PLL REN	Res.	PLLQ[1:0]		PLL QEN	Res.	Res.	PLLP	PLL PEN
rw	rw	rw	rw	rw	0	0	1		rw	rw	rw			rw	rw

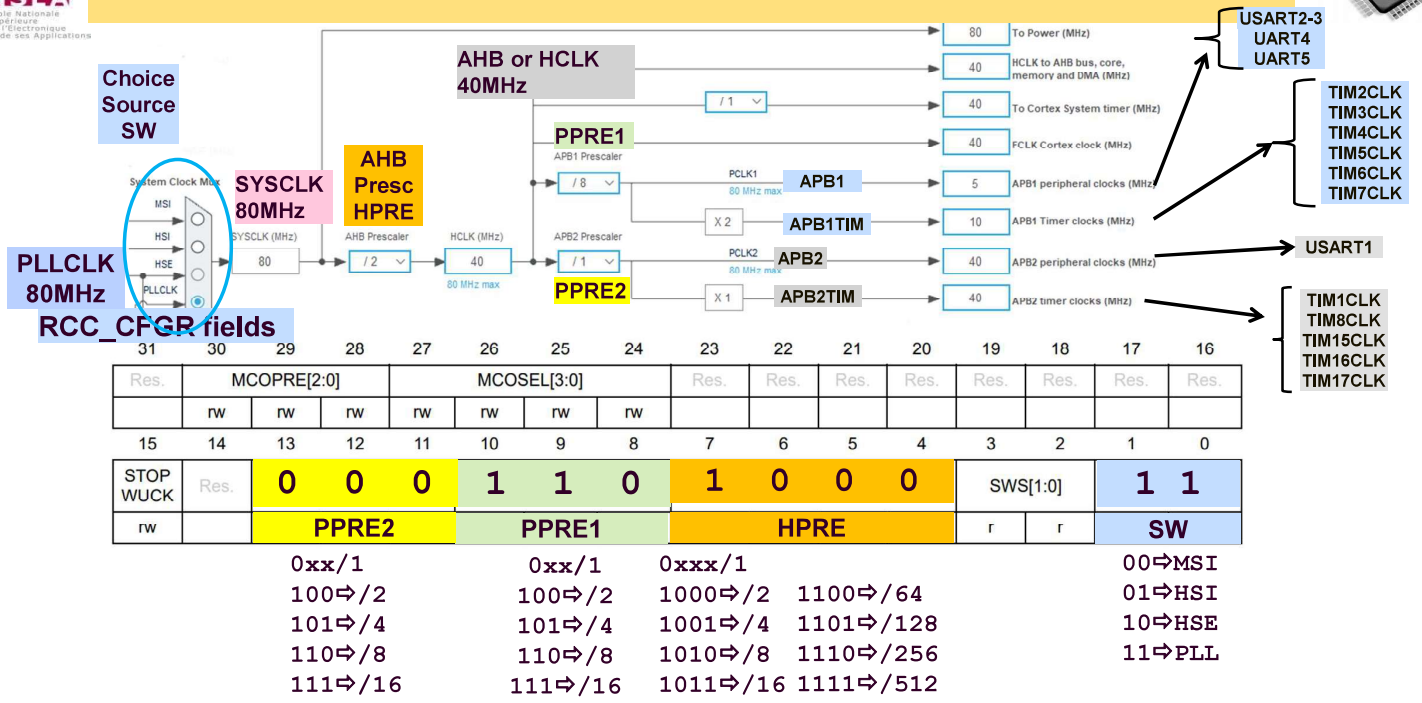
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	PLLN[6:0]							Res.	PLLM[2:0]			Res.	Res.	PLLSRC[1:0]	
	0	0	0	1	0	1	0		0	0	0			1	0

Binary value N=10

M-1 in binary

PLLSRC
00 ⇨ no
01 ⇨ MSI
10 ⇨ HSI
11 ⇨ HSE

Bus frequency for peripherals and Timers(L476)



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	MCOPRE[2:0]			MCOSEL[3:0]				Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
	rw	rw	rw	rw	rw	rw	rw								

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STOP WUCK	Res.	0	0	0	1	1	0	1	0	0	0	SWS[1:0]		1	1
rw		PPRE2			PPRE1			HPRE				r	r	SW	

0xx/1
100⇨/2
101⇨/4
110⇨/8
111⇨/16

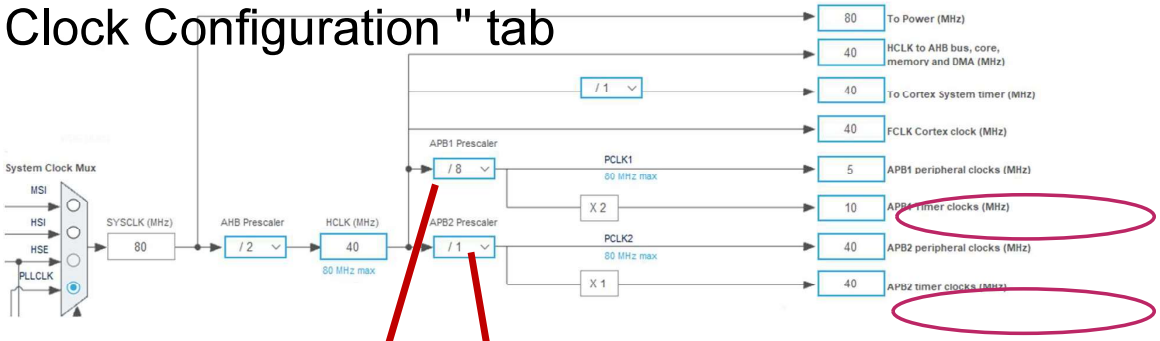
0xx/1
100⇨/2
101⇨/4
110⇨/8
111⇨/16

0xxx/1
1000⇨/2 1100⇨/64
1001⇨/4 1101⇨/128
1010⇨/8 1110⇨/256
1011⇨/16 1111⇨/512

00⇨MSI
01⇨HSI
10⇨HSE
11⇨PLL

Timer clock

- The " Clock Configuration " tab

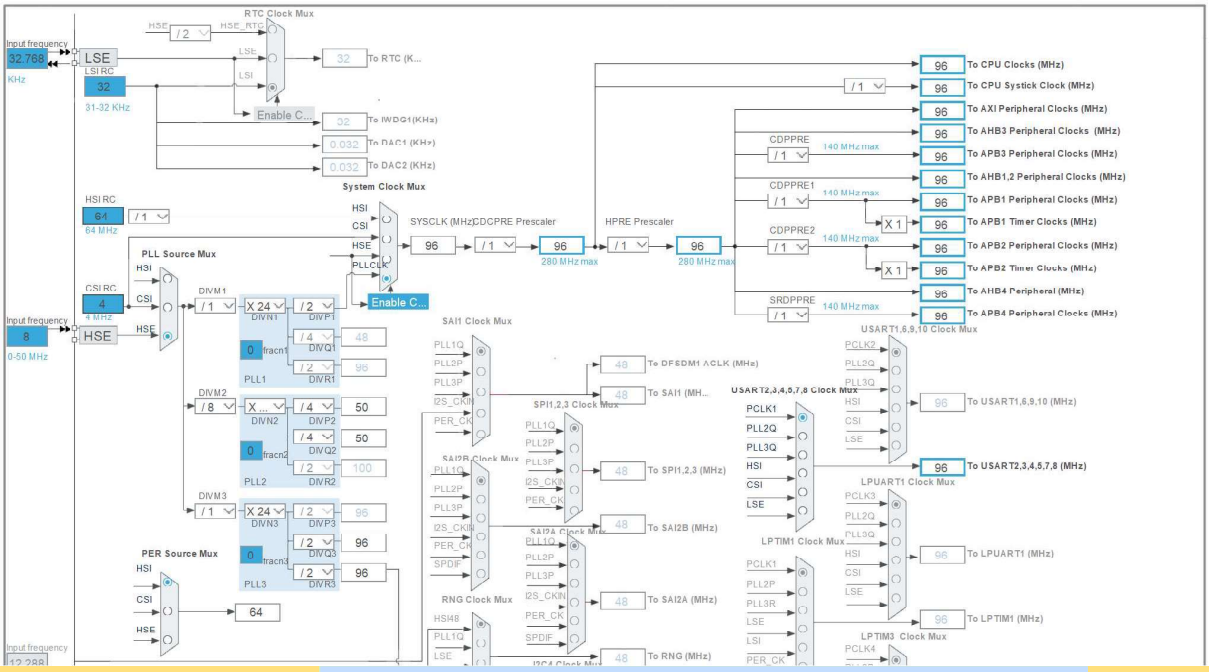


- Little subtlety:
  - If APBnPrescaler == 1
    - Then APBnTimerClock = **1** . APBnPeripheralClock
    - Otherwise APBnTimerClock = **2** . APBnPeripheralClock

Choice of Cortex frequency

- SYSCLK frequency
- High frequency
  - Advantage: high performance
  - Disadvantage: high energy consumption.
- Low frequency
  - Advantage: low energy consumption
  - Disadvantage: low performance
- How do I decide which clock frequency to send to the Cortex M4?
  - It will be up to you to decide according to your needs for autonomy vs. the targeted performance.
  - Note: there are "low power" modes available.

Clock configuration for STM32H7A3

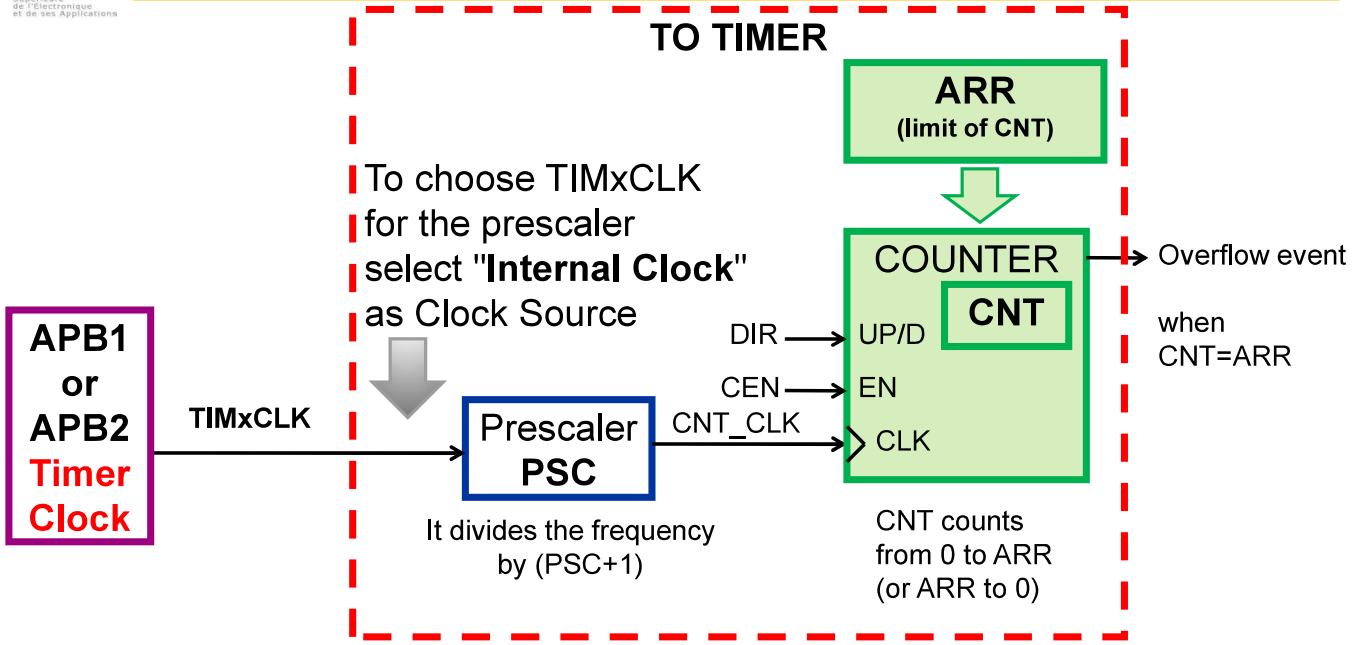


A more complex clock system than for the STM32F4xx

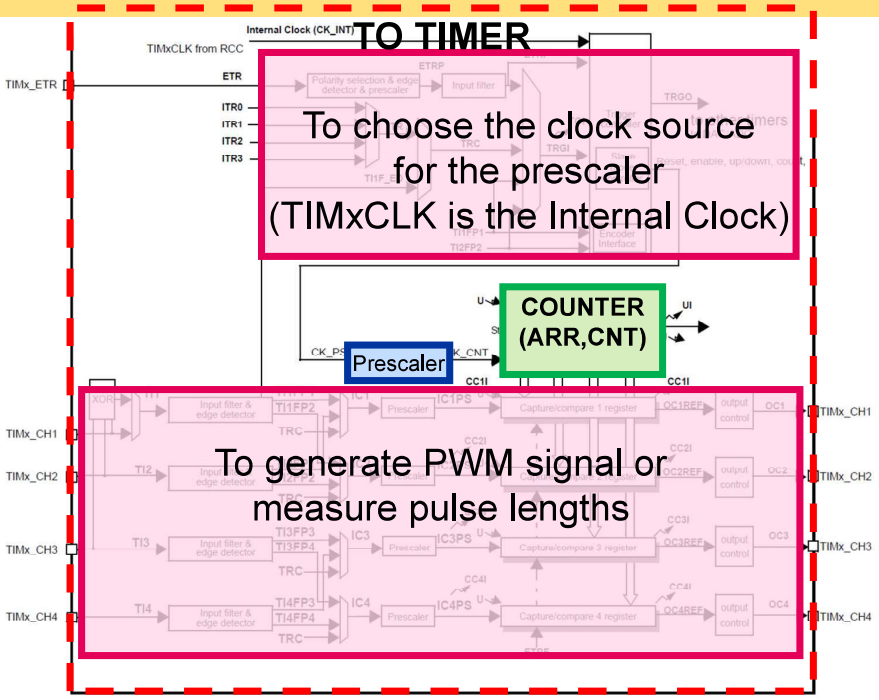
In the lab, we will use ioc view to configure the clocks

We will study the clock configuration for STM32F4xx but not for STM32H7A3

What is a TIMER?



A little more complex than expected

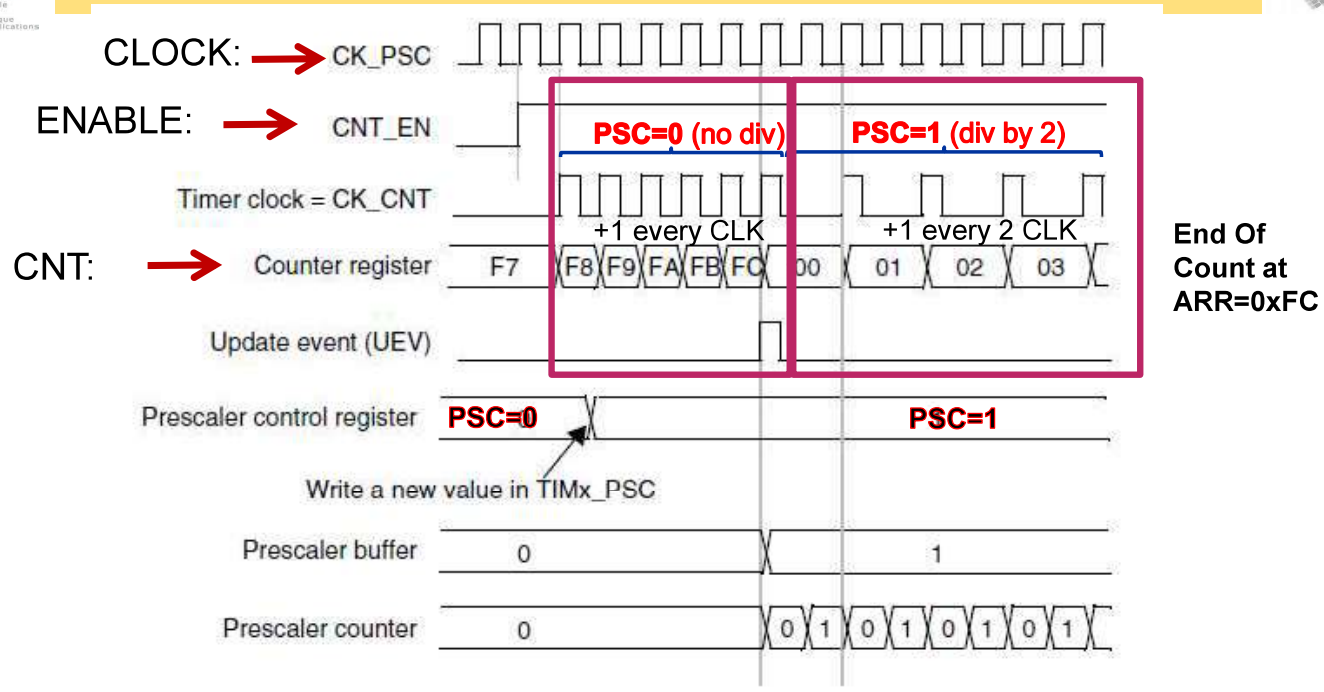


Utilities of a Timer

- To count at a fixed rate
  - It's a low level task.
  - The counter does this independently of the CPU.
  - It relieves the CPU from this task
- To generate an end of count event
  - This allows you to send an "interrupt" signal to interrupt the microprocessor and make it execute a specific code.
- To generate a PWM-type signal
  - This without microprocessor intervention (except for a short initialization).
- To measure the frequency and duty cycle of a PWM signal

A REVOIR

Division by 1 then by 2



TIMER registers: the ones that count!

- TIMx_PSC
  - Frequency divider division value (coded on 16 of the 32 bits): PSC+1
- TIMx_ARR
  - Count-up limit value or start value towards 0
  - On 16 bits (except for TIM2 and TIM5 on 32 bits)
- TIMx_CNT
  - 16-bit count value (32-bit for TIM2 and TIM5 only)
  - TIMx_CNT evolves from **0 to TIMx_ARR** or from TIMx_ARR to 0

Generation of a signal at the end of count

- Update" event: end of counting or CNT initialization
- UIE bit of TIMx_DIER
  - 0 prohibit update interruptions
  - 1 allow update interruptions

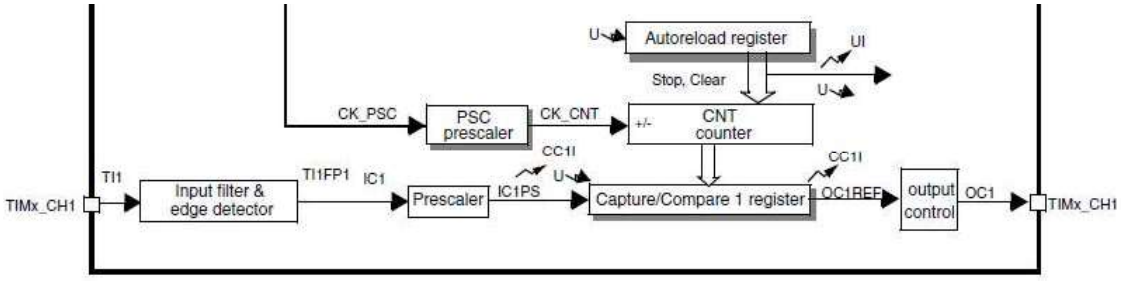
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	TDE	COMDE	CC4DE	CC3DE	CC2DE	CC1DE	UDE	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- UIF bit of TIMx_SR
  - This bit is set to 1 hardware
  - It is set to 0 by code
  - 0: no update
  - 1: an update interrupt is pending, i.e. an active interrupt request is pending.

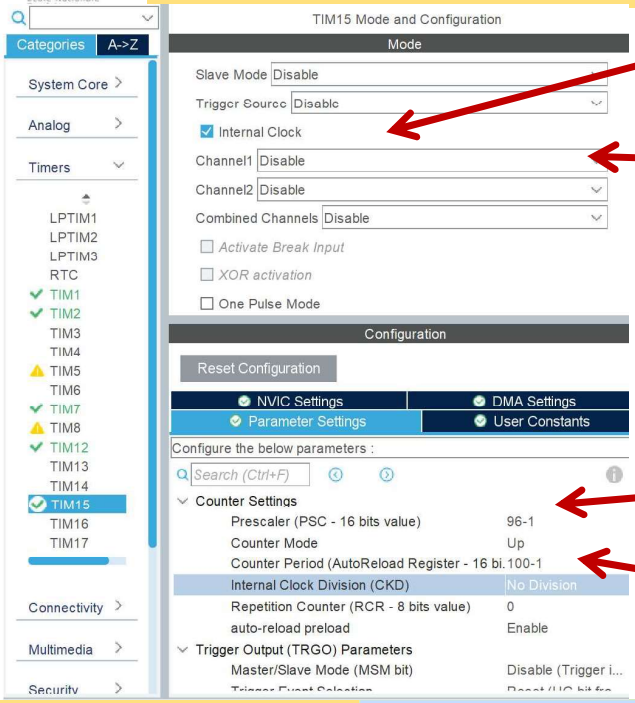
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CC4OF	CC3OF	CC2OF	CC1OF	Res.	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
			rc_w0	rc_w0	rc_w0	rc_w0	Res.	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	rc_w0

Other TIMER options

- It can count or count down
- You can choose between different sources for synchronization
- An interruption can be triggered on the acquisition of a signal (frequency calculation)
- A PWM signal can be generated using the counter and other registers.



Configuration of a Timer with CubeMX Tool (ioc file in STM32CubeIDE)



The input clock is the clock of the Peripheral bus

PWM can be selected there: it is associated with a pin

An “interrupt signal” is generated at the end of the count

PSC or Prescaler

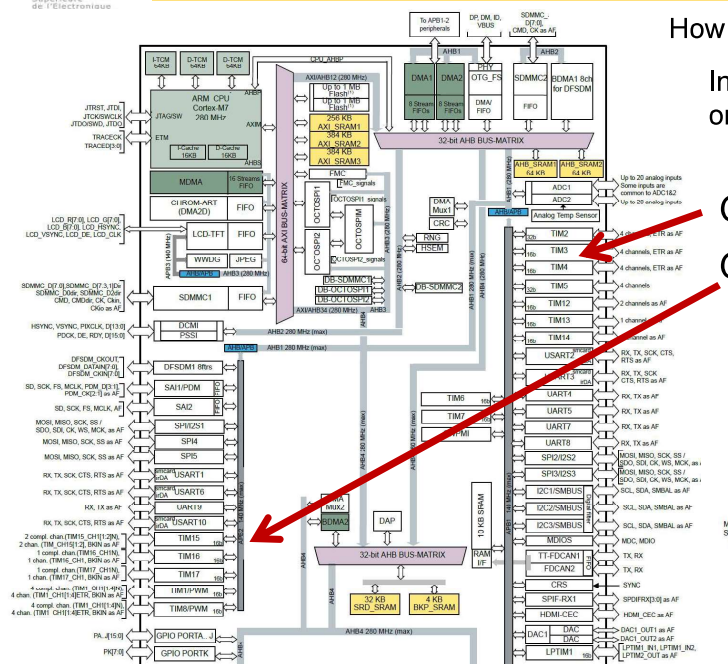
ARR

We will see next how to execute a code from this “event”

HAL initialization for Timer 2

- Code example for Timer2 (this code is generated by CubeIDE)
  - `TIM_HandleTypeDef htim2;`
  - `htim2.Instance = TIM2;`
  - `htim2.Init.Prescaler = 999; //PSC`
  - `htim2.Init.CounterMode = TIM_COUNTERMODE_UP;`
  - `htim2.Init.Period = 999; //ARR`
  - `htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1; // no division of the clock frequency`
  - `htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;`
  - `HAL_TIM_Base_Init(&htim2);`
- Then do not forget to launch the Timer! (to be added)
  - `HAL_TIM_Base_Start_IT(&htim2);`
  - or `HAL_TIM_Base_Start(&htim2);`

Block diagram of the STM32H7A3



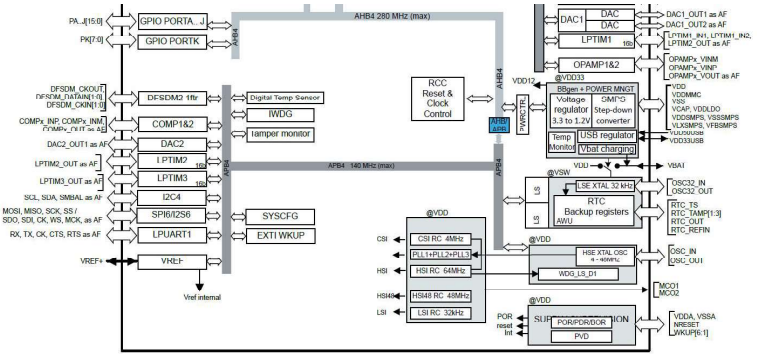
How to identify the clock of a Timer? (internal clock choice)

In the block diagram from the datasheet, it is possible to know on which buses (and so clocks) are connected the Timers

We read:

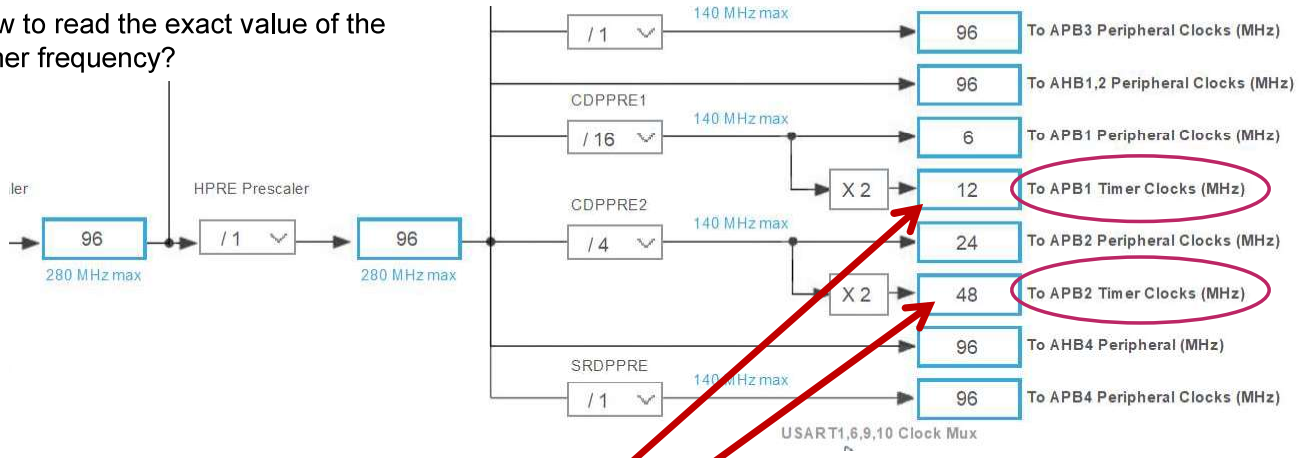
On APB1: TIM 2, 3, 4, 5, 12, 13 14

On APB2: TIM 1, 8, 15, 16, 17



Clocks for STM32H7A3 (example)

How to read the exact value of the Timer frequency?



for TIM 2, 3, 4, 5, 12, 13 or 14: APB1

for TIM 1, 8, 15, 16, 17 : APB2

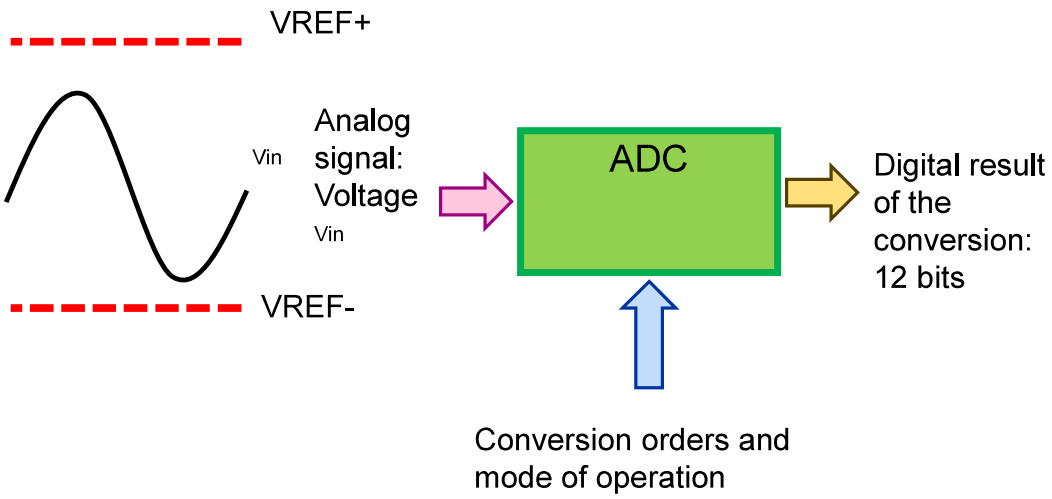
So 12 MHz clock sent to prescaler

So 48 MHz here sent to prescaler

ADC or Analog-to-Digital Converter

- **ADC=** Analog to Digital Converter
- In French: Convertisseur Analogique Numérique (CAN)
- Not to be confused with the CAN bus (Controller Area Network), which is widely used in the automotive industry.
- The ADC is therefore a component that allows an analog voltage present at its input to be presented in digital form.

ADC circuit

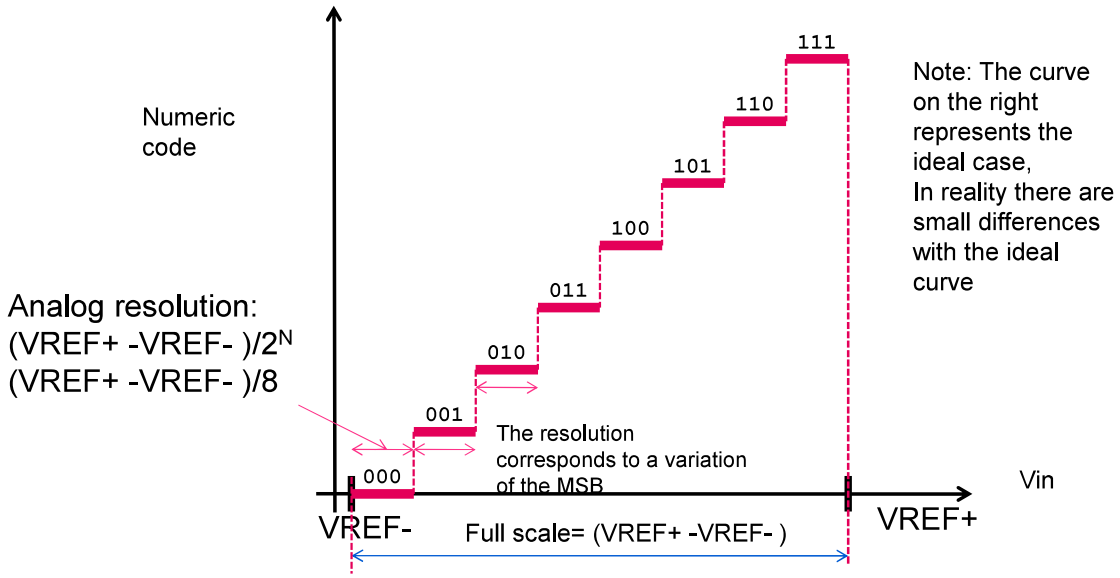


ADC of the STM32

- There are 1 to 3 ADCs on APB2
- They use conversion with successive approximations
  - The ADC response time is the time between the moment the conversion order is launched and the moment the numerical result is available.
  - Tconv for N bits=sampling Time + N cycles
  - The minimum for sampling time is 3 cycles.
  - The fastest conversion for 12 bits is therefore 12+3=15 cycles.
    - If ADCCLK at 10MHz then we get Tconv=1.5µs

ADC: vocabulary reminder

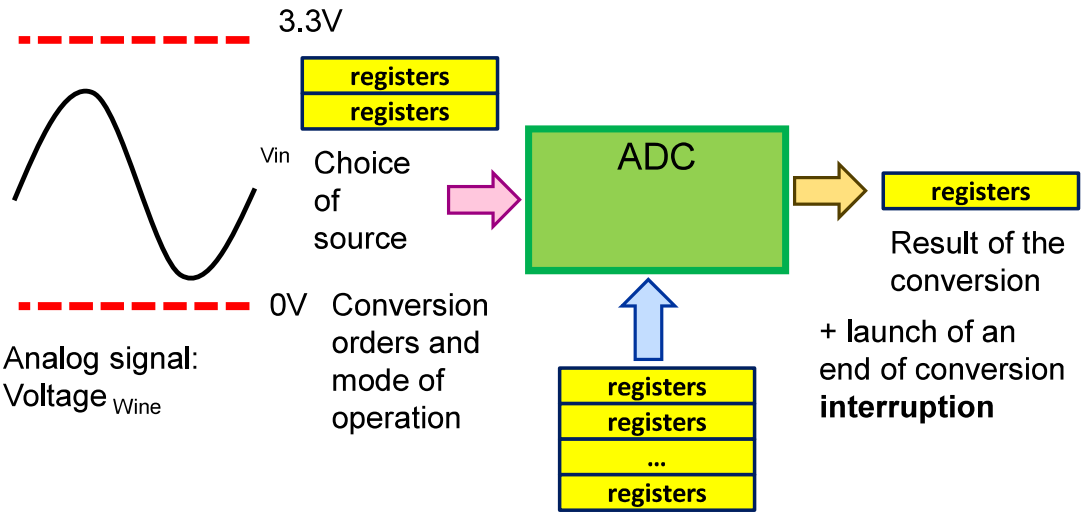
- Example with 3bits => $N=2^3=8$ levels

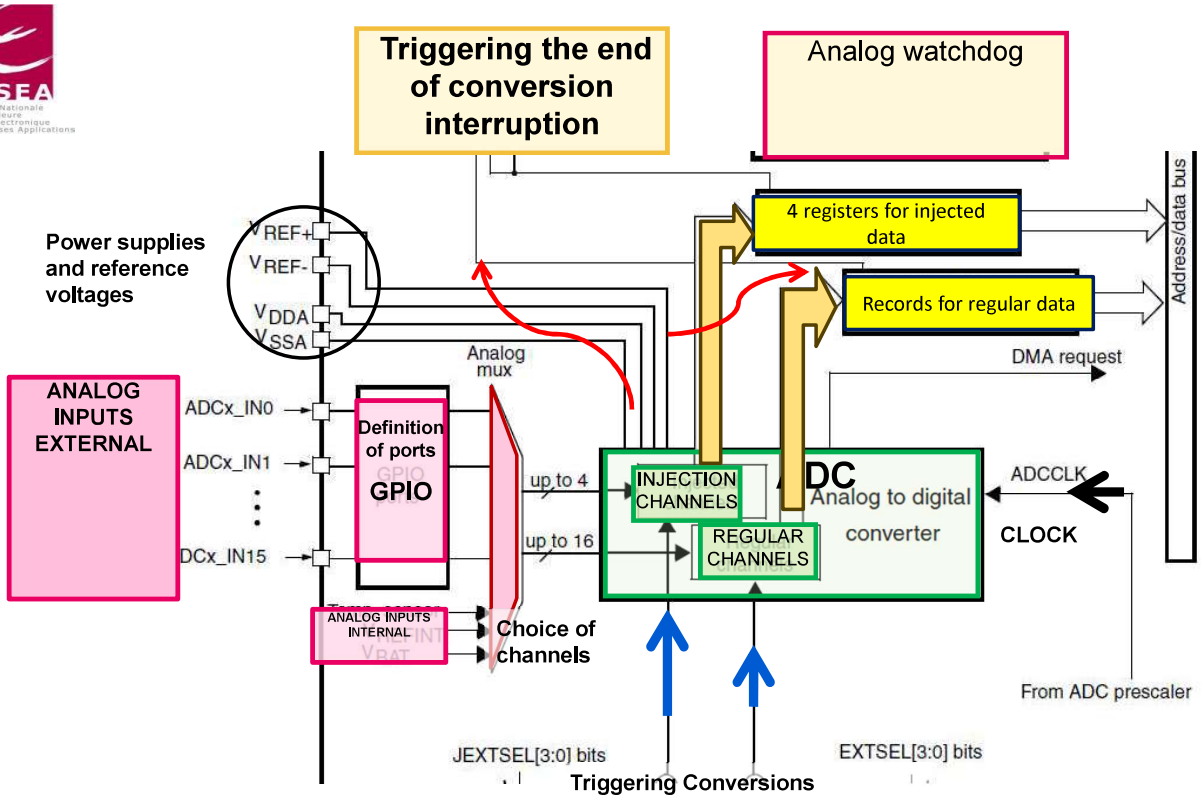


Characteristics of the ADC used

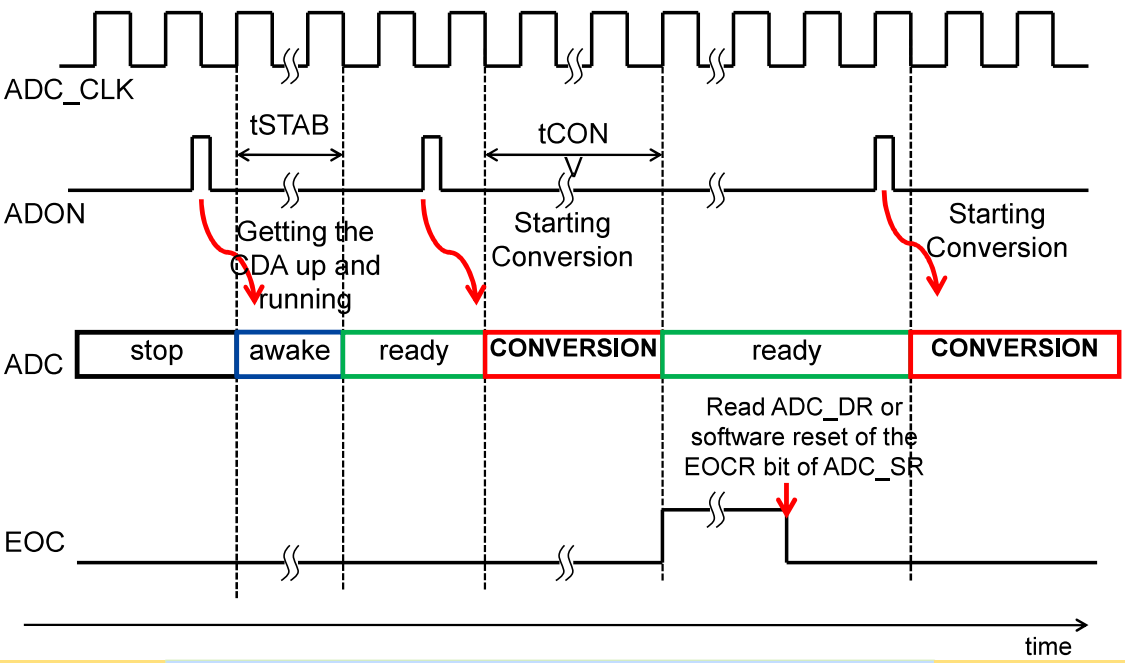
- VREF-=0V and VREF+=3.3V
  - full scale=3.3V
- Digital resolution of the STM32 ADC: 6, 8, 10 or 12 bits
  - For a digital resolution of 12 bits the analog resolution is therefore: $3.3/2^{12}=805.7\mu V$

ADC circuit in the STM32





Getting started and using ADC



Analog watchdog

Voltage observed on an input

High detection level

Warning zone:
If the observed voltage exceeds the high threshold Interruption

Determined by the posting in ADC_HTR

Authorized Zone

If the observed voltage remains in this area nothing happens.

Low detection level

Warning zone:
If the observed voltage falls below the low threshold Interruption

Determined by the posting in ADC_LTR

Events related to the ADC

- End of conversion: EOC
- Voltage overflow detected by the analog watchdog

HAL Initializations for ADC

```
ADC_HandleTypeDef hadc3;
hadc3.Instance = ADC3;
hadc3.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc3.Init.Resolution = ADC_RESOLUTION_12B;
hadc3.Init.ScanConvMode = ADC_SCAN_DISABLE;
hadc3.Init.ContinuousConvMode = DISABLE;
hadc3.Init.DiscontinuousConvMode = DISABLE;
hadc3.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc3.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc3.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc3.Init.NbrOfConversion = 1;
hadc3.Init.DMAContinuousRequests = DISABLE;
hadc3.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
 HAL_ADC_Init(&hadc3) ;
```

HAL channel Initializations for ADC

-	-	G10	18	K2	24	27	K2	PF6	I/O	FT	-	TIM10_CH1, SPI5_NSS, SAI1_SD_B, UART7_Rx, QUADSPI_BK1_IO3, EVENTOUT	ADC3_IN4

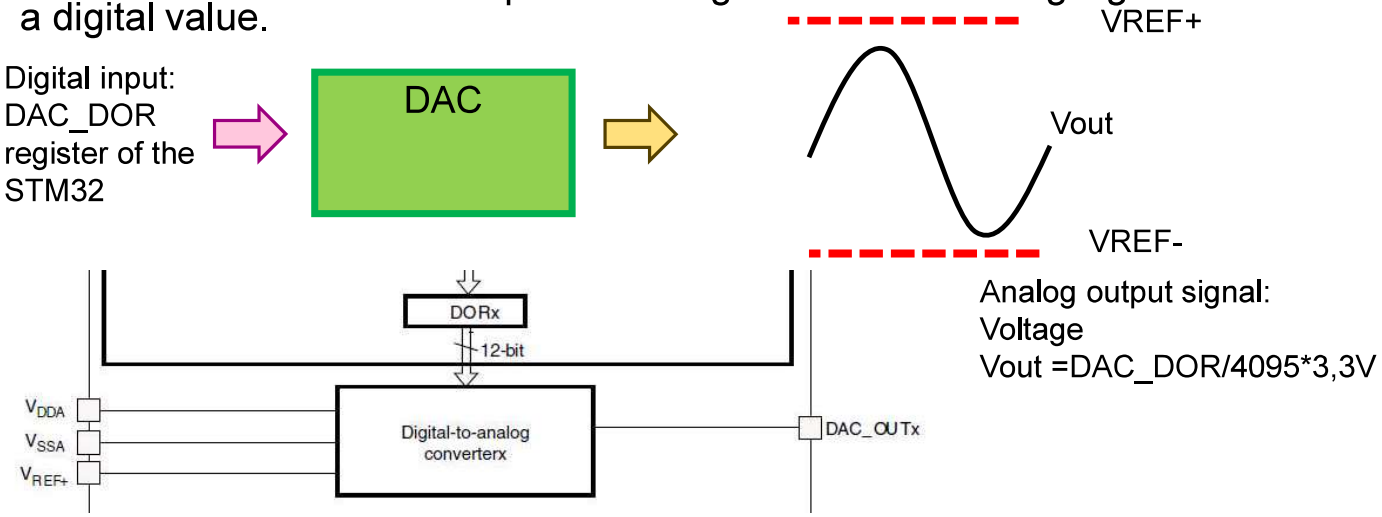
- In the datasheet:
- PF6 pin is channel 4 of ADC3 (configuration as Analog)
- `ADC_ChannelConfTypeDef sConfig = {0};`
- `  sConfig.Channel = ADC_CHANNEL_4;`
- `  sConfig.Rank = ADC_REGULAR_RANK_1;`
- `  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;`
- `  HAL_ADC_ConfigChannel(&hadc3, &sConfig);`

HAL functions for ADC

- To start a sofware conversion
- `HAL_ADC_Start(&hadc3);`
- Wait for the end of the conversion:
- `HAL_ADC_PollForConversion(&hadc3,10);`
- Get the value when it is ready:
- `val=HAL_ADC_GetValue(&hadc3);`

DAC (no DAC for F401)

- **DAC** = Digital to Analog Converter
- In French: Convertisseur Numérique Analogique (CNA)
- The DAC is therefore a component that generates an analog signal based on a digital value.

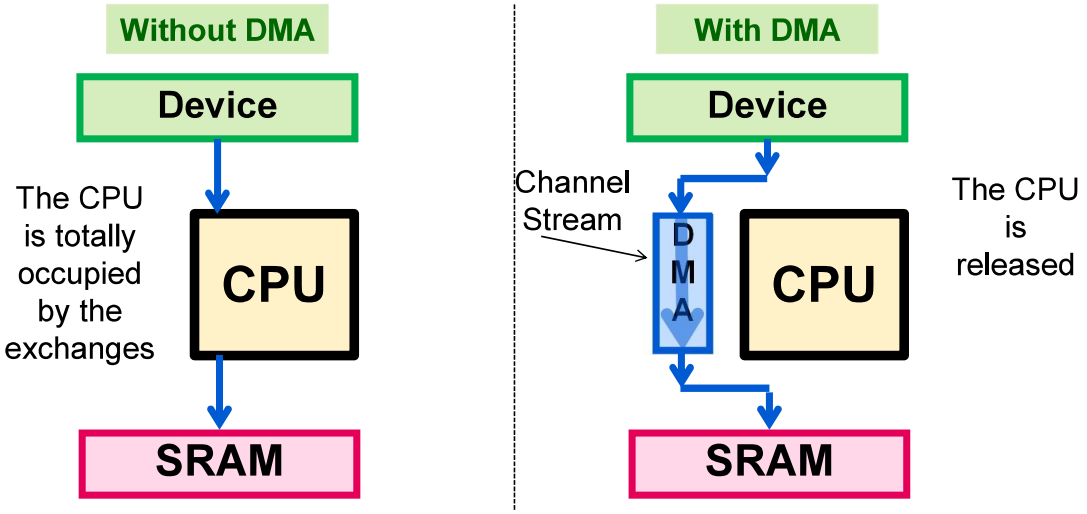


Initialization and use of the STM32 DAC

- **Activating the port** on which the DAC pin is located
- Analog input/output **configuration of the** DAC's output **pin**
  - MODER=11 and PUPDR=00
- **Activation of the DAC**
- In the default mode: any writing in the DAC_DHR register is immediately transferred to DAC_DOR and converted into an analog signal on the corresponding pin.
- It could have been more complicated!

DMA principle

- DMA: Direct Memory Access



DMA of STM32

- 16 possible channels
  - Programmable
    - From device to memory
    - From memory to device
    - From memory to memory
    - Peripheral to peripheral
  - Distributed on 2 controllers DMA1 and DMA2
- For channels
  - Give the start address (a device is associated with a memory)
  - Give the number of exchanges to be made

Key events related to peripherals

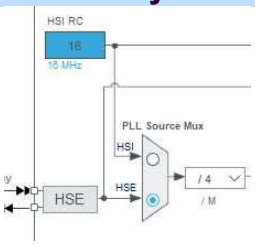
- Timer
  - Reaching an end of count
- ADC
  - End of conversion
  - Detection of tension by the watchdog
- DMA
  - End of data transfer
- These events will then be used to interrupt the execution in progress and to execute a code specific to each event (to be continued).

STM32CubeIDE

- Advanced C/C++ development platform for STM32 microcontrollers:
  - CubeMX (for easy configuration)
  - generation code
  - compilation code
  - debug features

Principles of STM32Cube

Step1: the settings are entered by the user



Step2: a C code is generated from the settings

```
157  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
158  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
159  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
160  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
161  RCC_OscInitStruct.PLL.PLLM = 4;
```

This code is integrated in a C project in which you can add your own code

Step3: this C code is built (or compiled)

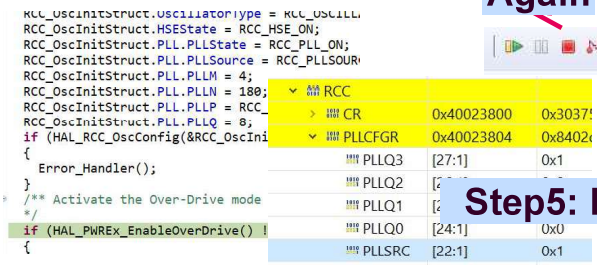
An elf file is generated if there are no errors. This file contains machine code of the ASM instructions and symbols of the iables and functions

Step4: the machine code is downloaded

The code is in the FLASH and is ready to be executed by the Cortex-M4 or M7

Step5: DEBUG!!

Again



Exercise 2 for Session 3

- Frequency of Timer interruption requests
  - We're considering Timer 7
  - The APB1 bus is set to synchronize Timer 7 to 20 MHz.
  - We initialize TIM7_PSC to 19999
  - During 1 s, how much does TIM7_CNT evolve?
  - If TIM7_ARR is set to 1999 and Timer7 is asked to generate interrupts at each end of count, how often does Timer7 send an end of count signal (or interrupt request) to the Cortex-M7?