
Introduzione alla STM32F4-Discovery Board



Luigi Coppolino, Giovanni Mazzeo

Outline

- I Sistemi Embedded, cosa sono e a cosa servono
- I microcontrollori, la loro architettura
- L'architettura del processore ARM Cortex M4
- La board STM32F4-Discovery
- Il nostro obiettivo
- Materiale necessario per lo sviluppo del progetto

I Sistemi Embedded

- Un Sistema Embedded (SE) (o Sistema Dedicato) è un sistema di elaborazione progettato per eseguire un insieme ristretto di funzioni per applicazioni specifiche (industriali, aerospaziali, automotive, ecc.)
- Solitamente, come nel caso di un SE per il controllo treni, sono sistemi che operano in tempo reale, ovvero devono rispondere ad eventi esterni in tempi prestabiliti (*deadline*). Si parla in questo caso di Sistemi Real-Time.
- Esempi di SE sono nel mondo che ci circonda ogni giorno: nelle lavatrici, nelle auto, nella macchina del caffè, nei cellulari.

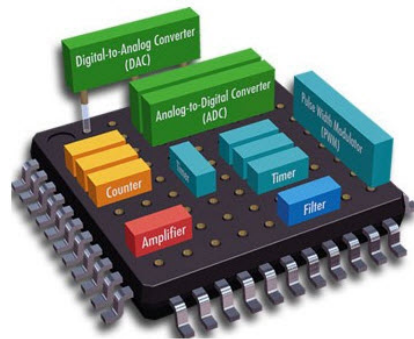
Cos'è un Microcontrollore

- I Sistemi Embedded sono basati sui Microcontrollori. I microcontrollori sono semplicemente “computer di dimensioni ridotte” all’interno di un singolo circuito integrato. Sono utilizzati per applicazioni specifiche (*Special Purpose*).
- Un microcontrollore, così come un computer *General Purpose*, ha una memoria, può essere programmato per qualsiasi computazione, riceve input e genera output.
- Nella maggior parte dei casi i microcontrollori possono essere dei System-on-Chip. Ovvero incorporano all’interno di un singolo chip tutte le unità tipiche di un calcolatore: CPU, memoria, bus, interfacce I/O, periferiche.

Embedded Systems vs General Purpose Computing

➤ Embedded Systems (Special Purpose)

- Eseguono singole applicazioni già note in fase di sviluppo del sistema
- Spesso hanno vincoli sul tempo di esecuzione. Non per forza, le performance devono essere alte
- In molte applicazioni hanno *hard-constraints* sul consumo di potenza



➤ General Purpose

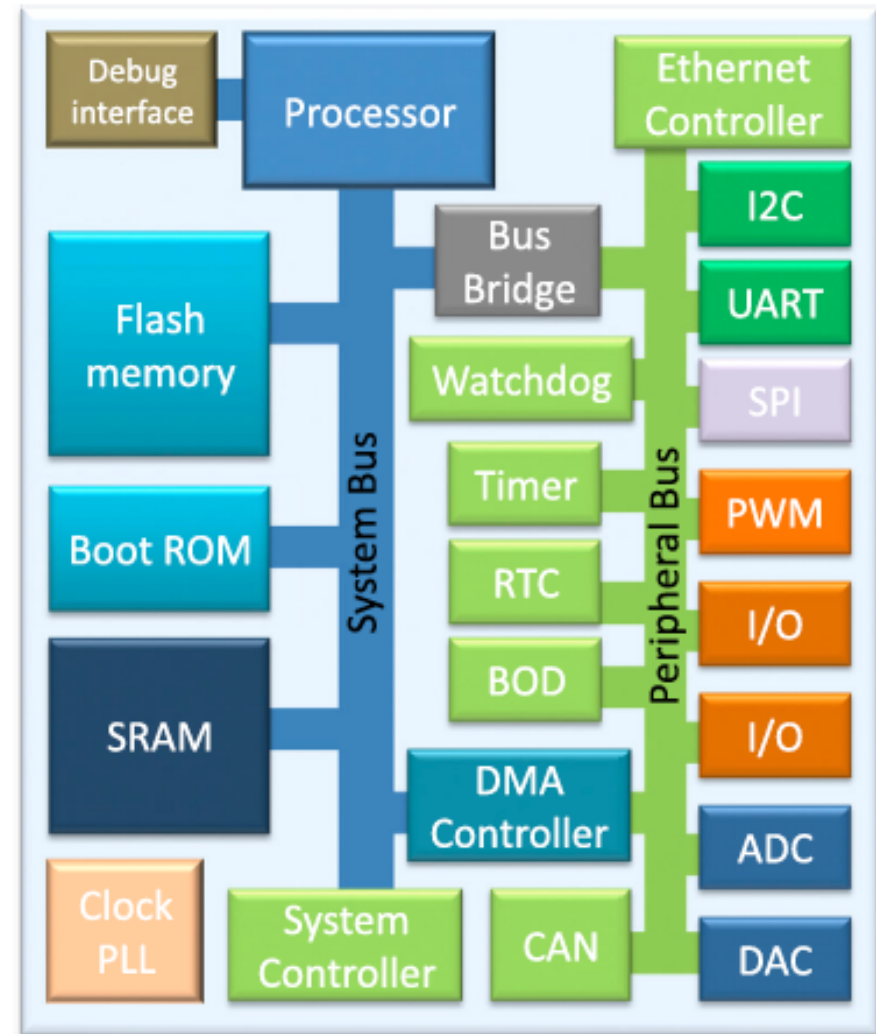
- Eseguono qualsiasi tipo di applicazioni
- Faster is always better
- Possono essere sempre riprogrammati da un utente finale



Architettura Generale di un Microcontrollore

➤ Ogni microcontrollore integra:

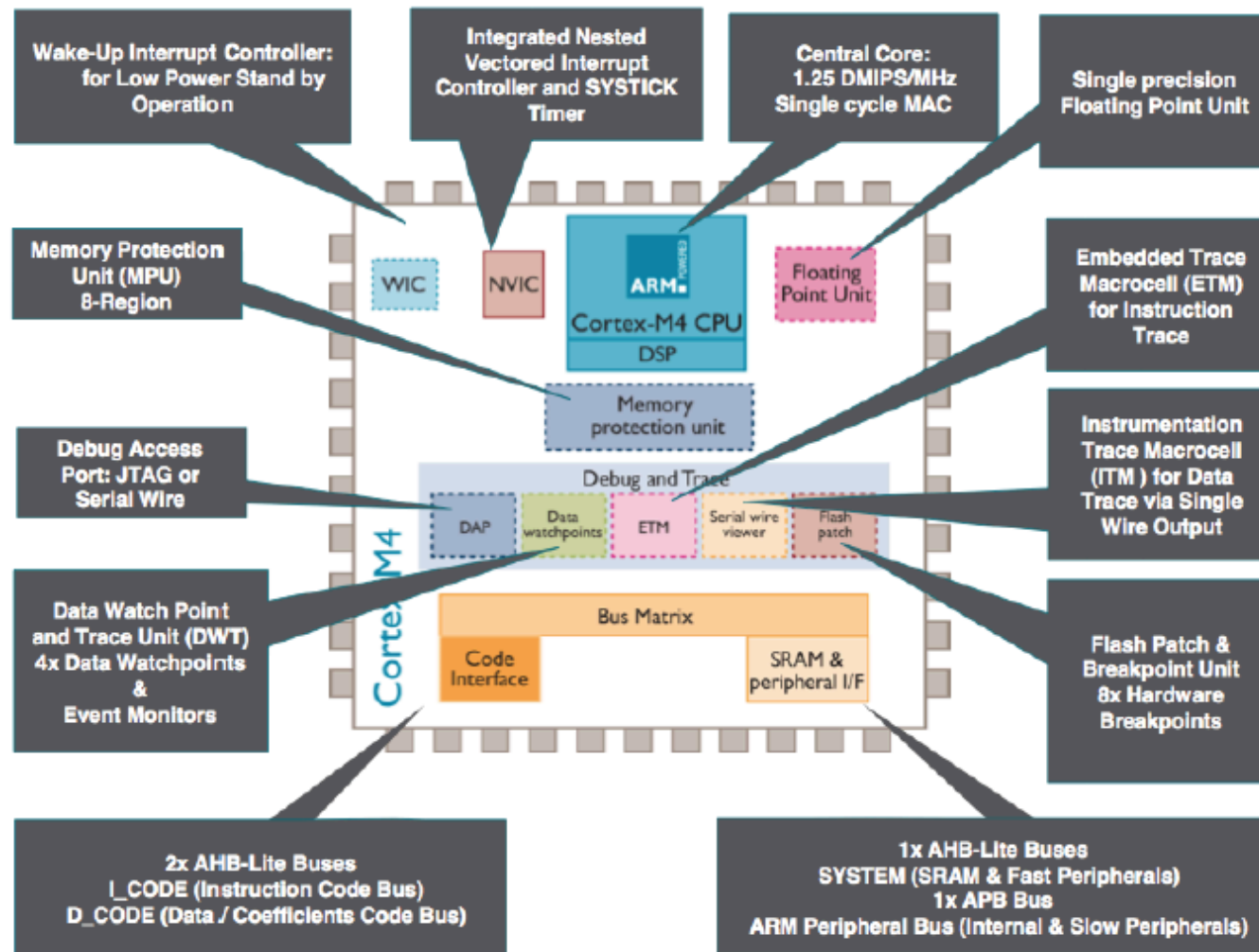
- Il Processore (e.g. Intel, Arm)
- Una memoria (SRAM o DRAM)
- Una memoria flash
- Bus di comunicazione (e.g. Advanced Microcontroller Bus Architecture (AMBA)). Quasi sempre due bus a diverse frequenze di clock.
- Interfacce di Comunicazione (I2C, SPI, UART, etc.)
- ADC/DAC
- Clock



Il Processore ARM Cortex M4

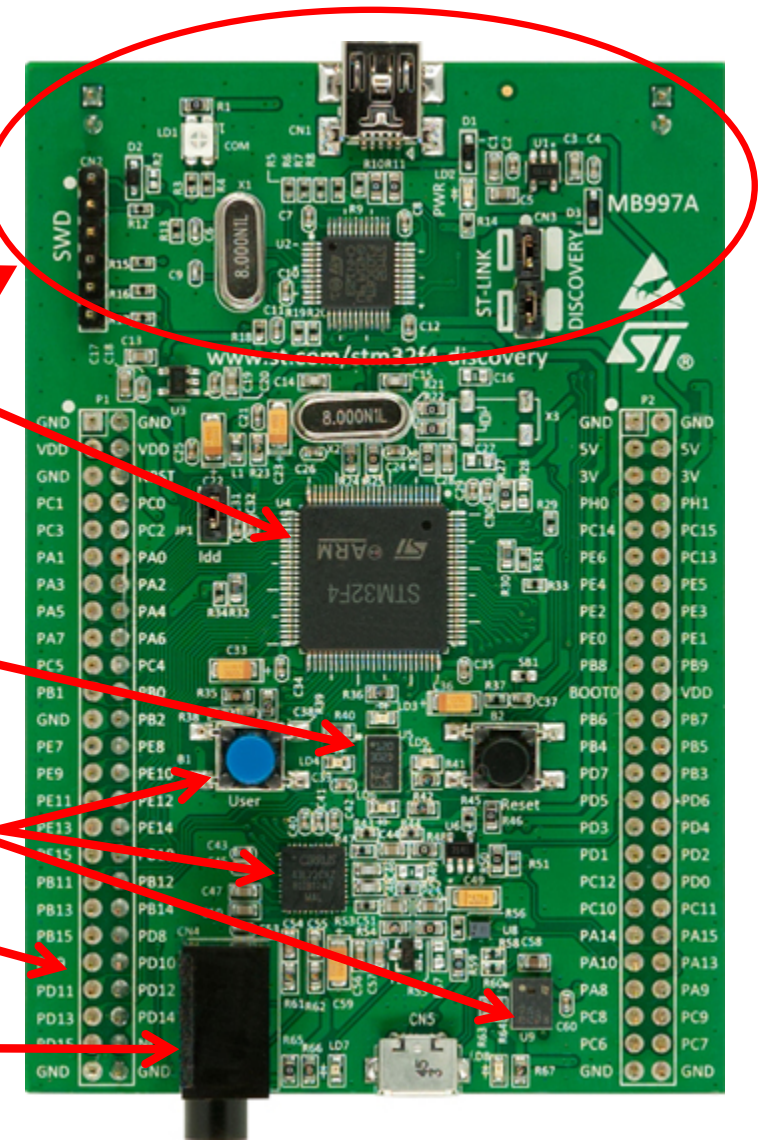
- Il processore ARM Cortex M4 e più in generale la famiglia M-series è un processore utilizzato per microcontrollori che garantisce bassi consumi di potenza con buone prestazioni
- E' un processore che implementa il set di istruzioni (ISA) Thumb a 16 bit che può essere visto come una forma compressa di un sottoinsieme dell'ARM Instruction Set (a 32 bit).
- Il processore presenta una pipeline a 3 stadi
- Gestisce le interruzioni in maniera innestata con meccanismi quale il Wake Up Interrupt Controller (WUIC) che permettono di ridurre il consumo di potenza

Il Processore ARM Cortex M4



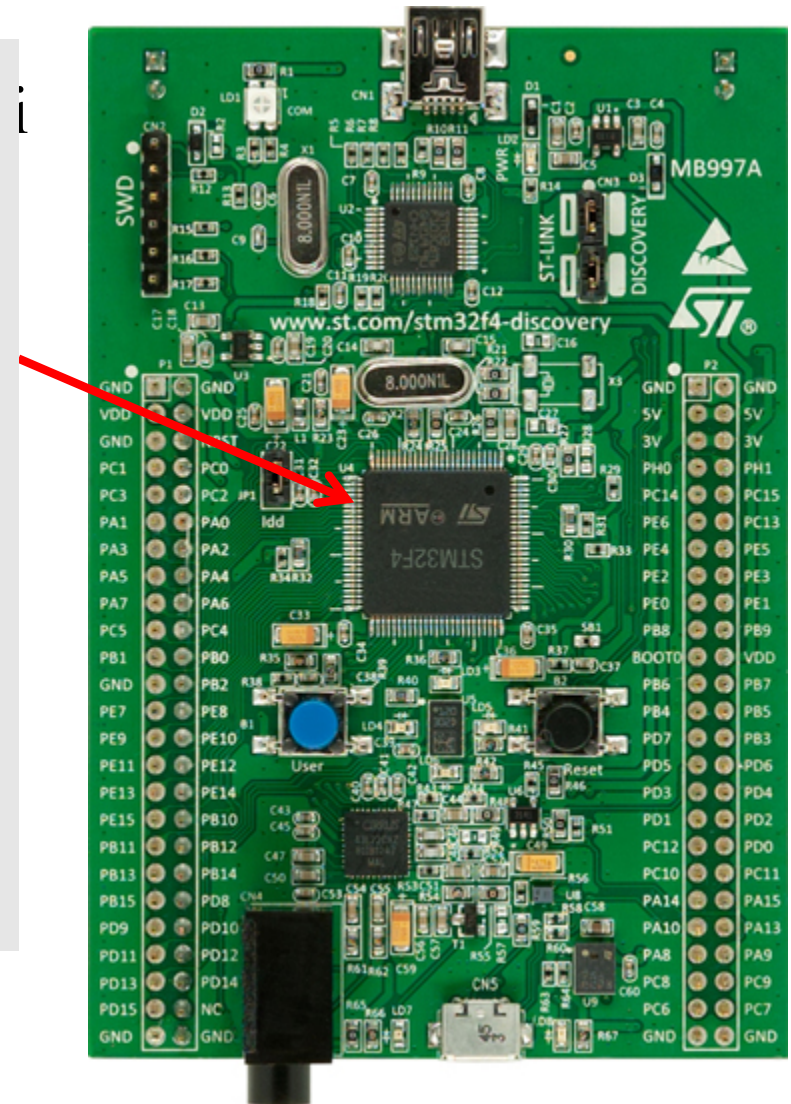
La board di sviluppo STM32F4-Discovery

- La STM32F4-Discovery è una scheda di sviluppo che contiene al suo interno:
 - Il microcontrollore STM32F4 basato sul ARM Cortex M4
 - Accelerometro 3-axis
 - Digital microphone
 - Audio DAC
 - Debugger
 - Pin di interfacciamento con il mondo esterno (GPIO)
 - Due pulsanti e 4 LED
 - Jack audio 3.5



Il Microcontrollore STM32F4

- Il microcontrollore STM32F4 presenta:
 - ARM M4 core processor 168MHz
 - 1MB Flash
 - 192KB SRAM
 - > 80 I/O Pins
 - 13 Timers
 - Serial Communications: 6 UARTs, 3 SPI, 2 I2C
 - USB OTG Controller
 - External memory controller
 - Internal DMA System
 - Ethernet Controller
 - SD Controller



Concetti di Base per la Programmazione dei Microcontrollori

- Programmare un microcontrollore significa istruirlo a fare una specifica funzione di interesse
- Per fare ciò noi scriveremo programmi in linguaggio C che saranno poi compilati (quindi tradotti in linguaggio macchina) per il nostro microcontrollore
- Il programma “tradotto” in linguaggio macchina sarà poi caricato nella memoria Flash del microcontrollore. Questa ha la caratteristica di mantenere la programmazione anche quando l'alimentazione al microcontrollore viene spenta
- Durante l'esecuzione la RAM conterrà i dati, ovvero tutte le variabili che si utilizzano all'interno del programma

Concetti di Base per la Programmazione dei Microcontrollori

- La programmazione dei microcontrollori si basa sulla scrittura in specifici registri di memoria (ad indirizzi prestabiliti)
- Ogni periferica/unità ha il suo set di registri dedicati dal quale andrà a leggere per sapere come si deve comportare
- Ogni periferica/unità di un microcontrollore per funzionare necessita di essere inizializzata

Sporchiamoci le mani: la Prima Programmazione della STM32F3

- **Obiettivo:** Si vuole programmare la scheda affinché premendo un pulsante si accenda un led.
- **Di cosa abbiamo bisogno?**
 - La STM32F3 discovery
 - Eclipse IDE
 - ARM-GCC toolchain
 - Debugger OpenOCD
 - ST-LINK

Intro

- Al fine di tenere un ambiente di sviluppo ordinato creeremo la cartella “ArmEnviroment” nel PATH principale “C:\”
- Questa operazione non è necessaria ma ci permetterà nelle varie esercitazioni di conoscere il PATH in cui sono localizzati i vari strumenti
- L’installazione dell’ambiente di sviluppo per la STM32F3 sarà effettuata attraverso gli eseguibili scaricabili da Google Drive:

<https://drive.google.com/open?id=0B5pdTArYAQ51Zi03ZkdOOUY3TTg>

- Tali eseguibili sono utili SOLO per installazioni su macchine Windows a 64bit
- Coloro i quali necessitassero di eseguibili per macchine a 32bit, potranno trovare opportuni link nel resto della presentazione

Java Runtime Environment

- L'ambiente di sviluppo che utilizzeremo necessita della Java Runtime Environment
- Eseguire dunque [jre-8u121-windows-x64.exe](#) e procedere nell'installazione
- Altrimenti scaricare da qui l'eseguibile di interesse

<http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

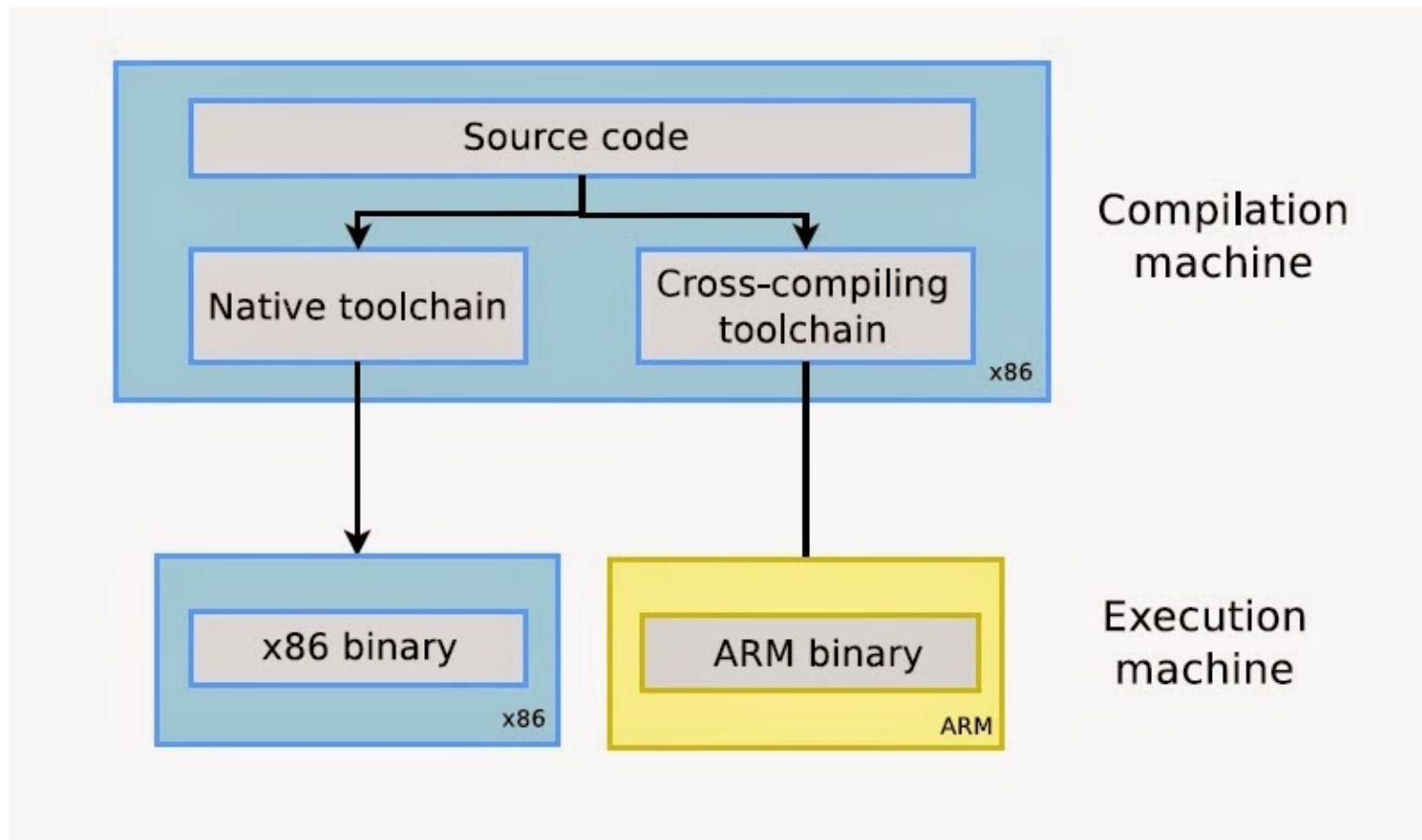
Eclipse

- Eclipse è un Integrated Development Environment (IDE) ovvero un ambiente di programmazione open source. Eclipse è utilizzato per programmare in diversi linguaggi di programmazione (C, C++, Java, Rust, PHP, JavaScript, ...)
- Eclipse non necessita di una procedura di installazione
- Scompattare l'archivio [eclipse-cpp-neon-3-win32-x86_64.zip](#) in C:\ArmEnvironment
- Altrimenti, scaricare Eclipse IDE for C/C++ developers da qui: <http://www.eclipse.org/downloads/packages/>

ARM-GCC Cross Toolchain 1/4

- Il programma che scriveremo per controllare il led della scheda è scritto nel linguaggio di programmazione C
- Tale programma dovrà essere compilato per generare il codice macchina che il microcontrollore saprà eseguire
- Il compilatore nativo (e.g. gcc) genererebbe il codice macchina per l'architettura su cui si sta eseguendo la compilazione
- Noi dobbiamo compilare il codice per l'architettura specifica del microcontrollore
- A tal fine necessitiamo un Cross-Compiler il quale permette di generare un file binario eseguibile su di un'architettura diversa da quella della macchina su cui è stato lanciato il cross compiler

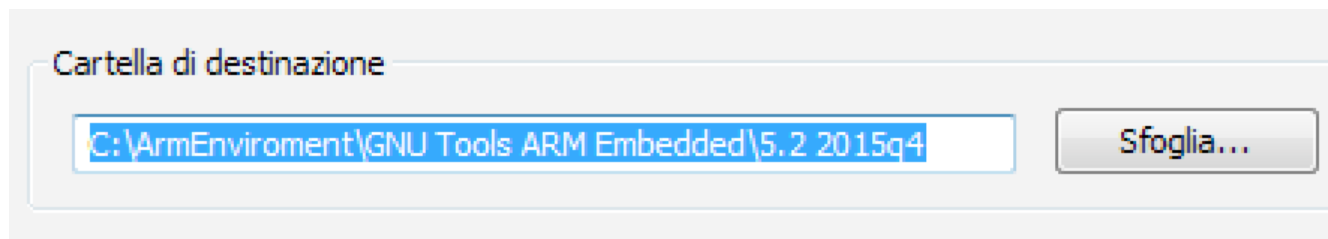
ARM-GCC Cross Toolchain 2/4



ARM-GCC Cross Toolchain 3/4

- Il primo passo da eseguire consiste nell'installazione del compilatore e degli strumenti di debugging per l'architettura ARM.
- Installare in C:\ArmEnvironment la nostra toolchain ARM armgcc-compiler.exe
- **N.B. Prima di porre fine all'installazione spuntare “Add path to environment variable”**
- E' possibile anche scaricare la tool-chain qui:

<https://launchpad.net/gcc-arm-embedded/+download>



ARM-GCC Cross Toolchain 4

- Successivamente è necessario installare, in Eclipse, il plug-in relativo al cross compiler GCC per la piattaforma ARM Cortex.
- L'installazione può essere eseguita nel seguente modo:
 - Scompattare in C:\ArmEnvironment il compresso plugin-eclipse-armgcc.zip. Tale archivio lo si può trovare o nella folder Drive o al seguente link <http://gnuarmeclipse.sourceforge.net/updates>
 - Andare in Eclipse

N.B. al primo avvio Eclipse chiederà di specificare il path relativo al workspace, quindi, per avere un facile accesso ai file di compilazione è consigliato specificare come path “C:\ArmEnvironment\workspace”

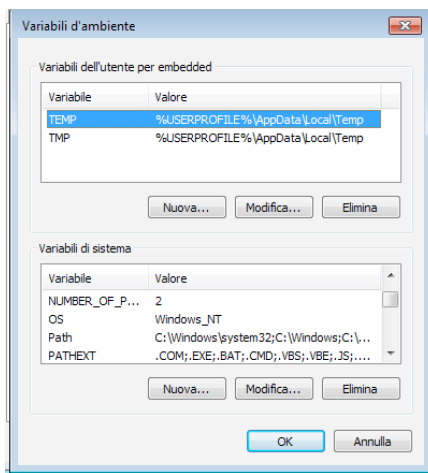
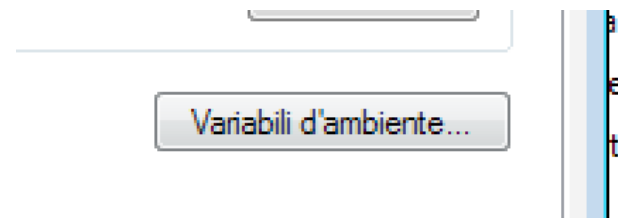
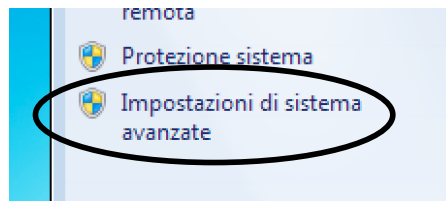
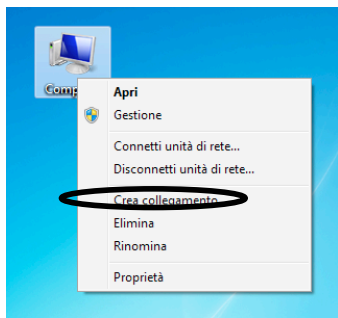
- Aprire il menu **Help->Install New Software** e cliccare su “Add” e poi “Local”
- Indicare il path del plugin scompattato in precedenza.
- **N.B. Non selezionare la sottocartella plugins ma la root della cartella scompattata**

Build Tools

- La compilazione dei driver e delle dipendenze necessitano di alcuni comandi presenti in ambiente linux.
- I comandi linux vengono aggiunti all'ambiente windows mediante l'installazione di "MinGw"
- Installare in C:\ArmEnvironment [mingw-w64-install.exe](#) presente su Drive
- Oppure scaricarlo da qui: <https://sourceforge.net/projects/mingw-w64/>
- Andare nella cartella di installazione di MinGw all'interno della sottocartella bin (o bin-x64) e rinominare il file "mingw-make" a "make"
- Una volta completata l'installazione è necessario settare la variabile di ambiente PATH affinché punti alla cartella dove sono presenti i binari di MinGW

Set Environment Variable

- Per settare manualment il PATH
“C:\ArmEnviroment\[InstallationFolderName]\bin” alle variabili d’ambiente di windows bisogna andare nelle impostazioni di sistema avanzate.



Aggiungere alla variabile “Path” il percorso “C:\Program Files\mingw-w64\[NomeCartella]\mingw64\[bin o bin-x64]”

NB:Se la variabile “Path” contiene altri valori utilizzare il simbolo “;” come separatore

ST-Link/V2

- ST-Link/v2 è il il modulo software che svolge le funzioni di linker.
- E' utilizzato per caricare il file binario (senza effettuare debug), compilato con il cross compiler, all'interno del cortex M4
- Installare in C:\ArmEnvironment STM32 ST-LINK Utility v4.0.0 setup.exe
- O scaricare il linker “STM32 ST-LINK utility” dall'indirizzo:
<http://www.st.com/web/en/catalog/tools/PF258168>

Debugger

- Il debugger è fondamentale nello sviluppo di software complessi.
- Permette di scorrere l'esecuzione del codice “step-by-step” e di legger i valori intermedi di tutte le variabili utilizzate nel codice
- Il più famoso debugger è il GNU GDB utilizzato per verificare programmi C/C++ (<https://www.gnu.org/software/gdb/>)
- Si possono realizzare due tipologie di attività debugging:
 - Local - Il debugging di un programma che esegue in locale sullo stesso sistema in cui si effettua il debug
 - Remote – Il debugging di un programma che esegue su un sistema (detto *target*) differente da quello (detto *host*) su cui si esegue il debug

STM32F4 Debugging

- Nel caso della STM32F4 ovviamente realizzeremo un *remote debugging*. Faremo uso di OpenOCD.
- Questo si basa su una comunicazione client/server realizzata tra la scheda e il nostro calcolatore
 - Un server OpenOCD, avente un file di configurazione specifico per la scheda su cui si vuole eseguire il programma, sarà lanciato sulla scheda e fornirà dunque informazioni al client sui valori nella memoria del microcontrollore.
 - Un client sarà lanciato e comunicherà con il server per ottenere informazioni da fornire allo sviluppatore

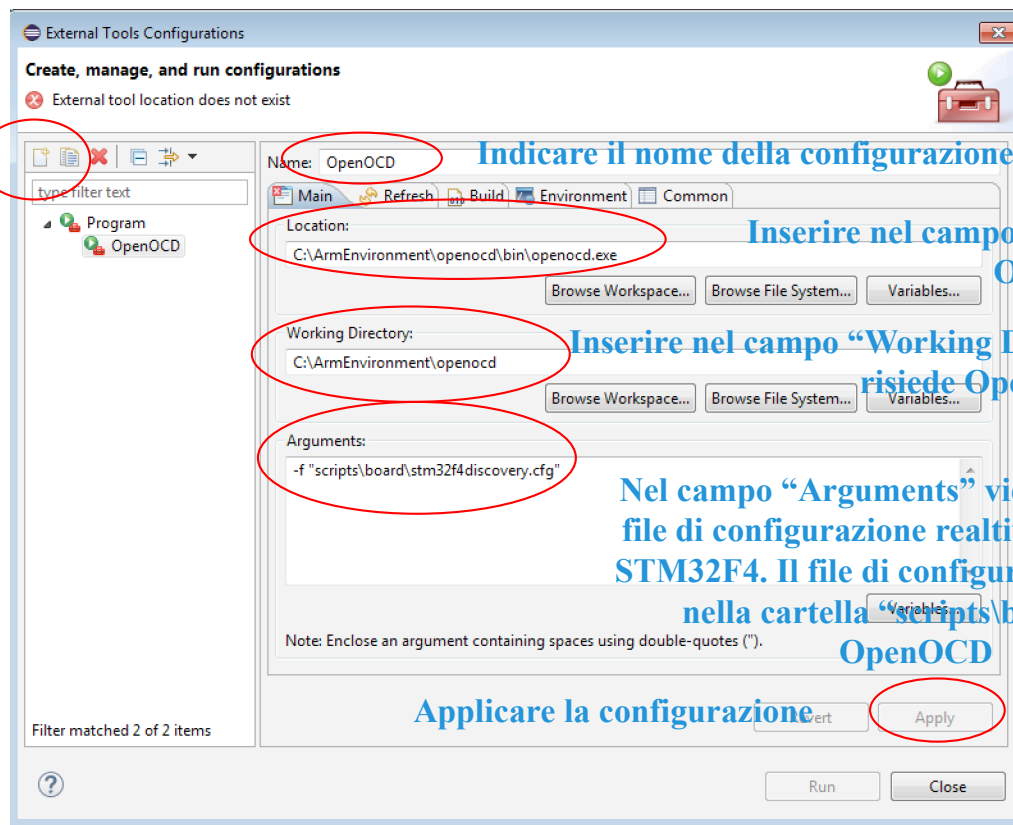
Debugger GDB and OpenOCD

- OpenOCD è un debugger universale On-Chip.
- Attraverso un driver interno interagisce con le board che utilizzano il protocollo ST-Link
- Scompattare il pacchetto openocd-0.9.0 dal link:
<http://www.freddiechopin.info/en/download/category/4-openocd>
 - Estrarre il pacchetto nella cartella “C:\ArmEnvironment”
 - Rinominare la cartella di “openocd-0.9.0” in “openocd”

Configurazione di OpenOCD in Eclipse

- È possibile configurare Eclipse, in modo che utilizzi OpenOCD, attraverso il menu: *Run -> External Tools -> External Tools Configurations*

Aggiungere una nuova configurazione



Indicare il nome della configurazione

Inserire nel campo "Location" l'eseguibile di OpenOCD

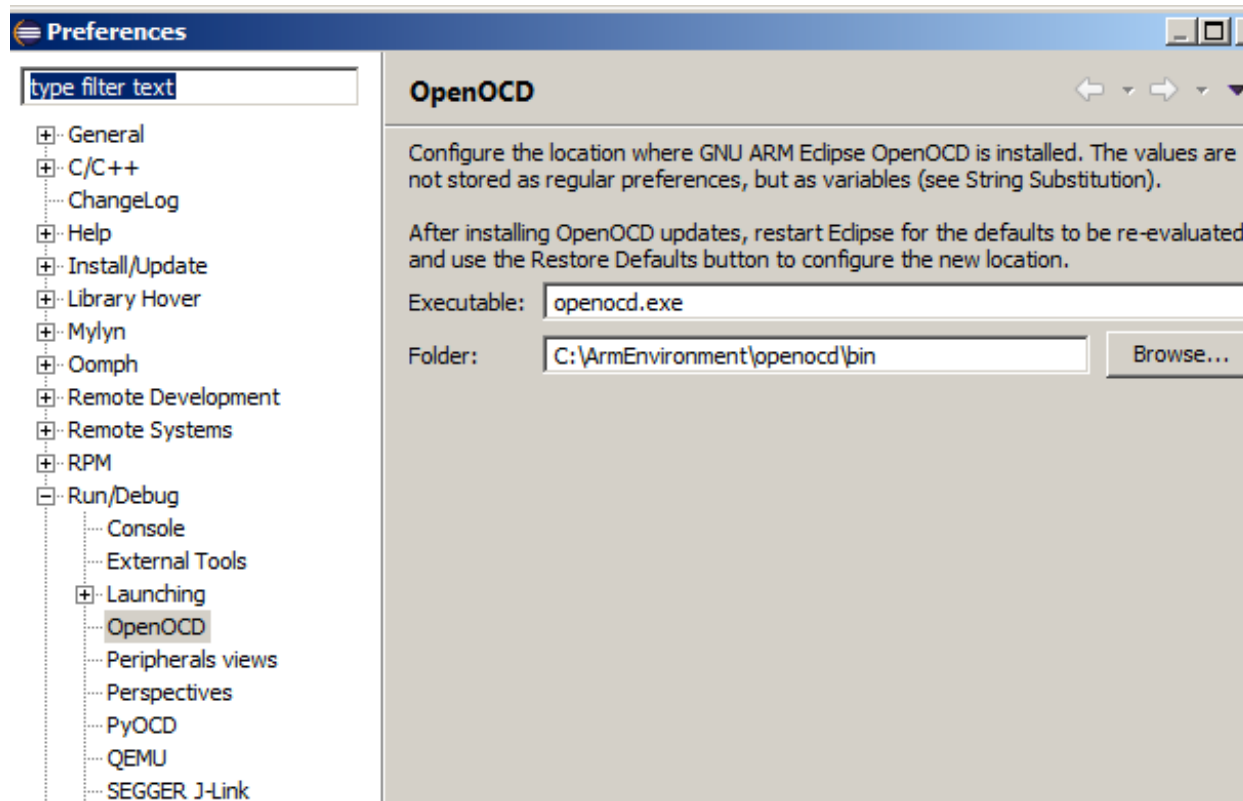
Inserire nel campo "Working Directory" la directory dove risiede OpenOCD

Nel campo "Arguments" viene indicato il file di configurazione relativo alla board STM32F4. Il file di configurazione risiede nella cartella "scripts\board\" di OpenOCD

Applicare la configurazione

Configurazione di OpenOCD in Eclipse

- È possibile configurare Eclipse, in modo che utilizzi OpenOCD andando a definire il path dove abbiamo installato il debugger:
Window → *Preferences* → *Run/Debug* → *OpenOCD*



Configurazione di OpenOCD in Eclipse

- Per la configurazione del client GDB in Eclipse, attraverso il menu selezionare il ragnò di *Debug->Debug Configurations -> GDB OpenOCD Debugging*

The screenshot displays the Eclipse IDE interface. On the left, the 'Debug Configurations' dialog is open, showing a tree view of configurations. The 'GDB OpenOCD Debugging' category is expanded, and 'OpenOCD-FirstAws' is selected. The main panel shows the configuration for 'OpenOCD-FirstAwsProjectDebug'. The 'OpenOCD Setup' section includes the following fields:

- Name:** OpenOCD-FirstAwsProjectDebug
- Start OpenOCD locally:**
- Executable:** `${openocd_path}/${openocd_executable}` (with 'Browse...' and 'Variables...' buttons)
- GDB port:** 3333
- Telnet port:** 4444
- Config options:** `-f "board\stm32f3discovery.cfg"` (highlighted with a red circle)
- Allocate console for OpenOCD:**
- Allocate console for the telnet connection:**

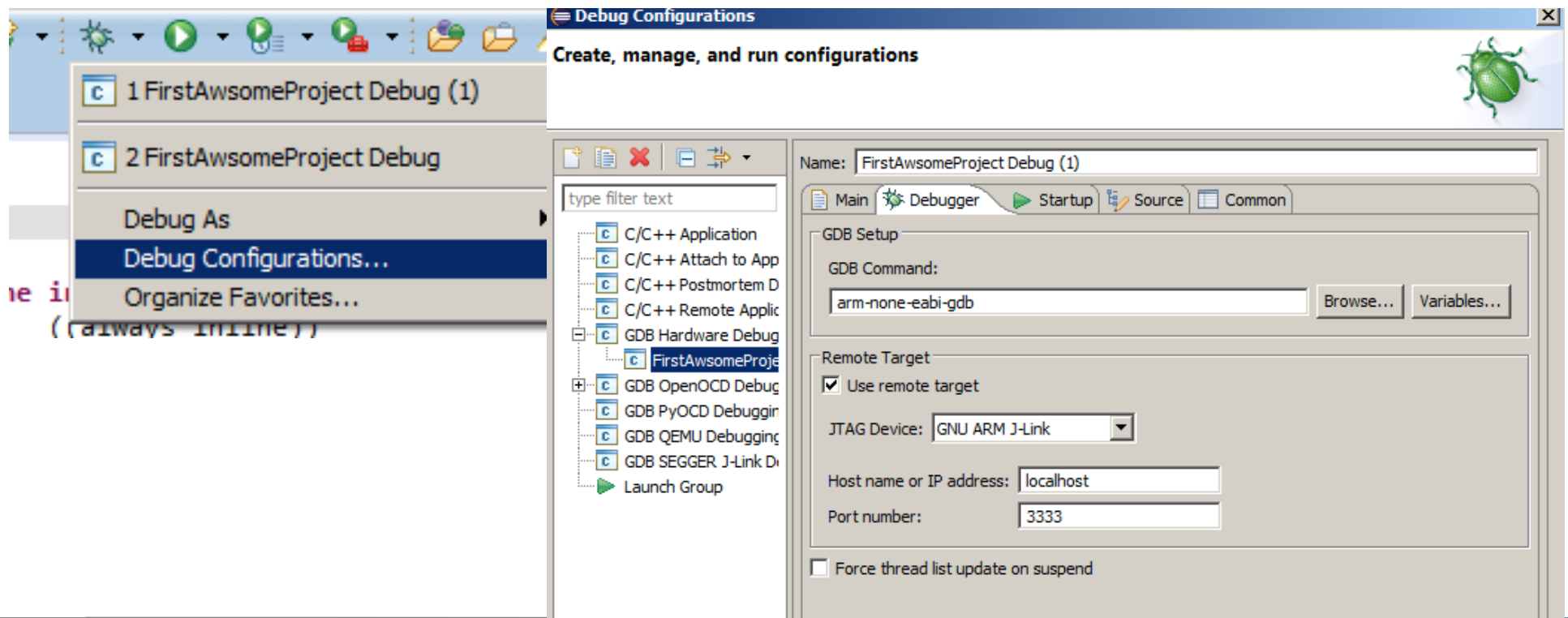
The 'GDB Client Setup' section includes:

- Executable:** `${cross_prefix}gdb${cross_suffix}` (with 'Browse...' and 'Variables...' buttons)
- Other options:** (empty text box)
- Commands:** `set mem inaccessible-by-default off`

At the bottom left, there is a logo for 'FITNESS' and the text 'The Fault and Intru'.

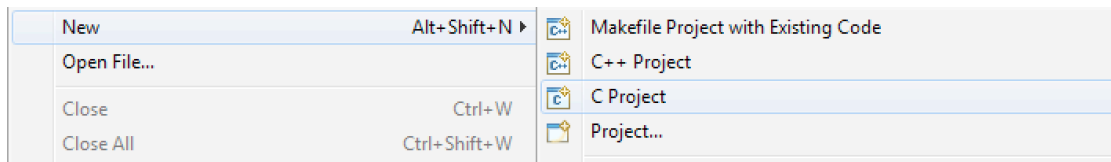
Configurazione client GDB

- Per la configurazione del client GDB in Eclipse, attraverso il menu selezionare il ragnò di *Debug->Debug Configurations*
- Modificare solo la scheda “debugger” aggiungendo quanto riportato nell’immagine

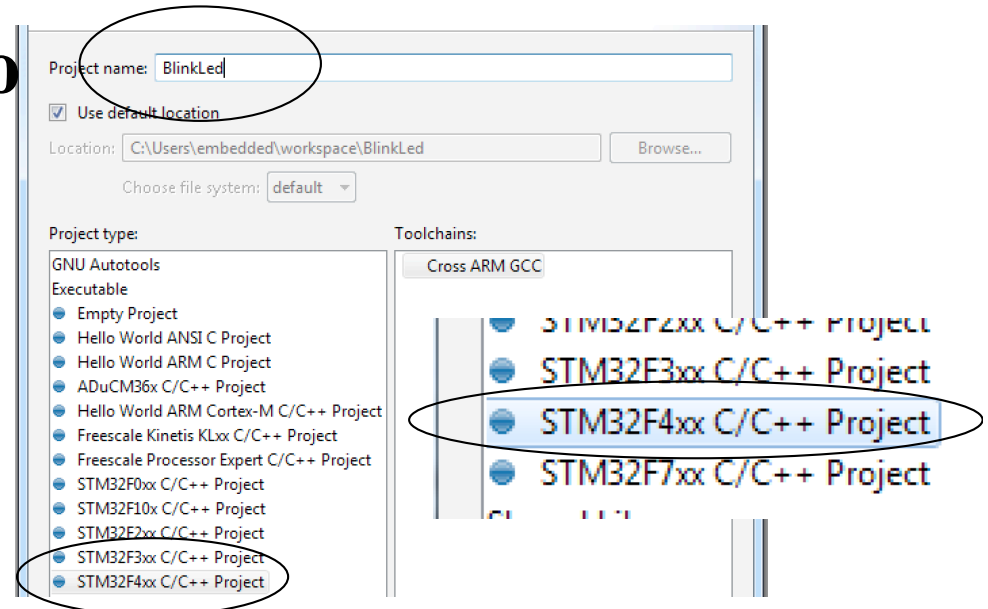


BlinkLed 1/

- Nei sistemi digitali il blink led costituisce l'esempio "Hello World"
- Creare un nuovo progetto dal menù "File->New->C Project"



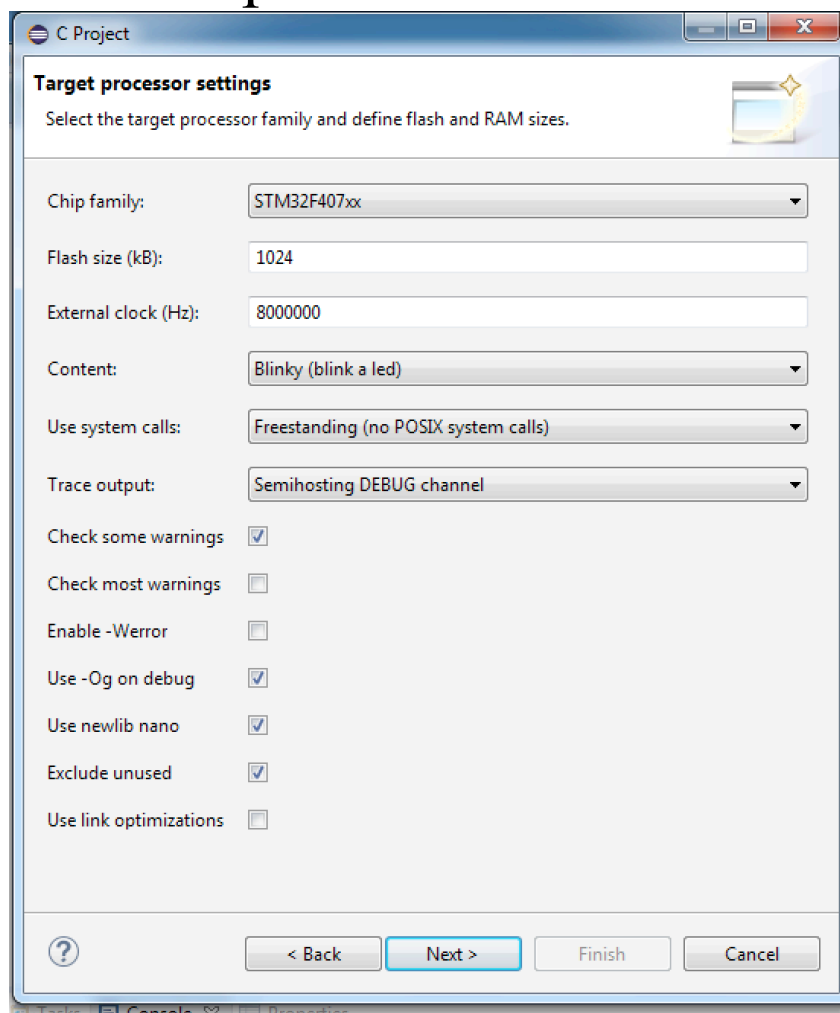
Inserire il nome del progetto



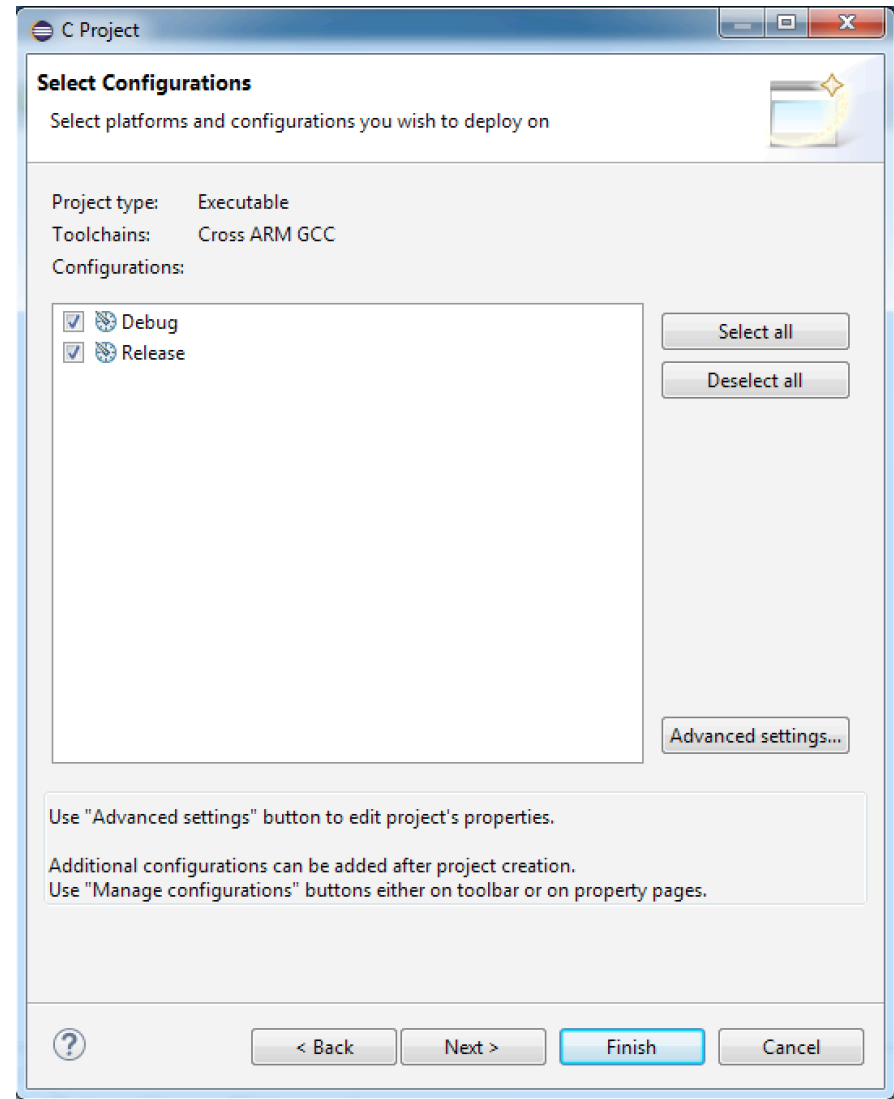
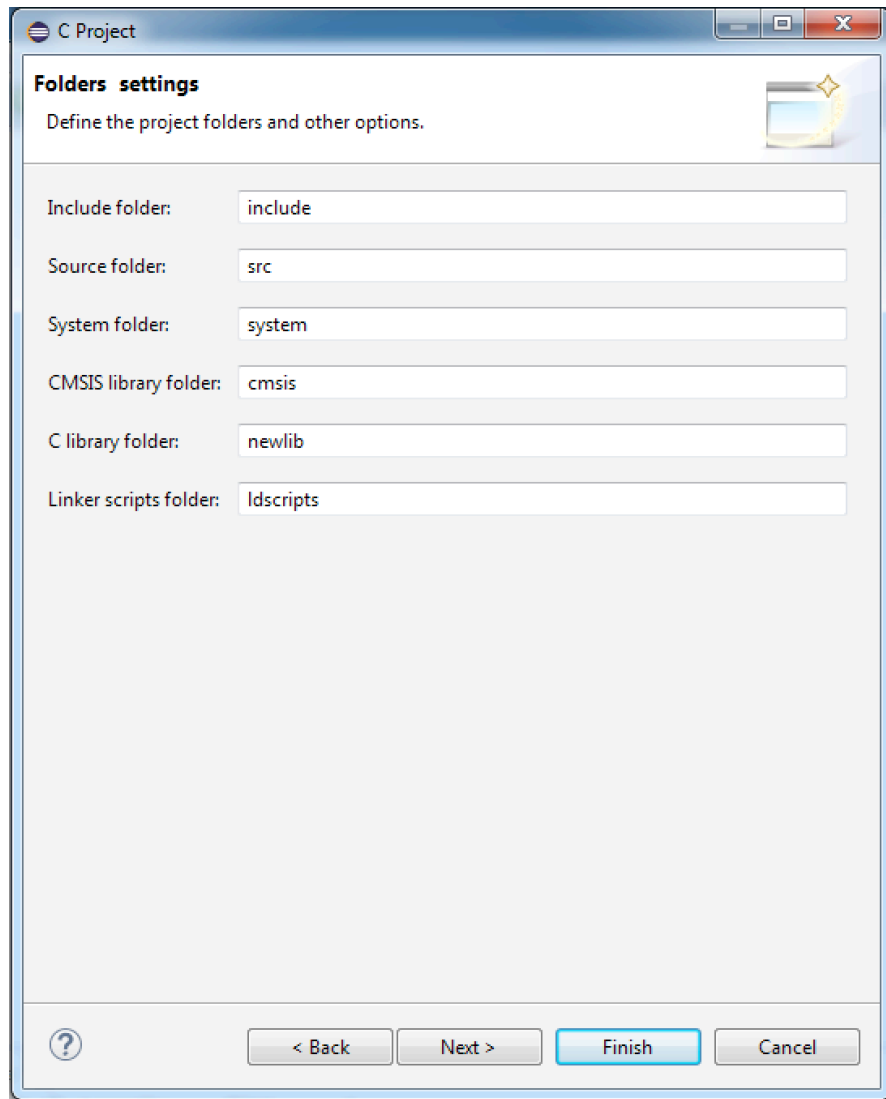
**Selezionare "STM32F4xx"
come tipo di progetto**

BlinkLed 2/

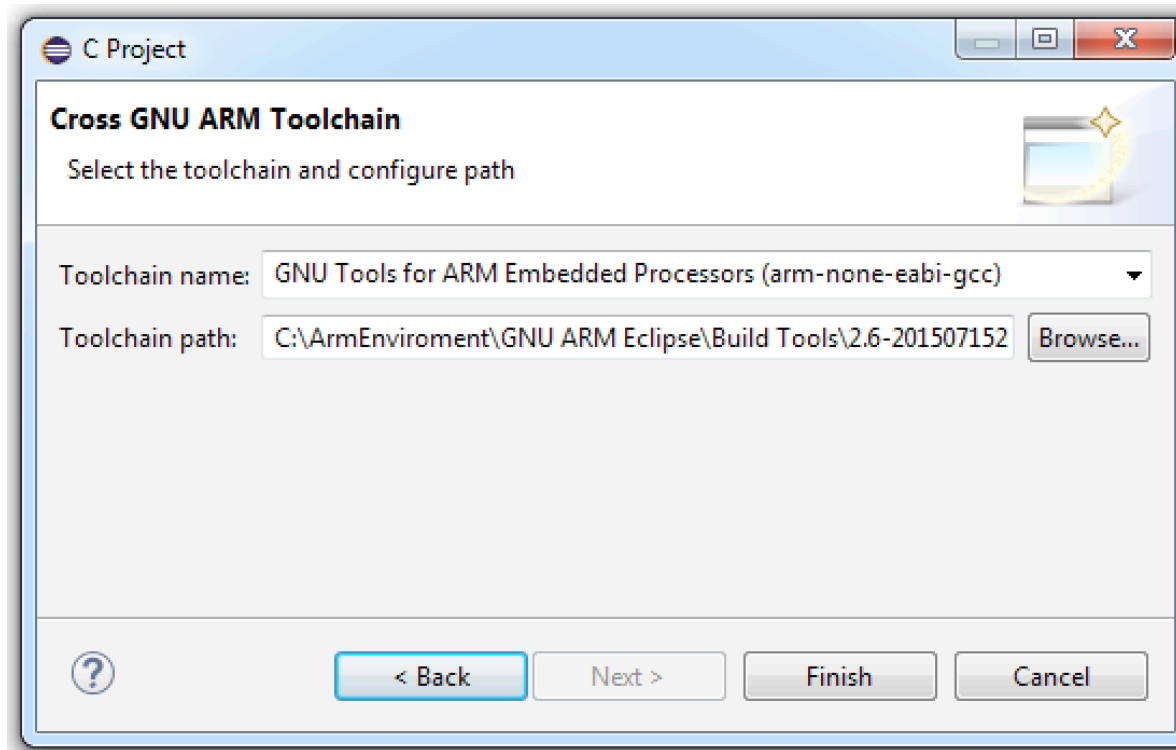
- Verificare che i vari campi siano settati come in figura:



BlinkLed 3/

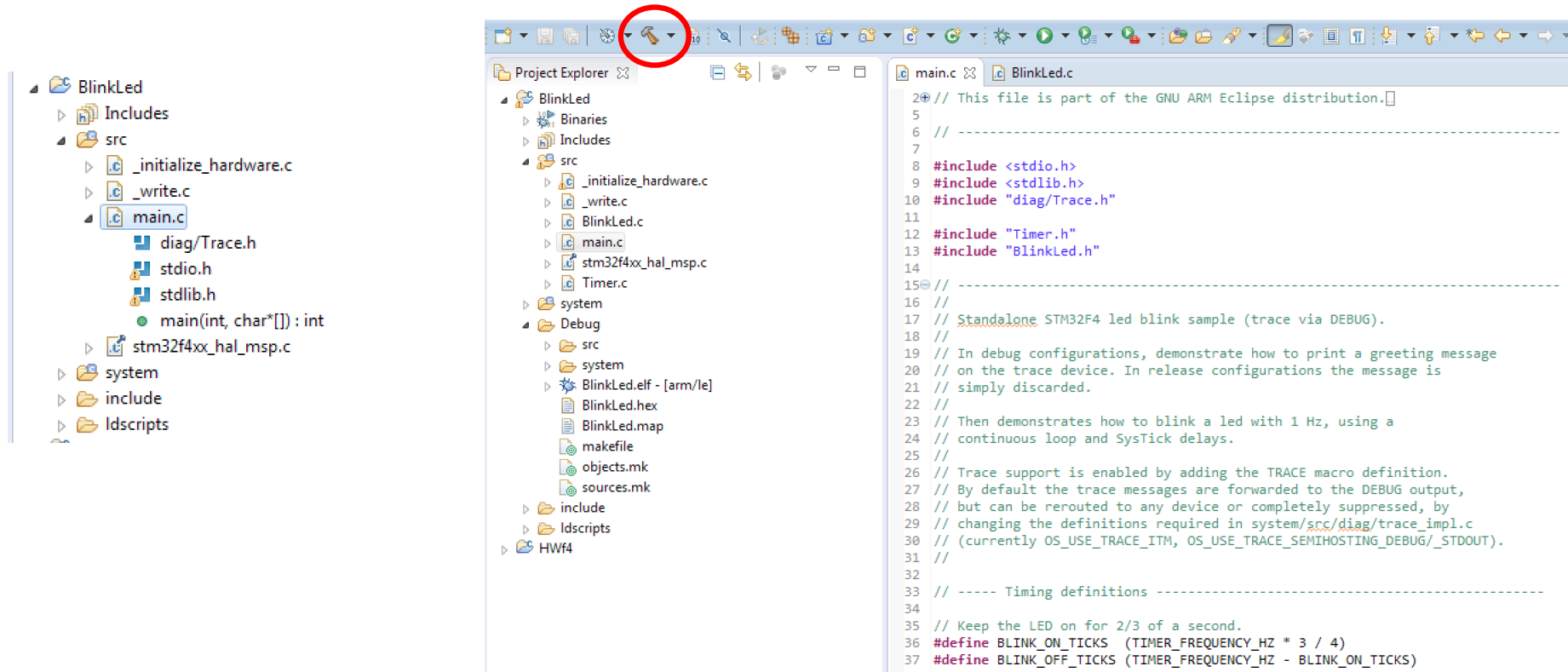


BlinkLed 4/



BlinkLed 5/

- Compilare il progetto cliccando sul martello



The screenshot displays the Eclipse IDE interface for the BlinkLed project. On the left, the Project Explorer shows the project structure with folders for Includes, src, system, and Idscripts. The main.c file is selected in the src folder. The central toolbar contains various icons, with the compile button (a hammer icon) circled in red. The main editor window shows the source code for main.c, which includes headers for stdio.h, stdlib.h, diag/Trace.h, Timer.h, and BlinkLed.h. The code contains comments and defines macros for BLINK_ON_TICKS and BLINK_OFF_TICKS.

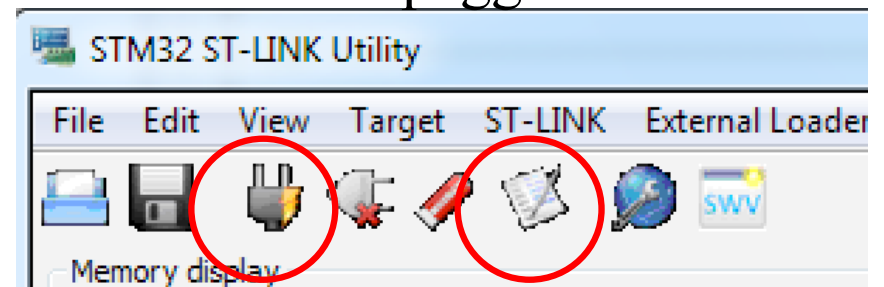
```
20 // This file is part of the GNU ARM Eclipse distribution.
5
6 // -----
7
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include "diag/Trace.h"
11
12 #include "Timer.h"
13 #include "BlinkLed.h"
14
15 // -----
16 //
17 // Standalone STM32F4 led blink sample (trace via DEBUG).
18 //
19 // In debug configurations, demonstrate how to print a greeting message
20 // on the trace device. In release configurations the message is
21 // simply discarded.
22 //
23 // Then demonstrates how to blink a led with 1 Hz, using a
24 // continuous loop and SysTick delays.
25 //
26 // Trace support is enabled by adding the TRACE macro definition.
27 // By default the trace messages are forwarded to the DEBUG output,
28 // but can be rerouted to any device or completely suppressed, by
29 // changing the definitions required in system/src/diag/trace_impl.c
30 // (currently OS_USE_TRACE_ITM, OS_USE_TRACE_SEMIHOSTING_DEBUG_STDOUT).
31 //
32
33 // ----- Timing definitions -----
34
35 // Keep the LED on for 2/3 of a second.
36 #define BLINK_ON_TICKS (TIMER_FREQUENCY_HZ * 3 / 4)
37 #define BLINK_OFF_TICKS (TIMER_FREQUENCY_HZ - BLINK_ON_TICKS)
```

BlinkLed 6/

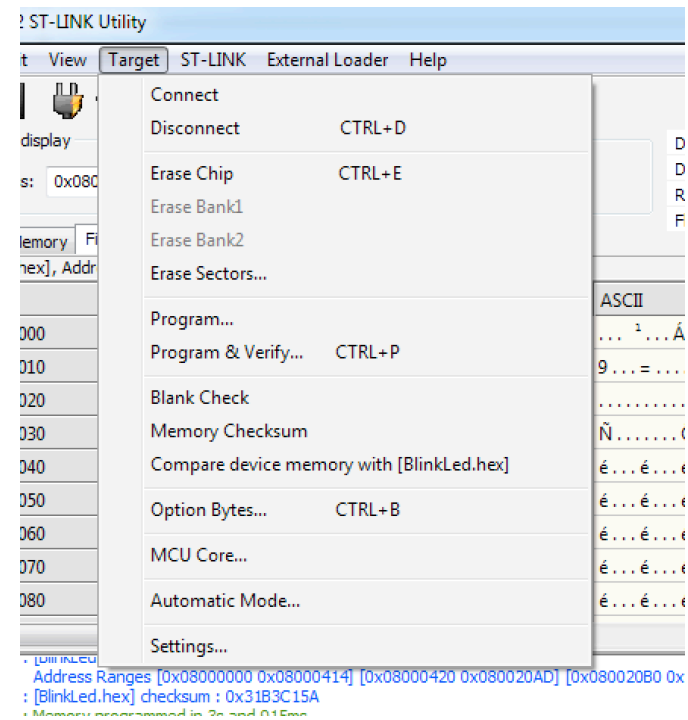
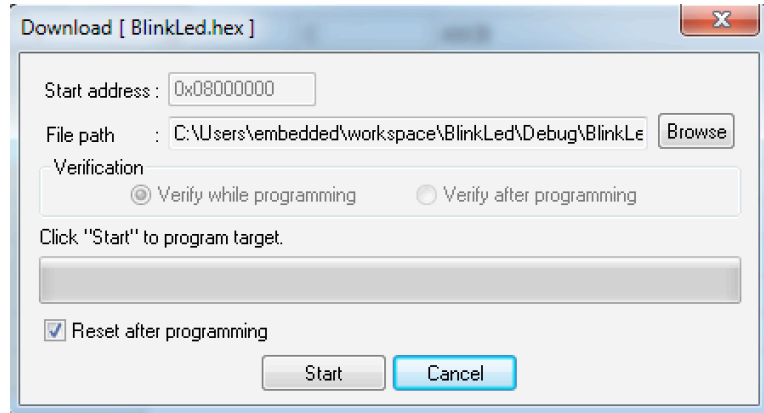
- Se la compilazione è andata a buon fine, provare a “flashare” la scheda STM32F3
- Collegarla, avviare STLINK, connetterlo alla scheda cliccando sulla icona di presa elettrica.
- Aprire (in file->open) il file da caricare sulla scheda. Tale file lo si può trovare nella cartella del progetto di Eclipse sotto (debug). Tale file ha estensione .hex
- Una volta aperto, lanciare il “program verify” tramite l’icona evidenziata.
- Scollegare e ricollegare la scheda per vedere se il led lampeggia

```
Invoking: Cross ARM GNU Create Flash Image  
arm-none-eabi-objcopy -O ihex "BlinkLed.elf" "BlinkLed.hex"  
Finished building: BlinkLed.hex  
  
Invoking: Cross ARM GNU Print Size  
arm-none-eabi-size --format=berkeley "BlinkLed.elf"  
text data bss dec hex filename  
8545 160 420 9125 23a5 BlinkLed.elf  
Finished building: BlinkLed.siz
```

14:37:26 Build Finished (took 13s.369ms)



BlinkLed 7/



Testare OpenOCD in Eclipse

- Una volta configurato il debugger, compilato il progetto di esempio, e collegato la scheda, lanciare il debug da Eclipse premendo sul ragnò di debug.

N.B. Chiudere ST-Link nel caso in cui lo si sia avviato in precedenza. In caso contrario, Eclipse riporterà un errore nel collegamento con la scheda

- Windows potrà chiedere che si accetti un rischio per la sicurezza, accettare!
- Eclipse chiederà se si vuole cambiare prospettiva su quella di debug, accettare!
- Se si è configurato il tutto correttamente il cambio di prospettiva di Eclipse sarà effettuato e vorrà dire che il debug potrà essere iniziato.

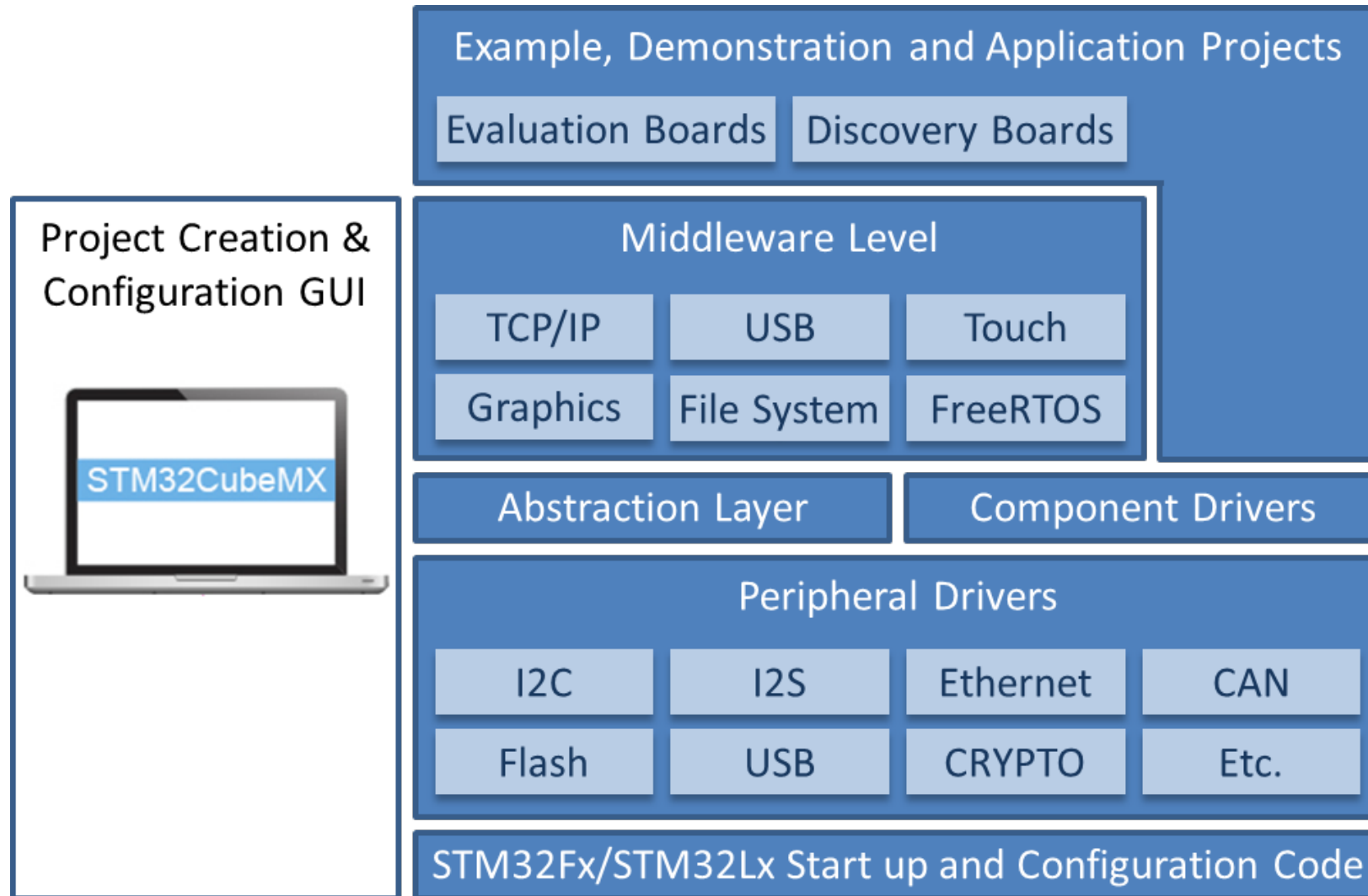
Esempi di Progetto da cui poter Iniziare

- La ST mette a disposizione degli sviluppatori un insieme di progetti utili per iniziare a sviluppare una specifica periferica.
- Il set di progetti che va sotto il nome di STMCubeMX è disponibile al seguente indirizzo:
 - <http://www.st.com/en/embedded-software/stm32cubef3.html>
 - Dovremo dunque creare un nuovo progetto vuoto in Eclipse e importare gli opportuni file

HAL 1/2

- Gli esempi riportati in STMCubeMX fanno uso del **Hardware Abstraction Layer (HAL)**
- Come il nome suggerisce, l' HAL introduce un livello di astrazione in più nell'interazione con il SoC
- In questo modo lo sviluppatore non dovrà preoccuparsi di interagire con l'HW dovendo quindi andare a scrivere nei registri delle differenti periferiche per configurarne l'uso.
- Avrà a disposizione delle API offerte dall'HAL che semplificheranno tale compito

HAL 2/2



Link

➤ Link Utili:

○ Gruppo Google del Corso

- <https://groups.google.com/forum/#!forum/corso-architetture-parthenope-2017/new>

○ Link Tutorial

- <http://www.robot-home.it/blog/software/tutorial-arm-stm32-stm32f4-discovery-eclipse-gnu-toolchain-openocd-chibios-sotto-windows/>