



Programmazione Microcontrollori



Programmazione Microcontrollori

Cosa Serve

- PC with **Windows** (XP / Vista / 7 / 8 /...)
- Development **board** (**STM32-XX Discovery**)
- **MINI USB** cable
- **Keil uVision IDE** for **ARM**

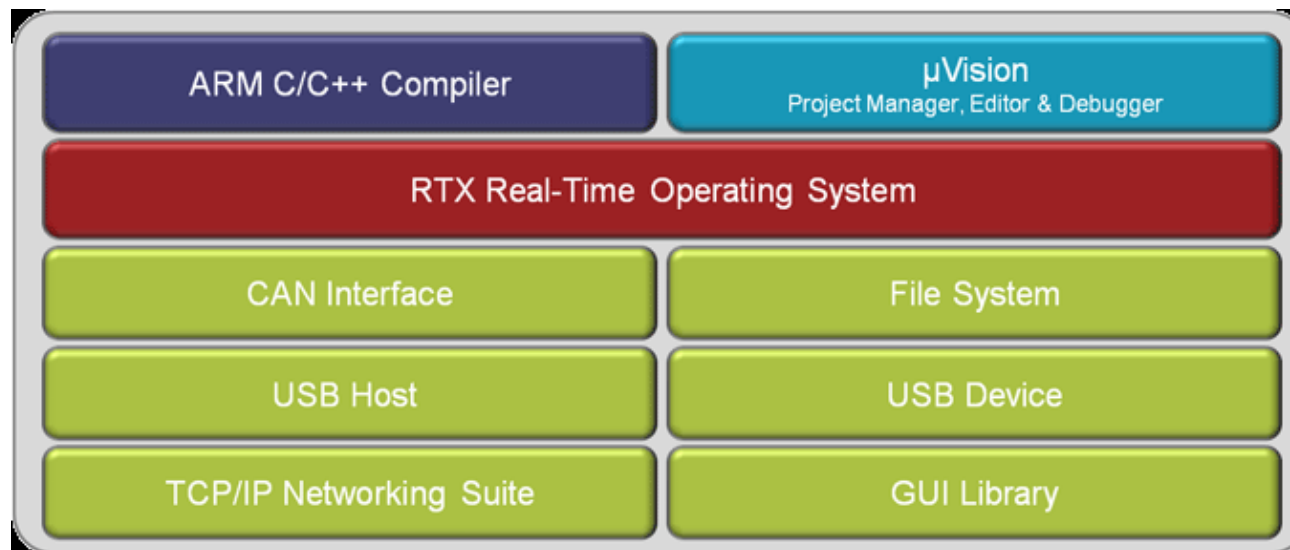


Keil uVision IDE for ARM

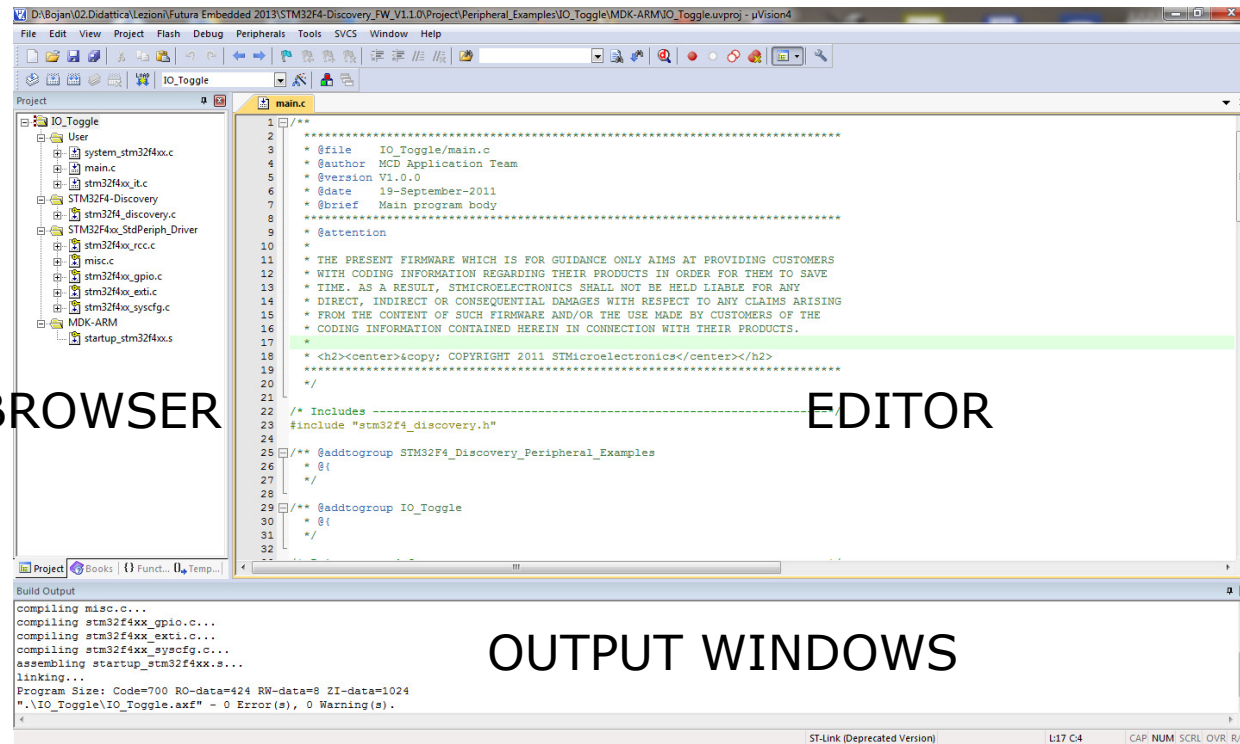
- Download the product from:
<http://www.keil.com/arm/mdk.asp>
- Run the downloaded executable
- Follow the instructions displayed by the SETUP program
- Install ST-Link Driver and update ST-Link Firmware

Keil uVision IDE for ARM

The MDK-ARM is a complete software development environment for Cortex™-M, Cortex-R4, ARM7™ and ARM9™ processor-based devices. MDK-ARM is specifically designed for microcontroller applications, it is easy to learn and use, yet powerful enough for the most demanding embedded applications.



Keil uVision IDE for ARM

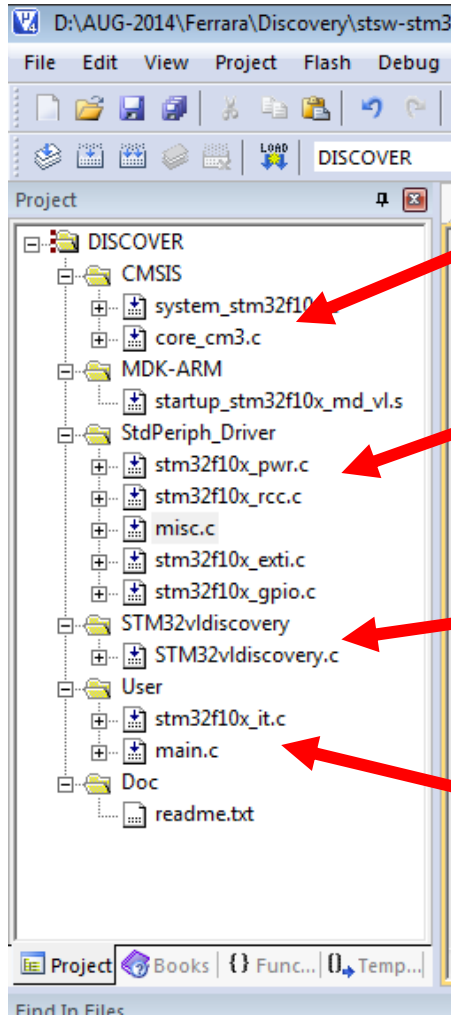


FILE BROWSER

EDITOR

OUTPUT WINDOWS

Keil uVision IDE for ARM



Core library CMSIS and MDK-ARM
(startup and system)

ST StdPeriph_Driver library (on-chip
devices init and use)

Discovery Library (on-board devices
init and use)

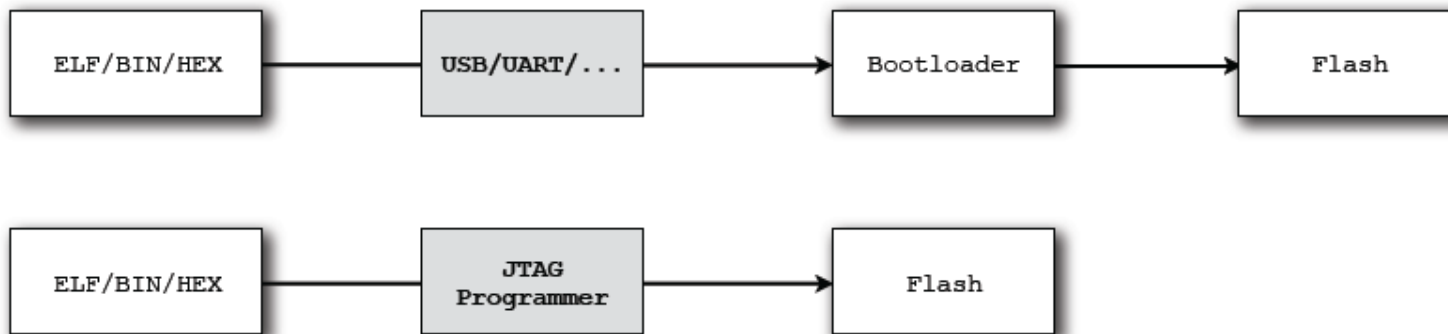
User code (main and all the
application code)

PROGRAMMING

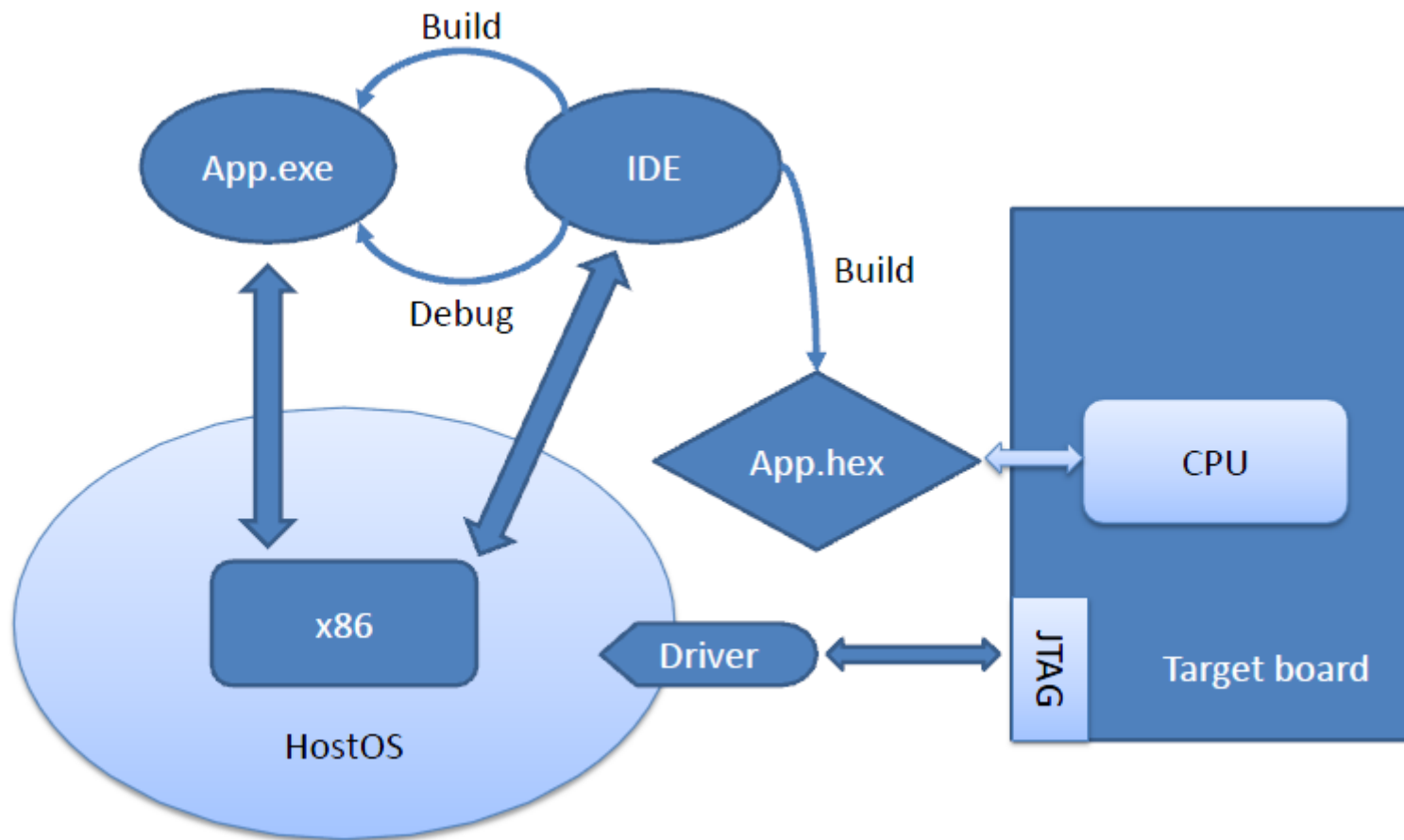
DUE OPZIONI:

–USB / UART / ... connection in **bootloader mode**

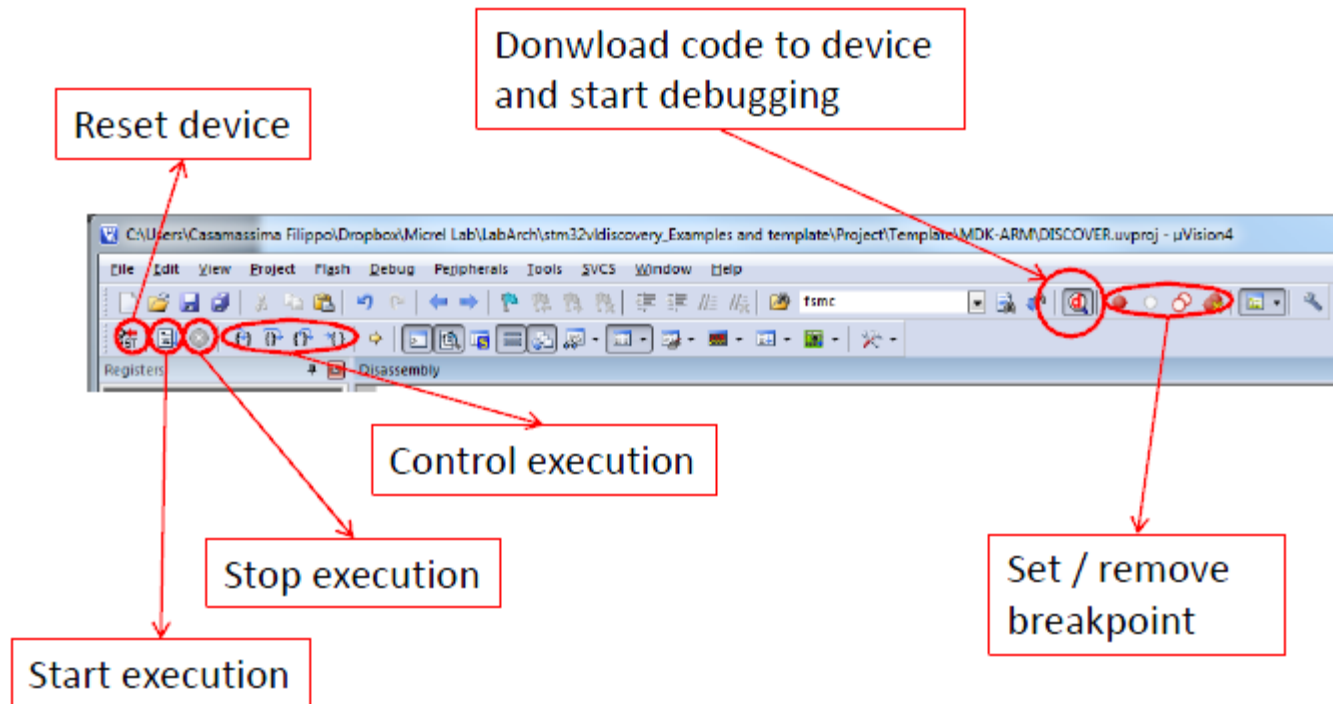
–JTAG and programmer **to write flash memory**



DEBUGGING



DEBUGGING



DEBUGGING

ARM Internal register status

The screenshot displays the Keil uVision4 IDE interface. The main window shows the disassembly of the `main` function. The current instruction location is highlighted in yellow, corresponding to the assembly instruction `MOV r1, #0x01` at address `0x080005F4`. A red arrow points to this instruction, with the label "Current instruction location". The registers window on the left shows the status of ARM internal registers, with the `R0` register highlighted. The command window at the bottom shows the load command: `Load "D:\AUG-2014\Ferrara\Discovery\stsw-stm32078\an3268\stm32vldiscovery_package\Project\Demo\MDK-ARM\DISCOVER.uvproj - μVision4"`.

Registers

Register	Value
R0	0x080005F5
R1	0x20000438
R2	0x00000000
R3	0x080005E7
R4	0x080007B0
R5	0x080007B0
R6	0x00000001
R7	0x000001F4
R8	0x00000000
R9	0x20000160
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x00000000
R14	0x00000000
R15	0x00000000

Disassembly window

```
53: RCC_APB1PeriphClockCmd(RCC_APB1Periph_FWR, ENABLE);
54:
55: /* Initialise LEDs LD3&LD4, both off */
56: MOV r1, #0x01
57: LSL r0, r1, #28
58: BL.W RCC_APB1PeriphClockCmd (0x08000350)
```

Current instruction location

```
47: * @retval None
48: */
49:
50: int main(void)
51: {
52: /* Enable GPIOx Clock */
53: RCC_APB1PeriphClockCmd(RCC_APB1Periph_FWR, ENABLE);
54:
55: /* Initialise LEDs LD3&LD4, both off */
56: STM32vldiscovery_LEDInit(LED3);
57: STM32vldiscovery_LEDInit(LED4);
58:
59: STM32vldiscovery_LEDOff(LED3);
60: STM32vldiscovery_LEDOff(LED4);
61:
62: /* Initialise USER Button */
63: STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_GPIO);
64:
65: /* Setup SysTick Timer for 1 msec interrupts */
66: }
```

Call Stack + Locals

Name	Location/Value	Type
main	0x080005F4	int f()

DEBUGGING

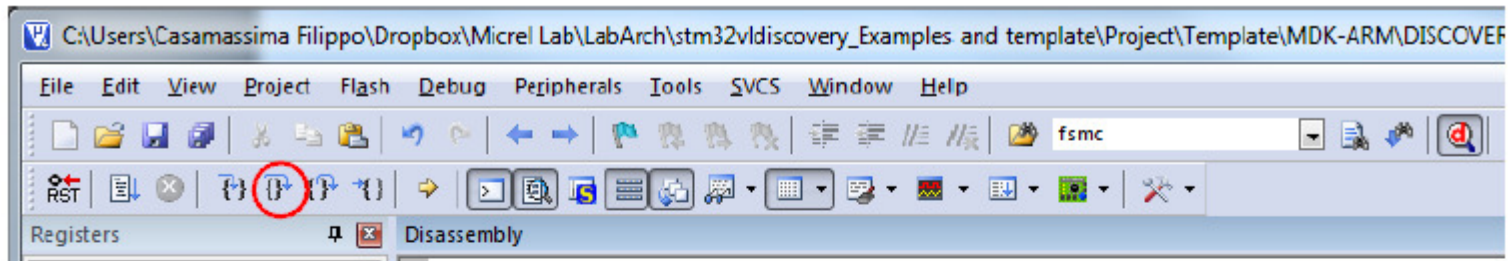


Step: executes next instruction

```
int main(void)
{
    u16 count = 0;
    u32 b = 0;
    int c = 0;
    c = SumValues(5, 18);
    /* main while */
    while(1)
    {
        count++;
        if (count==10000)
            b++;
    }
}

SumValues(int add1, int add2){
    volatile int sum_result = 0;
    sum_result = add1 + add2;
    return sum_result;
}
```

DEBUGGING



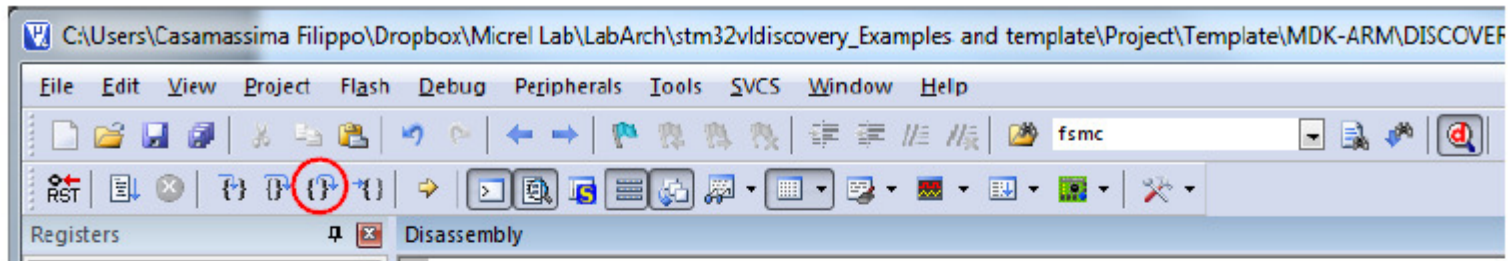
Step Over: executes next line



```
int main(void)
{
    u16 count = 0;
    u32 b = 0;
    int c = 0;
    c = SumValues(5, 18);
    /* main while */
    while(1)
    {
        count++;
        if (count==10000)
            b++;
    }
}

SumValues(int add1, int add2){
    volatile int sum_result = 0;
    sum_result = add1 + add2;
    return sum_result;
}
```

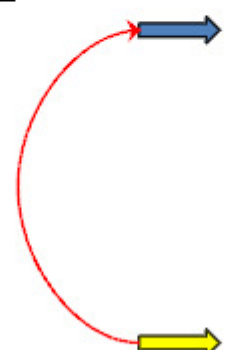
DEBUGGING



Step Out: Exit from the current function

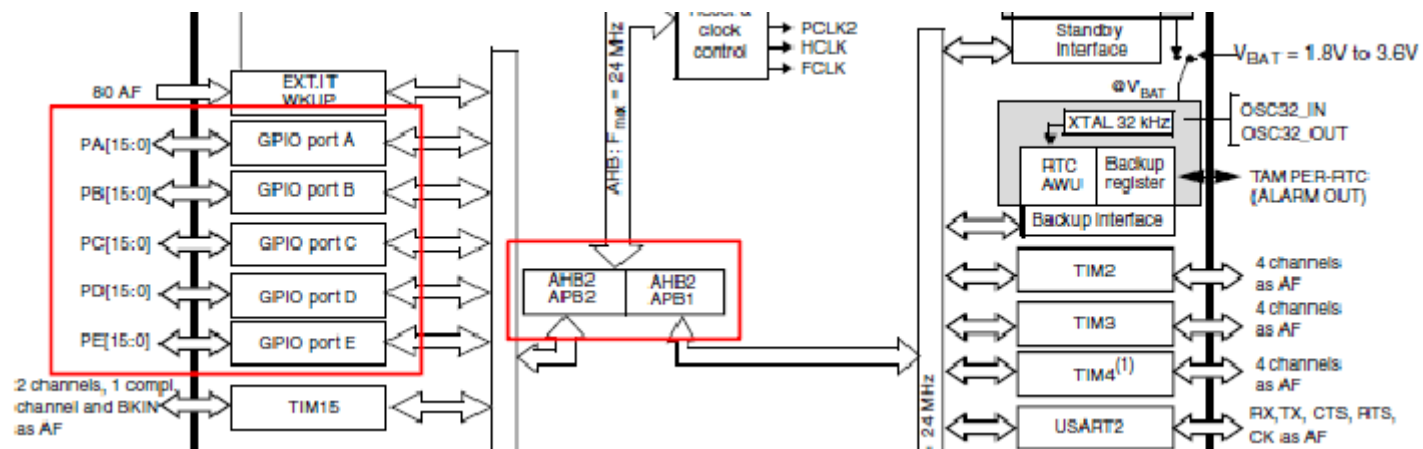
```
int main(void)
{
    u16 count = 0;
    u32 b = 0;
    int c = 0;
    c = SumValues(5, 18);
    /* main while */
    while(1)
    {
        count++;
        if (count==10000)
            b++;
    }
}

SumValues(int add1, int add2){
    volatile int sum_result = 0;
    sum_result = add1 + add2;
    return sum_result;
}
```



EXAMPLES: Ied

The STM32 is well served with general purpose IO pins, having up to 80 bidirectional IO pins. The IO pins are arranged as five ports each having 16 IO lines.





EXAMPLES: led – come usare i GPIO

Which bus GPIOs are connected to?

➔GPIO ports are always on the APB2 bus

Which port are we going to use?

➔Green LED is connected to the I/O Port C of STM32F100RB

➔Blue LED is connected to the I/O Port C of STM32F100RB

Which PINs the LEDs are connected to?

➔Green LED is connected to the pin 9 of Port C

➔Blue LED is connected to the pin 8 of Port C

What do I need to do with this GPIO? (input, output, ...)

➔I need to write (output)



EXAMPLES: led – Informazioni sulle connessioni

➔ The **datasheet** contains all the information we need

➔ Look at the **UM0919 User Manual**

https://www1.elfa.se/data1/wwwroot/assets/datasheets/STM32_discovery_eng_manual.pdf



EXAMPLES: led - Accensione

We need to enable the High Speed APB (APB2) peripheral.

➔void RCC_APB2PeriphClockCmd(uint32_t RCC_APB2Periph, FunctionalState NewState);

(Look at: stm32f10x_rcc.c)

We need to configure the GPIO Port

➔Fill up a GPIO_InitTypeDef structure (Look at: stm32f10x_gpio.h)

➔Init the GPIO Port with void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);

(Look at: stm32f10x_gpio.c)

Turn ON the LED (Look at :stm32f10x_gpio.c)

➔void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

EXAMPLES: led

main.c (green LED)

```
#include "stm32f10x.h"  
#include "stm32f10x_conf.h"
```

must include stm32f10x_gpio.h

```
int main(void)  
{
```

```
    GPIO_InitTypeDef GPIO_InitStructure;
```

```
    /* Enable the GPIO_LED Clock */  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
```

Enable APB2 bus Port C

```
    /* Configure the GPIO_LED pin */  
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;  
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;  
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;  
    GPIO_Init(GPIOC, &GPIO_InitStructure);
```

Configuration for Pin 9 Port C as output

```
    /* Turn ON */  
    GPIO_SetBits(GPIOC, GPIO_Pin_9);
```

"And light was made"

```
    while(1);  
}
```



SYSTick

SysTick is used to schedule **periodic** events

When the SysTick expires an **IRQhandler** is called

How can I use SysTick?

We need to setup the SysTick (Look at `core_cm3.h`)

➔ `static __INLINE uint32_t SysTick_Config(uint32_t ticks)`

ticks: the number of ticks between two interrupts

➔ `SystemCoreClock` is the number of ticks in 1 sec

We need to setup the callback (Interrupt Service Routine)

➔ The ISR is always define in `stm32f10x_it.c`

➔ The name of the ISR for SysTick is `void SysTick_Handler(void)`



SYSTick

main.c

```
#include "stm32f10x.h"
#include "stm32f10x_conf.h"

int main(void)
{
    if (SysTick_Config(SystemCoreClock / 1000)) {
        /* Capture error */
        while (1);
    }

    while (1);
}
```

ISR executed every 1 ms

stm32f10x_it.c

```
...

void SysTick_Handler(void){
    /* Here goes the code to periodically execute */
}

...
```