# Embedded Systems Design using CPU

Alessandro Palla
PhD student in
Embedded System Design
alessandro.palla@for.unipi.it

# Slides and other stuffs

Website: http://for.unipi.it/alessandro_palla/

You can find there (maybe)

- Today’s slides (Theoretical lesson)
- Tomorrow’s slides (Practical lesson)
- Link to some datasheet
  - Cortex M4
  - STM32F4
  - STM32F4Discovery Board
  - LIS3DSH (Accelerometer)

# Lessons Outline

- Short seminario on STM32F4 Embedded CPU. Theory and simple example
- Today
    - Challenges and Constraints designing an embedded system using a CPU
    - STM32F4 Introduction
    - Cortex M4 CPU
        - Simple approach to Thumb Instruction Set
    - Memory Model
    - AMBA Bus
    - Clock
    - NVIC

# Outline (...continue)

- Power Management
- DMA
- Timer

- Tomorrow
  - Simple application development: USB Mouse controlled by g acceleration.
  - How it works? When user tilts the board, accelerometer detects it and change mouse pointer's position on PC screen.
  - Developing and testing application on STM32F4-Discovery board

## Software for Tomorrow

- Unfortunately we have only one developer board :(
- You can follow software developing by download two software (not mandatory)
  - STM32Cube MX
  - IAR Embedded Workbench -> Kickstarter Edition (free)

If you have a STM32F4Discovery board you can take it!!

# Embedded System

- Embedded computing systems
  - Computing systems embedded within electronic devices
  - Hard to define. Nearly any computing system other than a desktop computer
  - Billions of units produced yearly, versus millions of desktop units
  - Perhaps 50 per household and per automobile

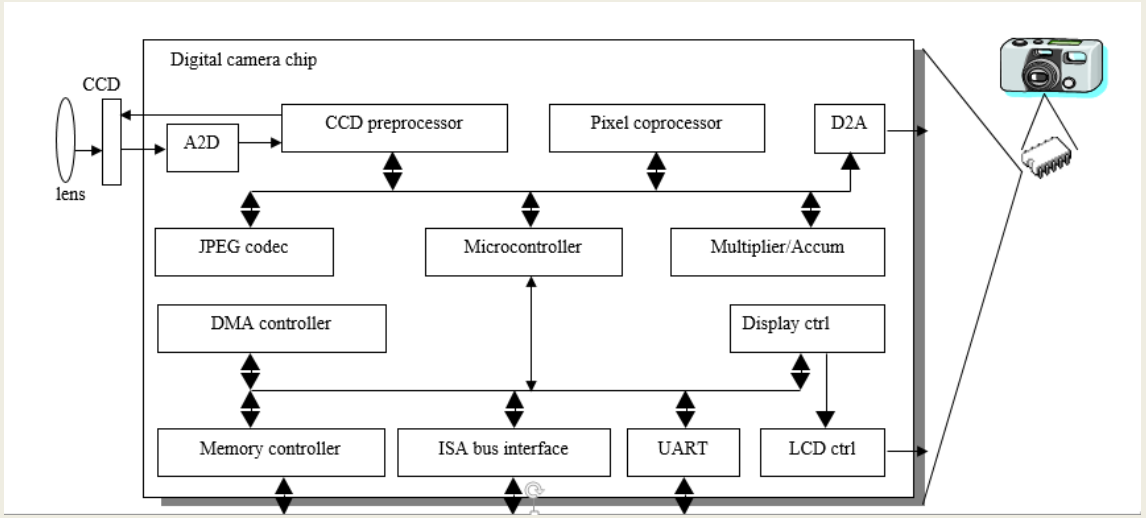
# Short list of embedded systems

- Anti-lock brakes
- Auto-focus cameras
- Automatic teller machines
- Automatic toll systems
- Automatic transmission
- Avionic systems
- Battery chargers
- Camcorders
- Cell phones
- Cell-phone base stations
- Cordless phones
- Cruise control
- Curbside check-in systems
- Digital cameras
- Disk drives
- Electronic card readers
- Electronic instruments
- Electronic toys/games
- Factory control
- Fax machines
- Fingerprint identifiers
- Home security systems
- Life-support systems
- Medical testing systems
- Modems
- MPEG decoders
- Network cards
- Network switches/routers
- On-board navigation
- Pagers
- Photocopiers
- Point-of-sale systems
- Portable video games
- Printers
- Satellite phones
- Scanners
- Smart ovens/dishwashers
- Speech recognizers
- Stereo systems
- Teleconferencing systems
- Televisions
- Temperature controllers
- Theft tracking systems
- TV set-top boxes
- VCR’s, DVD players
- Video game consoles
- Video phones
- Washers and dryers

## Common characteristics of embedded systems

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system’s environment
  - Must compute certain results in real-time without delay

# Design Example: digital camera



- Single-functioned -- always a digital camera
- Tightly-constrained -- Low cost, low power, small, fast
- Reactive and real-time -- only to a small extent

# Optimizing design metrics

- Obvious design goal:
  - Construct an implementation with desired functionality
- Key design challenge:
  - Simultaneously optimize numerous design metrics
- Design metric
  - A measurable feature of a system’s implementation
  - Optimizing design metrics is a key challenge

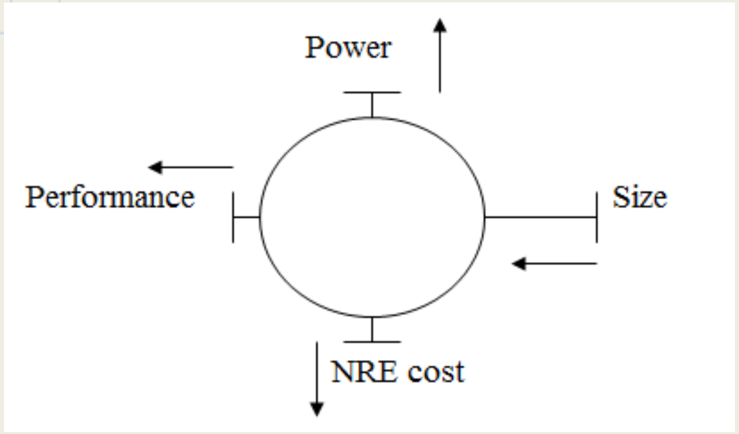
## Design Metrics

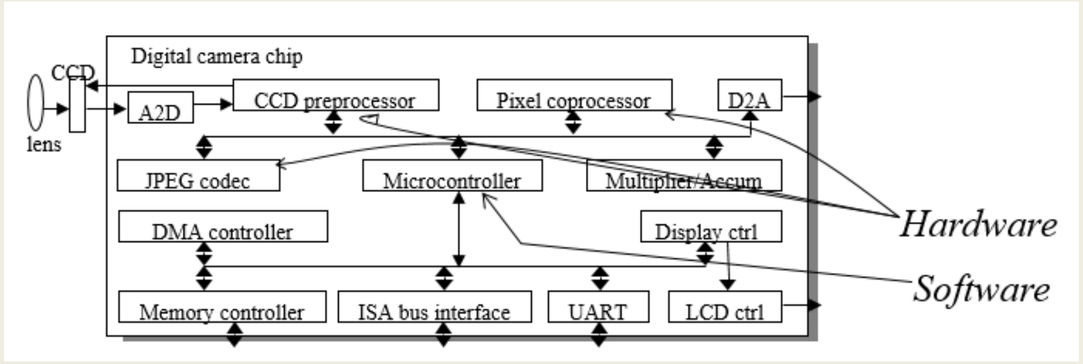
- Common metrics
  - Unit cost: the monetary cost of manufacturing each copy of the system, excluding NRE cost
  - NRE cost (Non-Recurring Engineering cost): The one-time monetary cost of designing the system
  - Size: the physical space required by the system
  - Performance: the execution time or throughput of the system
  - Power: the amount of power consumed by the system
  - Flexibility: the ability to change the functionality of the system without incurring heavy NRE cost

## Design Metrics

- Common metrics (continued)
  - Time-to-prototype: the time needed to build a working version of the system
  - Time-to-market: the time required to develop a system to the point that it can be released and sold to customers
  - Maintainability: the ability to modify the system after its initial release
  - Correctness, safety, many more

# Metrics Trade-Off





- Expertise with both **software and hardware** is needed to optimize design metrics
    - Not just a hardware or software expert, as is common
- A designer must be comfortable with various technologies in order to choose the best for a given application and constraints

# General-Purpose Processor

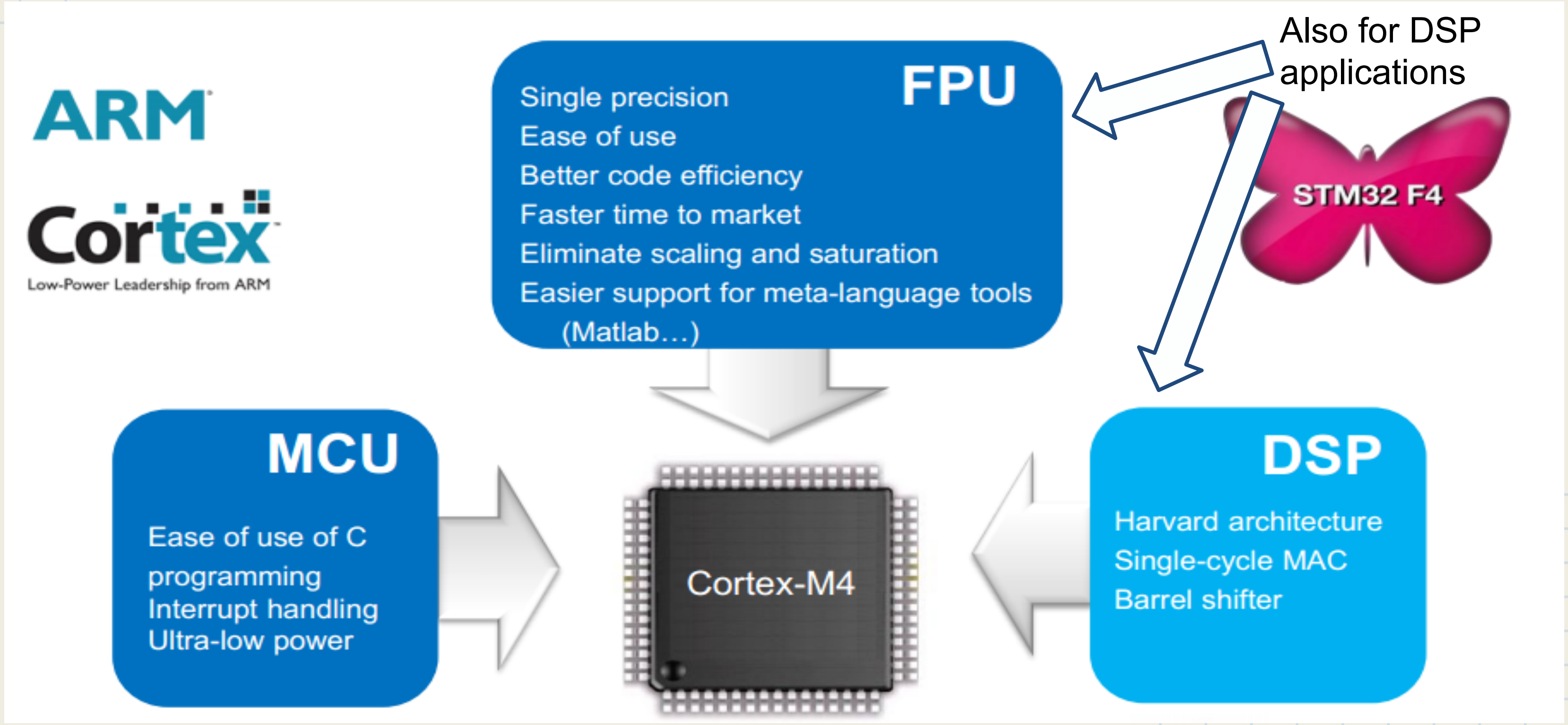
- General-Purpose Processor
    - Processor designed for a variety of computation tasks
    - Low unit cost, in part because manufacturer spreads NRE over large numbers of units
        - Motorola sold half a billion 68HC05 microcontrollers *only in 1996*
    - Carefully designed since higher NRE is acceptable
    - Can yield good performance, size and power
    - Low NRE cost, short time-to-market/prototype, high flexibility
        - User just writes software; no processor design
        - No need of complex hardware design, fast prototyping and error detection

## Case Study: STM32F4

- High performance embedded CPU using a Cortex M4 core
- Low cost development board (STM32F4Discovery costs approx 10$)
- Well documented
- Lots of feature and peripherals

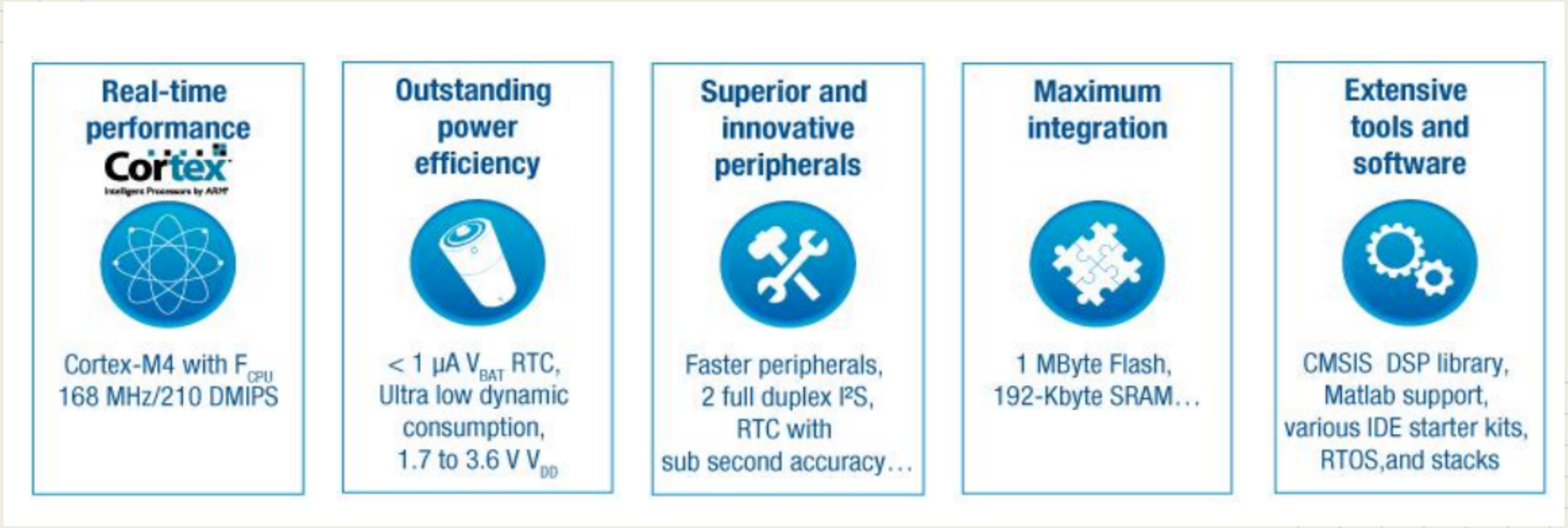
- M in Cortex M4 means “Microcontroller”. CPU designed for embedded systems that not requires too much computing power.
- Others Cortex family:
  - R: Realtime CPU, suitable for deep embedded real-time systems.
  - A: Application CPU, generic high performance processor

# STM32F4



Also for DSP applications

# Key Features



**Real-time performance**

Cortex-M4 with $F_{CPU}$ 168 MHz/210 DMIPS

**Outstanding power efficiency**

< 1 µA $V_{BAT}$ RTC, Ultra low dynamic consumption, 1.7 to 3.6 V $V_{DD}$

**Superior and innovative peripherals**

Faster peripherals, 2 full duplex I²S, RTC with sub second accuracy…

**Maximum integration**

1 MByte Flash, 192-Kbyte SRAM…

**Extensive tools and software**

CMSIS DSP library, Matlab support, various IDE starter kits, RTOS,and stacks

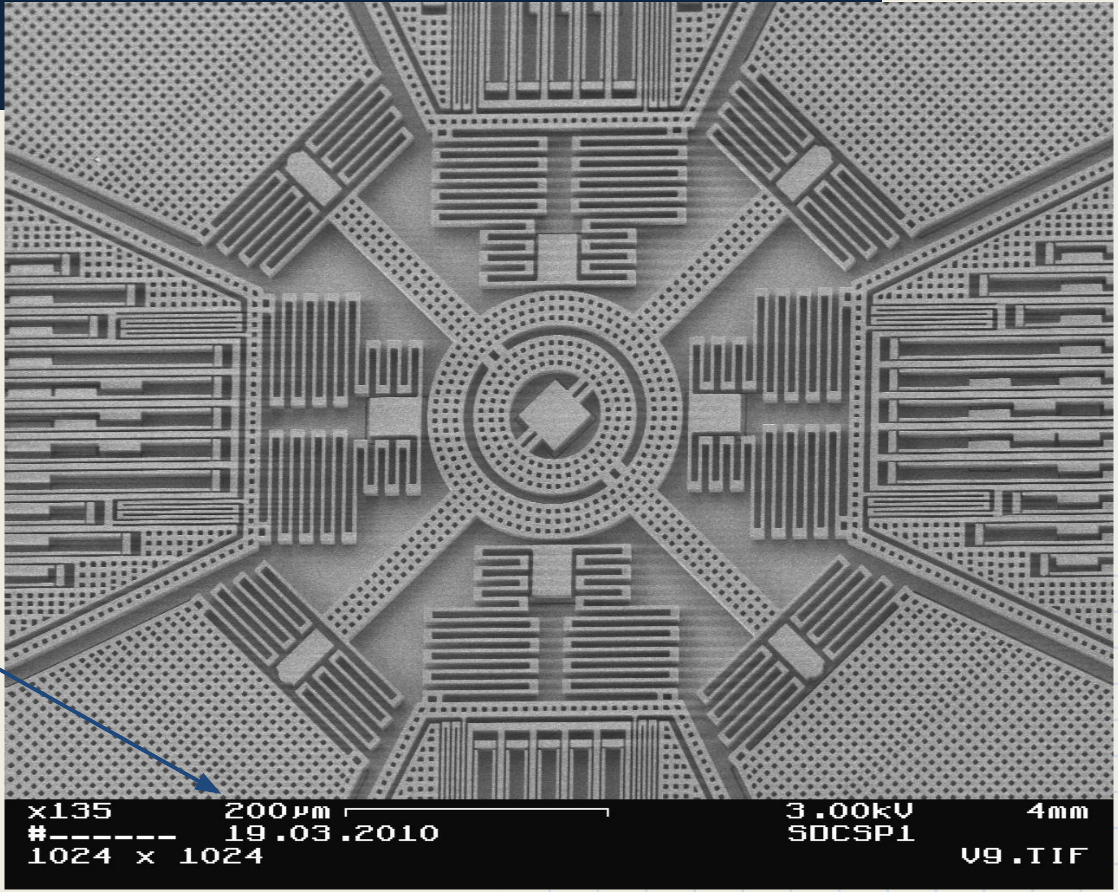
# STMicroelectronics

- One of the world top important productor of Electronics Devices
    - Embedded CPU
    - MEMS Sensor
        - MEMS means Micro-Electro-Mechanical-System, a sensor built in silicon into a chip to reduce dimension, costs and reliability
        - Example:
            - Microphone
            - Accelerometer, Gyro, Inertial Modules
            - Pressure Sensors and so on…
    - RFIC (Radio Frequency Integrated Circuit)
    - OPAMP and other analog electronics
    - ecc...
- An Italian-French company

# MEMS

Example of MEMS Accelerometer

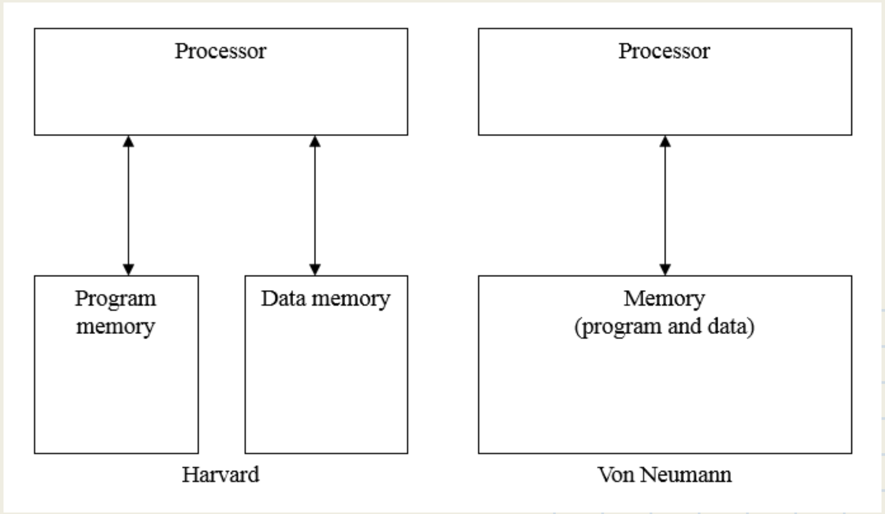
Embedded into a chip, note the scale!



# CPU Structure

- Von Neumann
  - Fewer memory wires
- Harvard
  - Simultaneous program and data memory access

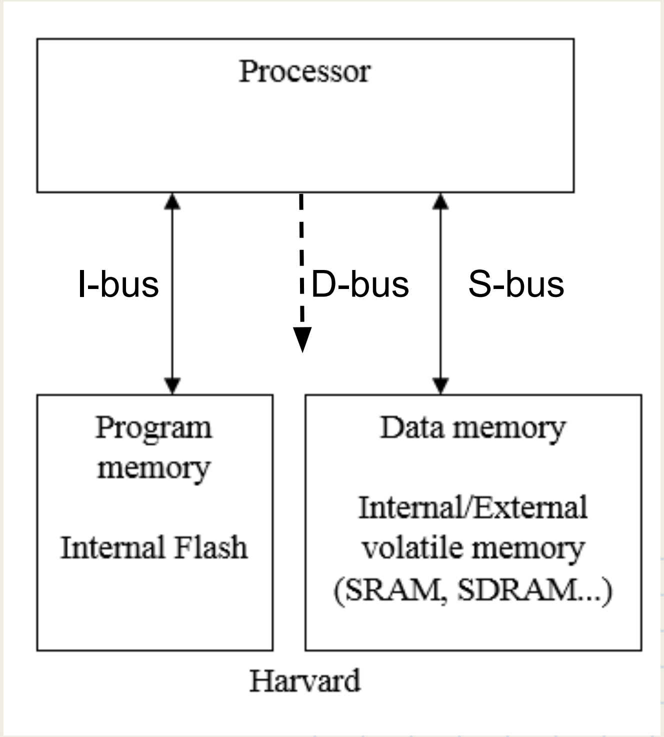
- In Harvard architecture CPU can access simultaneously both in program and data memory
- Cortex M4 has Harvard structure in order to improve CPU speed



# Memory Model

- Program is stored in a flash memory, data in a volatile memory like SRAM

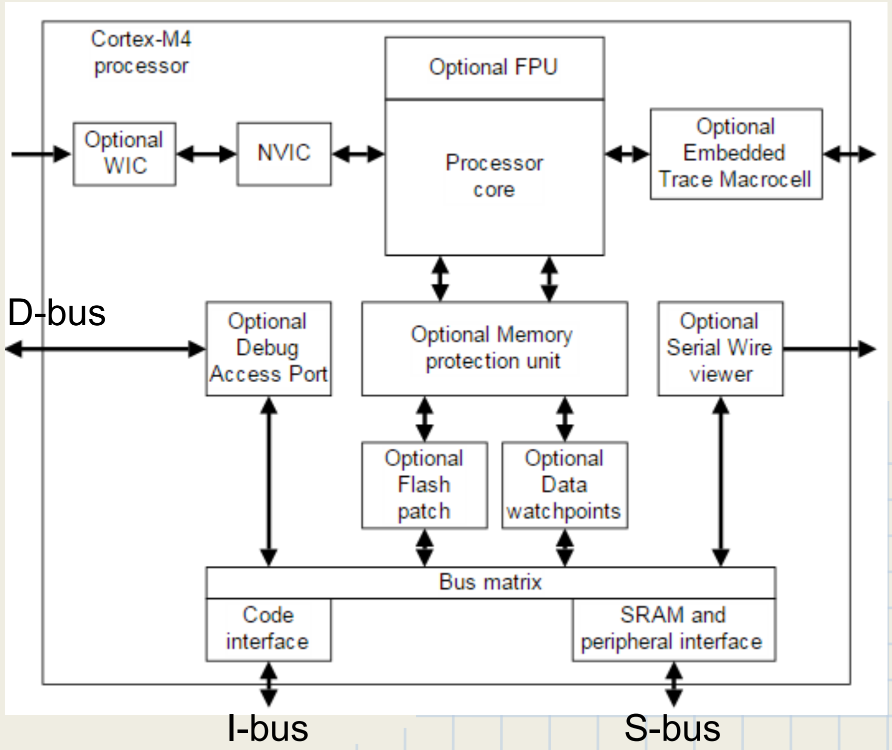
- I-bus: This bus is used by the core to fetch instructions.
- S-bus: This bus is used to access data located in a peripheral or in SRAM
- D-bus: This bus is used by the core for literal load and debug access.



# Cortex M4 Structure

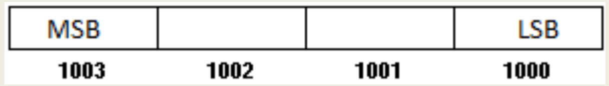
- Key Concept: from processor point of view everything is memory space!
  - Data Memory
  - Program Memory
  - Peripheral registers

- CPU access to memory only using LOAD and STORE instruction
  - RISC (Reduced Instruction Set Computer) paradigm

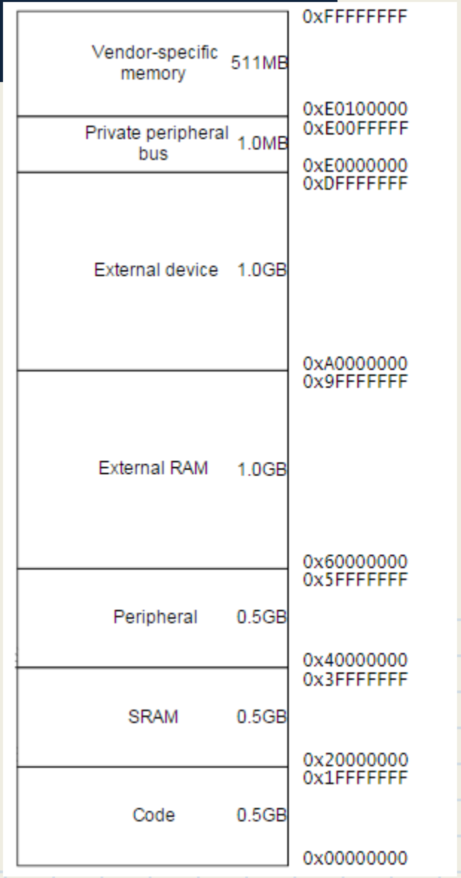

Cortex-M4 processor
Optional FPU
Optional WIC
NVIC
Processor core
Optional Embedded Trace Macrocell
D-bus
Optional Debug Access Port
Optional Memory protection unit
Optional Serial Wire viewer
Optional Flash patch
Optional Data watchpoints
Bus matrix
Code interface
SRAM and peripheral interface
I-bus
S-bus

# Memory Map

- 32 bit memory space (4 GB), byte-addressable, little endian:

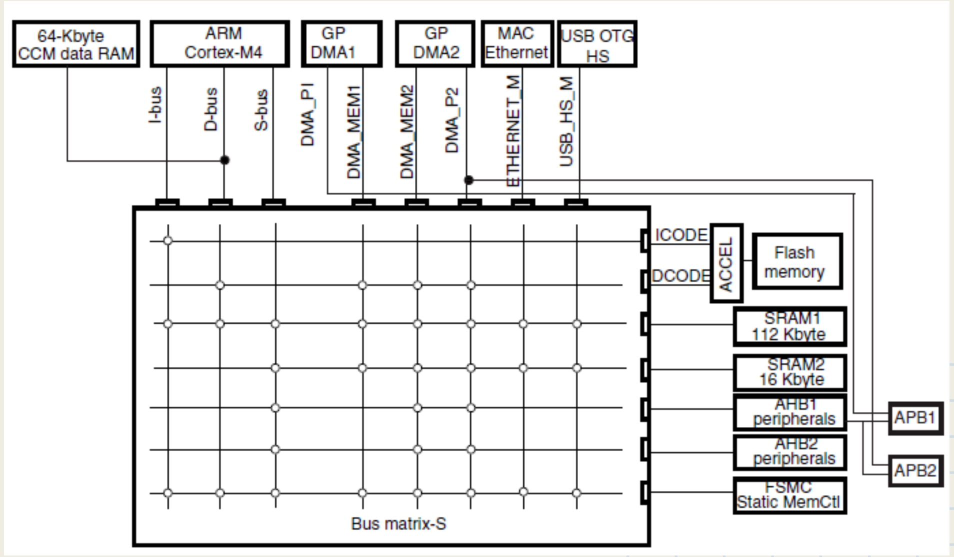


- Not everything of this space is available from user's application
  - Code
  - Internal SRAM
  - External SRAM
  - Peripherals
  - External Device
- Every peripheral is mapped into a specific address
- Depending of MSB bits of address bus arbiter selects which memory should be read/write



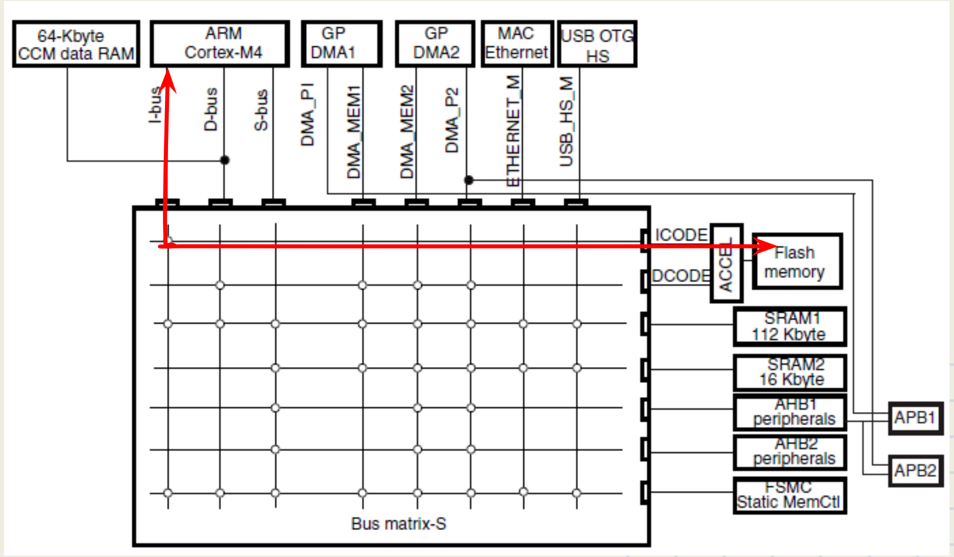
# Multi AHB bus matrix

- Using Bus matrix multiple bus master can perform simultaneously operation on different memory.



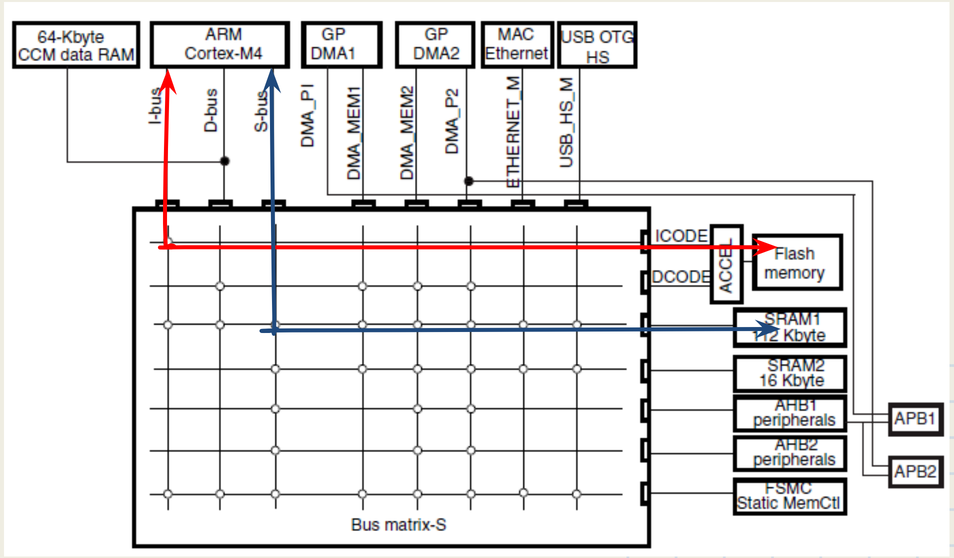
# Multi AHB bus matrix

- Using Bus matrix multiple bus master can perform simultaneously operation on different memory.
- For example in the same clock cycle:
  - CPU can fetch instruction from Flash



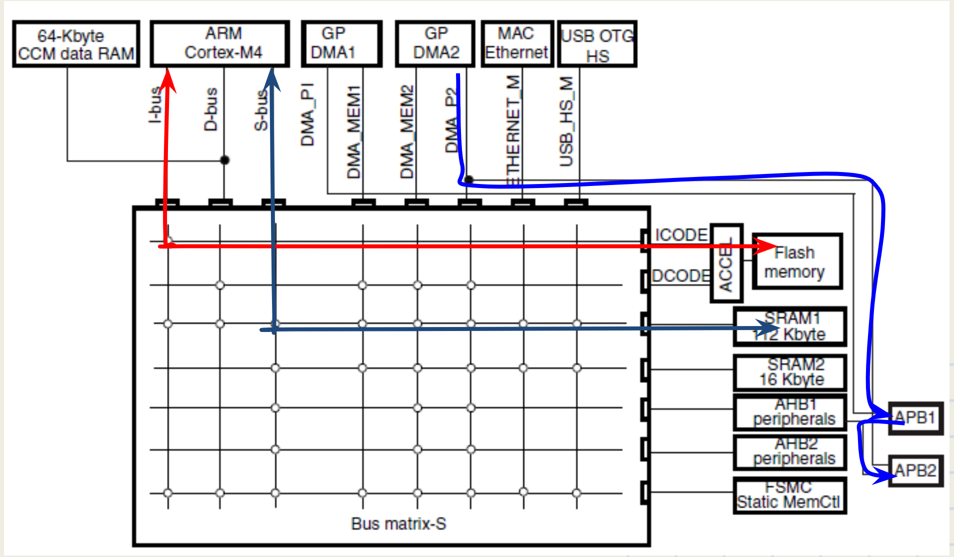
# Multi AHB bus matrix

- Using Bus matrix multiple bus master can perform simultaneously operation on different memory.
- For example in the same clock cycle:
    - CPU can fetch instruction from Flash
    - CPU can read/write data from SRAM



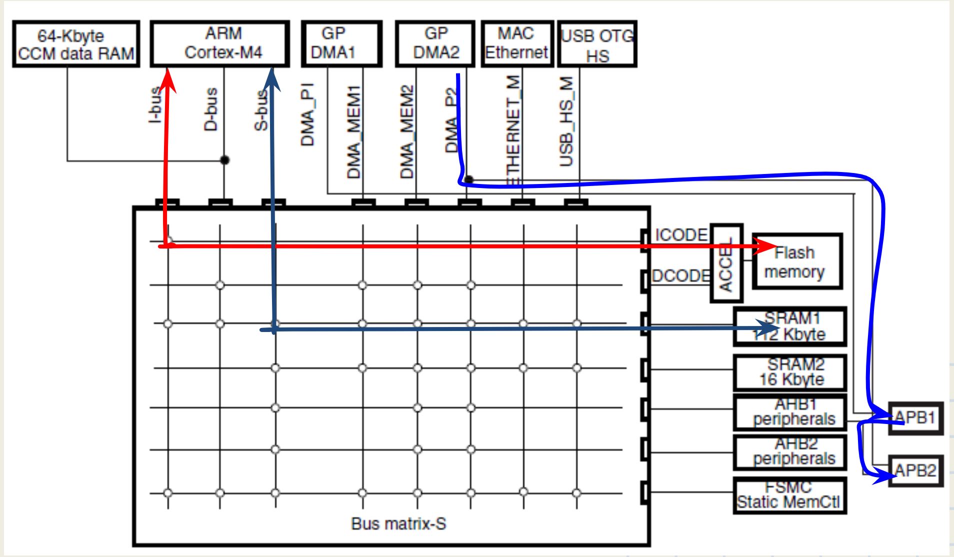
# Multi AHB bus matrix

- Using Bus matrix multiple bus master can perform simultaneously operation on different memory.
- For example in the same clock cycle:
  - CPU can fetch instruction from Flash
  - CPU can read/write data from SRAM
  - DMA can transfer data from two different peripherals



# Multi AHB bus matrix

- Using Bus matrix multiple bus master can perform simultaneously operation on different memory.
- For example in the same clock cycle:
    - CPU can fetch instruction from Flash
    - CPU can read/write data from SRAM
    - DMA can transfer data from two different peripherals



Multi AHB bus matrix automatically manages access arbitration between masters

# Flash memory latency

- Flash memory is perfect to store non-volatile data like program's code, static data etc… but is typically much slower than SRAM!
- CPU can't fetch instruction before Flash latency time.
  - Es: $f_{clk}$ = 168 MHz, $T_{clk}$ ~ 6 ns
  - Flash latency in clock cycle = floor($T_{r,flash}$ / $T_{clk}$)

- Flash latency time depending on
  - Supply voltage: low voltage increase latency time
  - Flash size: big flash has bigger parasitic capacitance
    - remember delay in CMOS:

$$T_{delay} \sim R_{on\text{-}mosfet} * C_p$$

    - Second order effect, voltage scaling is more important

# Wait states

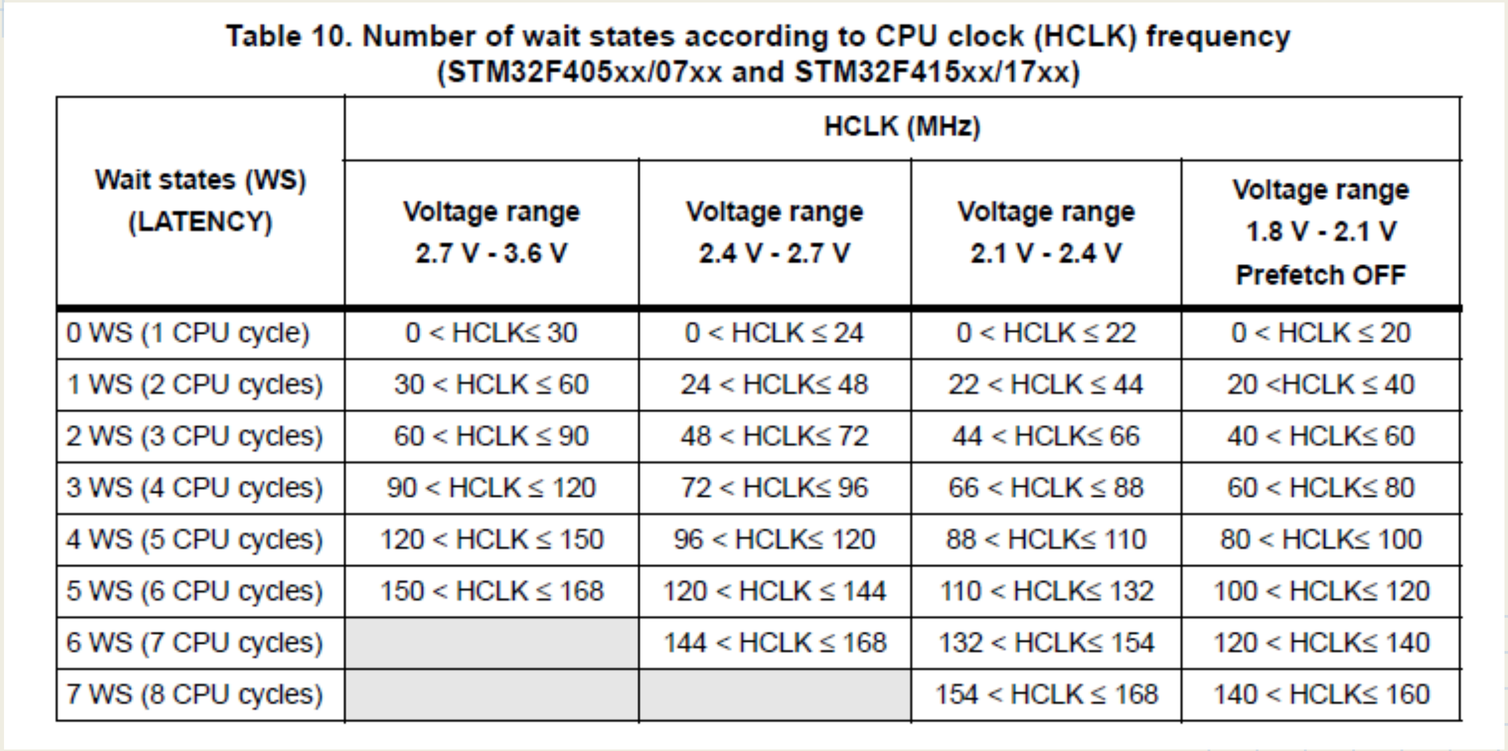
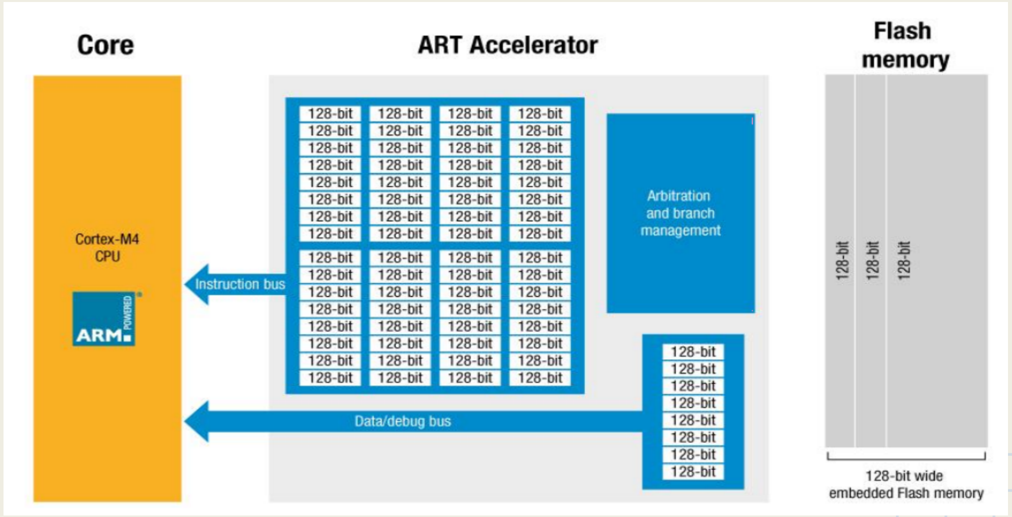


Table 10. Number of wait states according to CPU clock (HCLK) frequency (STM32F405xx/07xx and STM32F415xx/17xx)

| Wait states (WS) (LATENCY) | HCLK (MHz) | | | |
|---|---|---|---|---|
| | Voltage range 2.7 V - 3.6 V | Voltage range 2.4 V - 2.7 V | Voltage range 2.1 V - 2.4 V | Voltage range 1.8 V - 2.1 V Prefetch OFF |
| 0 WS (1 CPU cycle) | 0 < HCLK≤ 30 | 0 < HCLK ≤ 24 | 0 < HCLK ≤ 22 | 0 < HCLK ≤ 20 |
| 1 WS (2 CPU cycles) | 30 < HCLK ≤ 60 | 24 < HCLK≤ 48 | 22 < HCLK ≤ 44 | 20 <HCLK ≤ 40 |
| 2 WS (3 CPU cycles) | 60 < HCLK ≤ 90 | 48 < HCLK≤ 72 | 44 < HCLK≤ 66 | 40 < HCLK≤ 60 |
| 3 WS (4 CPU cycles) | 90 < HCLK ≤ 120 | 72 < HCLK≤ 96 | 66 < HCLK ≤ 88 | 60 < HCLK≤ 80 |
| 4 WS (5 CPU cycles) | 120 < HCLK ≤ 150 | 96 < HCLK≤ 120 | 88 < HCLK≤ 110 | 80 < HCLK≤ 100 |
| 5 WS (6 CPU cycles) | 150 < HCLK ≤ 168 | 120 < HCLK ≤ 144 | 110 < HCLK≤ 132 | 100 < HCLK≤ 120 |
| 6 WS (7 CPU cycles) | | 144 < HCLK ≤ 168 | 132 < HCLK≤ 154 | 120 < HCLK≤ 140 |
| 7 WS (8 CPU cycles) | | | 154 < HCLK ≤ 168 | 140 < HCLK≤ 160 |

# ART Accelerator

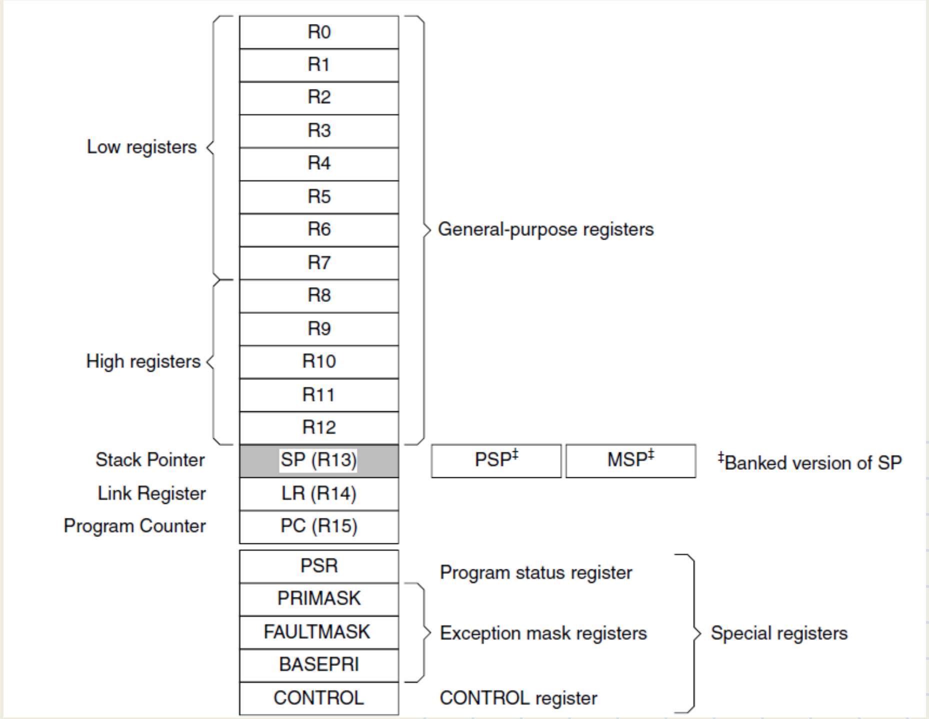
- Adaptive Real-Time memory Accelerator: a proprietary (STM) hardware block designed to reduce wait states in flash read access.
- Flash memory is organised in 128-bit blocks, up to 8 instruction-block using Thumb instruction set (16 bit x instr.)



- ART works like a cache memory. Using prefetch of next instruction theoretically it reduces to 0 wait states in sequential statements.
- For non-sequential instruction (like branch, call etc…) the penalty in terms of number of cycles is at <u>least equal to the number of wait states</u>
- A dynamic branch prediction system is used to improve performances

# Cortex M4 Register Map

- 13 General purpose register
  - Attention! in order to use 16 bit instruction, not all GP register can be used for all instruction



# Example: ADD/SUBTRACT

This instruction can do:

Rd = Rs ± Rn

Rd = Rs ± Offset3

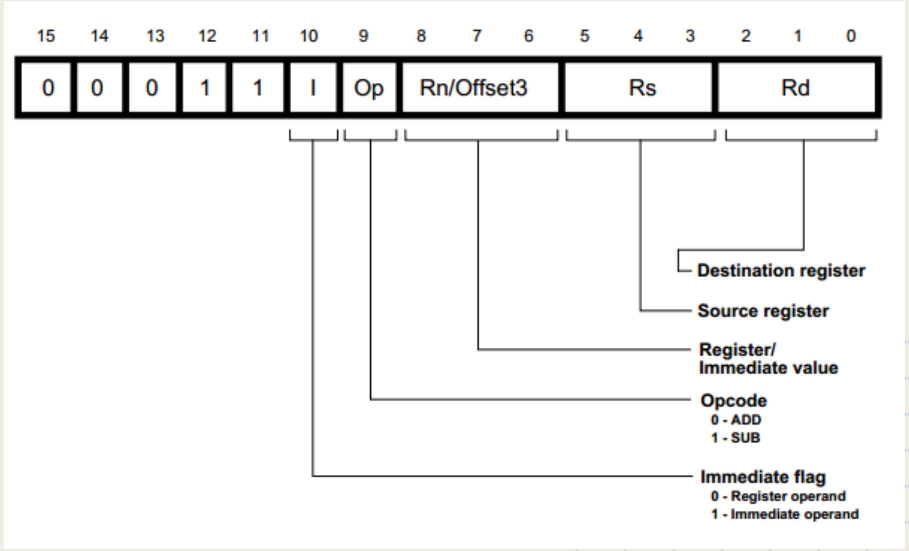
- 5 bit opcode
- flag for immediate operand, 3 bit
- 3 bit register, from R0 to R7

For more than 3 bit sum:

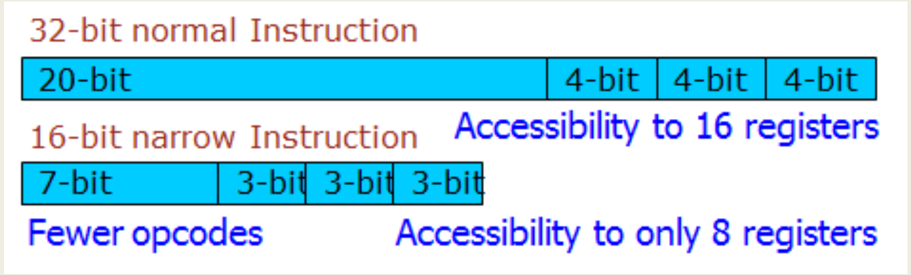
MOV Rn,#num

SUM Rd,Rs,Rn

- Same code size 32 bit Instruction set
- 2 clock cycle



# Thumb Instruction Set


32-bit normal Instruction
20-bit
4-bit
4-bit
4-bit
Accessibility to 16 registers
16-bit narrow Instruction
7-bit
3-bit
3-bit
3-bit
Fewer opcodes
Accessibility to only 8 registers

Thumb instructions in general requires more code than 32 bit RISC instruction set.

- More instruction for the same code that decreases performance of CPU

- Otherwise Thumb increases code density
    - Example: only 16 bit instruction size for simple increment like

      i++ => ADD R0,R0,#1

- Thumb code is typically 65% of ARM 32 bit code (infocenter.arm.com)

ARM7 and other high end processors can use both Thumb and ARM 32 bit instruction sets

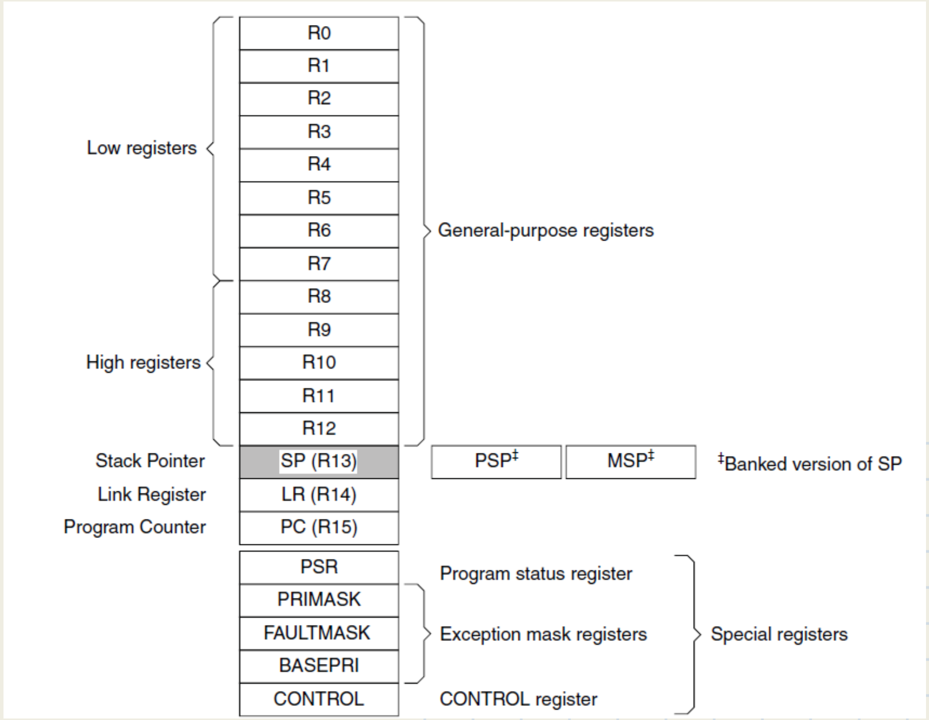
# Cortex M4 Register Map

- SP: Stack Pointer
- PC: Program Counter
- LR: Link Register, the register that contains return address from PC when *Branch and Link* (BL) instruction is executed

## Example:

```
.main
        BL    func          ; branch to func label
        …                   ; after BX CPU execute this instruction

.func
        …                   ; do stuffs
        BX LR               ; branch to LR register value
```



# Clock

- Three different clock sources can be used to drive the system clock
  - HSI oscillator clock (high-speed internal clock signal)
  - HSE oscillator clock (high-speed external clock signal)
  - PLL clock
- Device has also two secondary clock sources
  - 32 kHz low-speed internal clock
  - 32.768 kHz low speed external crystal
    - This two clock can be used to drive Real-Time Clock
    - $2^{15} = 32768$, so if this clock feed a 16 bit timer his MSB toggle every second

# External Clock

Figure 15. HSE external clock

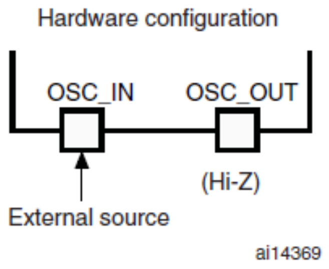
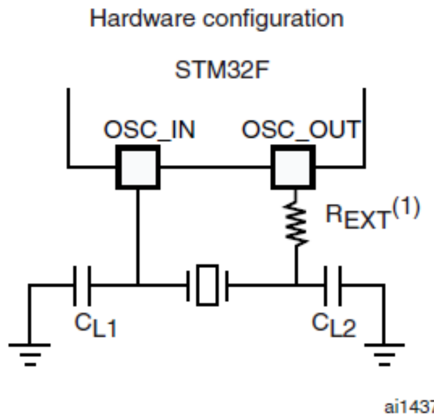
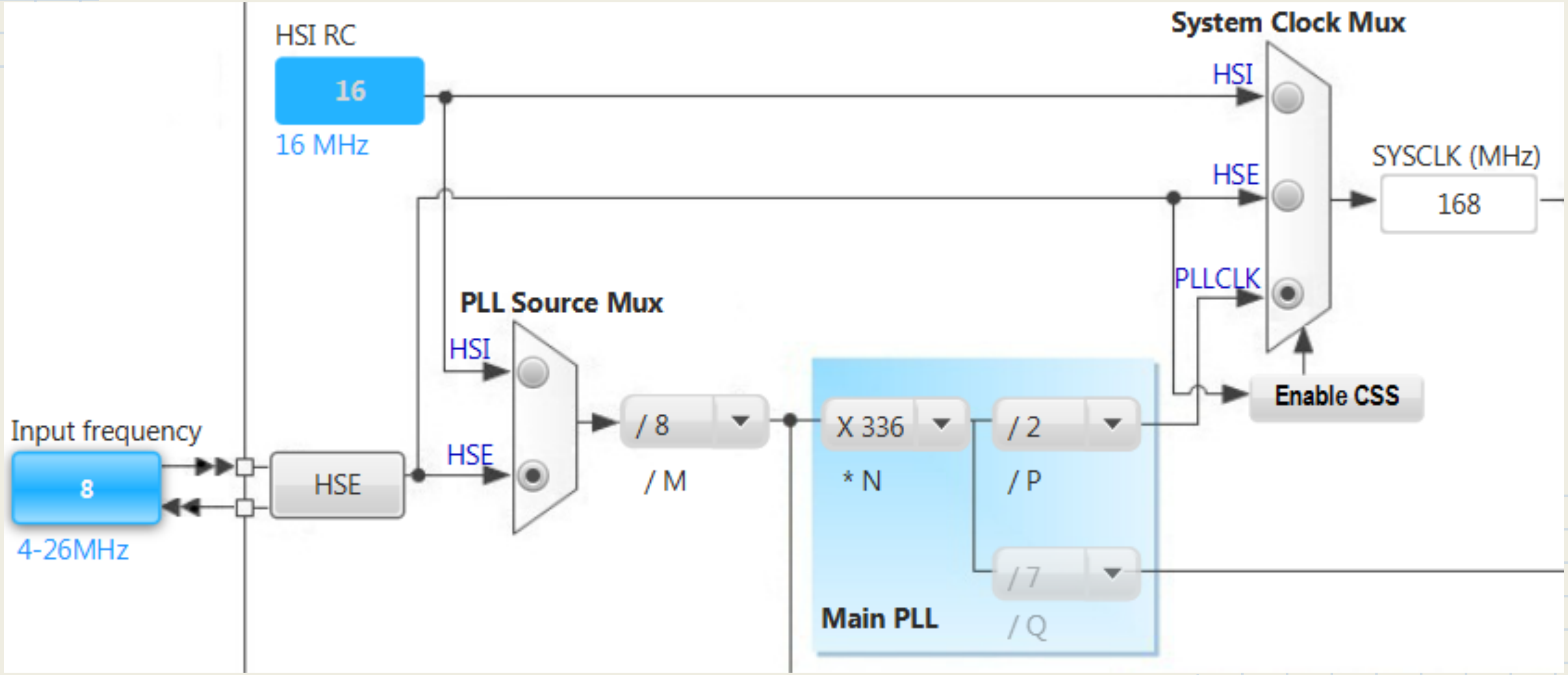


Figure 16. HSE crystal/ceramic resonators

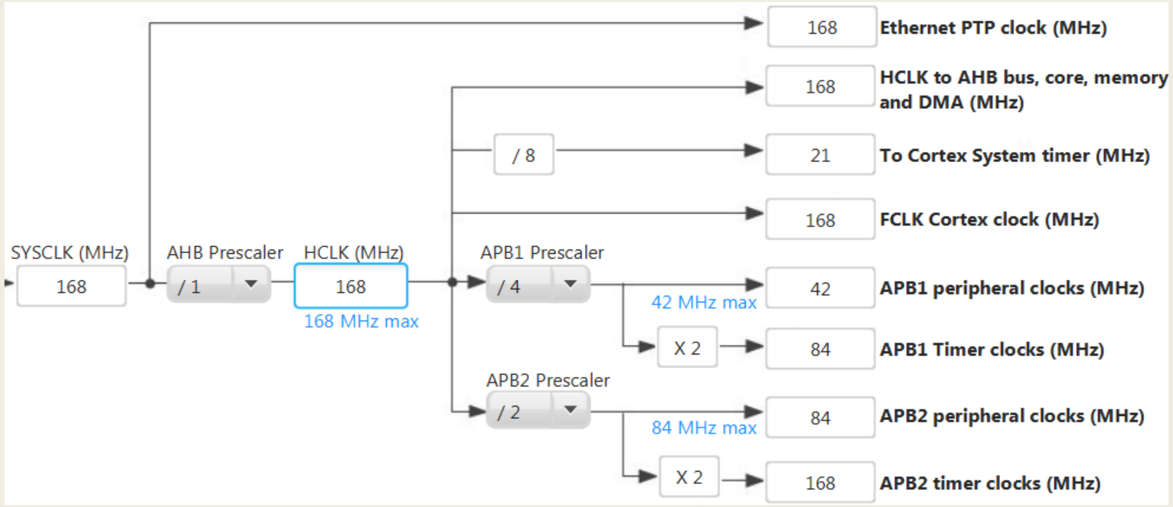


- With crystal/ceramic resonator in range from 4 to 26 MHz
- Using an external source. It can have a frequency from 1 to 50 MHz

# Select System Clock



# Other derived clock



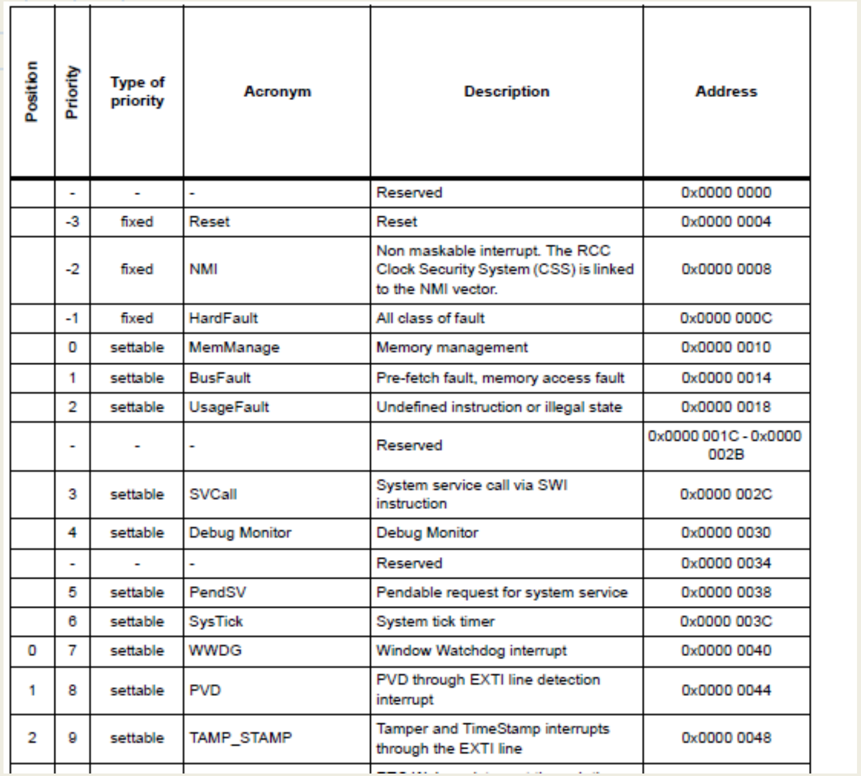
- APB1 and APB2 are peripheral lower speed buses
- Usually peripherals don’t need to go at maximum speed, so decrease clock frequency can save power.
- Prescalers can be changed runtime, like clock frequency

# Nested Vectored Interrupt Controller

- Nested Vectored Interrupt Controller (NVIC) provides configurable interrupt handling abilities to the processor.
  - Up to 91 maskable interrupt
  - Facilitates low-latency interrupt handling
  - Controls power management

- Three levels of priority:
  - Interrupt Number
  - Preempt Priority
  - Subpriority

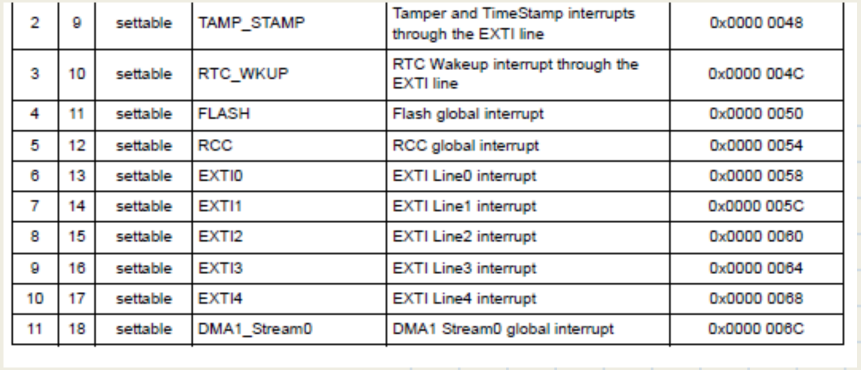
# Position Number



| Position | Priority | Type of priority | Acronym | Description | Address |
|---|---|---|---|---|---|
| | - | - | - | Reserved | 0x0000 0000 |
| | -3 | fixed | Reset | Reset | 0x0000 0004 |
| | -2 | fixed | NMI | Non maskable interrupt. The RCC Clock Security System (CSS) is linked to the NMI vector. | 0x0000 0008 |
| | -1 | fixed | HardFault | All class of fault | 0x0000 000C |
| | 0 | settable | MemManage | Memory management | 0x0000 0010 |
| | 1 | settable | BusFault | Pre-fetch fault, memory access fault | 0x0000 0014 |
| | 2 | settable | UsageFault | Undefined instruction or illegal state | 0x0000 0018 |
| | - | - | - | Reserved | 0x0000 001C - 0x0000 002B |
| | 3 | settable | SVCall | System service call via SWI instruction | 0x0000 002C |
| | 4 | settable | Debug Monitor | Debug Monitor | 0x0000 0030 |
| | - | - | - | Reserved | 0x0000 0034 |
| | 5 | settable | PendSV | Pendable request for system service | 0x0000 0038 |
| | 6 | settable | SysTick | System tick timer | 0x0000 003C |
| 0 | 7 | settable | WWDG | Window Watchdog interrupt | 0x0000 0040 |
| 1 | 8 | settable | PVD | PVD through EXTI line detection interrupt | 0x0000 0044 |
| 2 | 9 | settable | TAMP_STAMP | Tamper and TimeStamp interrupts through the EXTI line | 0x0000 0048 |

Position number is fixed and set by design

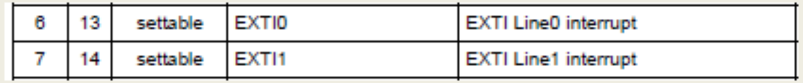
Rule: interrupt handler with lower priority/position number is prioritary to others



| | | | | | |
|---|---|---|---|---|---|
| 2 | 9 | settable | TAMP_STAMP | Tamper and TimeStamp interrupts through the EXTI line | 0x0000 0048 |
| 3 | 10 | settable | RTC_WKUP | RTC Wakeup interrupt through the EXTI line | 0x0000 004C |
| 4 | 11 | settable | FLASH | Flash global interrupt | 0x0000 0050 |
| 5 | 12 | settable | RCC | RCC global interrupt | 0x0000 0054 |
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt | 0x0000 0058 |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt | 0x0000 005C |
| 8 | 15 | settable | EXTI2 | EXTI Line2 interrupt | 0x0000 0060 |
| 9 | 16 | settable | EXTI3 | EXTI Line3 interrupt | 0x0000 0064 |
| 10 | 17 | settable | EXTI4 | EXTI Line4 interrupt | 0x0000 0068 |
| 11 | 18 | settable | DMA1_Stream0 | DMA1 Stream0 global interrupt | 0x0000 006C |

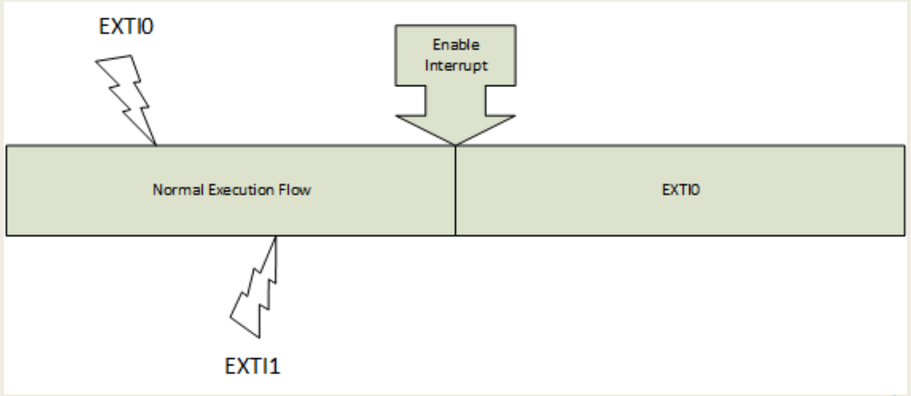
# How interrupt priority works

- It’s easy to understand using examples:
  - EXTI0 number = 13, EXTI1 number = 14



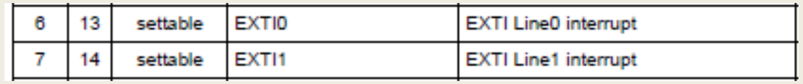
| | | | | |
|---|---|---|---|---|
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt |

Case 1: EXTI0 and EXTI1 pending, same preemption and subpriority



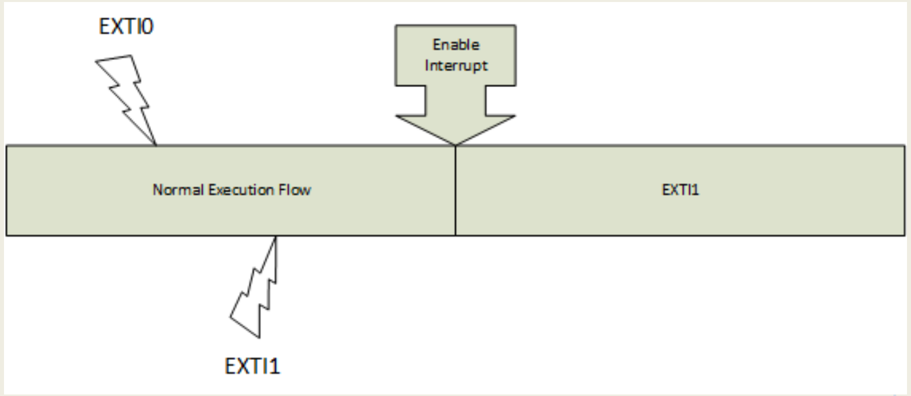
CPU executes highest priority interrupt

# How interrupt priority works



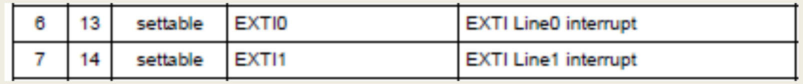
| | | | | |
|---|---|---|---|---|
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt |

Case 2: EXTI0 and EXTI1 pending, same preemption and but different subprioroty:   1 for EXTI0 and 0 for EXTI1



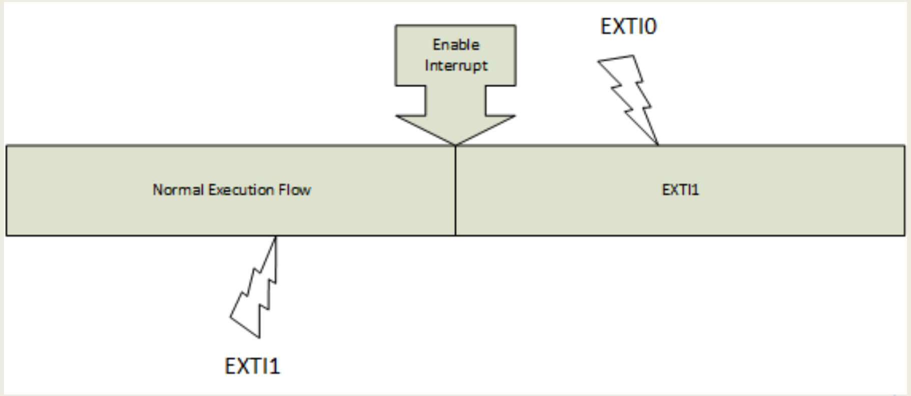
User can change priority interrupt using subpriority

# How interrupt priority works



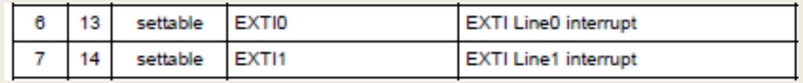
| | | | | |
|---|---|---|---|---|
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt |

Case 3: EXTI0 interrupt request while EXTI1 is running, same preemption and subpriority



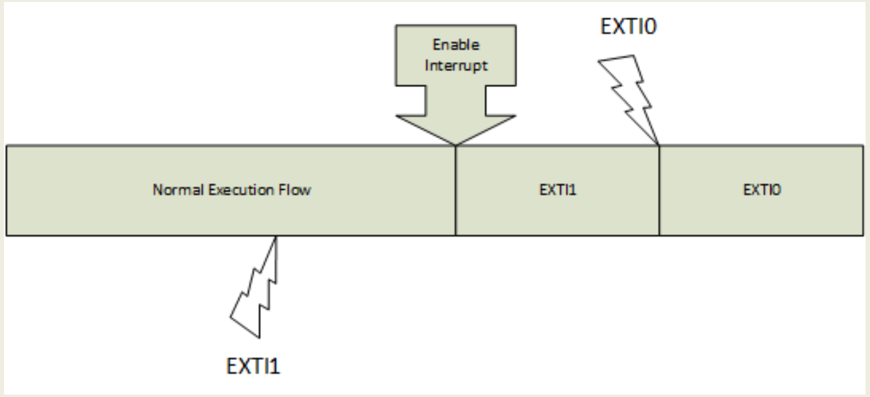
Interrupt service routine (ISR) can be preempted only from interrupt with higher preemption priority

# How interrupt priority works



| | | | | |
|---|---|---|---|---|
| 6 | 13 | settable | EXTI0 | EXTI Line0 interrupt |
| 7 | 14 | settable | EXTI1 | EXTI Line1 interrupt |

Case 4: EXTI0 interrupt request while EXTI1 is running, same subpriority but different preemption: 0 for EXTI0 and 1 for EXTI1



In that case EXTI0 has greater preemption priority than EXTI1

## Priority

- Stm32 has 4 bits of priority, 16 priority levels
- User can select how much of this 4 bits are for preemption priority and how much is for subpriority

- Example: 2 bits for preemption and 2 bits for subpriority
    - In this case we can have 4 levels of preempted ISR and 4 for subpriority
- Another Example: 0 bit for preemption and 4 for subpriority
    - No-preemptible ISR

## Interrupt Priority Summary

- The **preempt priority** level defines whether an interrupt can be serviced when the processor is already running another interrupt handler. In other words, preempt priority determines if one interrupt can preempt another

- The **subpriority** level value is used only when two exception with the same preempt priority level are pending. The exception with the lower subpriority will be handled first.

- The **position number** level value is used only when two exception with the same preempt priority and subpriority are pending. The exception with the lower position number will be handled first.

# Power Management

- Power consumption is one of the most important constraints in portable embedded device.
- STM32F4 provides several low-power mode
  - Sleep mode (CPU core stopped)
  - Stop mode (all clocks are stopped)
  - Standby mode (disable 1.2V power supply, lost of volatile data)
- In addition user can reduce power consumption in run mode
  - Slowing down system clock
  - Disable peripherals clock when they are unused

## Sleep Mode

| Sleep Mode | Description |
|---|---|
| Mode entry | WFI or WFE special instruction |
| Mode Exit | ● Interrupt if WFI was used<br>● Event if WFE was used |
| Latency | None |

- In sleep mode only CPU clock is stopped.
- To enter in this mode user can call special instruction
  - WFI (Wait for Interrupt) => wake CPU on any interrupt configured.
  - WFE (Wait for Event) => wake CPU on any event. Event is generated by EXTI peripherals (see documentation or TODO)

# Stop Mode

| Stop Mode | Description |
|---|---|
| Mode entry | WFI or WFE instruction plus configuration of some register |
| Mode Exit | ● EXTI lines configured in Interrupt mode if WFI was used<br>● EXTI lines configured in Interrupt mode if WFI was used |
| Latency | Oscillator startup time |

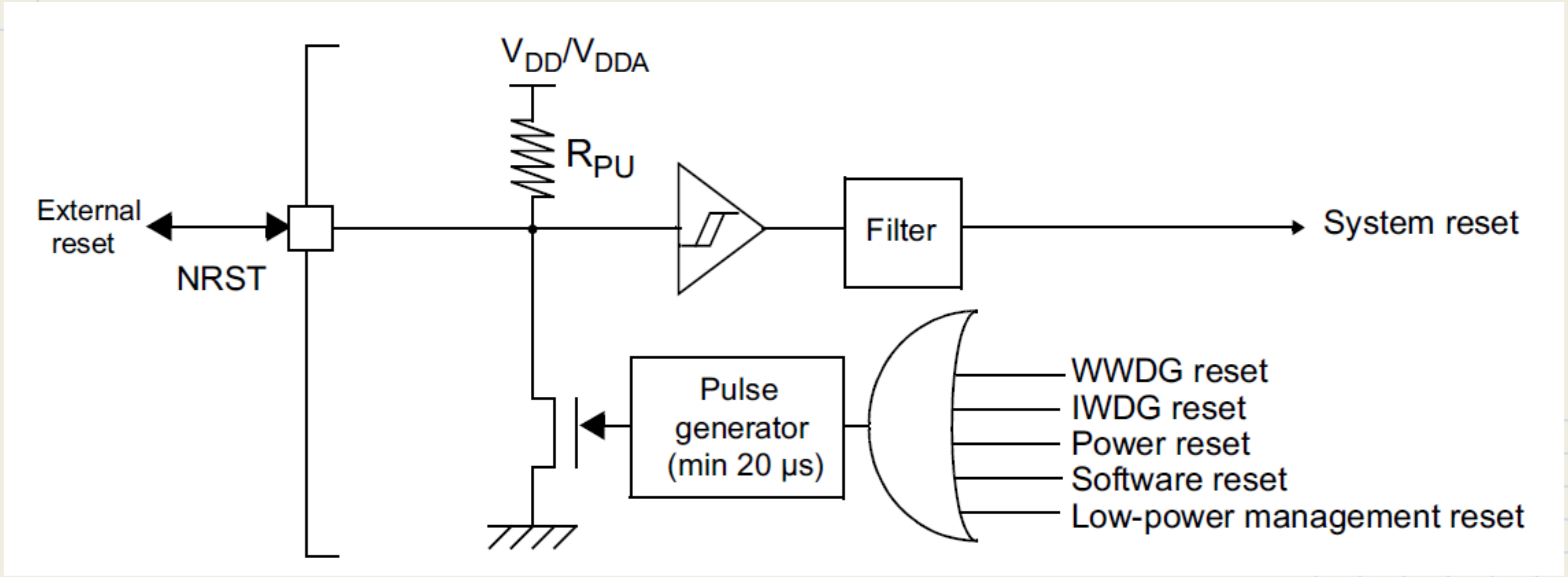
- In stop mode all clock are stopped. Voltage is still on
- User can decide to keep active few low consumption peripherals like RTC, Watchdog and low frequency oscillator (for RTC and Watchdog)

# Standby Mode

| Stop Mode | Description |
|---|---|
| Mode entry | WFI or WFE instruction plus configuration of some register |
| Mode Exit | Rising edge on WKUP pin, external reset, RTC or Watchdog reset |
| Latency | Oscillator startup time |

- In standby mode device disabled 1.2V regulator. All non volatile data was lost!!
- After waking up from standby, program execution restarts in the same way as after a Reset!
- User can decide to keep active few low consumption peripherals like RTC, Watchdog and low frequency oscillator (for RTC and Watchdog)

# Reset Circuit



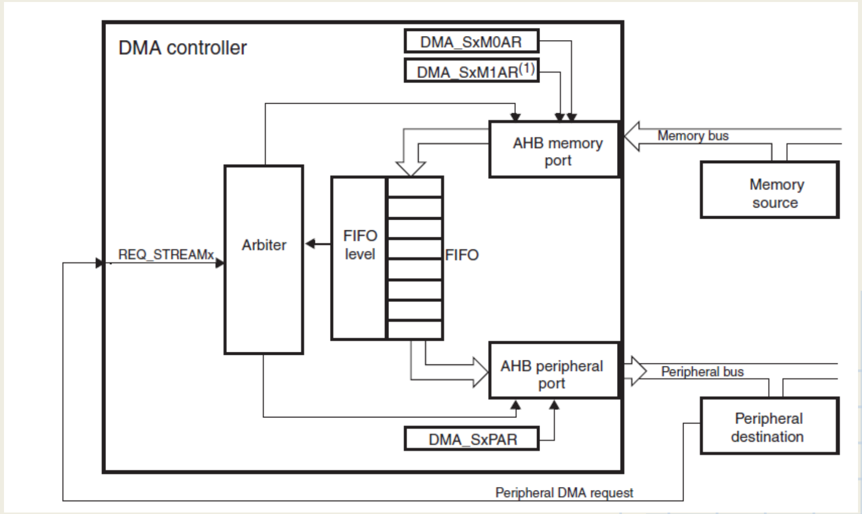
# DMA

- Direct memory Access(DMA) is used in order to provide high-speed data transfer between peripherals and memory and between memory and memory.
- Data can be quickly moved by DMA without any CPU action.
- This keeps CPU resources free for other operations

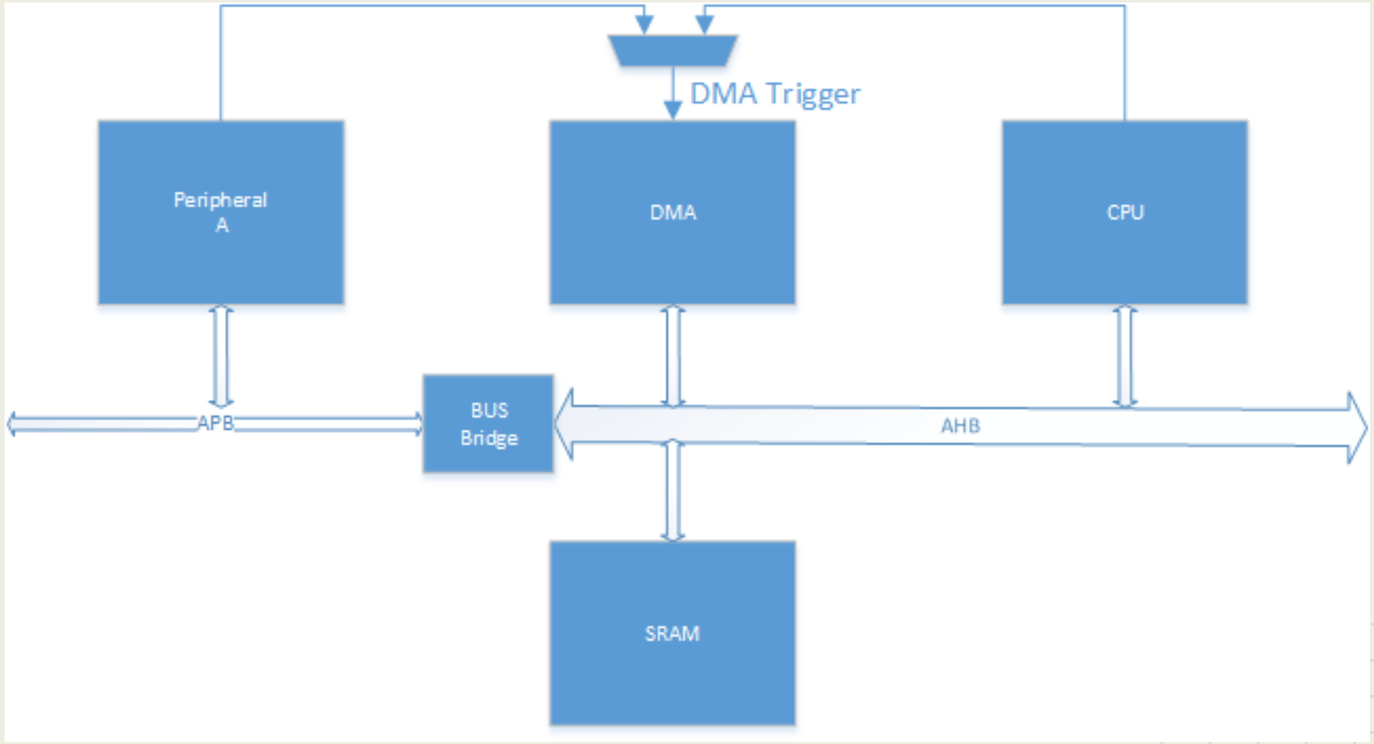
- Three modes:
  - Memory-to-Peripheral
  - Peripheral-to-Memory
  - Memory-to-Memory

# DMA

- DMA read data from Memory and place it to Peripheral destination address.
- It automatically increments source/destination address to perform multi-byte transfer, also in circular mode (i = (i+1)%N)
- 8 stream can be performed simultaneously, coordinated by a bus Arbiter depending on stream priority
- DMA transfer can be triggered both by software or peripherals.



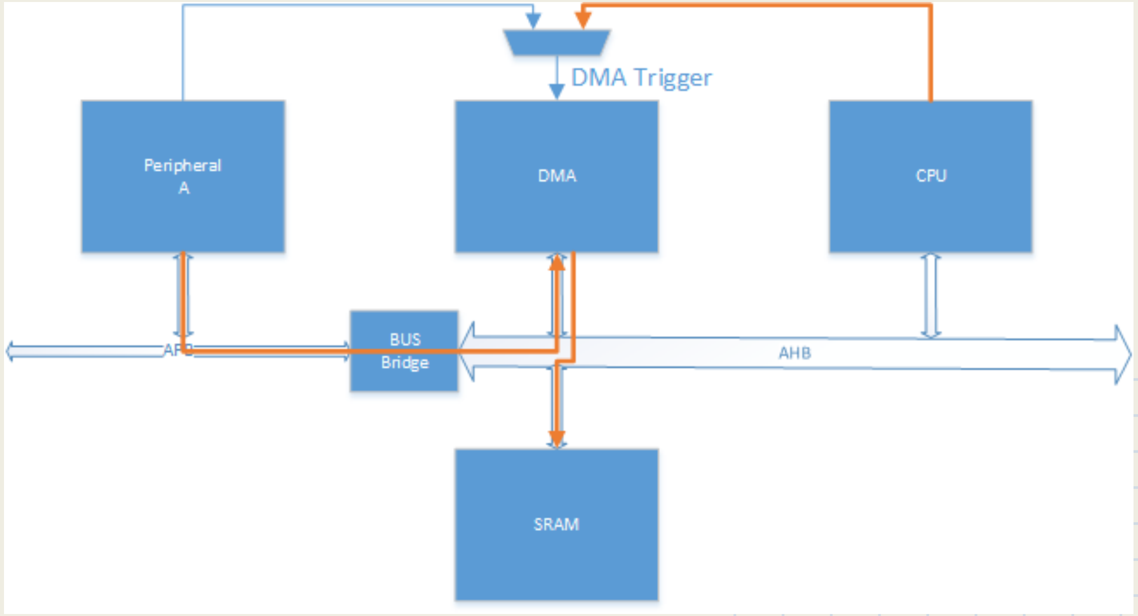
# DMA Example



# Peripheral-to-memory

Peripheral to memory transfer controlled by CPU

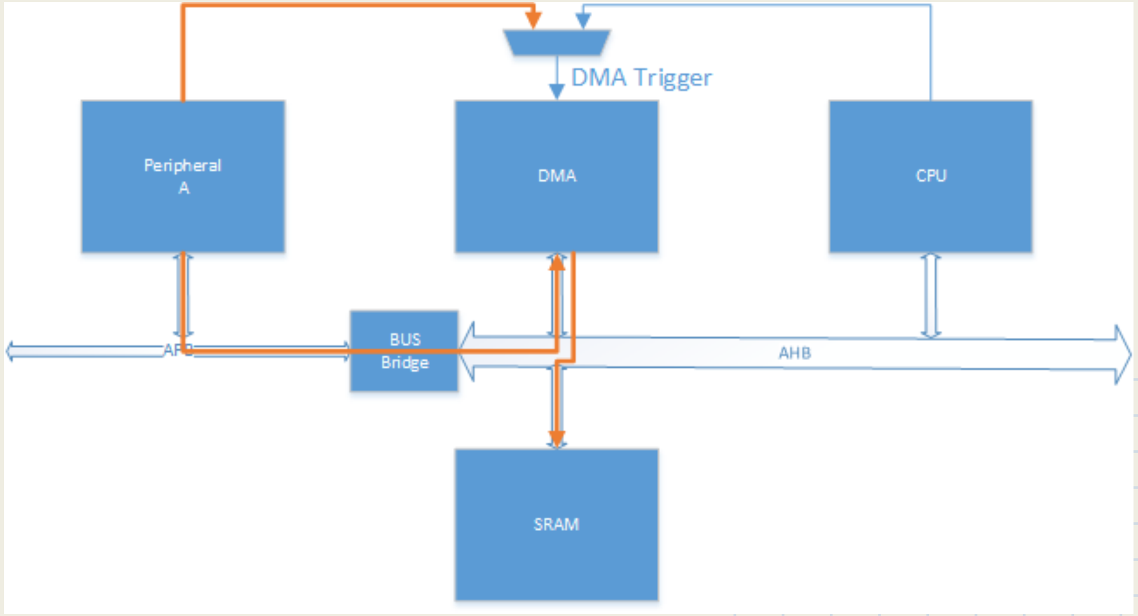
Example:
Transfer data array from SD card to memory



# Peripheral-to-memory

Peripheral to memory transfer controlled by peripheral

Example:
USB data transfer

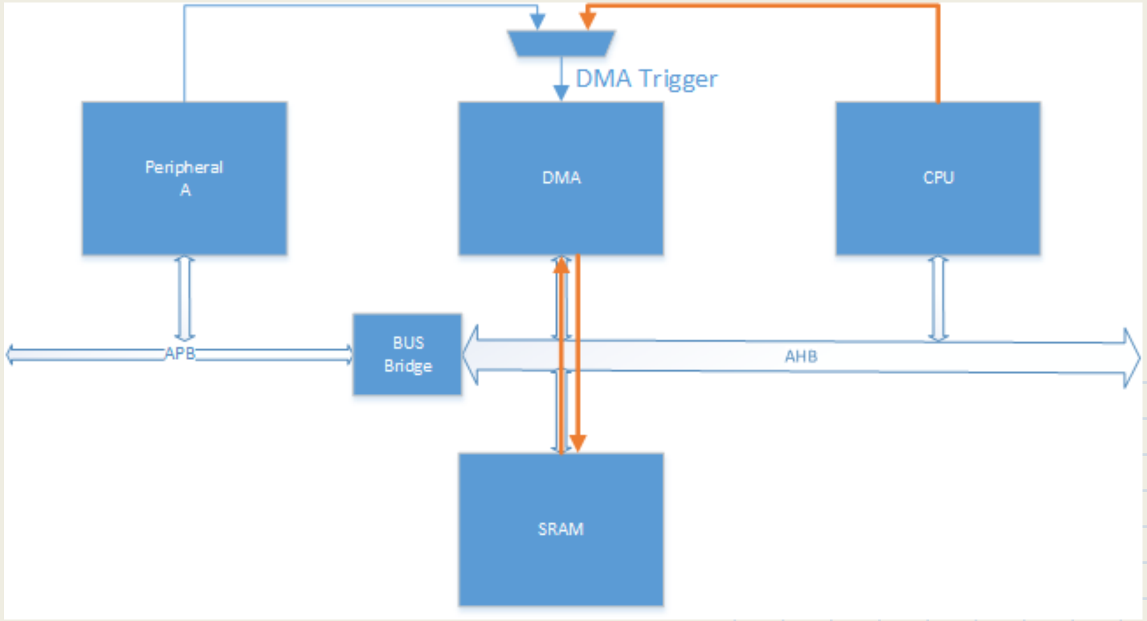


# Memory-to-memory

Memory to memory transfer.

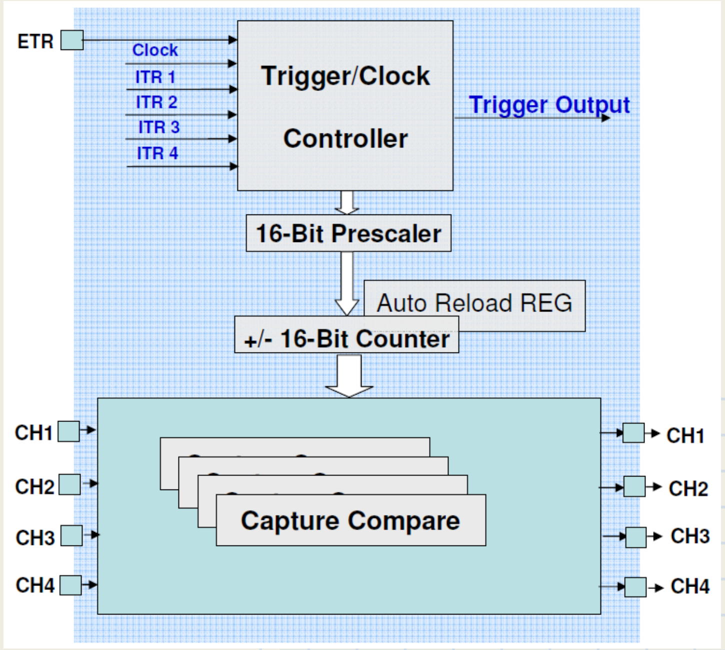
Example:
Non blocking memcpy

During memory transfer CPU is free to do other stuffs



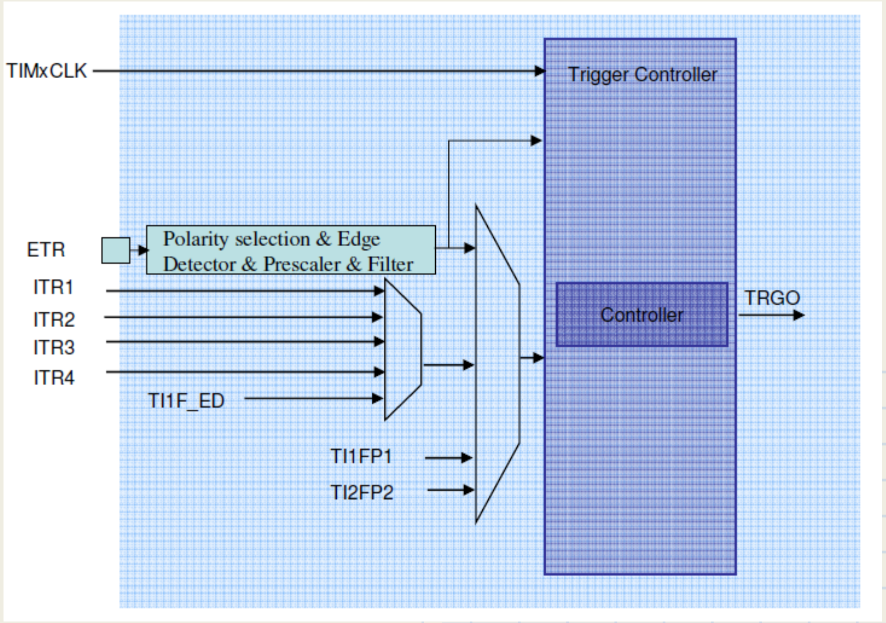
# Timer

- 16-bit Counter
- Up, down and centred counting modes
- Auto Reload
- 4 x 16-bit Capture/Compare Channels
  - Programmable channel direction: input/output
  - Input Capture, PWM Input Capture Modes
  - Output Compare, PWM, One Pulse Modes
- Independent IRQ/DMA Requests:
  - At each Update Event
  - At each Capture Compare Events
  - At each Input Trigger



# Clock Selection

- Clock can be selected from 2 sources
  - Internal clock TIMxCLK provided by the RCC
  - External pin ETR
- Timer Trigger can be
  - Internal trigger input 1 to 4:
    - ITR1 / ITR2 / ITR3 / ITR4
    - Using another timer as a prescaler
  - External Capture Compare pins
    - Pin 1: TI1FP1 or TI1F_ED
    - Pin 2: TI2FP2
  - External pin ETR

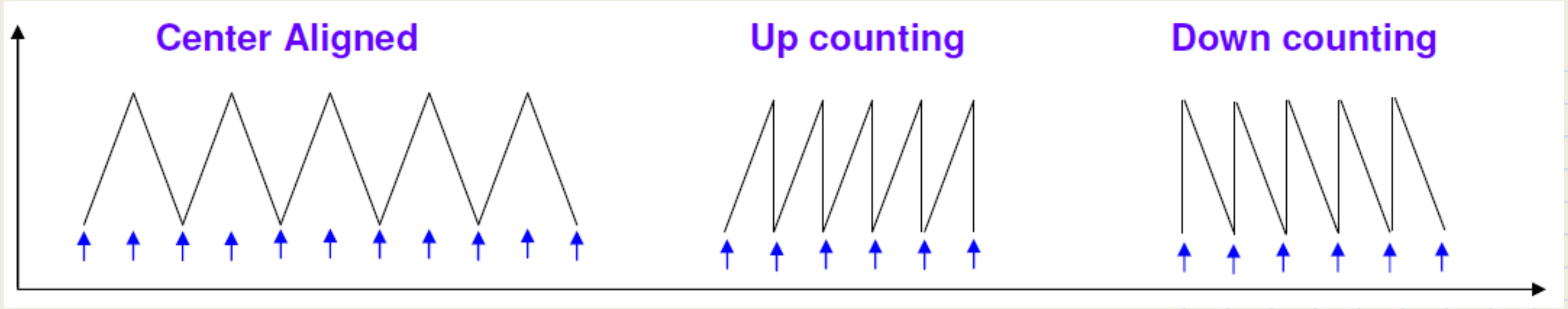


# Counting Modes

3 Counting Modes:

- Center Aligned
- Up counting
- Down Counting

All three mode have the same update-event interrupt period

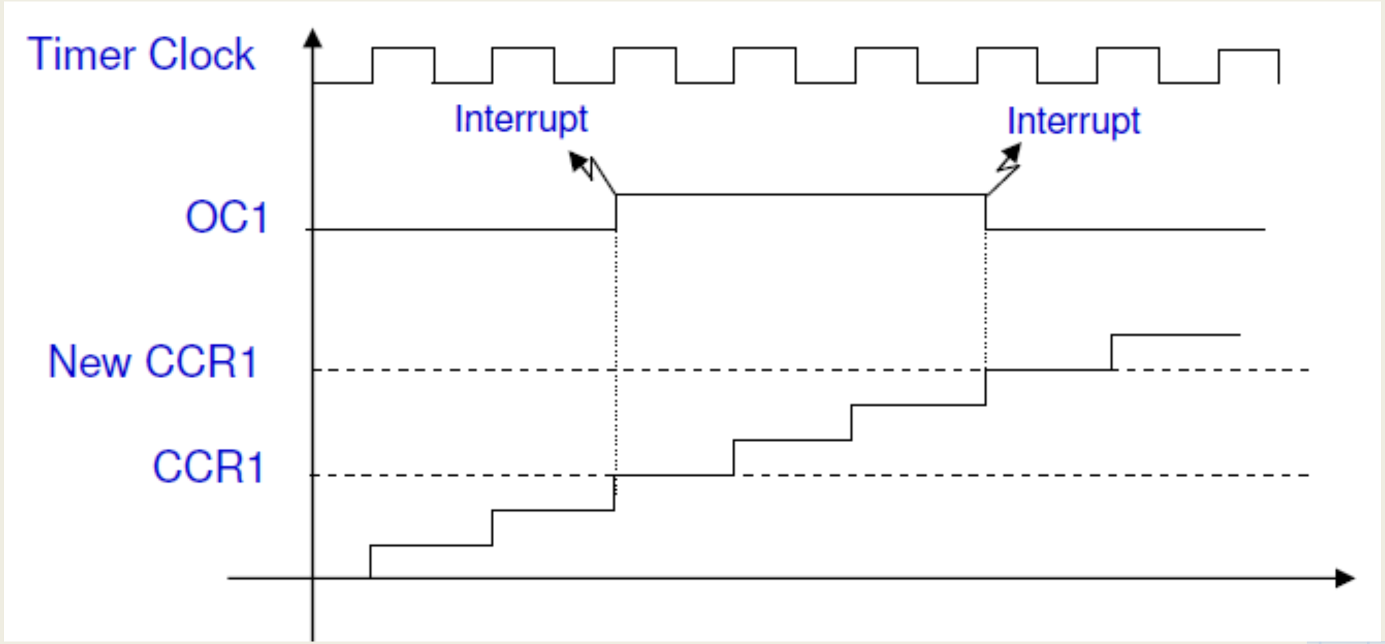


## Output Compare

The Output Compare is used to control an output waveform or indicate when a period of time has elapsed.

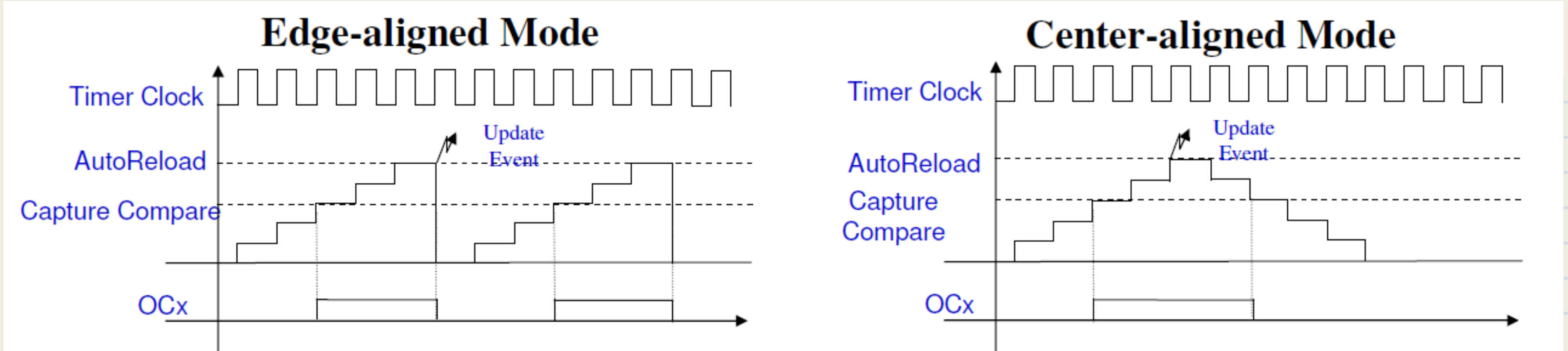
- When a match is found between the capture/compare register and the counter:
    - The corresponding output pin is assigned to the programmable Mode, it can be:
        - Set/Reset/Toggle
        - Remain unchanged
    - Generates an interrupt if the corresponding interrupt mask is set
    - Send a DMA request if the corresponding enable bit is set

# Output Compare



# PWM

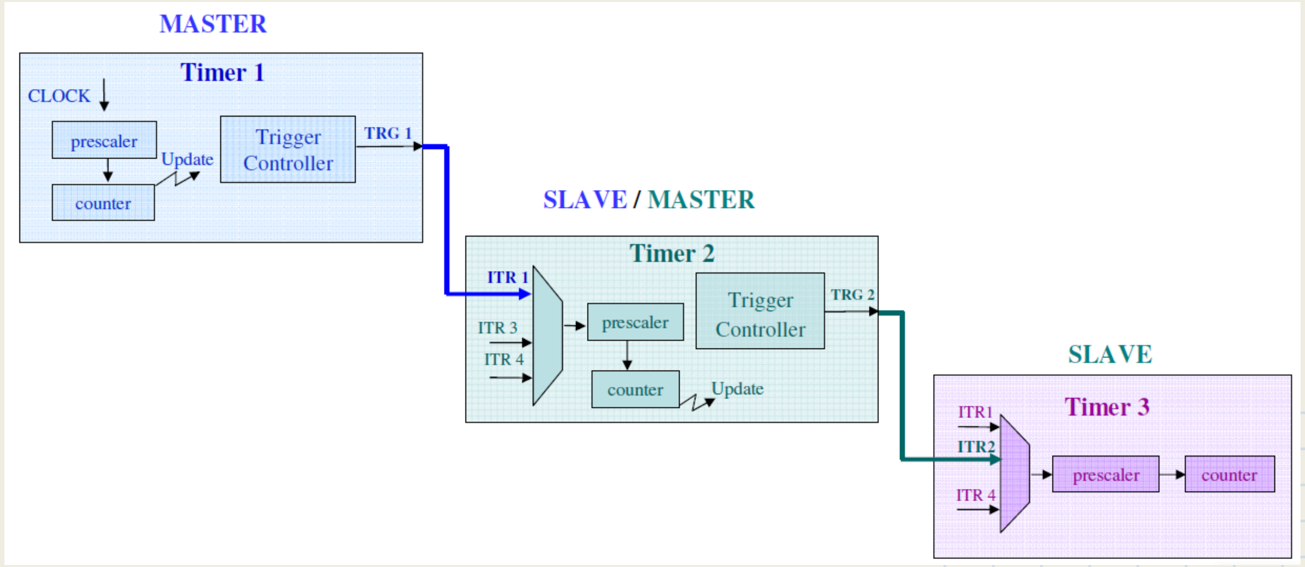
- The PWM mode allows to generate 4 independent signals.
- The frequency and a duty cycle determined as follow:
  - One auto-reload register to defined the PWM period.
  - Each PWM channel has a Capture Compare register to define the duty cycle.
- There are two configurable PWM modes:
  - Edge-aligned Mode
  - Center-aligned Mode



# Synchronization

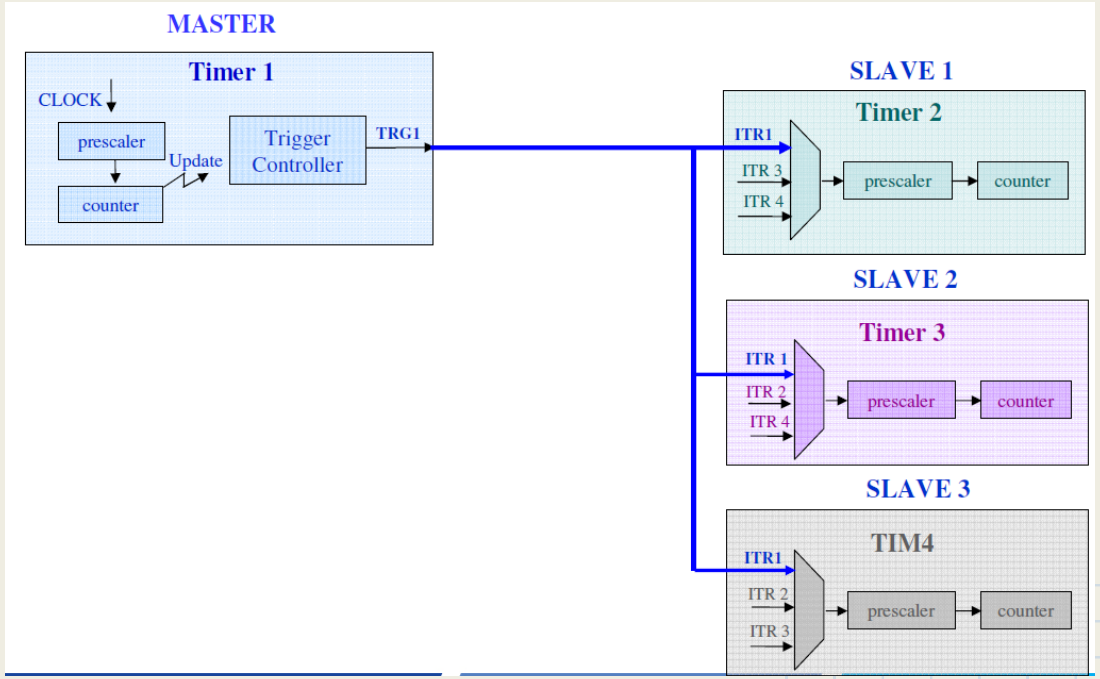
Timers can be linked together for synchronization pourposes

1) Cascade Mode: TIM1 used as master timer for TIM2, TIM2 configured as TIM1 slave and master for TIM3



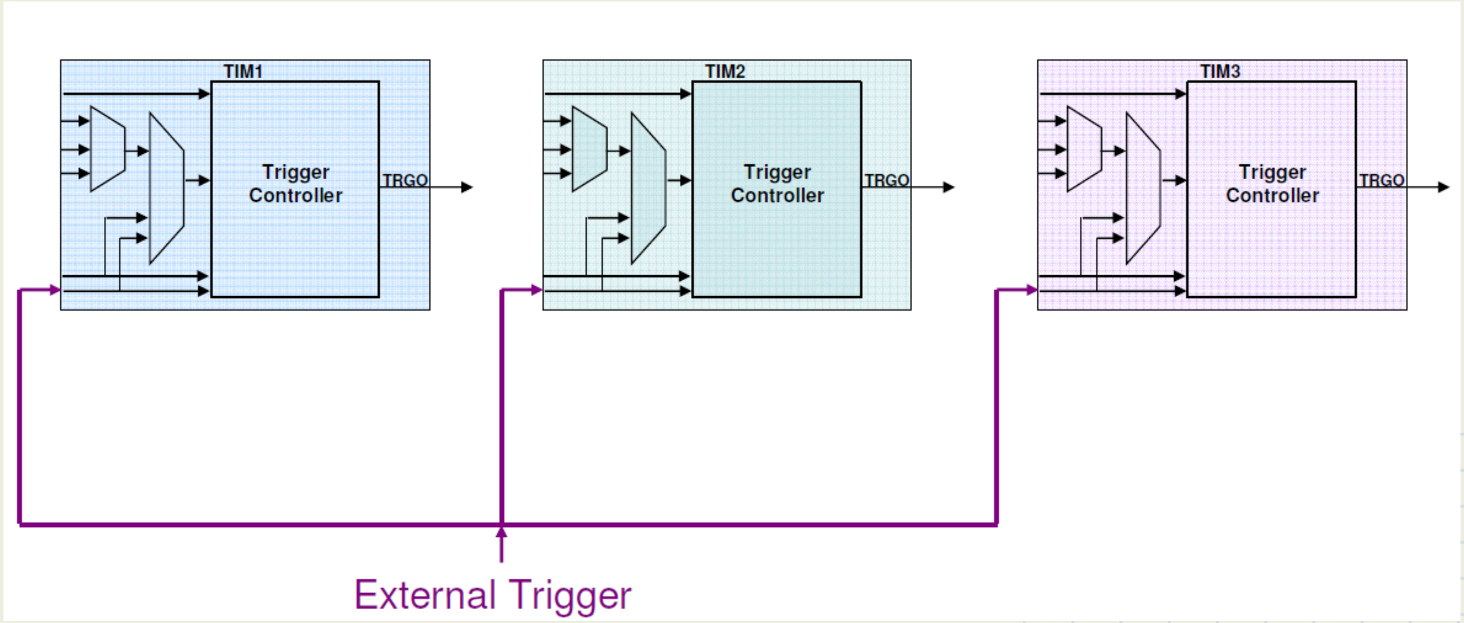
# Synchronization

2) One Master several slaves: TIM1 used as master for TIM2, TIM2 and TIM4.



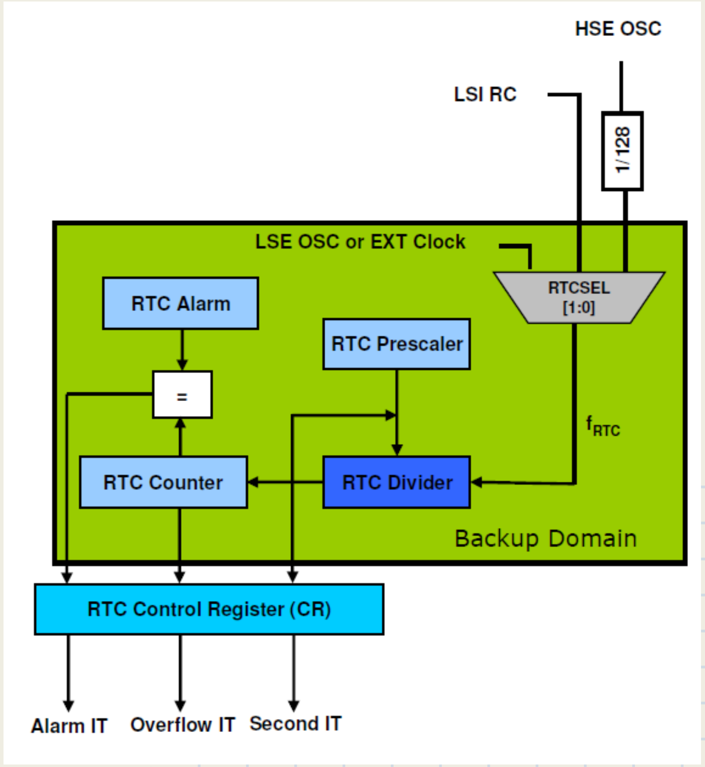
# Synchronization

3) Timers and external trigger synchronization: TIM1, TIM2 and TIM3 are slaves for an external signal connected to respective Timers inputs.



# RTC: Real Time Clock

- Clock sources
  - 32.768 kHz dedicated oscillator (LSE)
  - Low frequency (32kHz), low power internal RC(LSI)
  - HSE divided by 128
- 3 Event/Interrupt sources
  - Second
  - Overflow
  - Alarm (also connected to EXTI Line 17 for Auto Wake-Up from STOP)


HSE OSC
LSI RC
1/128
LSE OSC or EXT Clock
RTCSEL [1:0]
RTC Alarm
RTC Prescaler
=
$f_{RTC}$
RTC Counter
RTC Divider
Backup Domain
RTC Control Register (CR)
Alarm IT
Overflow IT
Second IT