Internet of Things

Marco Zecchini

Sapienza University of Rome,

zecchini@diag.uniroma1.it

Microcontrollers, its Components and ARM Mbed OS

Agenda

Section 1

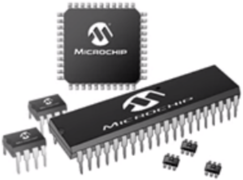
Introduction to Microcontroller

Embedded systems

An embedded system is a system whose main function is not computational but which is controlled by a computer embedded within it. It is designed to execute some *specific* task.

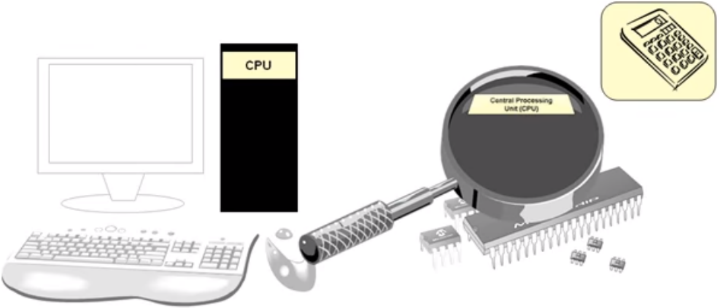
The main actor in embedded systems is the *microcontroller*.

What is a Microcontroller?

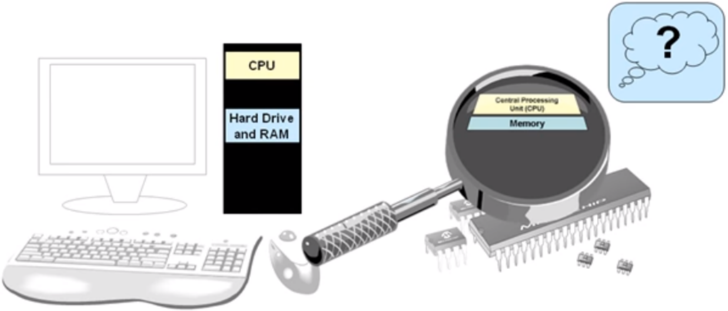


It is essentially a small computer on a chip.

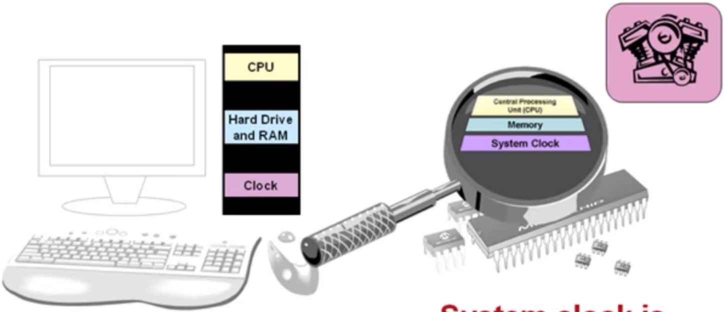
What is a Microcontroller?



What is a Microcontroller?

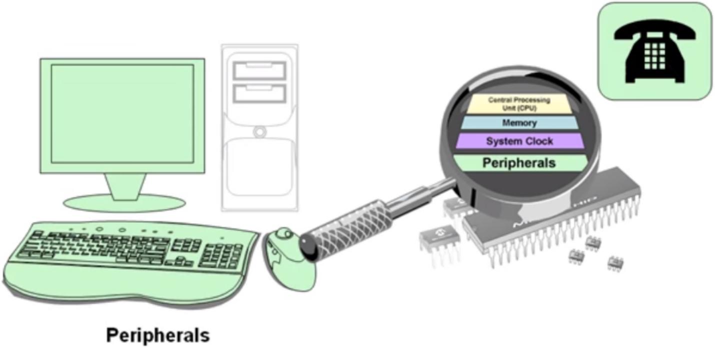


What is a Microcontroller?



System clock is derived from an Oscillator

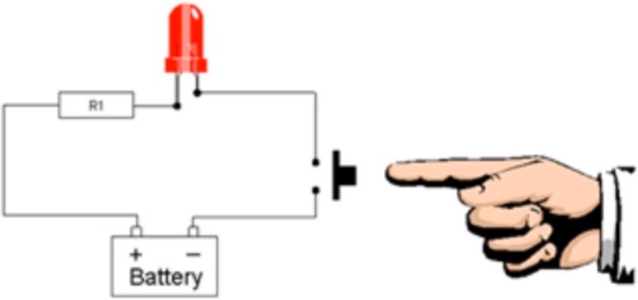
What is a Microcontroller?



Use cases

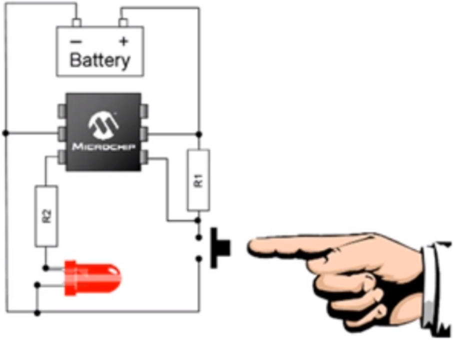
Where would I use a Microcontroller?

Anywhere I would like to add *intelligence* ...

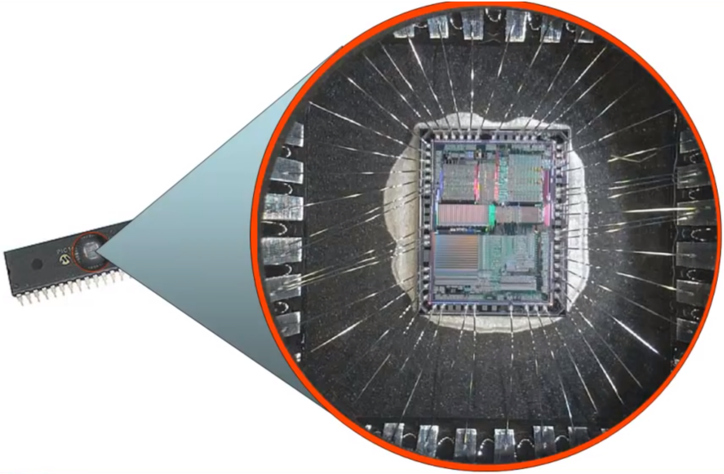


Where would I use a Microcontroller?

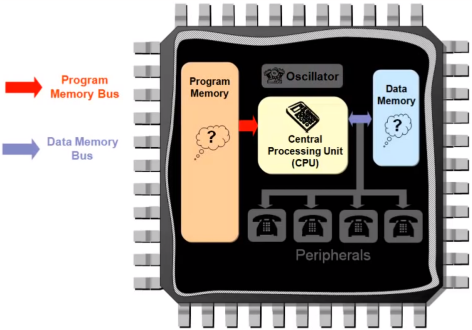
Anywhere I would like to add *intelligence* ...



How a MCU looks like



Harvard Architecture



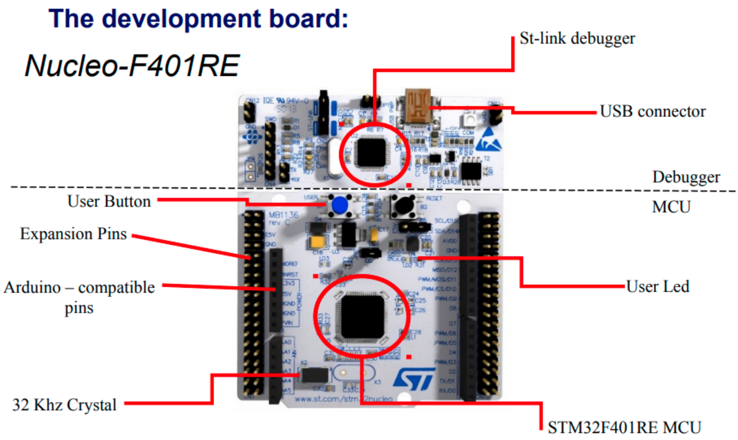
Von Neumann architecture has only one bus that is used for both instruction fetches and data transfers.

Harvard architecture allows to access data and program memory simultaneously.

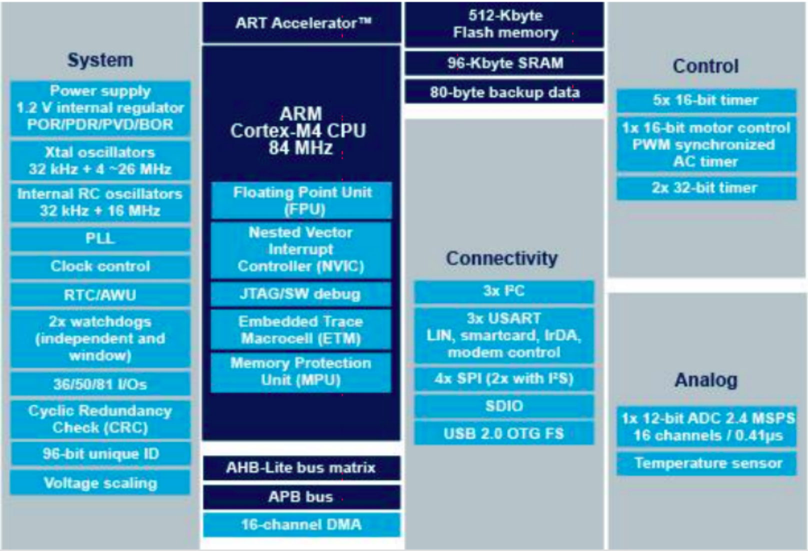
Section 2

ST Microelettronics board

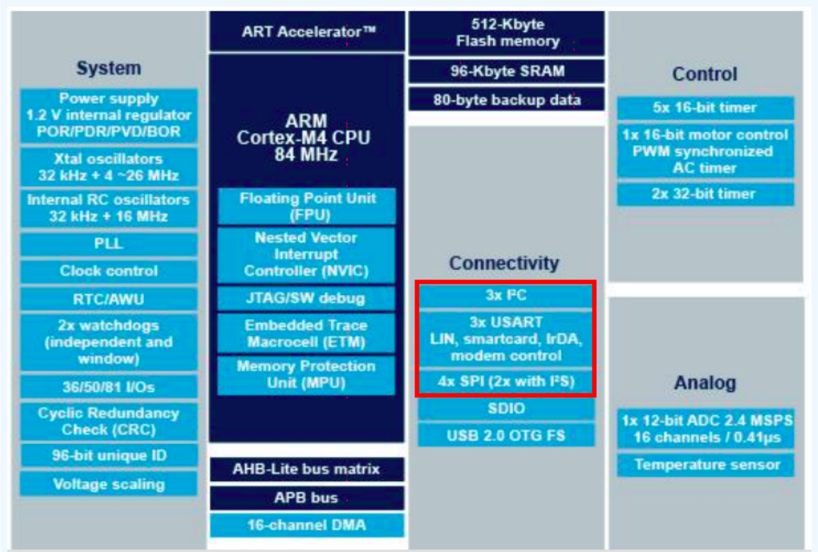
STMF401RE board



STMF401RE architecture

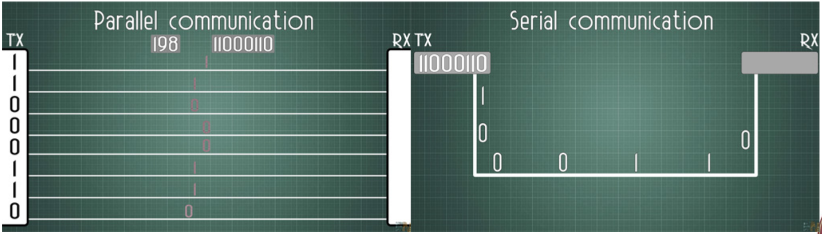


Connectivity



Serial communication

- In order to communicate with different external peripherals we use Serial Communication.
- In this kind of communication data is transferred serially (one after another) and not parallel (everything together).
- Fewer wires as compared to its parallel counterpart



Synchronicity and Bidirectional Communication

Serial communication can be categorized into:

- Synchronous and Asynchronous:
  - Synchronous serial communication: In this type of communication both transmitter and receiver share a common clock to remain in sync with each other.
  - Asynchronous serial communication: This type of serial communication does not require any common clock source between the transmitter and receiver
- Half-duplex and Full-duplex:
  - In a full-duplex system, both parties can communicate with each other simultaneously.
  - In a half-duplex system, both parties can communicate with each other, but not simultaneously; the communication is one direction at a time.

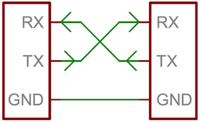
USART

Universal Synchronous and Asynchronous Receiver-Transmitter - is a type of a serial interface device that can be programmed to communicate asynchronously or synchronously.

We are interested in the asynchronous mode.

It is both half-duplex and full-duplex.

USART



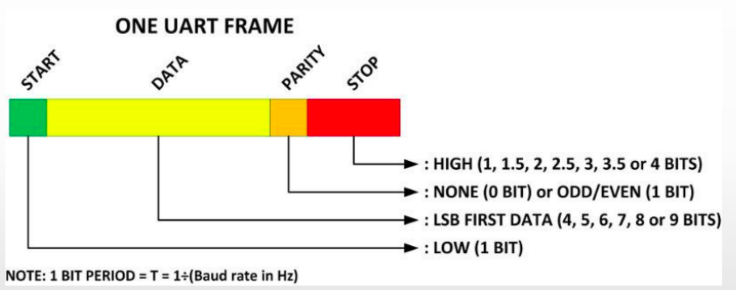
The basic bi-directional communication requires two lines:

- TX - Transmit
- RX - Receive

USART frame

Data is transferred within frames. A frame is composed by:

- START - single bit
- DATA - from 4 to 9 bits
- PARITY - from none to 1 bit
- STOP - from 1 to 4 bits



USART

Being asynchronous, we need to set extra parameters to allow RX and TX to communicate properly. These are:

- BAUD RATE (speed of data exchange)
- PARITY
- DATA SIZE
- STOP BITS

Example: 9600, 8, N, 1. Where:

- 9600 is the baud rate
- 8 is the number of bits in data field
- N is the parity, no parity used
- 1 is the number of stop bits

I^2C

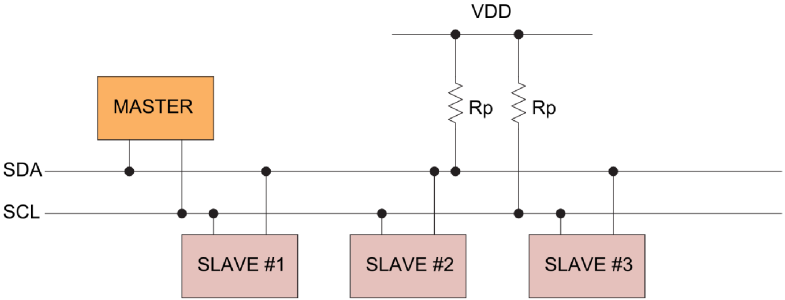
Inter-Integrated Circuit - is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus.

It is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed, like: EEPROMs, ADC / DAC, small displays, digital potentiometers, ...

I^2C

I2C uses only two bi-directional open-drain lines, pulled up with resistors:

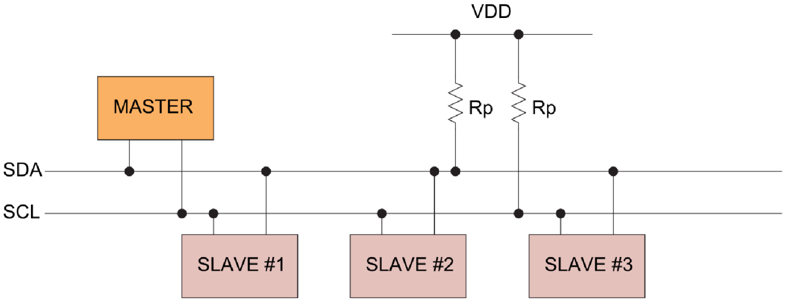
- Serial Data Line (SDA), used for sending and receiving data
- Serial Clock Line (SCL), used for synchronized the different devices



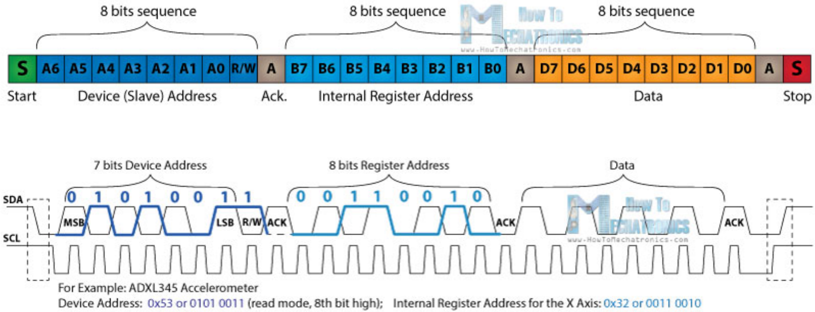
I^2C

I2C uses only two bi-directional open-drain lines, pulled up with resistors:

- Serial Data Line (SDA), used for sending and receiving data
- Serial Clock Line (SCL), used for synchronized the different devices



I^2C protocol



SPI

Serial Peripheral Interface - consists in a single master device and at least one slave device.

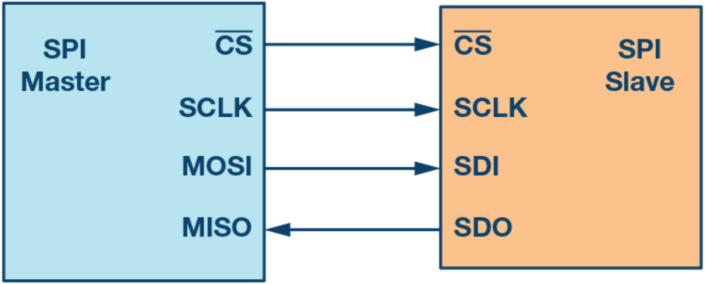
It is FULL-DUPLEX and synchronous

SPI

Only four signal lines are required:

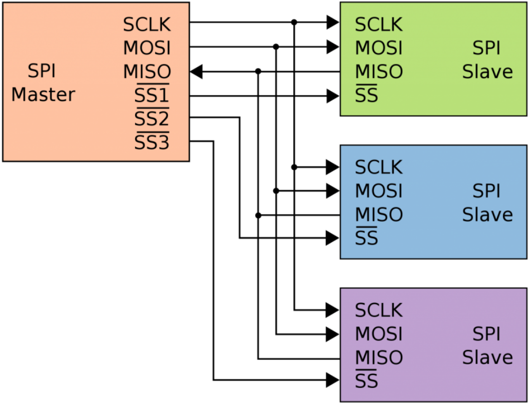
- MISO – Master Input Slave Output
- MOSI – Master Output Slave Input
- SCLK – Serial Clock
- SS – Slave Select

It is used for short distance communications, MISO and MOSI should be tri-state GPIOs.



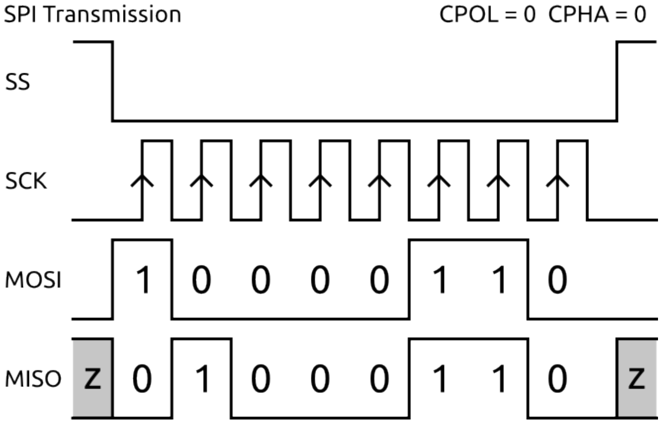
SPI

It is a shared bus with low GPIO requirements and it is sensibly faster than I2C (some peripherals exceed 10Mbit/s).

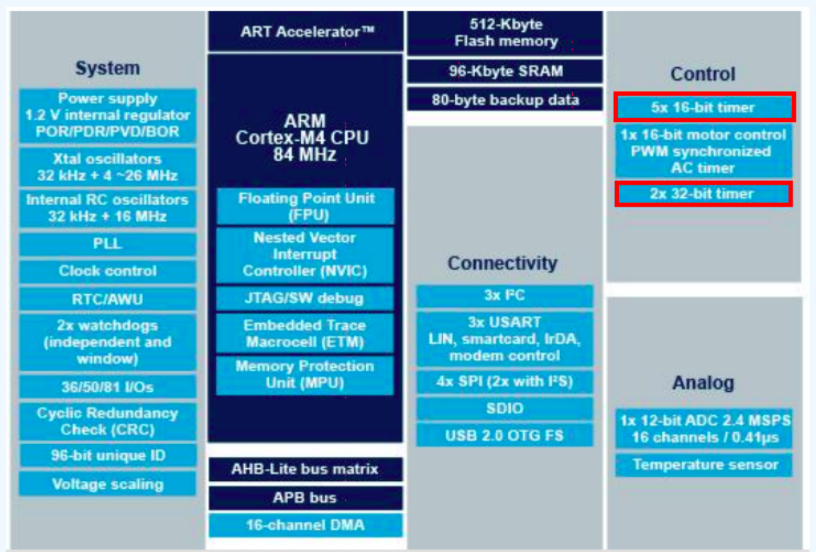


SPI

During each clock cycle, a bit is transferred from the MASTER to the SLAVE and a bit is transferred from the SLAVE to the MASTER.



Timer

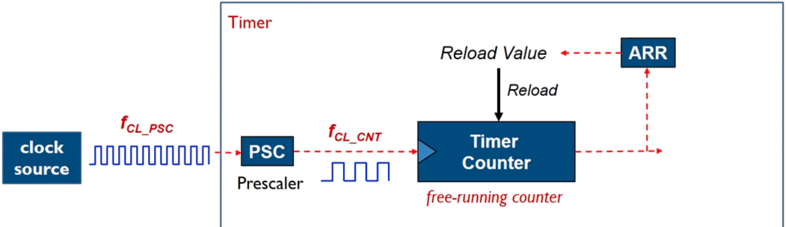

System
Power supply
1.2 V internal regulator
POR/PDR/PVD/BOR
Xtal oscillators
32 kHz + 4 ~26 MHz
Internal RC oscillators
32 kHz + 16 MHz
PLL
Clock control
RTC/AWU
2x watchdogs
(independent and window)
36/50/81 I/Os
Cyclic Redundancy Check (CRC)
96-bit unique ID
Voltage scaling
ART Accelerator™
ARM
Cortex-M4 CPU
84 MHz
Floating Point Unit (FPU)
Nested Vector Interrupt Controller (NVIC)
JTAG/SW debug
Embedded Trace Macrocell (ETM)
Memory Protection Unit (MPU)
AHB-Lite bus matrix
APB bus
16-channel DMA
512-Kbyte Flash memory
96-Kbyte SRAM
80-byte backup data
Connectivity
3x I²C
3x USART
LIN, smartcard, IrDA, modem control
4x SPI (2x with I²S)
SDIO
USB 2.0 OTG FS
Control
5x 16-bit timer
1x 16-bit motor control
PWM synchronized
AC timer
2x 32-bit timer
Analog
1x 12-bit ADC 2.4 MSPS
16 channels / 0.41µs
Temperature sensor

Timer

A *timer* is a digital counter that increments or decrements a variable according to a fixed input frequency – the clock source

The variable can be of any size, we use 16 bit and 32 bit timers. The size affects the maximum number of input clocks allowed before reaching an overflow or underflow situation

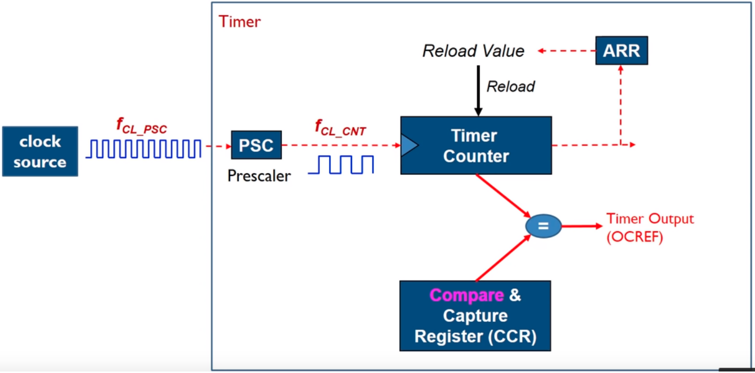
Timer



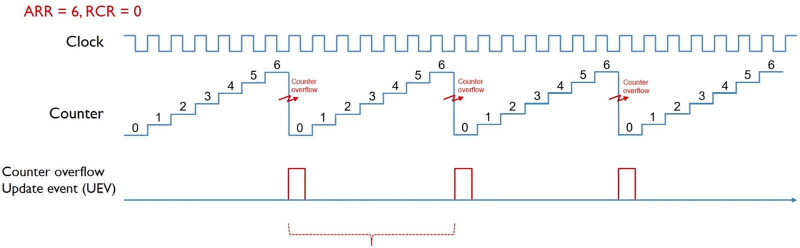
$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

Timer

STM timer have comparison logic to compare the timer value against a specific value, set by software, that triggers some action when the timer value matches the preset value



Upcounting Timer



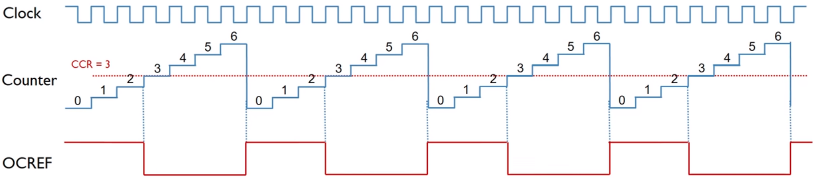

```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

Upcounting Timer

PWM Mode 1 (Low-True)

Mode I
Timer Output = High if counter < CCR; Low if counter ≥ CCR

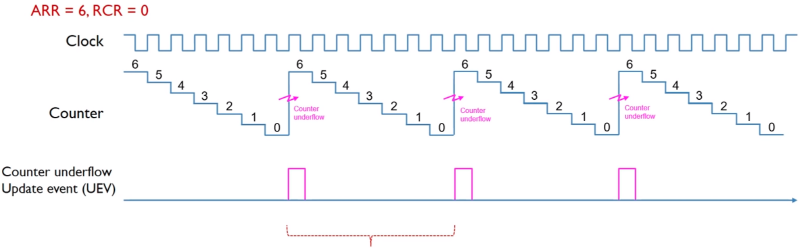
Upcounting mode, ARR = 6, CCR = 3, RCR = 0



$$\text{Duty Cycle} = \frac{CCR}{ARR + 1} = \frac{3}{7}$$

```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

Downcounting Timer




```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

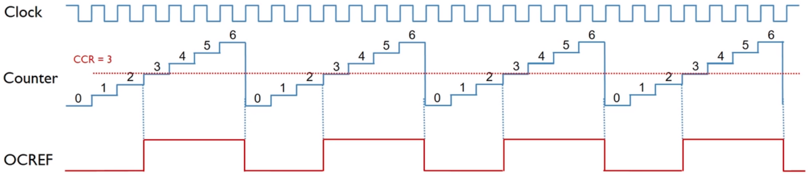
Downcounting Timer

PWM Mode 2 (High-True)

Mode 2

Timer Output = Low if counter < CCR; High if counter ≥ CCR

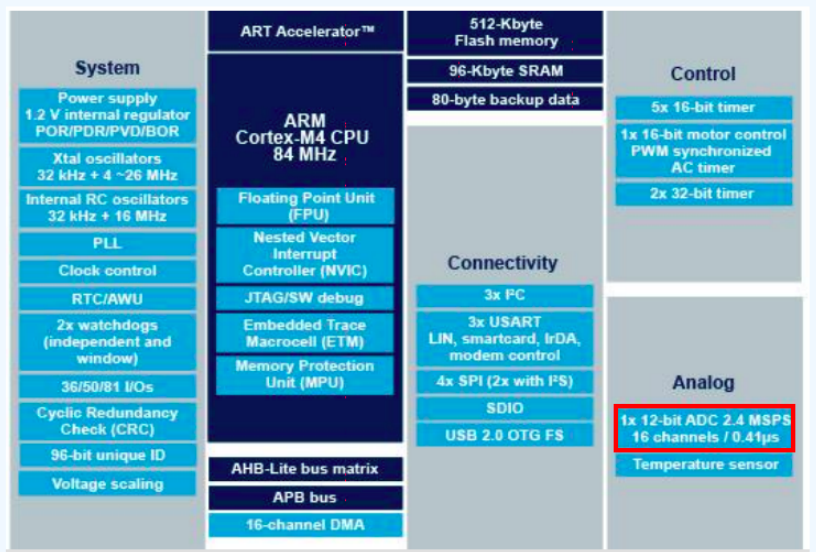
Upcounting mode, ARR = 6, CCR = 3, RCR = 0



$$\text{Duty Cycle} = 1 - \frac{\text{CCR}}{\text{ARR} + 1} = \frac{4}{7}$$

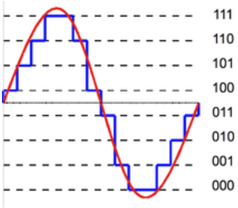
$$\text{Period} = (1 + \text{ARR}) * \text{Clock Period} = 7 * \text{Clock Period}$$

Analog-to-Digital Converter



Analog-to-Digital Converter

Analog-to-Digital Converter - is a system that converts an analog signal into a digital signal.



Physical values are often *analog*. A digital circuit needs to convert them into *digital* values in order to handle them.

ADC Resolution

Our MCU has a single ADC with 12bit resolution and 2.4 Mbps.
12 bit means that the result of an AD conversion gives up to 4096 different values.

If the working range of our ADC is from 0V to 4.096V, then each bit represents 1 mV.

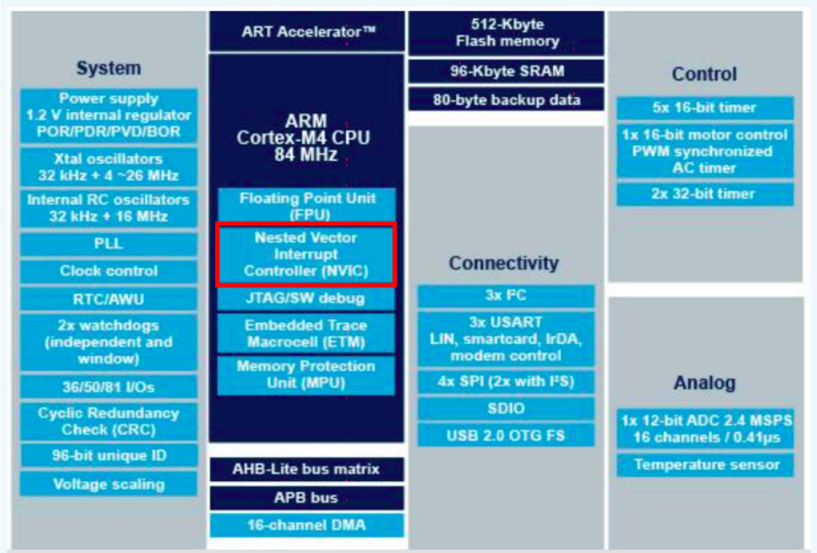
If the working range of our ADC is from -1.5V to 1.5V, then each bit represents $(1.5 + 1.5) / (2^{12}) = 3 / 4096 = 0.73$ mV.

ADC Sampling sequence

ADC is considered a slow peripheral, thus a reading sequence should follow this approach:

1. Initialize the ADC peripheral
2. Define an interrupt routine
3. Send the sampling command and do other stuff while waiting
4. The interrupt routine will be executed on ADC sampling completed

Interrupts


System
Power supply
1.2 V internal regulator
POR/PDR/PVD/BOR
Xtal oscillators
32 kHz + 4 ~26 MHz
Internal RC oscillators
32 kHz + 16 MHz
PLL
Clock control
RTC/AWU
2x watchdogs
(independent and window)
36/50/81 I/Os
Cyclic Redundancy Check (CRC)
96-bit unique ID
Voltage scaling
ART Accelerator™
ARM
Cortex-M4 CPU
84 MHz
Floating Point Unit (FPU)
Nested Vector Interrupt Controller (NVIC)
JTAG/SW debug
Embedded Trace Macrocell (ETM)
Memory Protection Unit (MPU)
AHB-Lite bus matrix
APB bus
16-channel DMA
512-Kbyte Flash memory
96-Kbyte SRAM
80-byte backup data
Connectivity
3x I²C
3x USART
LIN, smartcard, IrDA, modem control
4x SPI (2x with I²S)
SDIO
USB 2.0 OTG FS
Control
5x 16-bit timer
1x 16-bit motor control PWM synchronized AC timer
2x 32-bit timer
Analog
1x 12-bit ADC 2.4 MSPS
16 channels / 0.41µs
Temperature sensor

Interrupts

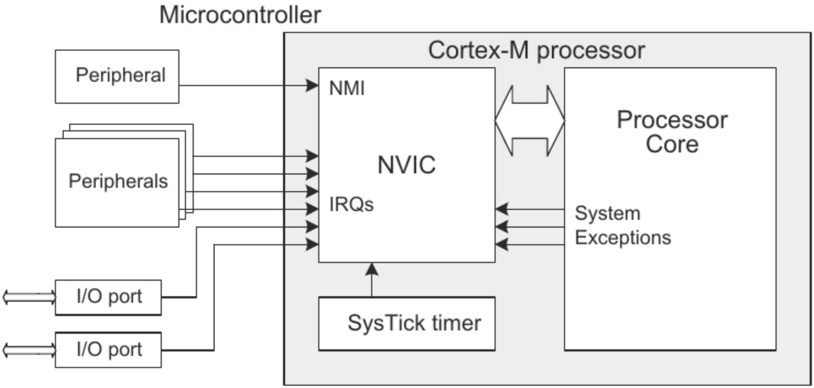
Sometimes an application MUST react quickly to events

Interrupts associate events like timer overflow, ADC sampling completion or the press of a button to a function.

When an interrupt happens, the current function is stopped and instead the interrupt handler is executed.

Interrupts

Nested Vector Interrupt Controller - is an advanced system for handling interrupts and their priorities.



Section 3

Introduction to ARM Mbed OS

ARM Mbed OS

Arm Mbed OS is a free, open-source embedded operating system designed specifically for the "things" in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller (32 bit).

Features

Modular - Ease of Use

With a modular library structure, the necessary underlying support for your application will be automatically included on your device. By using the Mbed OS API, your application code can remain clean, portable and simple.

End to End Security

It addresses security in device hardware, software, communication.

Open Source

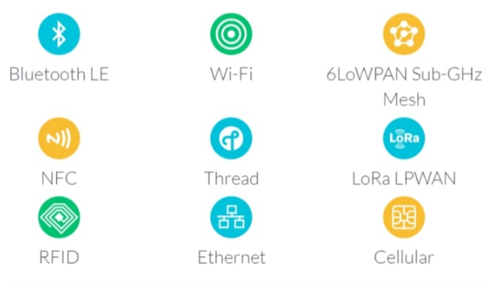
`https://github.com/ARMmbed/mbed-os`

Community

Big community support and *forum*.

Features

Connectivity



Drivers and Support Libraries

Possibility to use other drivers and libraries developed by other users.

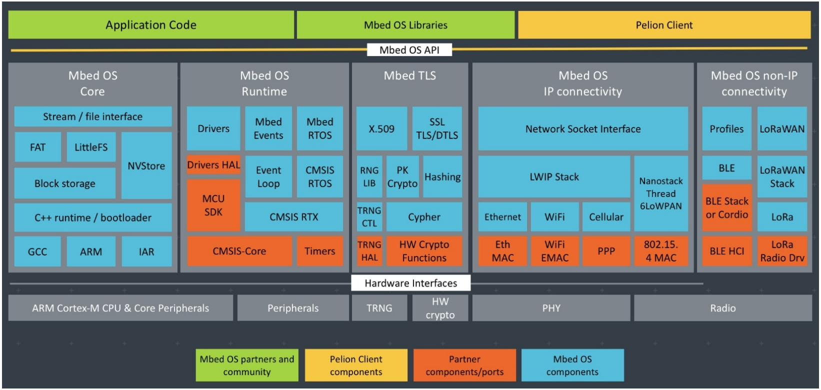
Toolchains and IDE

Two possibilities:

Online IDE, no setup and easiest way to get started.

Offline toolchains, (Arm Compiler 6, GCC and IAR) that support Arm Mbed CLI.

Architecture Diagram on an Mbed board



Blinking example

main.cpp

```
#include "mbed.h"

/*-------------------------------------------------------------------------------

-- HelloWorld example

    Printing hello world and toggling LED1

-------------------------------------------------------------------------------*/

DigitalOut led(LED1);

int main()
{
    int i = 1;

    printf("Hello World !\n");

    while(1) {
        wait(1); // 1 second
        led = !led; // Toggle LED
        printf("This program runs since %d seconds.\n", i++);
    }
}
```

https://os.mbed.com/users/marcozecchini/code/Esempio_helloworld/

Thank you