simo85

STM32F4 DISCOVERY - TUTORIAL CON KEIL MVISION

18 October 2012

Premessa

Ho deciso di scrivere questo articolo per descrivere il procedimento di configurazione e programmazione di un microcontrollore **ARM Cortex M4**. Non è stato seguito nessun tutorial in particolare, tutto il procedimento è stato svolto individualmente durante un pò di *troubleshooting*, e dopo una piccola esperienza ho deciso di riportare qui alcune informazioni. Ho ancora molto da imparare ma è anche vero che mi piacerebbe lavorare seriamente con questa architettura.

L'articolo non è intenzionato a descrivere l'architettura ARM nei suoi più profondi dettagli, anche perché non basterebbe un libro, ma bensì descrive il procedimento di configurazione e stesura di quello che può essere un primissimo codice di esempio, usando istruzioni dirette in linguaggio C.

Volevo ringraziare **TardoFreak** per avermi dato direttamente ed indirettamente gli stimoli ed aver fatto crescere i miei interessi per i sistemi embedded, **GuidoB** che mi ha aiutato in un post riguardo ad un codice C (ed in bocca al lupo per tutto)..

Buona lettura.

Introduzione

In questo articolo si descrive il procedimento e la tecnica di programmazione di un microcontrollore di architettura ARM Cortex M4. Per esattezza l'hardware utilizzato è una scheda

- **[1] STM32F4 Discovery**

(al link è possibile scaricare anche lo schema elettronico), la quale incorpora il circuito per il debugger/programmatore

- **[2] ST-Link**

ed un microcontrollore

- **[3] STM32F407VGT6**.

I 168 MHz di clock, 1 Mbyte di Flash, 17 Timers, 16 canali di DMA e tutte le altre caratteristiche riportate al link sotto la voce "**Key Features**" fanno di questo microcontrollore un bel ragnaccio a 100 zampe!

L'ambiente di sviluppo usato è l'IDE

- **[4] Keil µVision**

installato su Windows XP.

Sfortunatamente la compilazione delle **ARM toolchain** sotto ambiente Linux non è andata del tutto a buon fine, e causa anche mancanza di tempo, non sono riuscito a far funzionare il tutto come avrei voluto. Per il momento posticipo questo compito per un altra occasione, chissà che potrò descriverlo in un successivo articolo.

Come al solito l'articolo sarà arricchito di link utili e necessari, riportati all'interno dell'articolo come alla fine nella sezione "Link utili".

Link di riferimento per l'architettura ARM

Personalmente non mi sento in grado di descrivere in un articolo l'architettura **ARM** (**A**dvanced **R**educed Instruction Set Computer **M**achine), anche perché non è lo scopo dell'articolo, però prima di proseguire mi sembra d'obbligo indirizzare i lettori al sito di riferimento ARM.

Al sito ufficiale

- **[5] ARM**

è possibile consultare la documentazione online

- **[6] ARM Infocenter**

dove si possono trovare tutte la documentazioni ufficiali di riferimento all'architettura, così come i manuali per le varie serie di processori e MCU e le funzioni **CMSIS**

- **[7] CMSIS - Cortex Microcontroller Software Interface Standard**.

Materiale necessario

Come al solito prima di cominciare è bene procurarsi TUTTO il materiale. Nel nostro caso, oltre all'hardware, dobbiamo procurarci il DATASHEET del microcontrollore in questione, 1422 paginette che si possono trovare al link a seguire

- **[7] STM32F407VG Datasheet**.

Questo è l'unico datasheet di cui abbiamo bisogno per conoscere e programmare il microcontrollore in C (nel nostro caso).

La ST mette a disposizione gratuitamente le librerie C

- **[8] STM32F4 DSP and standard peripherals library**

per i microcontrollori Cortex M4 STM32F4xx.

Al link è possibile scaricare l'intero file zippato contenente tutti i sorgenti delle librerie e progetti. Il file **stm32f4_dsp_stdperiph_lib.zip** deve essere scaricato e salvato sul disco duro in quanto contiene file necessari alla compilazione del programma.

All'interno del file zippato si trova anche la documentazione approfondita in formato **chm**. Si ricorda che per poter aprire un file di tale estensione è necessario disporre appunto di un lettore file **chm**. Nell'esempio non useremo le librerie ma è utilissimo consultarle.

Oltre alle librerie viene messo a disposizione il

- **[9] STM32F4DISCOVERY board firmware package**

quindi un file contente i progetti di esempio per la scheda di valutazione in uso. Per lo scopo dell'articolo però interessa il link **[8]**.

Nel mio caso ho usato l'IDE per la stesura e la compilazione del codice, mentre per caricare il file esadecimale all'interno della memoria Flash del microcontrollore ho usato la **STM32 Link Tool** scaricabile al link **[2]** sotto la voce "**Related Tools and Software**".

L'IDE **Keil µVision** offre l'opzione di installare automaticamente il driver per il debugger/programmatore **ST-Link V2**, tuttavia come si può osservare sempre al link **[2]**, si trova anche il download del singolo driver.

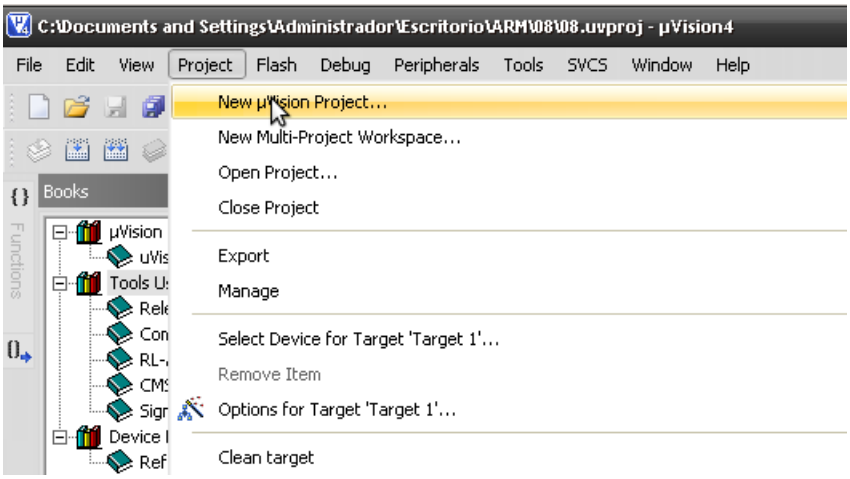
Il primo progetto

Il primo programma di prova consiste sempre nell'accensione di un LED con lampeggio per una prima prova visiva. Però, prima di cominciare con la stesura del codice, proseguiamo passo per passo nel settare i vari parametri di configurazione all'interno dell'IDE.

Quindi, dopo aver scaricato ed installato l'IDE (in questo caso ricordiamo che si tratta del **Keil µVision**) e aver svolto la eventuale procedura per ottenera la licenza, l'unica cosa da fare è lanciare il programma. :)

Screenshot.png

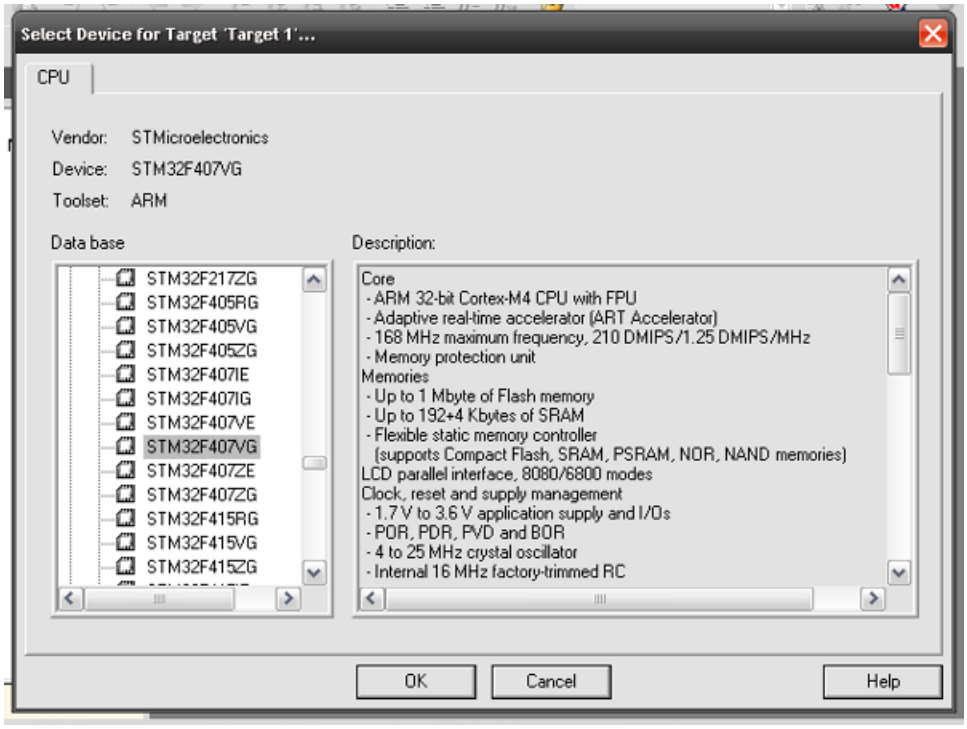
Una volta aperto l'IDE dovremo creare un nuovo progetto. Muoveremo il cursore sulla voce **Project** e selezioneremo **New µVision Project**



Screenshot-1.png

A seguire si aprirà una "classica" finestra nella quale bisognerà scegliere la cartella in cui salvare il nuovo progetto. Ovviamente si è liberi di sceglier e il percorso che più si preferisce. Si consiglia comunque di creare una cartella in cui salvare il progetto al suo interno. Ad esempio se creiamo un progetto di nome **Test**, creare la cartella **Test**..

Una volta scelto il percorso e la cartella dedicata, si aprirà la finestra dedicata alla selezione del dispositivo (MCU) a programmare. Nel nostro caso bisogna cercare nella cartella **ST Microelectronics** (nella sotto-finestra **Data Base:**) il microcontrollore **STM32F407VG**.



Screenshot-2.png

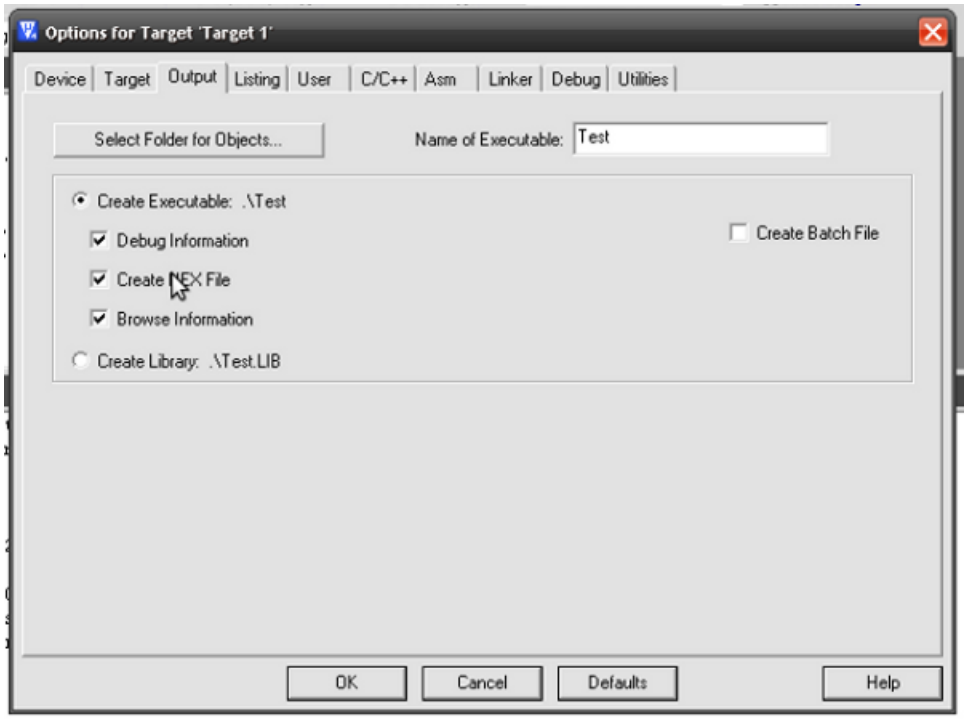
Una volta che si ha cliccato su OK si aprirà un'altra finestra nella quale vi verrà chiesto se volete aggiungere il file **startup_stm32f4xx.s** (che è il file Assembly per la mappatura di memoria) al progetto. Cliccare su OK!

Fatto ciò si aprirà l'ambiente dedicato al progetto, apparentemente solo provvisto del file Assembly appena citato.

A questo punto il primo passo a seguire è quello di settare la configurazione di compilazione, debug e caricamento del file esadecimale nella Flash del microcontrollore.

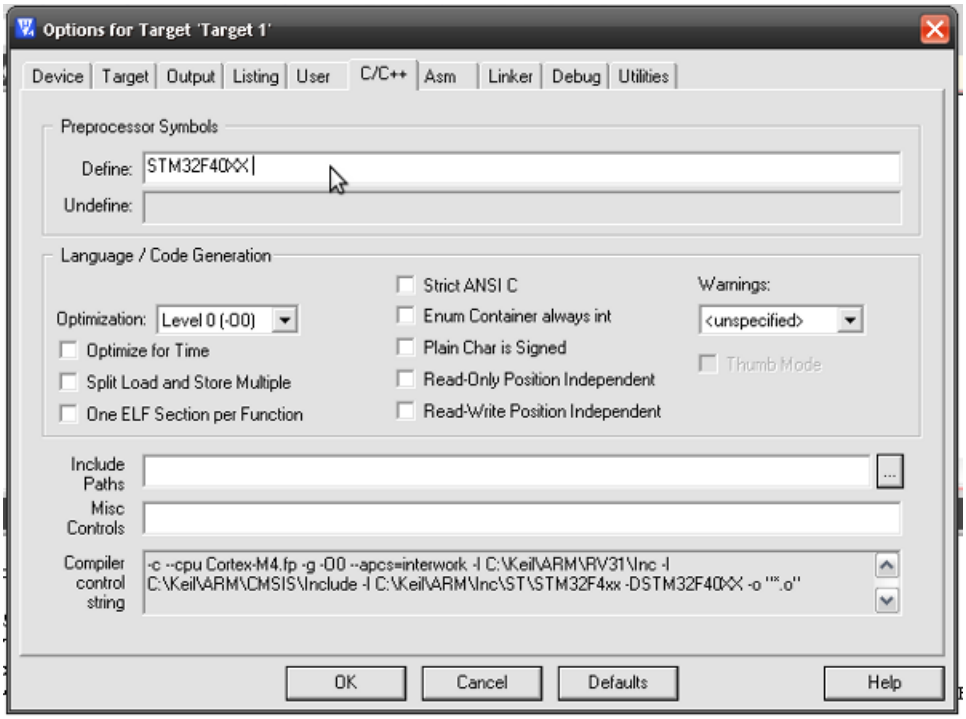
Cliccare quindi la voce **Flash** del menù principale ed a seguire **Configure Flash Tools...**.

Nella finestra che si aprirà, bisogna entrare nel sottomenù **Output** ed abilitare la creazione del file esadecimale.



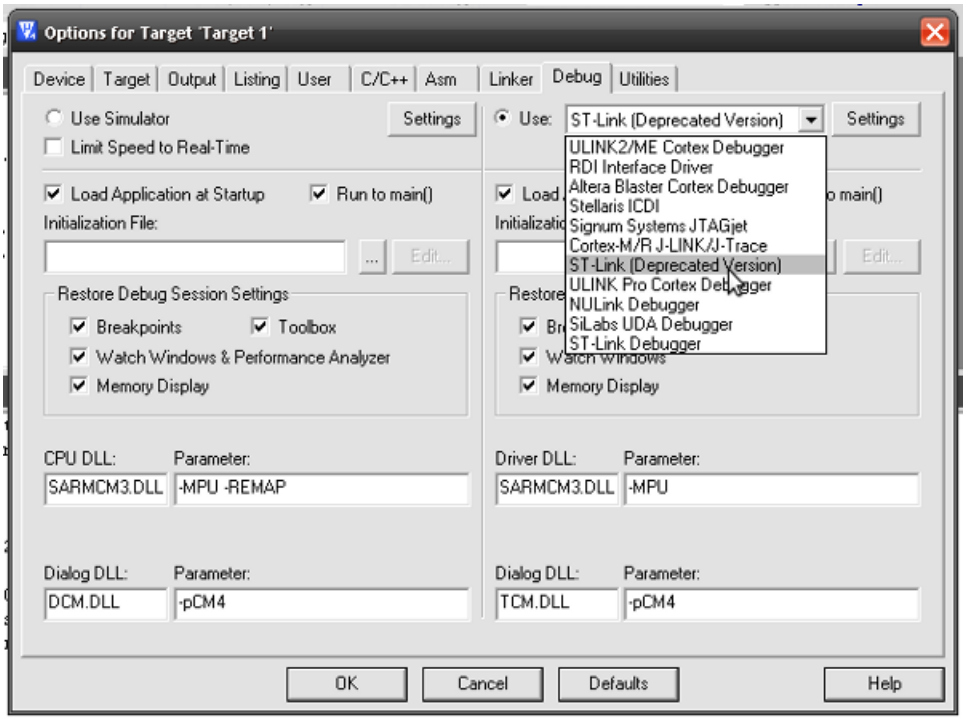
Screenshot-4.png

A partire dalla versione **V1.1.0** delle librerie, è necessario definire la macro **STM32F40XX** nel sotto menù **C/C++** alla voce **Define**. Questo serve ad istruire il compilatore nel generare codice per la famiglia STM32F40x e non STM32F427x. Per maggiori dettagli osservare l'header **stm32f4xx.h** righe 68 - 80.



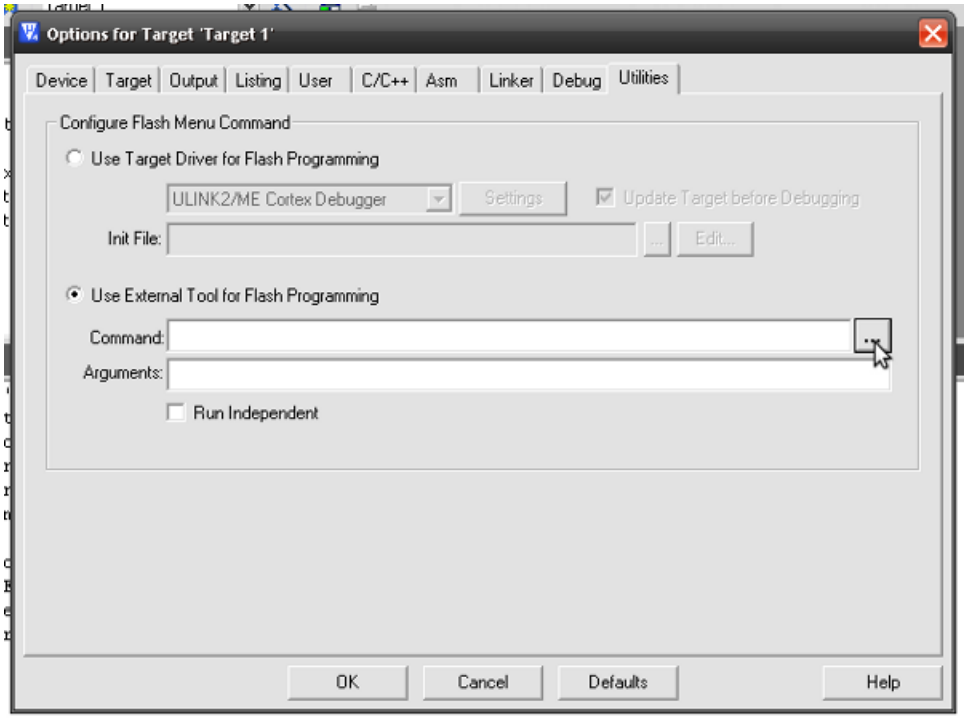
Screenshot-8.png

Nel sottomenù **Debug** selezionare **ST-Link (Deprecated Version)**.



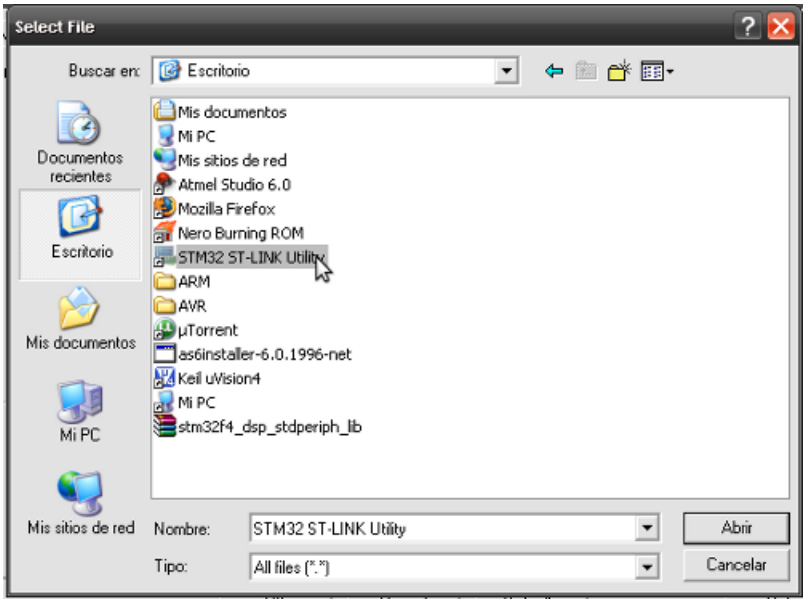
Screenshot-5.png

Non resta che selezionare il FLASH downloader. Nel mio caso ho installato la tool **STM32 ST-Link Utility** scaricabile al link **[2]** riportato in precedenza, causa problemi di funzionamento con l'IDE ed il downloader interno. Entrando nel sottomenù selezionare **Use External Tool for Flash Programming**.



Screenshot-5.png

E quindi selezionale la utility dal Desktop (link creato dall'installazione del software).



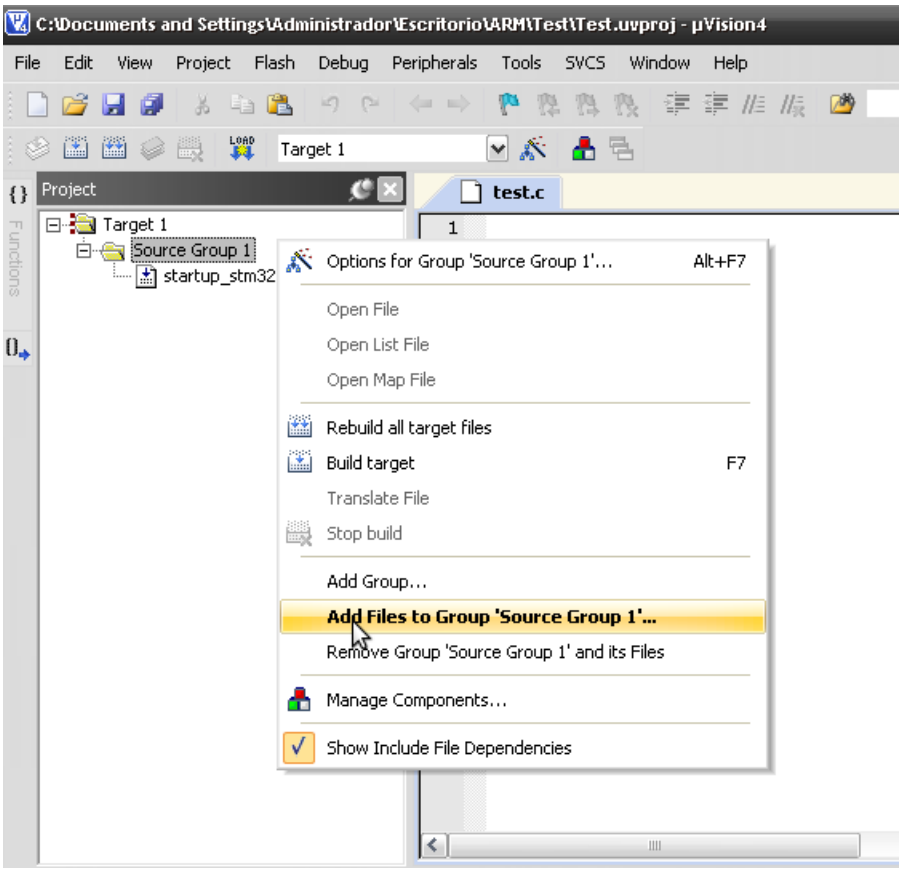
Screenshot-7.png

Nel caso non si voglia usare la utility, selezionare l'opzione **Use Target Driver for Flash Programming** ed a seguire **ST-Link (Deprecated Version)**.

Fatto ciò si è pronti a mettere mano al codice del progetto.

Cliccare **Ctrl-N** per aprire una sezione per un nuovo file di testo che salveremo (all'interno della cartella dedicata al progetto) premendo **Ctrl-S**con il nome **test.c**, all'interno del quale risiederà la funzione main.

A questo punto bisogna aggiungere il file **test.c** al progetto. Con il cursore posizionato sulla voce **Source Group 1**, clicchiamo con tasto destro del mouse e selezioniamo la voce **Add Files to Group 'Source Group 1'** come rappresentato nella immagine a seguire.



Screenshot-3.png

Una volta eseguita l'operazione selezionare il file **test.c**. Si ricorda che questa operazione deve essere effettuata per TUTTI i file necessari alla compilazione del progetto, siano questi header o file C.

A questo punto, per lo scopo dell'articolo, mancano altri 3 file necessari alla compilazione del progetto, bensì:

- **stm32f4xx.h**
- **system_stm32f4xx.h**
- **system_stm32f4xx.c**

Il file **system_stm32f4xx.c** implementa funzioni di configurazioni generali di sistema, come per esempio il clock. Ad ogni modo è sempre bene leggere la descrizione iniziale al principio del/i file per documentarsi a dovere.

Scompattato il file **stm32f4_dsp_stdperiph_lib.zip** i primi due file si trovano nella cartella

STM32F4xx_DSP_StdPeriph_Lib_V1.1.0\Libraries\CMSIS\Device\ST\STM32F4xx\Include

mentre il terzo si trova in

STM32F4xx_DSP_StdPeriph_Lib_V1.1.0\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates.

Io, a causa di problemi di compilazione, ho copiato i tre file in questione in una cartella a parte ed ho usato quelli per i progetti, causa problemi di compilazione..

Una volta svolto con successo il procedimento descritto, possiamo passare alla stesura del primo firmware di prova.

Il primo programma

Come preannunciato in precedenza, il primo programma è il classico lampeggio di un LED. Per la sua stesura è stato consultato il datasheet del link **[7]**.

Il seguente e semplicissimo codice implementa una piccola funzione di ritardo usando il Timer 1 **TM1** (sezione 14 pag. 353 del datasheet).

Andando con ordine è bene sottolineare che ogni periferica interna dispone di un registro di reset e clock **Reset and clock control (RCC)** (sezione 6.1 pag. 111 del datasheet).

Prima di usare ciascun dispositivo, sia questo timer, GPIO (General Purpose Input Output), DMA etc. è necessario abilitare il relativo clock. Questo vuol dire che se vogliamo usare anche solo il pin GPIOD:2 (è irrilevante il numero di pin in questo caso), dovremo abilitare il clock per il GPIOD.

Per esempio l'istruzione

```
RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;
```

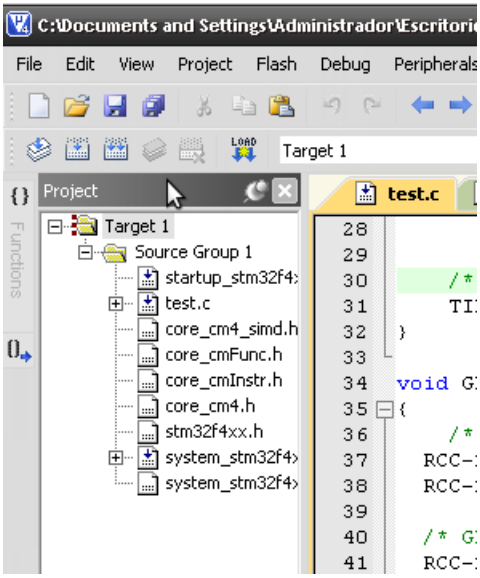
abilita il clock per il GPIOD.

```
/*
   STM32F4 1st test
   Author: A. Simone (simo85)
*/

#include <stm32f4xx.h>

void TIM1_config(void)
{
    /* Reset TM1 peripheral */
    RCC->APB2RSTR |= RCC_APB2RSTR_TIM1RST;
    RCC->APB2RSTR &= ~(RCC_APB2RSTR_TIM1RST);

    /* TIM1 CLOCK ENABLE*/
    RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;

    /* SET the autoreload value */
    TIM1->ARR &= 0xffff;

    /* TIM1 UP Counting & CLK Division x 2, downcount & autoreload enabled */
    TIM1->CR1 = TIM_CR1_CKD_0 | TIM_CR1_DIR | TIM_CR1_ARPE;

}

void delay_TM1(uint16_t CNTR_VALUE)
{
     /* SET the counter value */
     TIM1->CNT = CNTR_VALUE;

     /* Enable TIM1 - Turn it on*/
     TIM1->CR1 |= TIM_CR1_CEN;

     while(!TIM1->CNT)
        ;

     /* Turn off TM1 */
     TIM1->CR1 &= ~(TIM_CR1_CEN);
}
```

```
void GPIO_config(void)
{
    /* Reset GPIOD peripheral */
    RCC->AHB1RSTR |= RCC_AHB1RSTR_GPIODRST;
    RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIODRST);

    /* GPIOD CLOCK ENABLE*/
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIODEN;

    /* MODER - General I/O purpose (Digital) Output */
    GPIOD->MODER = GPIO_MODER_MODER15_0;

    /* 100 MHz High speed or maximum high speed */
    GPIOD->OSPEEDR = GPIO_OSPEEDER_OSPEEDR15_0;

    /* Push Pull Output Stage */
    GPIOD->OTYPER = 0x0000;

    /* No pull-up or pull-down */
    GPIOD->PUPDR = 0x0000;
}

int main(void)
{
    TIM1_config();
    GPIO_config();

    while(1)
    {
       /* Set the pin to logic level 1 */
       GPIOD->BSRRL = 0x00008000;

       delay_TM1(0x00ff);
       /* Set the pin to logic level 0 */
       GPIOD->BSRRH = 0x00008000;

       delay_TM1(0x00ff);
    }

    return 0;
}
```

Caricamento in Flash

Una volta scritto e salvato il codice all'interno del progetto, è il momento di compilare il programma. Facendo riferimento all'immagine seguente:

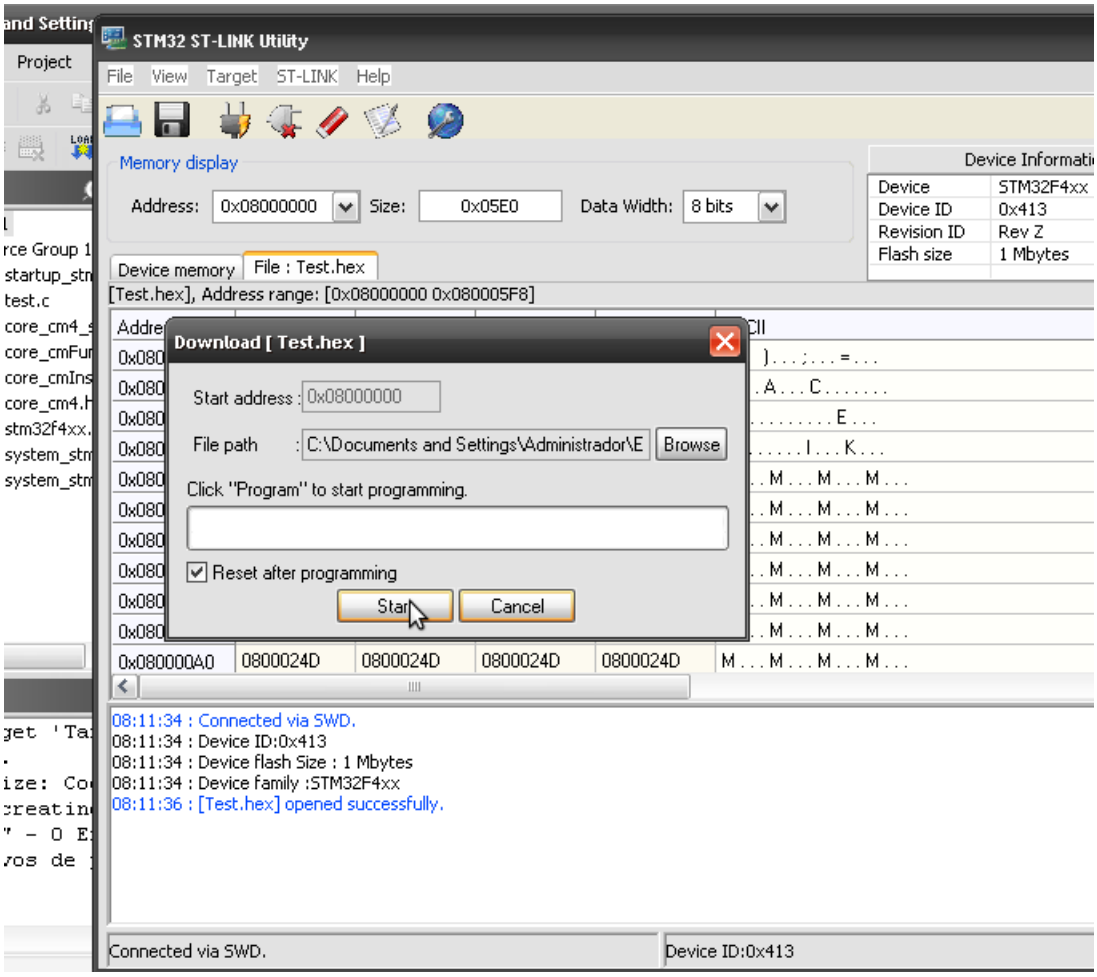


Screenshot-11.png

Il codice può essere compilato cliccando sulla icona **Build** (quella sotto l'icona **Open**), o più velocemente premendo **F7**.

A questo punto il di caricamento del file esadecimale può essere effettuato cliccando sull'icona **Load**.

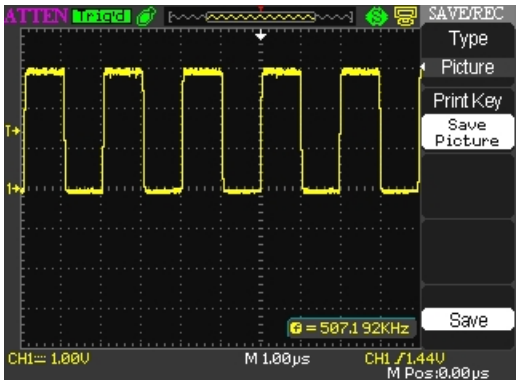
Se si è scelto di usare la *utility* **ST-Link Utility** secondo le impostazioni mostrate precedentemente, questa si aprirà automaticamente (può anche essere eseguita al di fuori dell'IDE senza problemi).



Screenshot-12.png

A questo punto dovrebbe essere sufficente cliccare sull'icona **Program Verify** (penultima icona sotto la voce del menù **Help**). Si aprirà quindi la finestra di **Download**. Eventualmente bisognerà cercare il file con estensione .hex cliccando su **Browse**, in tal caso lo si seleziona dando l'OK.

Ora si, è tutto pronto per caricare in memoria Flash il firmware scritto. Ecco quindi una cattura con l'oscilloscopio dell'esecuzione del codice riportato in precendenza.



ADS00001.jpg

Esempio con interrupt

Il codice a seguire mostra il semplice uso di una interruzione. Il codice accende in sequenza, alla pressione di un pulsante, i LED montati sulla scheda. Anche il pulsante fa parte della stessa scheda. Si ricorda che seguendo il link **[1]** è possibile scaricare lo schema elettronico della scheda.

```
/*    STM32F4 Interrupt test   Author: A. Simone (simo85)*/

#include <stm32f4xx.h>

unsigned char x;

void EXTI0_IRQHandler(void)
{
    /*
     EXTI->PR must be reset to 1 in order
     to clear the pending edge event flag


     FROM DATASHEET page 205
     Pending register (EXTI_PR)

     "This bit is set when the selected edge event
     arrives on the external interrupt line.
     This bit is cleared by writing a 1 to the bit
     or by changing the sensitivity of the edge detector."
    */

    if(EXTI->PR == EXTI_PR_PR0)
        EXTI->PR = EXTI_PR_PR0;
```

```
    x++;

    if(x > 3)
        x = 0;
}

void GPIO_config(void)
{
     /* Reset GPIOD pheriperal */
    RCC->AHB1RSTR = RCC_AHB1RSTR_GPIODRST | RCC_AHB1RSTR_GPIOARST;
    RCC->AHB1RSTR &= ~(RCC_AHB1RSTR_GPIODRST | RCC_AHB1RSTR_GPIOARST);

    /* GPIOD and GPIOA CLOCK ENABLE*/
    RCC->AHB1ENR = RCC_AHB1ENR_GPIODEN | RCC_AHB1ENR_GPIOAEN;

    /* GPIOA:0 configuration */
    /* Set GPIOA0 as input */
    GPIOA->MODER |= 0x55555554;

    /* Set GPIOA0 as Open Drain and GPIOA[15:1] as PUSH PULL reset state */
    GPIOA->OTYPER = (uint16_t) 0x0000;

    /* GPIOAx no pull-up or pull-down */
    GPIOA->PUPDR = (uint32_t) 0x00000000;

    /* GPIOD Configuration */
    /* Set GPIOD[12 ··· 15] as output */
    GPIOD->MODER = GPIO_MODER_MODER12_0 | GPIO_MODER_MODER13_0 | GPIO_MODER_MODER14_0 |

    /* Set GPIOD[12 ··· 15] speed up to 25MHz */
    GPIOD->OSPEEDR = GPIO_OSPEEDER_OSPEEDR12_0 | GPIO_OSPEEDER_OSPEEDR13_0 | GPIO_OSPEE

    /* Set GPIODx as PUSH PULL reset state */
    GPIOD->OTYPER = 0x0000;

    /* GPIOD no pull-up or pull-down */
    GPIOD->PUPDR = 0x00000000;
}

void EXTI0_config(void)
{
    /* Reset SYSCFG peripheral */
```

```
    RCC->APB2RSTR |= RCC_APB2RSTR_SYSCFGRST;
    RCC->APB2RSTR &= ~(RCC_APB2RSTR_SYSCFGRST);

    /* Enable SYSCFG clock */
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    /* Configure external interrupt for PA0 */
    SYSCFG->EXTICR[0] = 0xfff0;

    /* Enable Interrupt and event generation on line #0 */
    EXTI->IMR |= EXTI_EMR_MR0;

     /* Enable rising edge detection on line #0 */
    EXTI->RTSR |= EXTI_RTSR_TR0;


    /* CMSIS Functions */

    /* Set EXTI0 to the lowest priority */
    NVIC_SetPriority(EXTI0_IRQn, 0x0f);

    /* Enable EXTI0 Interrupt */
    NVIC_EnableIRQ(EXTI0_IRQn);
}

void main(void)
{
    /*
        0x1000 -> PD12 -> LD4 Green ON
        0x2000 -> PD13 -> LD3 Orange ON
        0x4000 -> PD14 -> LD5 Red ON
        0x8000 -> PD15 -> LD6 Blue ON
    */

    uint16_t VALUE[4] = {0x1000, 0x2000, 0x4000, 0x8000};

    GPIO_config();
    EXTI0_config();

    x = 0;

    while(1)
    {
```

```
        /* Set the GPIOXx bit */
        GPIOD->ODR = VALUE[x];
    }
}
```

Nella funzione **EXTI0_config** vengono settati i registri e bit necessari alla abilitazione della interruzione sul pin di I/O GPIOA:0. Per maggiori dettagli si consiglia di consultare il datasheet.

Le funzioni

```
    /* Set EXTI0 to the lowest priority */
    NVIC_SetPriority(EXTI0_IRQn, 0x0f);

    /* Enable EXTI0 Interrupt */
    NVIC_EnableIRQ(EXTI0_IRQn);
```

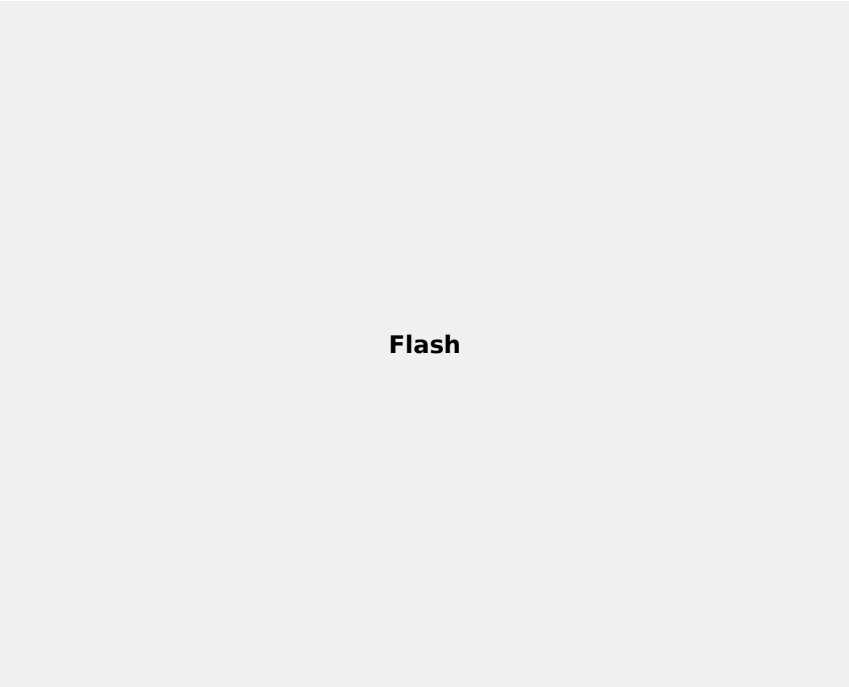
settano la priorità dell'interruzione e la abilitano. **NVIC** è l'acronimo di **N**ested **V**ector **I**nterrupt **C**ontroller, overo il controllore del vettore delle interruzioni.

EXTI0_IRQn è riconosciuta come interruzione sulla esterna 0 (consultare la sezione 10 **Interrupts and events** del datasheet a pag. 249 per maggiori informazioni). La funzione **EXTI0_IRQHandler** non può chiamarsi diversamente, è previamente dichiarata e mappata in **startup_stm32f4xx.s**.

```
__Vectors       DCD     __initial_sp               ; Top of Stack
                DCD     Reset_Handler              ; Reset Handler
                DCD     NMI_Handler                ; NMI Handler
                DCD     HardFault_Handler          ; Hard Fault Handler
                DCD     MemManage_Handler          ; MPU Fault Handler
                DCD     BusFault_Handler           ; Bus Fault Handler
                DCD     UsageFault_Handler         ; Usage Fault Handler
                DCD     0                          ; Reserved
                DCD     0                          ; Reserved

                ...
                ; External Interrupts
                ...

                DCD     RCC_IRQHandler             ; RCC
                DCD     EXTI0_IRQHandler           ; EXTI Line0
                DCD     EXTI1_IRQHandler           ; EXTI Line1
                DCD     EXTI2_IRQHandler           ; EXTI Line2
                DCD     EXTI3_IRQHandler           ; EXTI Line3
```

```
                DCD     EXTI4_IRQHandler          ; EXTI Line4
                DCD     DMA1_Stream0_IRQHandler   ; DMA1 Stream

                ...
```

A seguire si mostra un video di esecuzione del firmware appena postato.



Conclusioni

Nonostante la disponibilità delle librerie C, l'articolo era intenzionato ad descrivere il procedimento di programmazione usando le istruzioni dirette, non essendo troppo esaustivo nella descrizione dei codici di esempio, invogliando gli utenti a leggere le documentazioni per capire i perché delle istruzioni e chissà, invogliarli a fare sempre di più.

L'uso di istruzioni dirette permette di conoscere più a fondo il microcontrollore, ma è anche vero che le librerie, con la documentazione a disposizione, non nascondono nulla all'utente.

Infine, la scelta dipende da quest'ultimo. In un ambiente lavorativo, dove il tempo corre, l'uso delle librerie può essere d'obbligo. In casa, a scopo didattico, vale la pena leggere il datasheet ed addentrarsi un pò di più con lo scopo di imparare.

Spero che l'articolo sia piaciuto ai lettori e che lo abbiano trovato allo stesso tempo molto utile. Altrettanto invito alla segnalazione di errori nel caso ci fossero e mi siano sfuggiti.

Infine, un saluto a tutte le persone che credono in me, all'interno della community e non.

A presto.

Simo

Link utili

- **STM32F3xxx and STM32F4xxx Cortex-M4 programming manual**
- **ARM Cortex-M4 Overview**
- **ARM Cortex M4 Technical Reference manual**
- **ARM Cortex M3 Technical Reference manual**
- **ARM Assembly Language**