Laboratorio microcontrollori e open source
Seconda parte

Politecnico Open unix Labs

17 aprile 2012

Introduzione *Per chi si è perso la prima lezione*

Questo corso è una breve introduzione ai microcontrollori, concentrandosi sia sugli aspetti software di programmazione che hardware di costruzione di circuiti.

Verranno presentate due piattaforme per lo sviluppo di applicazioni:

- Arduino che è una piattaforma molto popolare basata su microcontrollori Atmel a 8bit
- STM32, una architattura di microcontrollori più potente a 32bit, usando il sistema operativo Miosix.

Tutto il corso sarà tenuto in ambiente Linux, usando solo strumenti Open Source.

Struttura del corso *Per chi si è perso la prima lezione*

Il corso si compone di tre lezioni.

- Lezione 1: Basi di Arduino
  - Breve introduzione ai microcontrollori
  - Breve introduzione ad Arduino
  - Ampia sessione di sperimentazione pratica con semplici esempi usando Arduino
- Lezione 2 (oggi): Basi di STM32 e Miosix
  - Breve introduzione ai microcontrollori STM32
  - Breve introduzione a Miosix
  - Ampia sessione di sperimentazione pratica con semplici esempi usando STM32 e Miosix
- Lezione 3: Progetti avanzati
  - Verranno mostrati progetti più complessi basati sia su STM32 che Arduino

Introduzione a questa lezione

Proseguiamo quindi il nostro percorso didattico sui microcontrollori. Nella lezione precedente abbiamo visto Arduino, una piattaforma che semplifica lo sviluppo di applicazioni con microcontrollori.

Sebbene il microcontrollore dell'Arduino sia un ATmega328, finora non ci siamo dovuti preoccupare dello specifico modello, e non ne abbiamo letto il *datasheet* per vedere che caratteristiche ha, e come usarlo, ma abbiamo solo chiamato delle funzioni come `pinMode()` senza sapere cosa fanno veramente.

In questa lezione vedremo un altro tipo di microcontrollore, molto più potente, l'STM32, e nella prossima scenderemo più in dettaglio sulle sue caratteristiche.

Vedremo anche un sistema operativo per microcontrollori, Miosix.

STM32

STM32F407VGT6

LQFP100 14 x 14 mm

1 Mbyte of Flash memory
192 Kbytes of RAM

STM32 è una "famiglia" di microcontrollori a 32 bit, prodotti da ST.

Con questo si intende che esistono molti microcontrollori con diverse periferiche, diversa capacità di memoria e diversa velocità, ma con lo stesso processore.

STM32

Al di fuori degli ambienti di sviluppo semplificati, come si usa un microcontrollore?

Intanto, bisogna scegliere il modello. Per fare questo, i vari produttori hanno sui loro siti delle funzionalità di "parametric search" che consentono di comparare facilmente le caratteristiche dei loro microcontrollori, alla ricerca di quello adatto.

In questo corso useremo l'STM32F407, un microcontrollore a 32bit, 168MHz, con 192KB di RAM e 1MB di FLASH.

In confronto, l'ATmega328 dell'arduino è a 8bit, 16MHz, con 2KB di RAM e 32KB di FLASH.

STM32

Una volta scelto il microcontrollore, conviene vedere se si riesce a trovare una board di sviluppo. Di questi tempi i microcontrollori sono sempre più spesso prodotti in package SMD, molto piccoli e quindi difficilmente saldabili a mano.

Il prossimo passo consiste nel scaricare il datasheet del microcontrollore e iniziare a darci un occhiata.

Un *datasheet* è un documento scritto dal produttore di un componente elettronico, che ne specifica le caratteristiche.

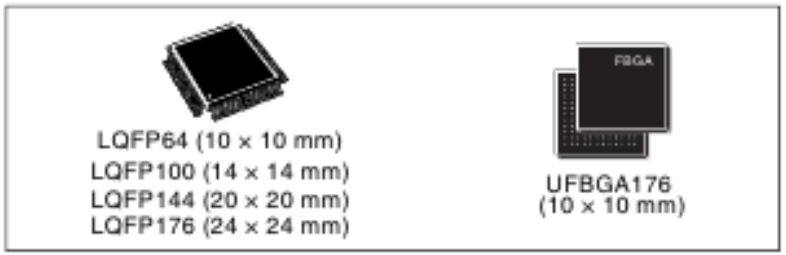
In alcuni casi, può essere accompagnato da un secondo documento, solitamente detto "programmer's manual" che spiega più in dettaglio come accedere alle varie periferiche via software.

STM32F405xx
STM32F407xx

ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM,
USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera

Features

- Core: ARM 32-bit Cortex™-M4F CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 168 MHz, memory protection unit, 210 DMIPS/ 1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 1 Mbyte of Flash memory
  - Up to 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
  - Flexible static memory controller



- Up to 140 I/O ports with interrupt capability
  - Up to 136 fast I/Os up to 84 MHz
  - Up to 138 5 V-tolerant I/Os
- Up to 15 communication interfaces
  - Up to 3 x I^2C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/2 UARTs (10.5 Mbit/s, ISO 7816 interface, LIN, IrDA, modem control)

La prima pagina del datasheet di un microcontrollore STM32 (154 pagine).

RM0090
Reference manual

STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx microcontroller memory and peripherals. The STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx will be referred to as STM32F40x and STM32F41x throughout the document, unless otherwise specified.

The STM32F40x and STM32F41x constitute a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics please refer to the STM32F40x and STM32F41x datasheets.

For information on programming, erasing and protection of the internal Flash memory please refer to the *STM32F40x and STM32F41x Flash programming manual.*

La prima pagina del programmer's manual di un microcontrollore STM32 (1315 pagine).

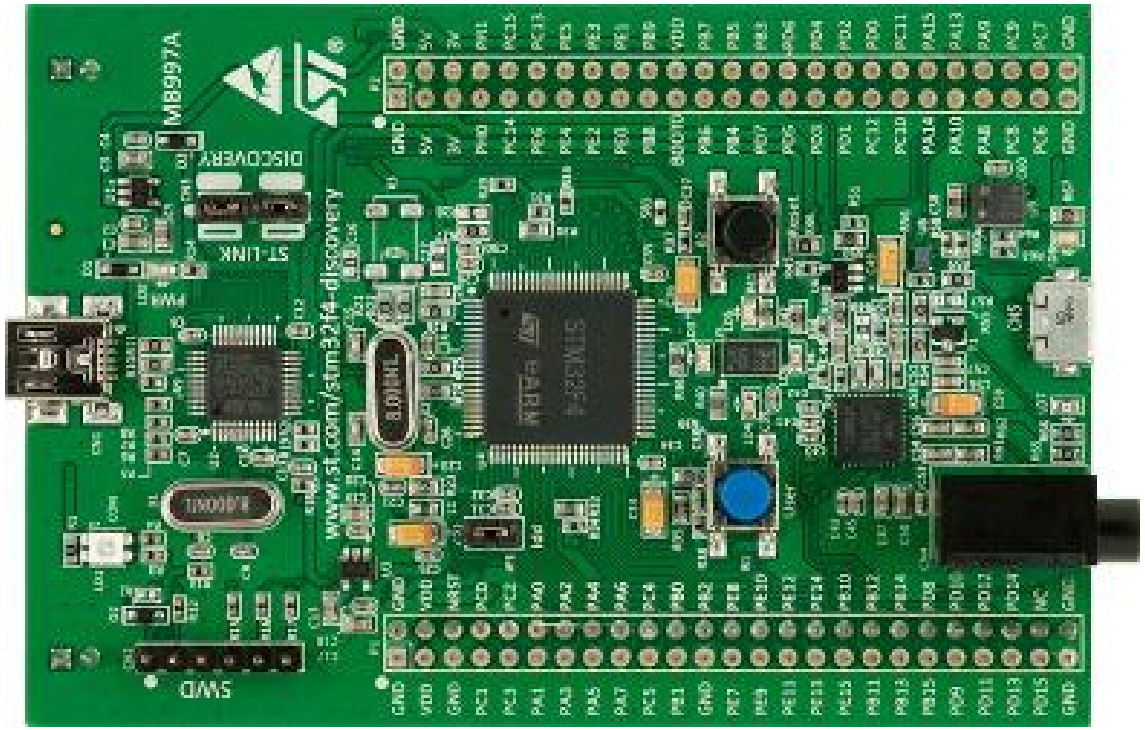
STM32

Non occorre spaventarsi per via del numero di pagine di questi documenti in quanto ci sono molti trucchi, che si imparano con l'esperienza, per evitare di leggere la maggior parte di questi documenti:

- *learn as you go:* basta dare un occhiata veloce al contenuto e poi legggere e programmare insieme, invece che leggere prima tutto e poi programmare
- *leggere solo quello che serve:* il datasheet spiega tutte le caratteristiche del processore e delle periferiche. Basta non leggere i capitoli delle periferche che non ci interessano.
- *librerie:* come le librerie di Arduino evitano di leggere il datasheet dell'ATmega32, anche per STM32 esistono librerie o runtime che semplificano lo sviluppo.

Nel seguito useremo delle librerie mentre nell'ultima lezione accederemo direttamente all'hardware.

STM32



In questo corso useremo una board, la *stm32f4discovery*. Questa board, come Arduino, contiene il microcontrollore e i componenti necessari al suo funzionamento, ed espone i GPIO su due conettori laterali.

STM32

In aggiunta, sulla board ci sono

- Un programmatore/in circuit debugger per progammare e debuggare i propri programmi.
- Una seconda porta USB, connessa alla periferica USB del microcontrollore e usabile per comunicare ad alta velocità tra il microcontrollore e il PC.
- Un pulsante e 4 led controllabili dall'utente.
- Un accelerometro a 3 assi.
- Un microfono.
- Un uscita cuffie stereo.

Nonostante ciò, il prezzo della board è comparabile a quello di un Arduino.

STM32

I GPIO negli STM32 sono organizzate in *porte* identificate da una lettera. Ogni porta ha 16 GPIO. Quindi per esempio PB0 è il GPIO 0 della porta B.

La board stm32f4discovery ha tante periferiche a bordo, questo fa si che alcuni GPIO siano "già occupati" e quindi occorre sapere quali sono liberi. La lista dei GPIO liberi è la seguente:

	PB0		PD0	
PA1	PB1	PC1	PD1	
	PB2	PC2	PD2	
			PD3	PE3
		PC4		PE4
	PB5	PC5		PE5
		PC6	PD6	PE6
	PB7		PD7	PE7
PA8	PB8	PC8	PD8	PE8
		PC9	PD9	PE9
			PD10	PE10
	PB11	PC11	PD11	PE11
	PB12			PE12
	PB13			PE13
	PB14			PE14
	PB15			PE15

UM1472
User Manual

STM32F4DISCOVERY
STM32F4 high-performance discovery board

Introduction

The STM32F4DISCOVERY helps you to discover the STM32F4 high-performance features and to develop your applications. It is based on an STM32F407VGT6 and includes an ST-LINK/V2 embedded debug tool interface, ST MEMS digital accelerometer, ST MEMS digital microphone, audio DAC with integrated class D speaker driver, LEDs, pushbuttons and a USB OTG micro-AB connector.

Figure 1. STM32F4DISCOVERY

Anche la board ha un documento di specifica, che contiene gli schemi elettrici. Sebbene non si possa definire open hardware, almeno è possibile vedere come è fatta.

Miosix

Miosix è un sistema operativo per microcontrollori, sviluppato a partire dal 2008. Il codice è open source/free software, rilasciato con licenza GPL (+linking exception).

Le linee guida usate per lo sviluppo del kernel sono:

- Offrire un ambiente di sviluppo il più possibile "standard compliant", compatibilmente con le limitazioni hardware dei microcontrollori, in modo da cercare di colmare il gap tra lo sviluppo di applicazioni desktop ed embedded.
- "You don't pay for what you don't use", fare in modo (entro certi limiti) che non ci sia performance o code size penalty per le feature non utilizzate. Questo viene effettuato quando possibile automaticamente oppure dove non possibile tramite opzioni di configurazione.

Miosix

Le principali feature offerte dal kernel sono:

- Multithreading, tramite un API standard (PTHREADS).
  Consente di spezzare le applicazioni in unità di lavoro eseguite in parallelo, semplificando lo sviluppo di codice.
- Supporto ai linguaggi C e C++.
  Consente la programmazione object oriented anche nei microcontrollori.
- Supporto alla libreria standard del C e del C++, inclusa la STL, exception handling, etc.
  Evita di dover reinventare la ruota.
- Astrazione dell'hardware, tramite API inidipendenti dalla piattaforma per le periferiche più complesse (USB, Display).
  Semplifica la portabilità del codice tra varie architetture hardware.
- Filesystem, integrato con le librerie standard.
  Consente di accedere ai file tramite la stessa API disponibile su desktop PC (`fopen()`, `fstream`).
- Pluggable schedulers.
  Un API per scegliere lo scheduler da usare nel kernel, a compile time, consente di scegliere lo scheduler più adatto per una data applicazione, e consente di sviluppare e testare scheduler innovativi.

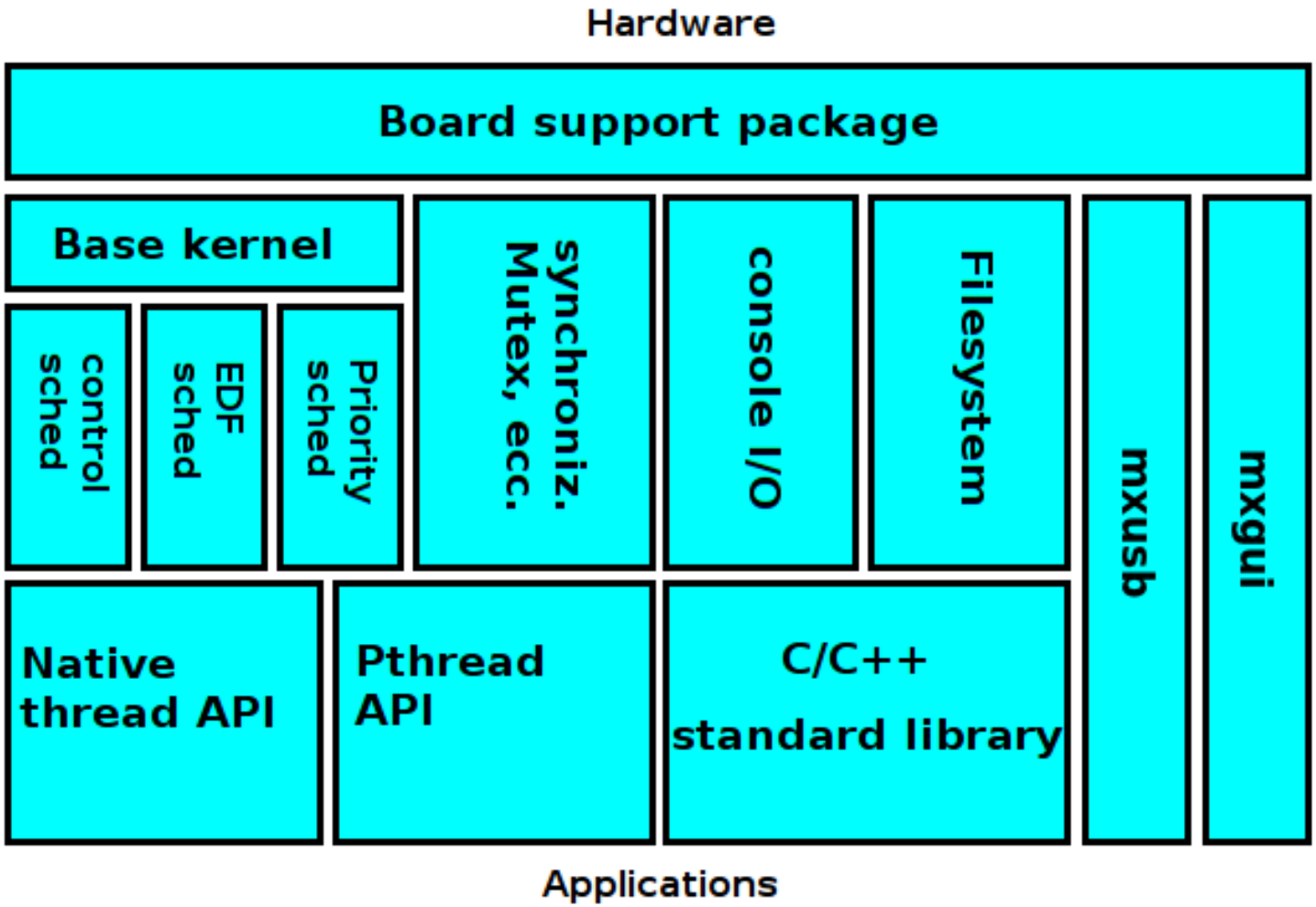
Miosix

Oltre al kernel, esistono delle librerie specifiche per gestire classi di periferiche particolarmente complesse in modo indipendente dalla piattaforma hardware:

- `mxgui` Una libreria grafica in grado di gestire display comunemente usati in dispositivi a microcontrollori.
- `mxusb` Uno stack USB device (non host, o almeno, not yet) con un API priva di dettagli implementativi delle periferiche USB.

Miosix

Riepilogo dell'architettura del kernel.



Miosix

Come cominciare a usare il kernel Miosix?

Per prima cosa, andare sul sito del kernel
www.webalice.it/fede.tft/miosix/index.html

Lì si può trovare:

- La guida all'installazione del compilatore e dell'ambiente di sviluppo (*www.webalice.it/fede.tft/miosix/download_menu.html*)
- L'URL del repo GIT dove scaricare i sorgenti del kernel (*gitorious.org/miosix-kernel*)
- La guida su come configurare il kernel per le varie board supportate, inclusa la stm32f4discovery (*www.webalice.it/fede.tft/miosix/hardware_menu.html*)
- La documentazione dell'API del kernel (*www.webalice.it/fede.tft/miosix/doxygen_menu.htm*)

Miosix

Il kernel Miosix si appoggia a molti altri tool opensource:

Per compilare il codice si usa il compilatore GCC per architettura ARM

Per la scrittura del codice Miosix si appoggia all'IDE Netbeans.

Per la programmazione delle board si usano differenti tool a seconda della board.
Nel caso della stm32f4discovery useremo QStlink2 (*code.google.com/p/qstlink2/*)

Per l'in circuit debugging si usano gdb e openocd.

Sperimentazione pratica

Completata questa breve introduzione ai microcontrollori STM32 e a Miosix, si può subito passare alla sperimentazione pratica.

Progettino #1: pulsanti e LED

Specifiche del progettino:

Collegare un pulsante e un LED a una board stm32f4discovery.
Il led deve lampeggiare quando si preme il pulsante.

Progettino #1: pulsanti e LED

Organizzazione del progettino

- Software: configurare il kernel Miosix per la board stm32f4discovery
- Software: struttura di un programma in Miosix
- Software: la funzione `usleep()`
- Software: come accedere ai GPIO da Miosix
- Hardware: i led e il pulsante della board stm32f4discovery
- Pratica: costruzione del circuito

Progettino #1: pulsanti e LED | Configurare Miosix

Il kernel Miosix supporta molte board, perciò prima di compilarlo è necessario configurarlo per la board che si vuole utilizzare.

Questa procedura si compone di tre fasi:

- Editare il file `miosix/config/Makefile.inc` per selezionare la board
- Editare il file `miosix/config/miosix_settings.h` con eventuali opzioni specifiche per la board
- Configurare l'IDE Netbeans per avere il supporto alla code completion

La procedura di configurazione è documentata sul sito, ma verrà qui riproposta per semplicità.

Progettino #1: pulsanti e LED | Configurare Miosix

Il primo passo è aprire il file `miosix/config/Makefile.inc`

```
##
## Target board, choose one. This also implicitly select the target
## architecture
##
#OPT_BOARD := lpc2138_miosix_board
OPT_BOARD := stm32f103ze_stm3210e-eval
#OPT_BOARD := stm32f103ve_mp3v2
#OPT_BOARD := stm32f100rb_stm32vldiscovery
#OPT_BOARD := stm32f103ve_strive_mini
#OPT_BOARD := stm32f103ze_redbull_v2
#OPT_BOARD := stm32f407vg_stm32f4discovery
#OPT_BOARD := stm32f207ig_stm3220g-eval
#OPT_BOARD := stm32f207zg_ethboard_v2
```

In questo file, # indica una linea di commento. Occorre togliere il # alla linea con il nome della board selezionata, e assicurarsi tutte le altre siano commentate.
In questo caso la board da selezionare è la `stm32f407vg_stm32f4discovery`

Secondo passo, alcune board richiedono alcune opzioni specifiche. Nel caso della

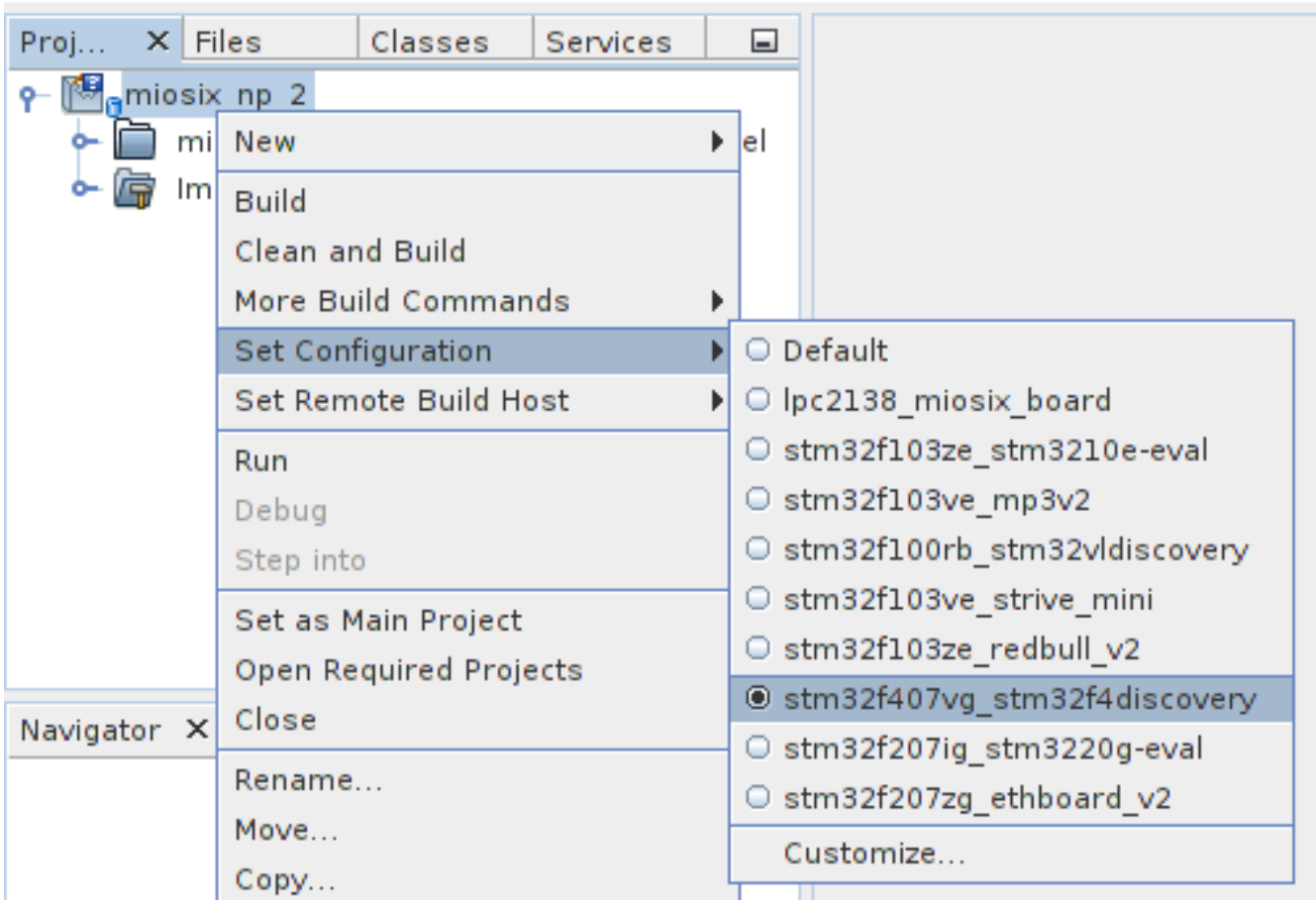
stm32f4discovery è necessario disattivare il filesystem, in quanto la board non hadell'hardware in grado di supportarlo.

Per farlo è necessario riportate editare il file `miosix/config/miosix_settings.h` e commentare la riga:

```
#define WITH_FILESYSTEM
```

Senza questa modifica si avrebbero errori nella compilazione del kernel.

Terzo e ultimo passo, occorre selezionare la board nell'IDE.



Clic destro sul progetto > Set configuration > nome board

In Miosix, contrariamente ad Arduino, esiste il `main()` come in un comune programma C/C++. Questa funzione si trova nel file *main.cpp* che non contiene altro, ed è dove il programmatore può scrivere la sua applicazione.

```
int main()
{

}
```

Questa funzione viene chiamata quando si fornisce corrente alla board, dopo il boot del kernel.

Se il main ritorna, la board si spegne. In una tipica applicazione embedded il `main()` non dovrebbe mai ritornare, in modo che l'applicazione continui a funzionare finchè non viene tolta corrente alla board.

Progettino #1: pulsanti e LED | `usleep()`

Esistono molti modi per ottenere dei ritardi di tempo in Miosix, tra questi il più portabile è la funzione `usleep()`. Questa funzione prende come parametro un tempo espresso in microsecondi.

Il kernel può approfittare di questa funzione per schedulare altri task.

Per usarla, è necessario aggiungere una

```
#include <unistd.h>
```

all'inizio del programma.

Progettino #1: pulsanti e LED | GPIO in Miosix

Miosix offre una libreria C++ per accedere ai GPIO indipendentemente dalla piattaforma.
Per rendere più pulito il codice, Miosix forza l'utente a *dichiarare* i GPIO prima di usarli, e a dargli un nome, come ad esempio "led" o "pulsante".

Per cominciare occorre aggiungere all'inizio del programma

```
#include <miosix.h>
using namespace std;
```

Dopodichè, immaginando di avere un led connesso al GPIO PD15, si può dichiararlo in questo modo:

```
typedef Gpio<GPIOD_BASE,15> led;
```

Progettino #1: pulsanti e LED | GPIO in Miosix

La sintassi precedente dichiara una classe template di nome *led*, connessa al GPIO PD15.

Si possono chiamare i seguenti metodi su questa classe:

- `led::mode(Mode::OUTPUT);` Configura il GPIO come uscita
- `led::mode(Mode::INPUT);` Configura il GPIO come ingresso
- `led::high();` Porta un GPIO configurato come uscita a livello alto
- `led::low();` Porta un GPIO configurato come uscita a livello basso
- `led::value();` Chiamato su un GPIO configurato come ingresso, ritorna 0 o 1 a seconda che sul GPIO arrivi un livello logico 0 o 1.

Nota: i metodi vanno chiamati con `::` perchè si tratta di metodi statici.

Progettino #1: pulsanti e LED | board stm32f4discovery

Dato che la board stm32f4discovery ha 4 led e un pulsante, possiamo evitare di collegare questi componenti esternamente.

- Il pulsante è connesso al GPIO PA0, ed è già dotato della resistenza di pulldown
- Il led è connesso al GPIO PD15, ed è già dotato della resistenza di limitazione della corrente

Progettino #1: pulsanti e LED

Pratica: costruzione del circuito

Progettino #2: LCD

Specifiche del progettino:

Collegare un LCD da 2 righe per 16 caratteri compatibile HD44780 alla stm32f4discovery. Stampare "Hello world" sulla prima riga, mentre il numero di secondi dall'avvio della board sulla seconda riga.

Progettino #2: LCD

Organizzazione del progettino

- Teoria: ripasso sui display compatibili HD44780.
- Software: Miosix e la classe Lcd44780
- Pratica: costruzione del circuito

Progettino #2: LCD | HD44780

Gli LCD compatibili HD44780 hanno 14 o 16 piedini, a seconda della presenza di una retroilluminazione per renderli visibili al buio.

- 1 Da collegare a massa
- 2 Da collegare a +5V
- 3 Contrasto, collegare un trimmer tra +5V e massa
- 4 Register Select, da collegare a un GPIO
- 5 Read/Write (R/W), da collegare a massa (write)
- 6 Clock (Enable), da collegare a un GPIO
- 7 Bit 0 non collegare
- 8 Bit 1 non collegare
- 9 Bit 2 non collegare
- 10 Bit 3 non collegare
- 11 Bit 4, da collegare a un GPIO
- 12 Bit 5, da collegare a un GPIO
- 13 Bit 6, da collegare a un GPIO
- 14 Bit 7, da collegare a un GPIO

Progettino #2: LCD | Lcd44780

Per pilotare un LCD compatibile HD44780 da Miosix, esiste la classe `Lcd44780` disponibile includendo `#include <util/lcd44780.h>`

Per iniziare a usare la classe, occorre prima dichiarare i vari GPIO a cui sono collegati i pedini RS, E, D4, D5, D6, D7 del display:

```
typedef Gpio<GPIOB_BASE,0> rs;
typedef Gpio<GPIOB_BASE,1> e;
typedef Gpio<GPIOB_BASE,11> d4;
typedef Gpio<GPIOB_BASE,12> d5;
typedef Gpio<GPIOB_BASE,13> d6;
typedef Gpio<GPIOB_BASE,14> d7;
```

Progettino #2: LCD | Lcd44780

Una volta dichiarati i Gpio è possibile ustanziare la classe.
Il costruttore prende 8 parametri: i 6 gpio, il numero di righe e il numero di colonne del display.

```
Lcd44780 lcd(rs::getPin(),e::getPin(),d4::getPin(),d5::getPin(),
                    d6::getPin(),d7::getPin(),2,16);
```

Nota: la classe si può istanziare solo all'interno di una funzione, come ad esempio il `main()` e non come "variabile" globale.

Progettino #2: LCD | Lcd44780

La classe ha poi i seguenti metodi per interagire con il display:

- `lcd.go(x,y)` Sposta il cursore alla posizione (x,y)
- `ldc.printf()` Funziona esattamente come la comune printf, ma stampa sul display.

Progettino #2: LCD

Pratica: costruzione del circuito

Progettino #3: Multithreading

Specifiche del progettino:

Collegare un pulsante, un LED, e un LCD da 2 righe per 16 caratteri compatibile HD44780 alla stm32f4discovery.
Stampare "Hello world" sulla prima riga, mentre il numero di secondi dall'avvio della board sulla seconda riga.
Il led deve lampeggiare quando si preme il pulsante.

Progettino #3: Multithreading

Organizzazione del progettino

- Teoria: threads
- Software: multithreading in Miosix
- Pratica: costruzione del circuito

Progettino #3: Multithreading | thread

Un *Thread* è un contesto di esecuzione indipendente.

Se dal `main()` istanzio un thread, ci saranno due contesti di esecuzione che verranno eseguiti in parallelo.

Sui sistemi multicore più thread possono effettivamente essere eseguiti in parallelo. Su un sistema con una sola CPU l'esecuzione parallela è *simulata* togliendo periodicamente la CPU a un thread per assegnarla agli altri.

I thread, contrariamente ai process possono condividere variabili.

Progettino #3: Multithreading | PTHREADS

Miosix supporta il multithreading tramite l'API standard PTHREADS.

Per usare i thread occorre includere `#include <pthread.h>`

Quando si crea un thread, questo esegue una specifica funzione, che è come il `main()` del thread.

Questa funzione deve avere come argomento `void*` e ritornare `void*`, esempio:

```
void *mythread(void*)
{

}
```

Progettino #3: Multithreading | PTHREADS

Dopo aver creato questa funzione, occorre far partire il thread, con questo codice:

```
pthread_t threadid;
pthread_create(&threadid, NULL, mythread, NULL);
```

Il thread termina quando la sua funzione (in questo caso `myhtread()` ritorna.

E' anche possibile fermare il chiamante in attesa che il thread appena avviato termina. Questo si può fare chiamando `pthread_join()` passandogli come parametro il threadid:

```
pthread_join(threadid, NULL);
```

Progettino #3: Multithreading

Pratica: costruzione del circuito