Programmazione di sistemi multicore

Michele Martinelli

Michele.martinelli@uniroma1.it

Google form!

Da compilare subito… https://forms.gle/yEXUsGyvsndcaWt49

Orario delle lezioni

Aula 1 - Via del Castro Laurenziano 7A

Martedì 16 -19
Giovedì 16-18

- Portate il vostro computer

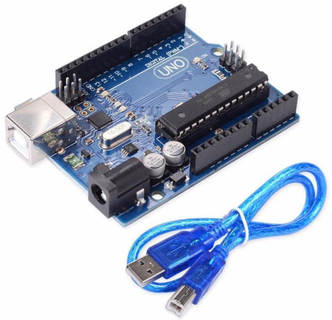
- Ricevimento su appuntamento (email)

Programma del corso

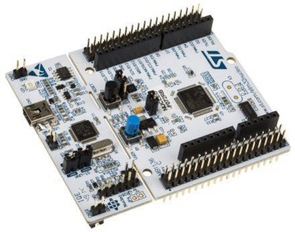
Ripasso programmazione C (circa 2 lezioni)

Programmazione multithread in C

Sistemi embedded: Arduino e elettronica



FreeRTOS



MPI

CUDA/OpenCL

Calendario ottobre

Differenze rispetto allo scorso anno

L’accento sarà posto molto di più sui sistemi EMBEDDED

Il corso affronta problematiche e tecniche di programmazione delle moderne piattaforme di calcolo, dai sistemi embedded ai sistemi multicore presenti sui più comuni laptop, fino alle moderne GPU.

Il parallelismo offre enormi opportunità nello sviluppo di applicazioni con requisiti computazionali stringenti: basti pensare ai benefici in aree come computer graphics e big data computing. Ma un maggior parallelismo nell'hardware risulta inefficace se non è opportunamente sfruttato a livello software. Ciò impone cambiamenti fondamentali nello stile di programmazione e, ad un più alto livello di astrazione, nella progettazione degli algoritmi e delle strutture dati utilizzate. Ad oggi, la maggior parte degli algoritmi sono pensati secondo una visione sequenziale del modello di calcolo soggiacente e i linguaggi di programmazione offrono poche astrazioni per programmare sistemi multiprocessore.

Scopo del corso è di insegnare agli studenti a "pensare in parallelo", rivisitando sia il modo in cui i programmi sono espressi che alcune tecniche algoritmiche tradizionalmente pensate per modelli sequenziali. La prima parte del corso utilizza alcune piattaforme embedded per sviluppare e testare applicazioni multithread. Presenteremo successivamente tecniche di programmazione di GPU e - tempo permettendo - alcune applicazioni pratiche (droni aerei e sottomarini).

Modalità di esame

Esonero a metà corso

Secondo esonero / esame a fine corso

Massimo voto 30

progetto (facoltativo)

Fino a 5 punti bonus + lode

Materiale didattico

- Pagina twiki: http://twiki.di.uniroma1.it/twiki/view/PSMC/WebHome
- Slide
- Materiale su Arduino, Freertos, CUDA, MPI sarà dato durante le lezioni

Google form!

Da compilare subito… https://forms.gle/yEXUsGyvsndcaWt49

Sistema Embedded - definizione

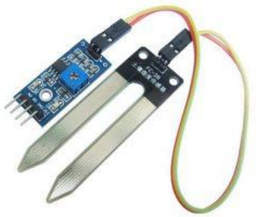
Qualsiasi dispositivo che includa una logica programmabile

ma che non risulti un general purpose computer

Un sistema embedded è costituito da una parte hardware

E una parte software (firmware)

Monitora, controlla l’ambiente esterno usando sensori



risponde usando attuatori.

Definizione di Wikipedia

*An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.*

It is embedded as part of a complete device often including hardware and mechanical parts.

By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs.

Dimensioni del mercato IoT

How big is the Internet of Things market?
Source: IDC Forecasts Worldwide Technology Spending
Internet of Things to reach $1.2 Trillion in 2022.

How big is the artificial intelligence market?
IDC predicts worldwide spending on
cognitive and **Artificial Intelligence** systems will reach $77.6B in 2022

the global **Big Data** and business analytics **market**
estimated $189 billion in 2019

Mercato in forte crescita = richiesta di figure professionali

Lavorare nel mondo embedded

Sviluppatore Software Embedded / FW Engineer
Electronic Engineer - Embedded Hardware Designer

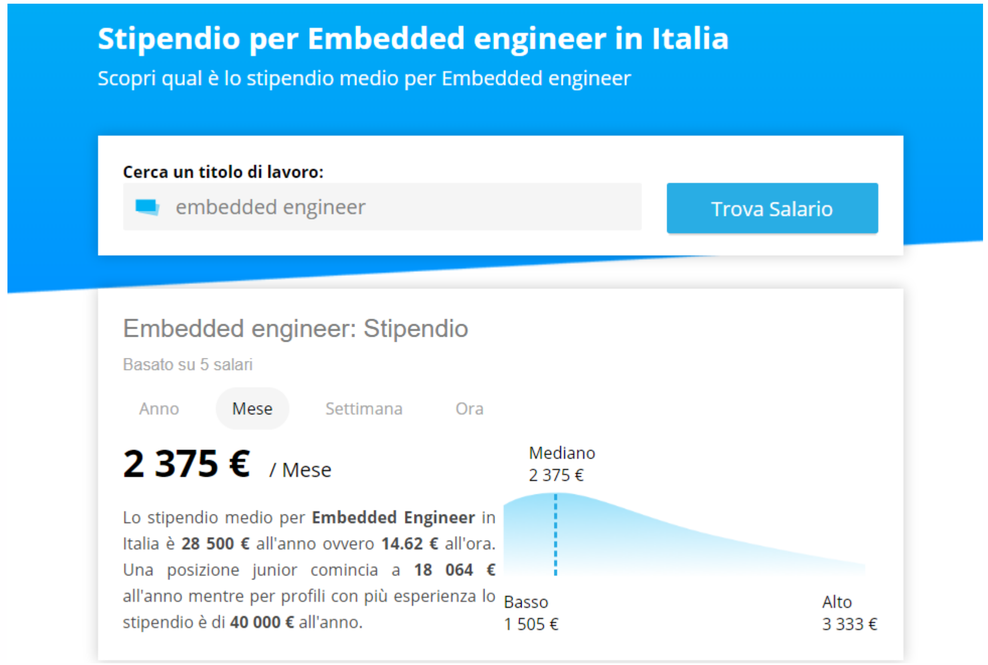
- Laurea magistrale in Ingegneria elettronica, ingegneria informatica o informatica

- esperienza almeno triennale nella progettazione e sviluppo di software
- Conoscenza della lingua inglese
- padronanza del linguaggio C
- esperienza nello sviluppo di codice in linguaggio C su sistemi embedded (FreeRTOS)
- buona conoscenza di IDE dedicati allo sviluppo e debug embedded (KEIL, TrueStudio)
- conoscenza protocolli di comunicazione (I2C, SPI, RS232, USART)
- esperienza nell'aggiornamento/manutenzione di soluzioni software esistenti
- conoscenza dei protocolli di comunicazione e routing (CTP, IP)

Completano il profilo:
- esperienza nell'utilizzo di strumentazione da banco (oscilloscopio, tester, alimentatore)
- basi di elettronica digitale e analogica (capacità di leggere schematici elettronici)

- esperienza di programmazione in python
- conoscenza di Linux, anche su piattaforme embedded (Raspbian)
- buona capacità di redigere documentazione tecnica in inglese
- esperienza nell'uso di sistemi di controllo di versione (Git)
- esperienza nell'utilizzo di tecnologie WAN (Sigfox, LoRa) e PAN (ZigBee, BLE)

Lavorare nel mondo embedded

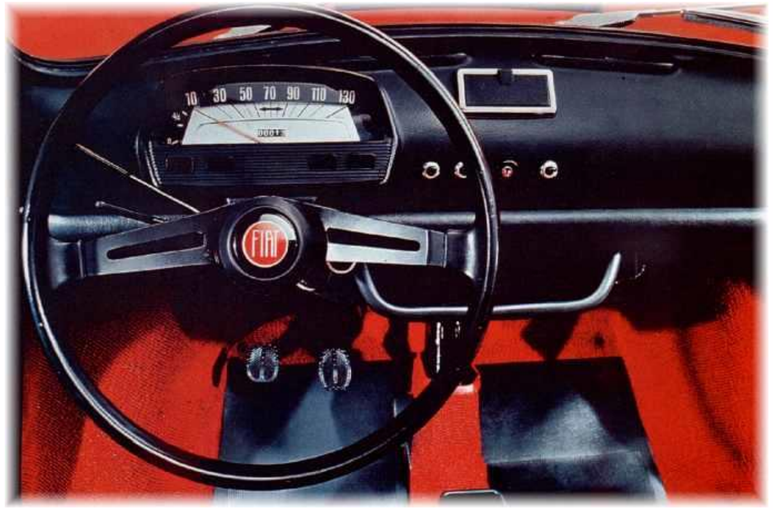


https://neuvoo.it

Motivazioni

Elettronica dilagante

- Crescita di esigenze computazionali
- Intelligenza diffusa
- Elaborazione e processamento di dati in realtime



Esigenze tecniche e di mercato

Le richieste di flessibilità ed integrazione hanno generato diverse necessità:

- Parallelismo delle operazioni
- Numero delle variabili da controllare
- Modularità
- affidabilità
- flessibilità
- Riduzione dimensioni e costi

- SE riprogrammabili (aggiornamenti fw)
- Logiche riprogrammabili (FPGA)

Sistemi Embedded – Dove?

Esempi consumer

- Mobile

- Home

Sistemi Embedded – Dove?

Sistemi Embedded – Esempi

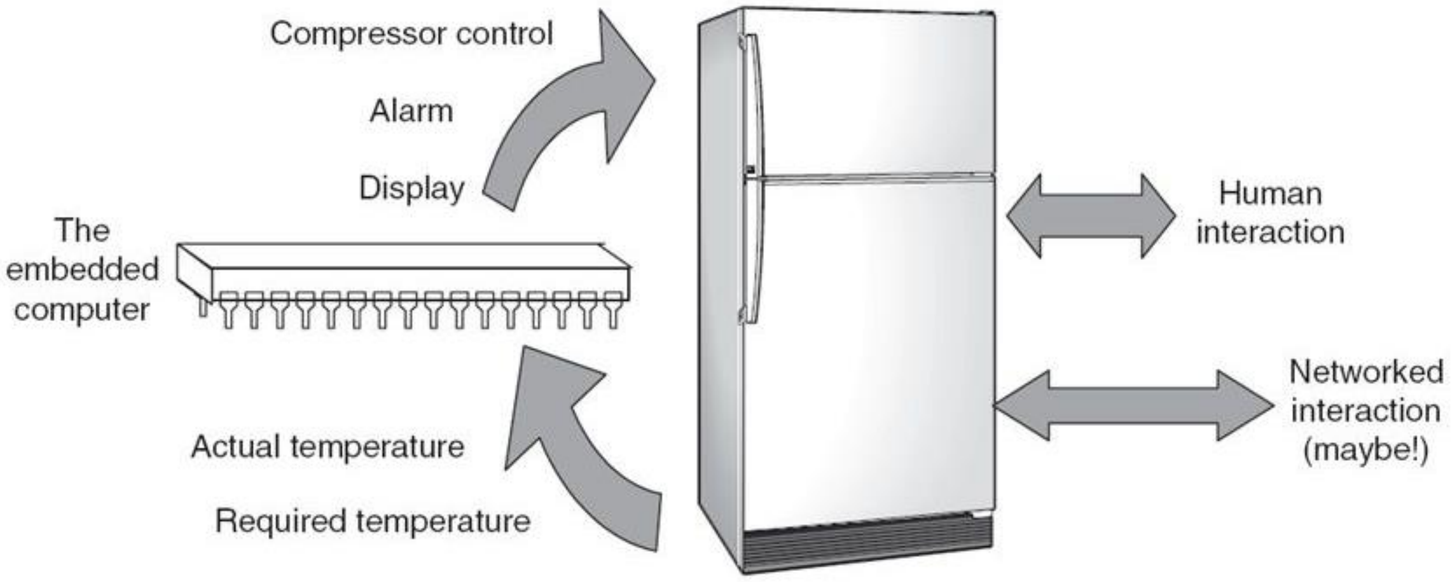
Prodotto: spazzolino elettrico

Microprocessore: 8bit

- Accensione / spegnimento
- Velocità programmabile
- timer
- circuito di ricarica

Sistemi Embedded – Esempi

Prodotto: frigorifero



Sistemi Embedded – Esempi

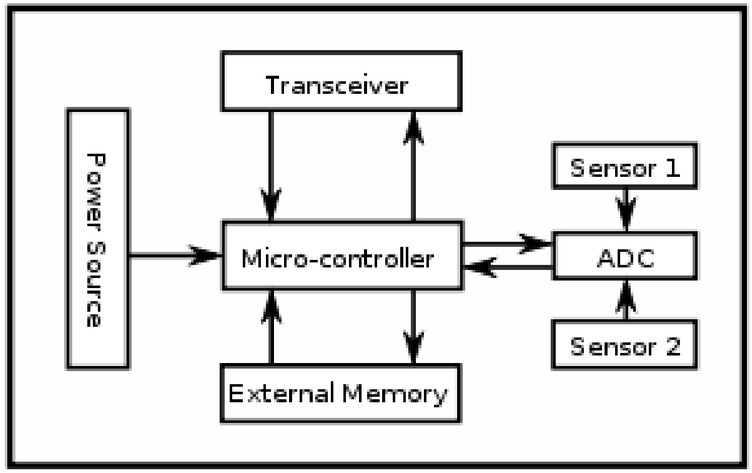
In a car, we have a lot of embedded systems as shown below.
We will have more and more as cars evolve into future car.

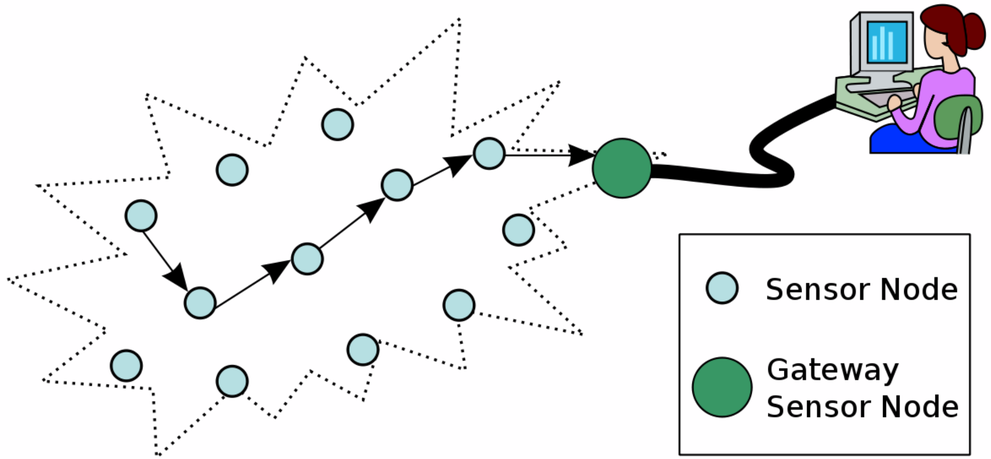
Sistemi Embedded – Esempi

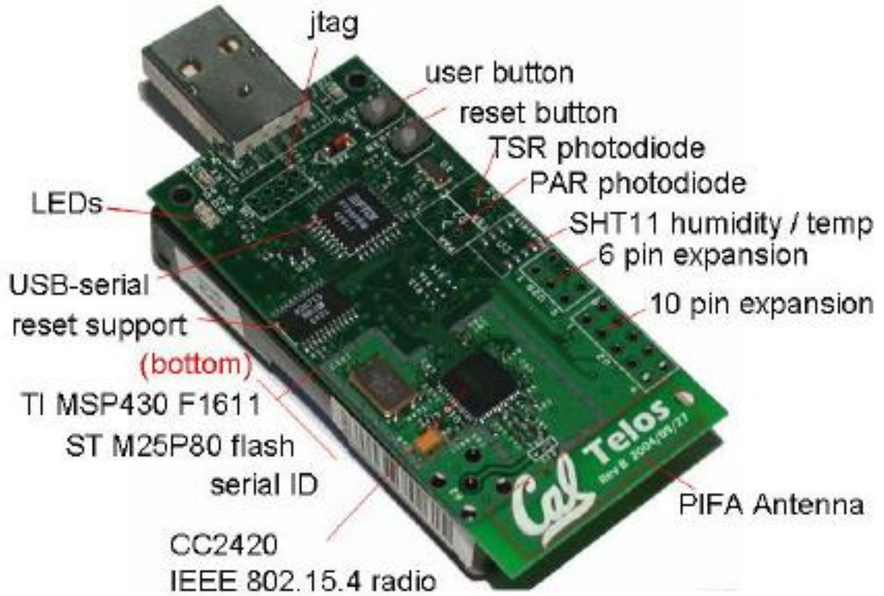
Wireless Sensor Networks

collezione di dispositivi elettronici costituiti da:

- Sensori
- Radio transciver
- CPU
- Memory
- Power source







Caratteristiche dei Sistemi Embedded

Operazioni Real-Time

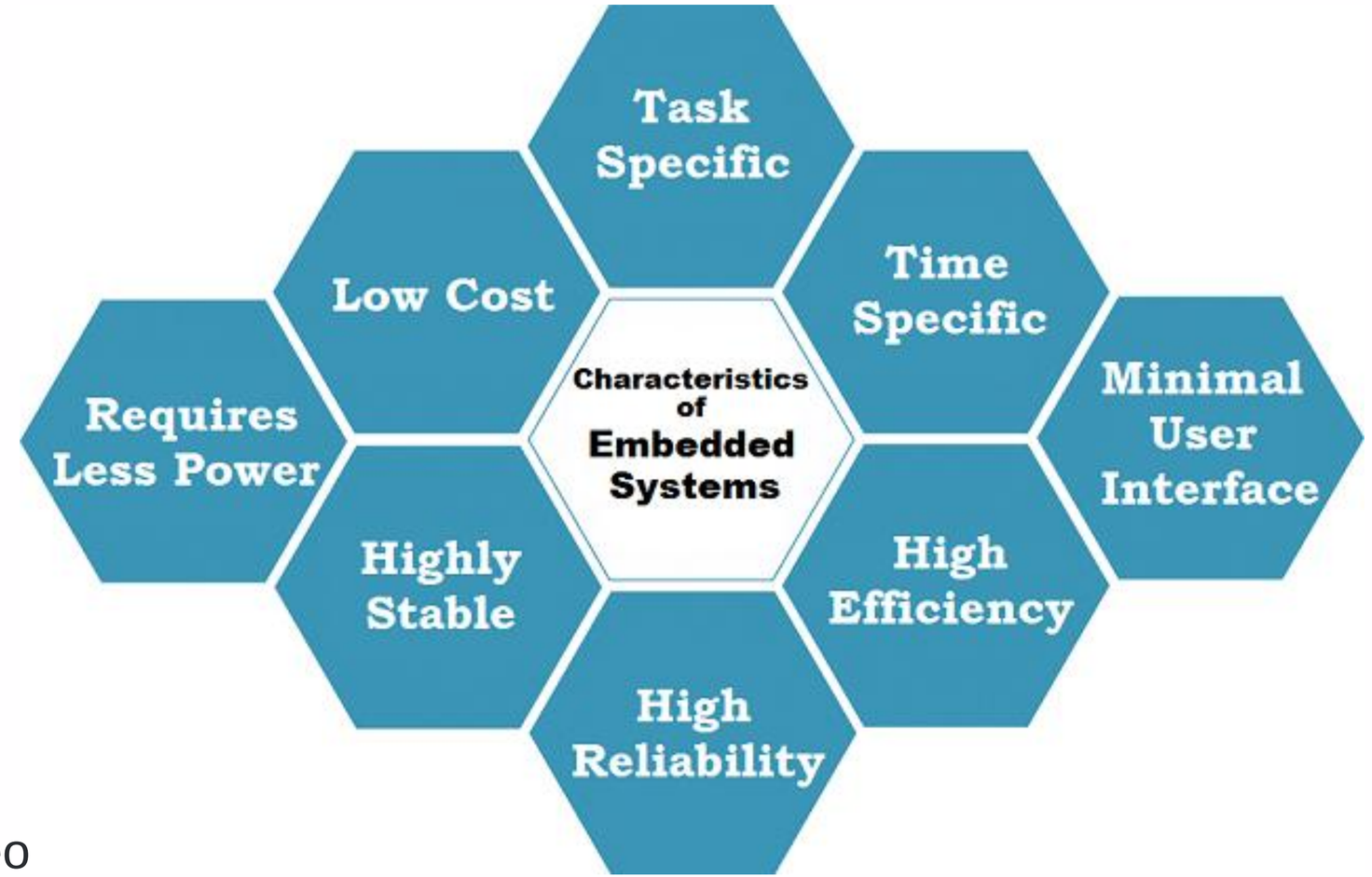
Basso costo di produzione

Basso consumo

Basso Time To Market

Algoritmi sofisticati o multipli.

- Fast Fourier Trasform;
- Encoder/Decoder Audio-Video



Caratteristiche dei Sistemi Embedded: risorse energetiche limitate

PRO:
Massimizzare il tempo di idle del sistema
low-power mode: consumo energetico minimo

CONTRO:
Overhead di transizione
Difficile prevedere le performance richieste

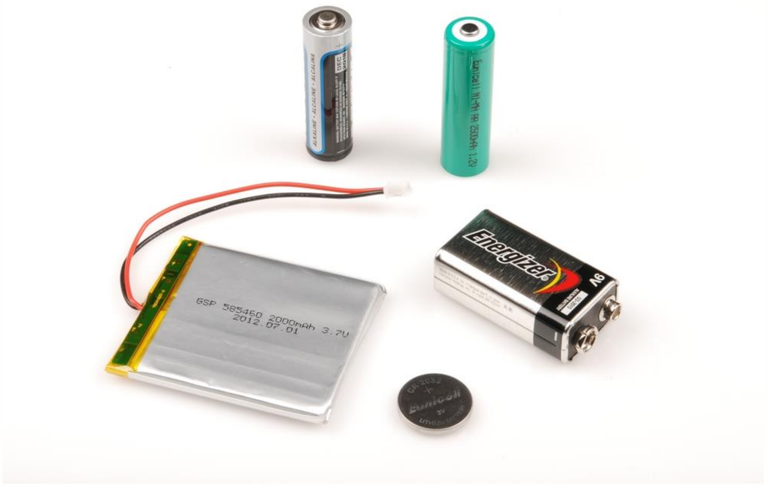
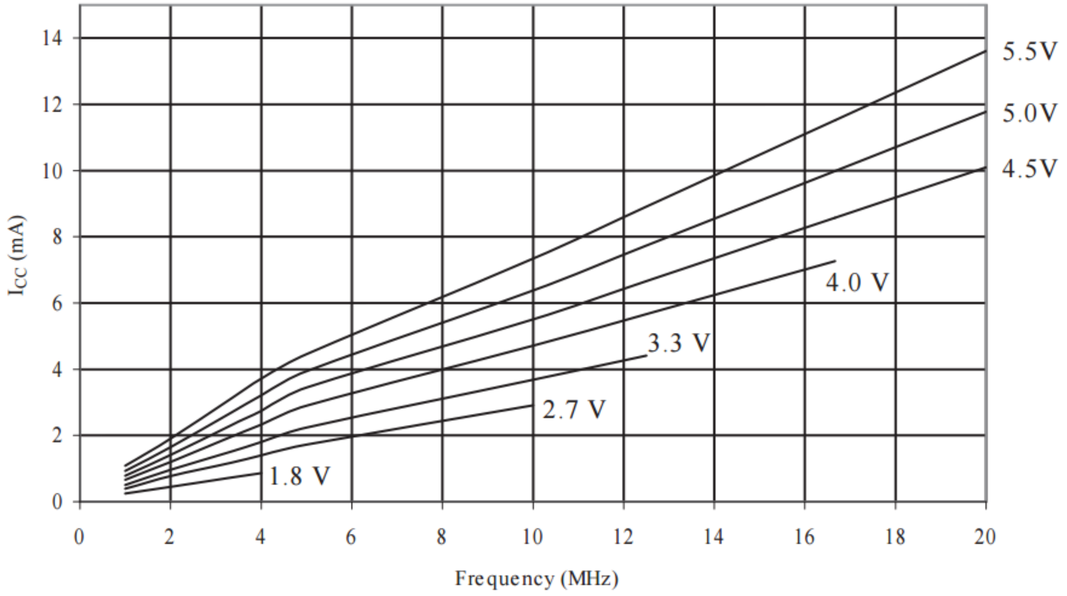
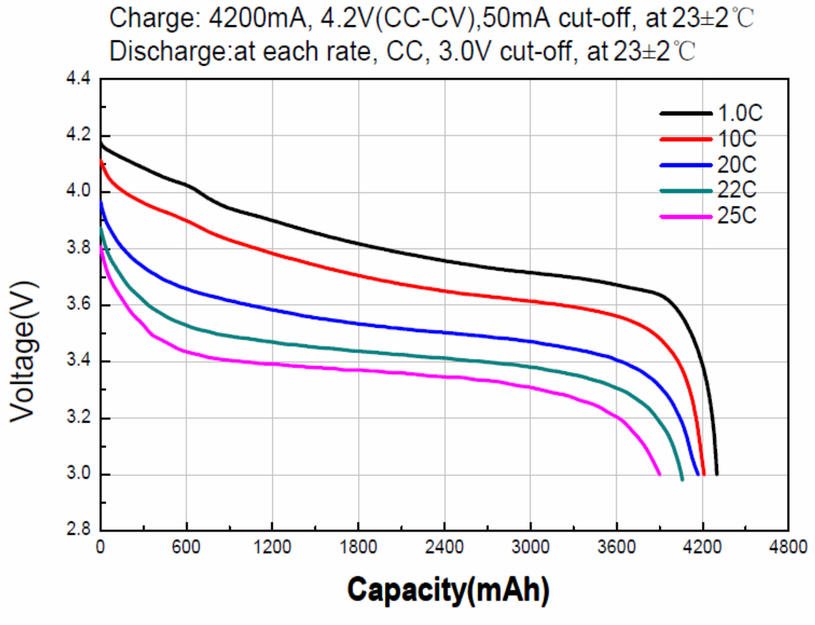


Figure 33-2. ATmega328: Active Supply Current vs. Frequency (1MHz - 20MHz)





Caratteristiche dei Sistemi Embedded: risorse computazionali limitate

Esempio: Arduino

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Esempio 2: Teensy 4

Teensy 4.0 features a 600 MHz Cortex-M7 processor,

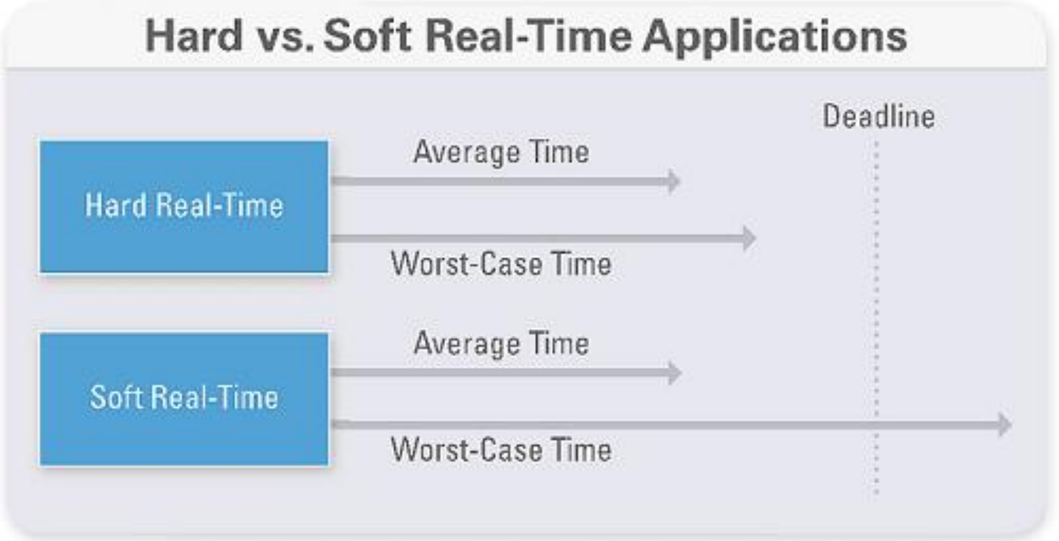
Caratteristiche dei Sistemi Embedded: operazioni Real-time

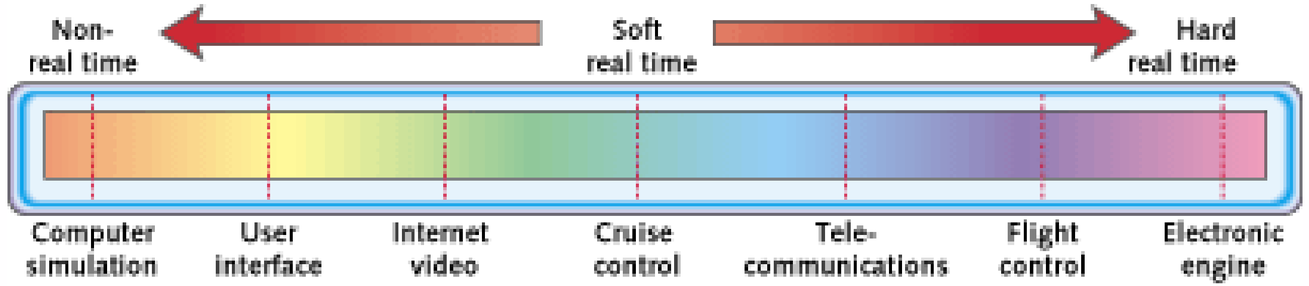
La computazione deve essere effettuata prima di

una deadline.

•Hard real-Time: Missing deadline = Failure

•Soft Real Time: Missing deadline = degradation





Evoluzione dei Sistemi Embedded

• Dagli anni ’80 i microprocessori sono stati sviluppati con diversi livelli di complessità

• sono classificati in base alla dimensione della loro word:

Si parte da un microcontrollore a 8-bit è progettato per applicazioni a basso costo ed include una memoria ed un dispositivo di I/O

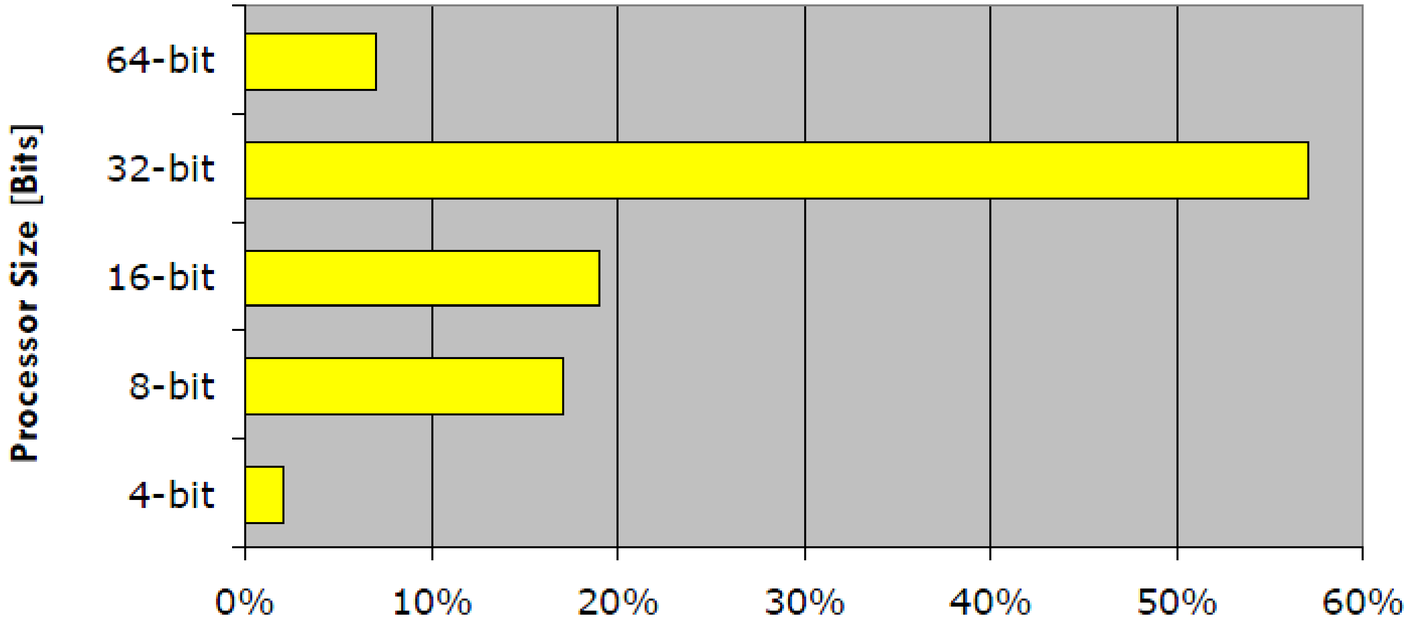
Fino ad arrivare ad un microcontrollore a 32-bit RISC che offre alte prestazioni ed una grande potenza di calcolo

Evolution of Embedded systems

Company	Processor	Year
INTEL 4004	4-bit	1971
INTEL 8085	8-bit	1974
INTEL 8048	8-bit	1976
INTEL 8031	8-bit(ROM-LESS)	-
INTEL 8051	8 bit(MASK ROM)	1980
INTEL 8086	16-bit	1978
Atmel At89C51	8-bit(Flash Memory)	1984
Microchip PIC16C64	8-bit	1985
Motorola 68HC11	8-bit(on chip ADC)	1985
AVR	8-bit RISC	1996

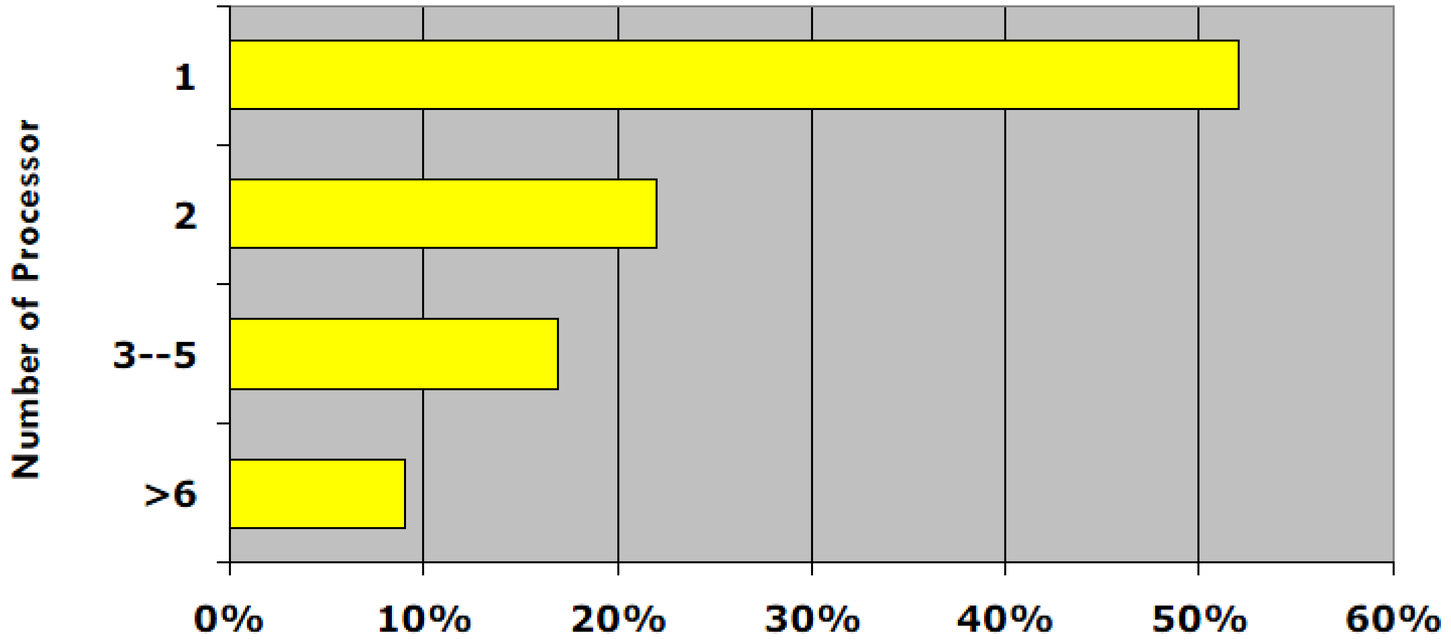
in una data architettura il formato standard di una variabile semplice (intero, puntatore, handle ecc.) è di n **bit** di lunghezza.
Generalmente questo riflette la dimensione dei registri interni della CPU usata per quell'architettura.

CPU bit- size



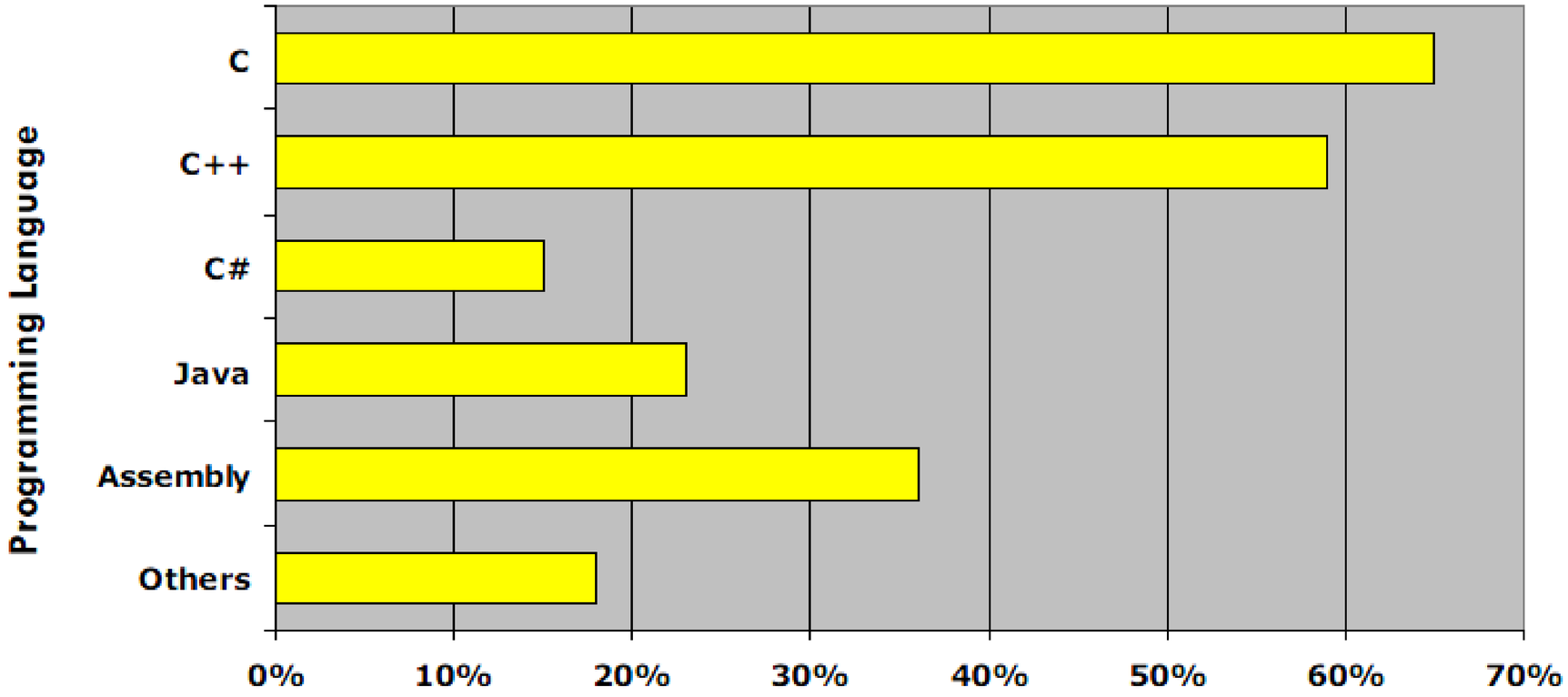
Indagine di mercato di sviluppatori ingegneri appartenenti ad EETimes e Embedded System Design Magazine

Numero di CPU



Indagine di mercato di sviluppatori ingegneri appartenenti ad EETimes e Embedded System Design Magazine

Linguaggi di programmazione utilizzati



Indagine di mercato di sviluppatori ingegneri appartenenti ad EETimes e Embedded System Design Magazine

Software

Differenti soluzioni software:

- Applicazione standalone (bare metal): l’applicazione gestisce autonomamente tutte le risorse di sistema

- Con Sistema Operativo: un Boot Loader è caricato in ROM ed ha lo scopo di inizializzare tutto l'hardware di sistema, verificarne il corretto funzionamento e successivamente caricare il sistema operativo in memoria RAM ed avviarne l'esecuzione

Vedremo entrambi gli approcci in applicazioni pratiche

Programmazione bare metal

Nessun supporto per multitasking, scheduling e sincronizzazione

Nessuno scheduler

Nessuna astrazione sull’hardware

Programmazione diretta dell’hardware

Maggior controllo delle risorse

Un sistema operativo…

Supporto per multitasking, scheduling e sincronizzazione

Supporto per un ampio range di dispositivi di I/O

Supporto per il networking

Supporto per la gestione della memoria

Sicurezza (accesso alle risorse) e gestione della potenza

Microprocessore

Il microprocessore diventa indispensabile nel controllo di sistemi complessi

- Numero delle variabili da controllare elevato
- Elevata comunicazione con l'esterno
- Necessità di espansione non conosciuta a priori (periferiche, memoria…)
- Esegue esclusivamente operazioni logiche, aritmetiche e di controllo
- Elabora sia dati prodotti internamente che provenienti da dispositivi esterni
- Totalmente dipendente da elementi periferici

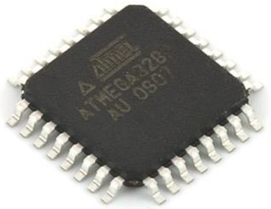
Microcontrollore

INTEGRA («embedda») un microprocessore, aggiungendo:

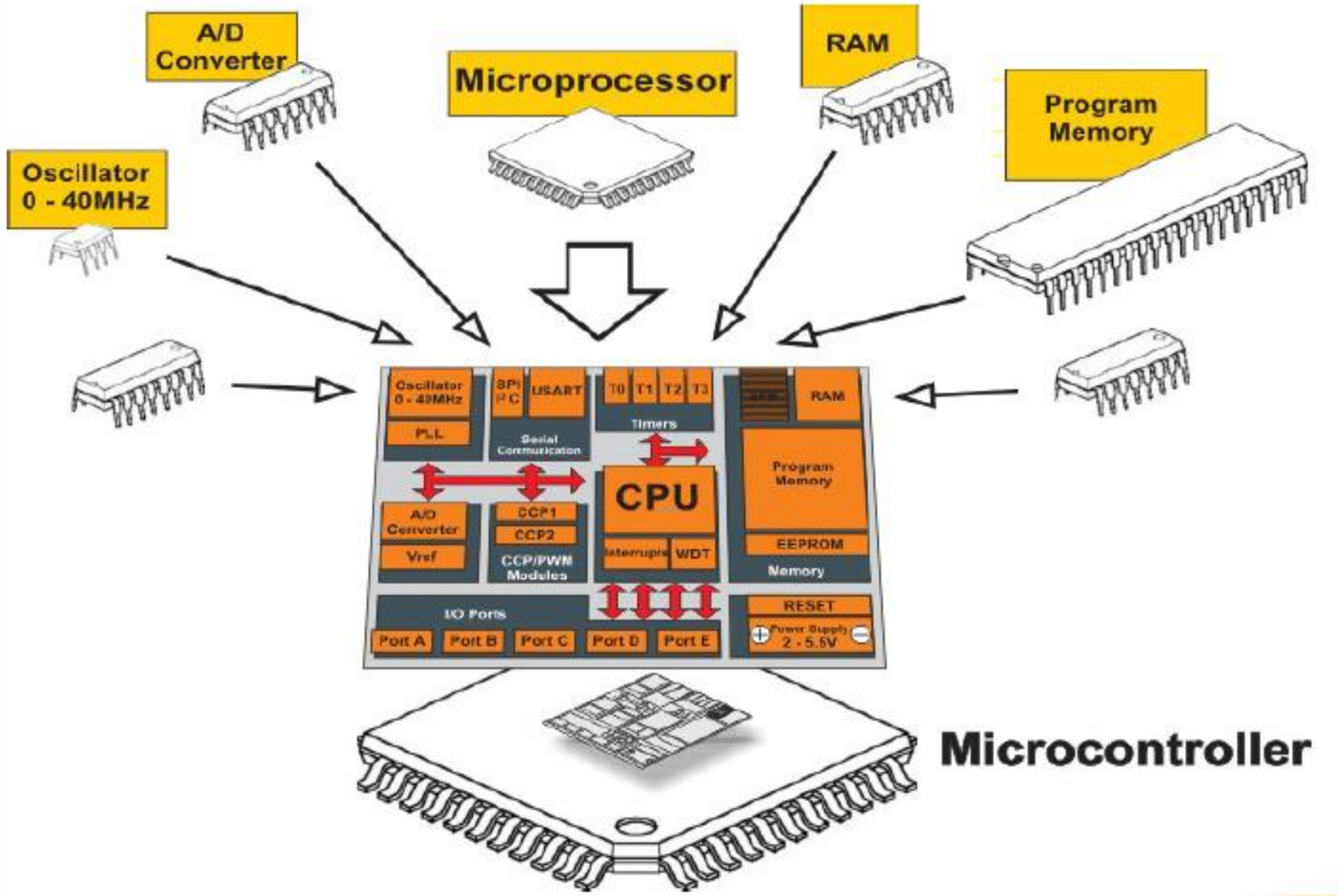
- Comunicazione diretta con dispositivi esterni integrando periferiche interne
- Memorizzazione di dati o programmi

Contro:

- Molto difficile da espandere (es. memoria esterna comunque più lenta, assenza di veri BUS)
- Esegue esclusivamente le operazioni legate al firmware con il quale sono è programmato



Microcontrollore



Microcontrollore vantaggi

Integrazione delle periferiche su un singolo chip:

- Minor numero di dispositivi discreti per la realizzazione di un sistema
- Dimensioni ridotte del sistema
- Costi inferiori
- Complessità inferiore
- Maggiore affidabilità
- Protezione dalle copiature
- Risparmio energetico
- Ri-programmabilità del sistema
- Comunicazione diretta con altri sistemi

Memoria nei sistemi embedded

Occorre sia memoria non volatile che volatile (equivalente a RAM + HD sui pc)

RAM veloce e costosa (quindi ridotta)

FLASH lenta e limitata nei cicli di scrittura

Il Sistema Operativo e l'applicativo (firmware) sono memorizzati in FLASH

Boot dalla Flash all'accensione poi il codice è copiato nella ram interna

Periferiche nei sistemi embedded

Necessarie per svolgere alcune funzioni non banali:

- ADC (convertitore analogico-digitale: es. microfono)
- DAC (convertitore digitale-analogico: es. stereo)
- EEPROM (memoria aggiuntiva)

Interrupt

Un meccanismo fondamentale nei microcontrollori è la gestione delle interruzioni (interrupt)

In risposta ad eventi esterni (pulsanti, timer…), il microcontrollore sospende l'esecuzione del programma principale, esegue una certa funzione e poi ritorna all'esecuzione principale

Esempio:

Applicazione in cui il microcontrollore legge continuamente dati da un sensore (es. temperatura).

Una periferica Timer (contatore) viene programmata per generare un interrupt ogni secondo.

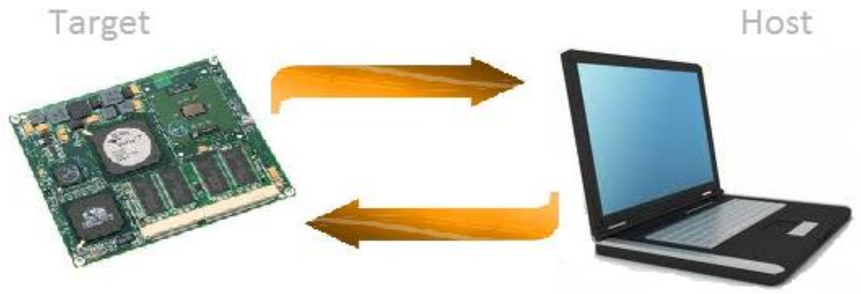
Una funzione viene eseguita ad ogni interrupt per aggiornare un display con l'indicazione dell'orario e della temperatura.

Compilazione

Non si può fare «javac» o «gcc» sul sistema embedded destinazione…!

Per la programmazione si utilizza un ambiente di sviluppo sul PC, detto *Host*.

(*Target* identica la piattaforma su cui verrà eseguita l'applicazione; mentre l'host e la macchina su cui si sviluppa il programma)



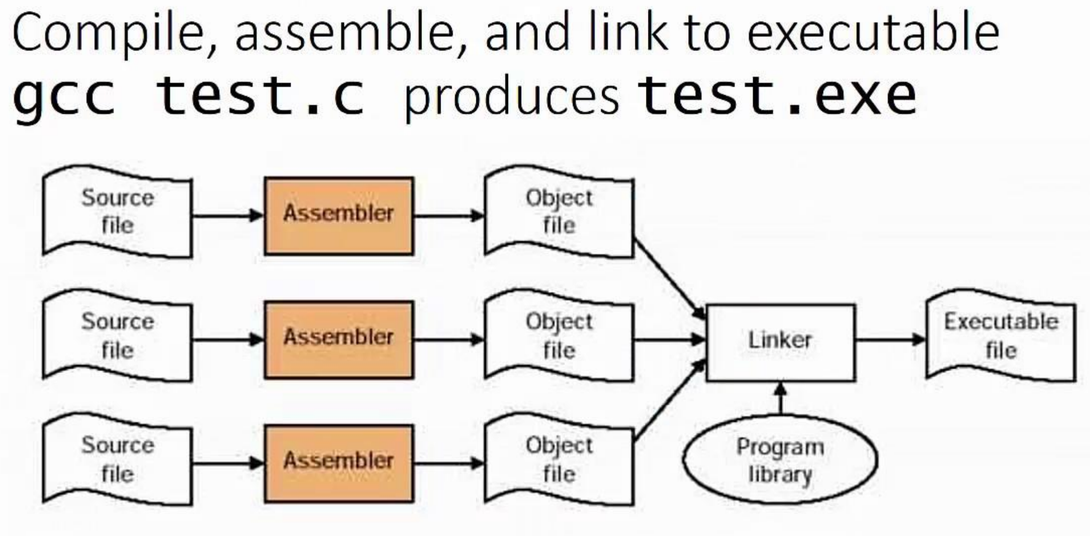
In generale e possibile programmare tutti i processori in un linguaggio a basso livello (Assembler):
massima efficienza a scapito della portabilità (differenti MCU, differenti «istruzioni macchina»)

Il linguaggio utilizzato più versatile e il C, ma alcune volte è necessario aggiungere istruzioni in assembler

Compilazione

Un progetto software è costituito da uno o piu file.c (sorgenti) e file.h (header)

1. Pre-processing Si tratta di un'operazione testuale consistente in una sostituzione letterale del codice; ad esempio la stringa #include xxx.h fa sì che il file in questione venga sostituito così com'è in quella porzione di codice;
2. Compilazione Si ha la conversione dei file xxx.c in codice macchina producendo dei file oggetto (.o);
3. Linking Partendo dai differenti file oggetto generati, li si «linka» generando un unico eseguibile finale

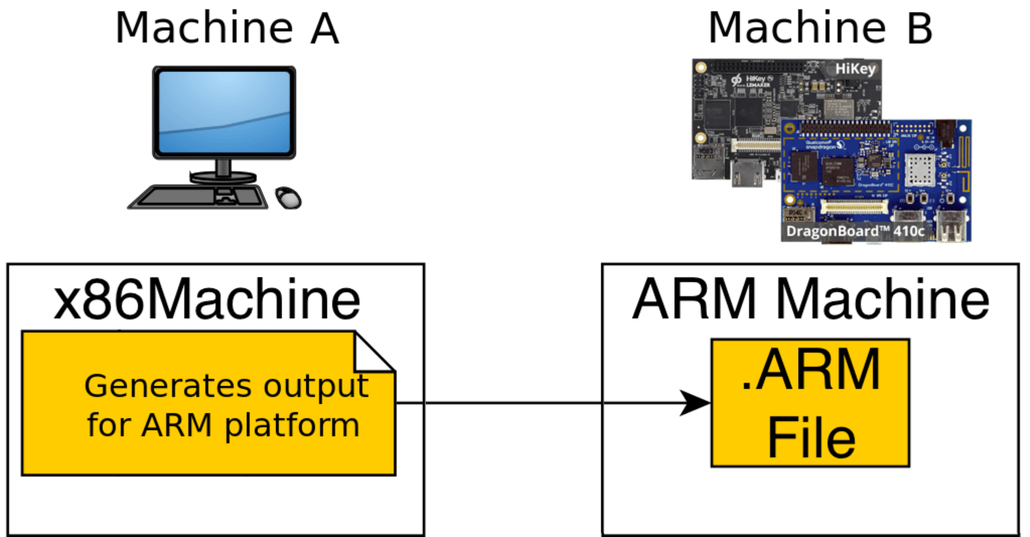


Cross-Compilazione

Flussi di processo (toolchain) che permettono di ottenere un eseguibile, per mezzo dei tre passi descritti precedentemente, in grado di “girare" su di un target differente dalla piattaforma in cui e stato sviluppato il sorgente.

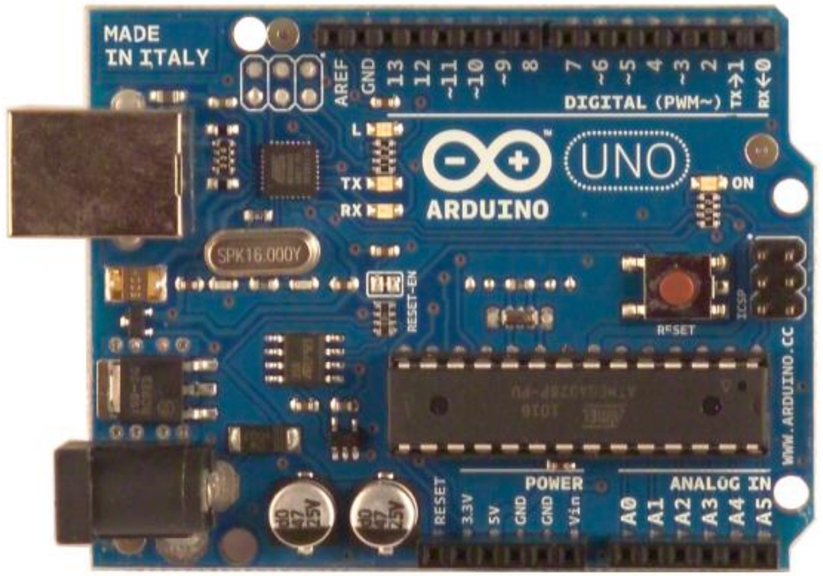
- Si lavora sempre con lo stesso ambiente per diverse piattaforme
- Si può sviluppare per piattaforme che non integrano un compilatore

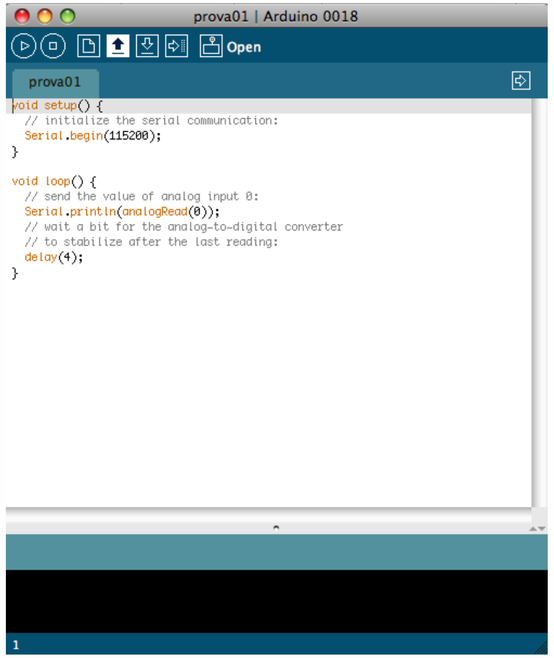
Esempio: sviluppo di app IOS/Android



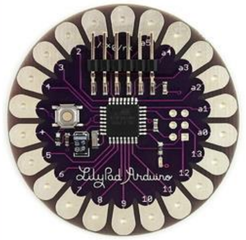
Che cos’è ARDUINO?

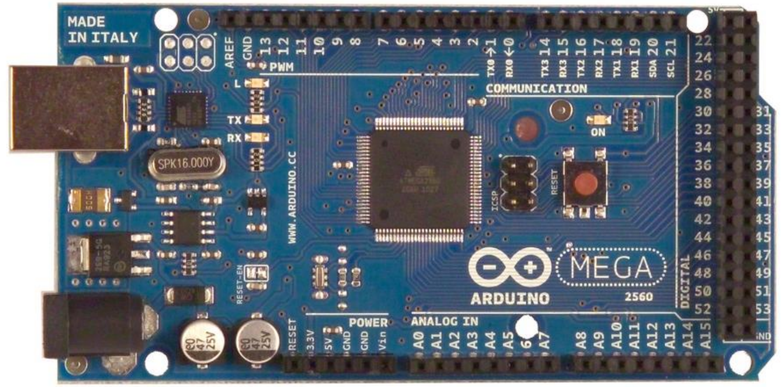
“Arduino is an open-source physical computing platform based on a simple I/O board and a development environment that implements the Processing / Wiring language. Arduino can be used to develop stand-alone interactive objects or can be connected to software on your computer.“ (www.arduino.cc, 2006)





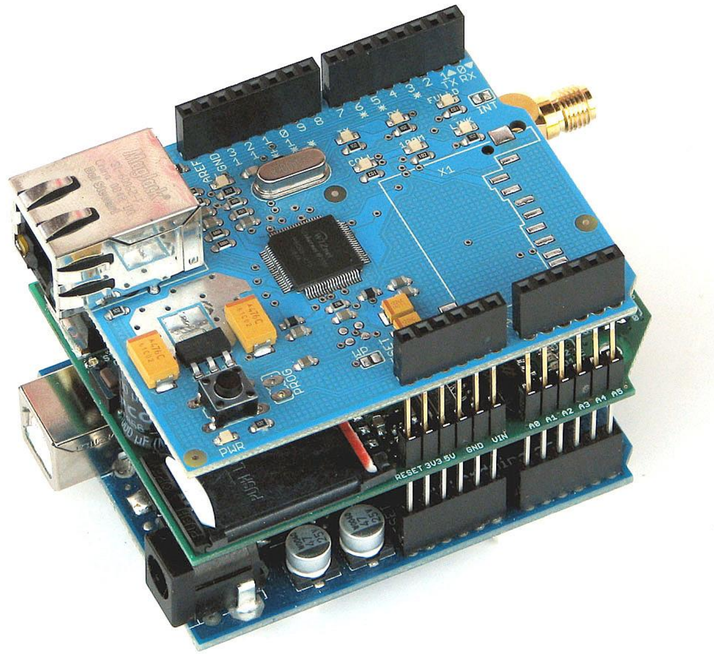
Nano Lilly Mini Mega…





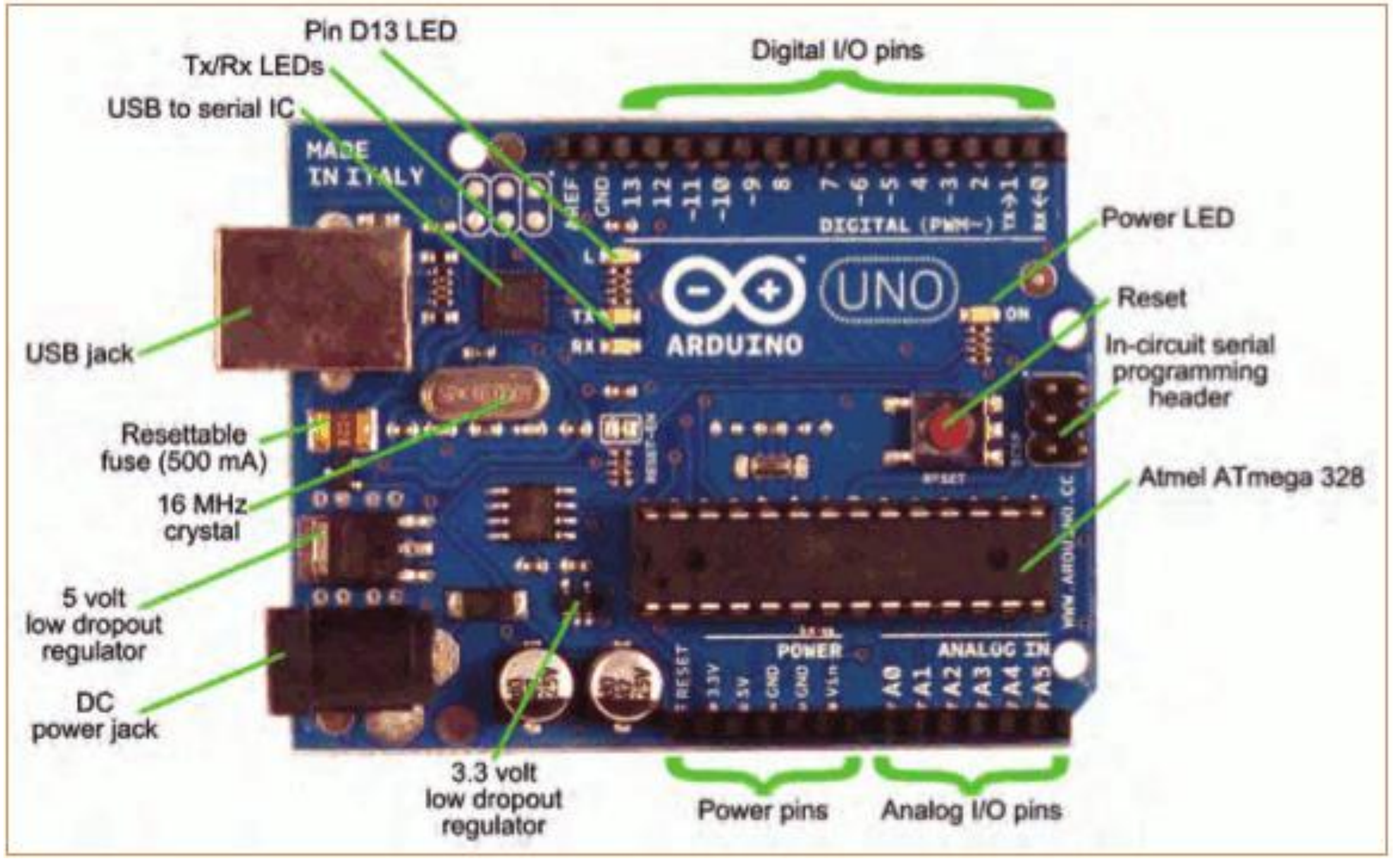
- semplice (nato per la prototipazione da parte di non elettronici)
- essenziale (minimo dei componenti)
- immediato (IDE C/JAVA)
- supportato (community, forum)
- USB (bootloader)
- modulare (shield)

modulare - shield

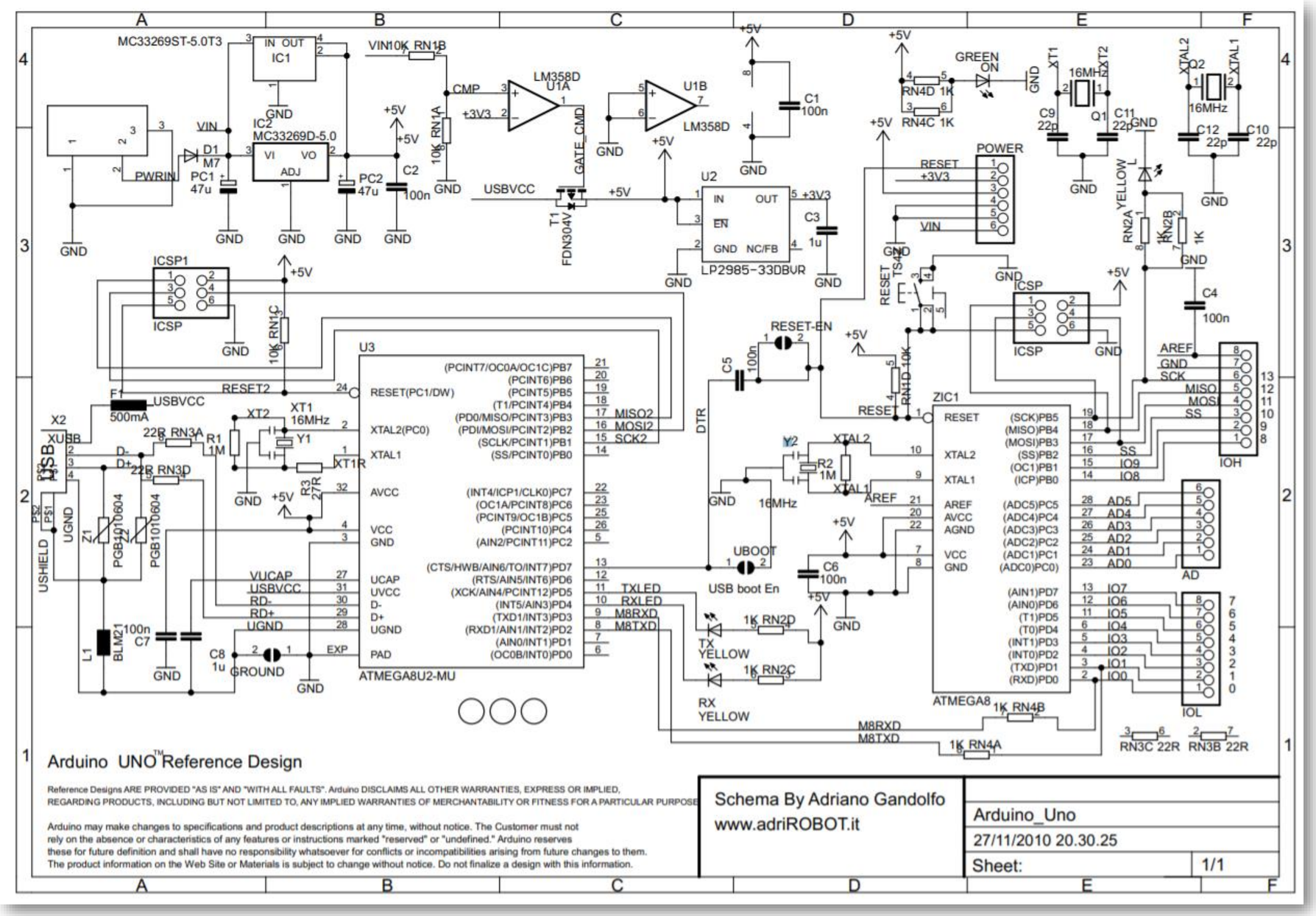


Permettono l’aggiunta di funzionalità in modo semplice

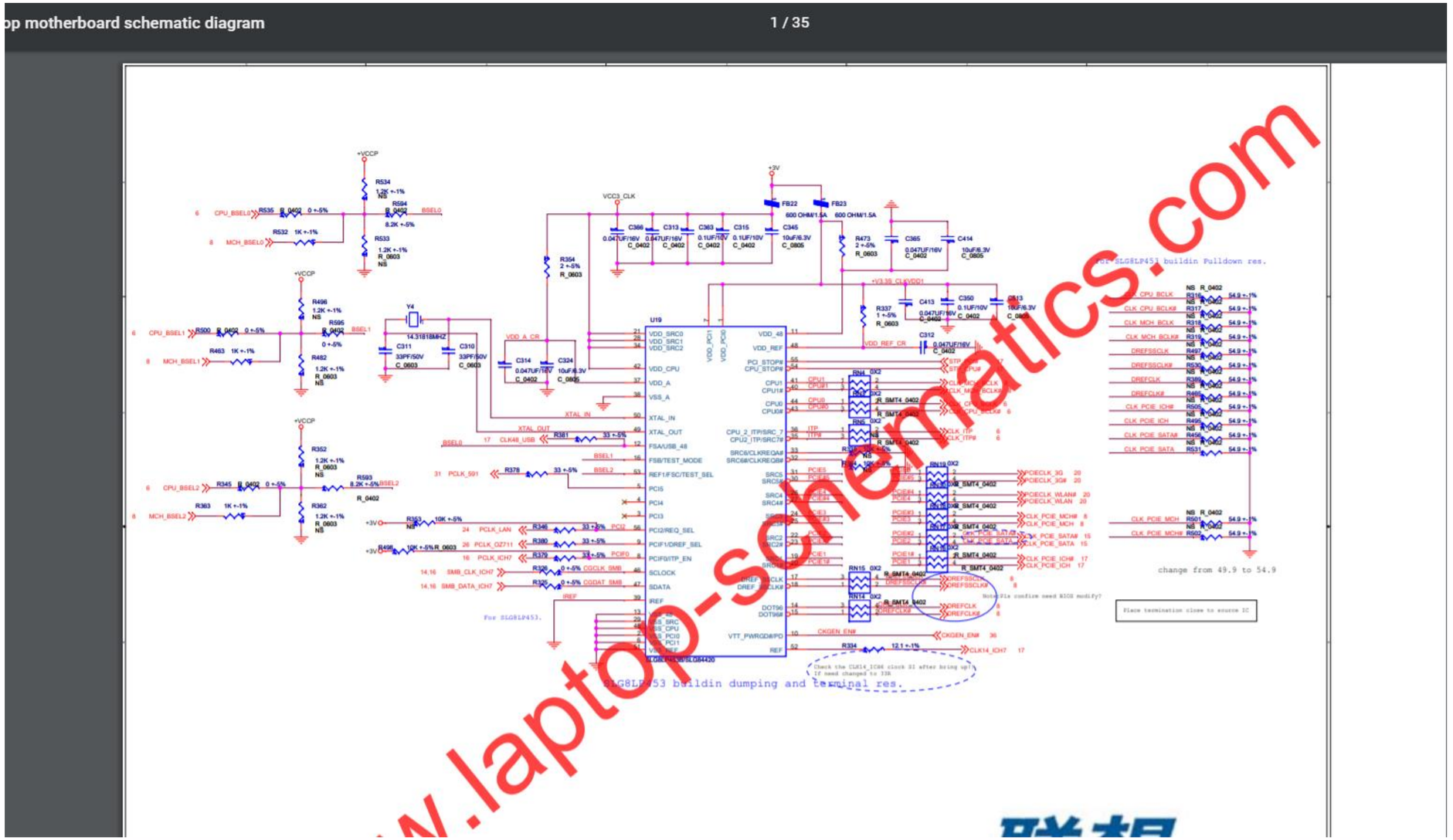
La struttura hardware di Arduino



Arduino: Schema circuitale

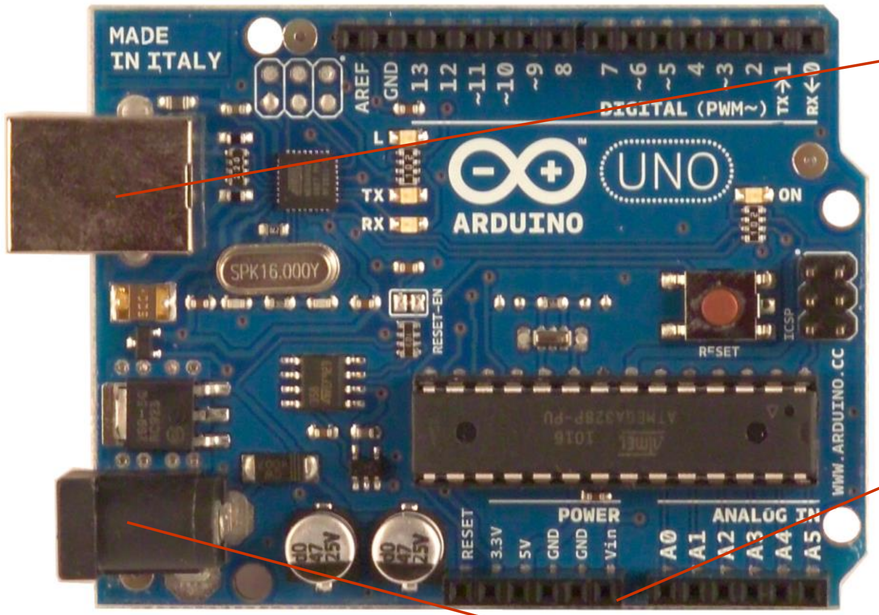


Lenovo: Schema circuitale MB



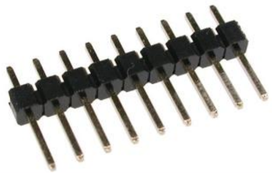
https://www.informaticanapoli.it/download/SCHEMIELETTRICI/IBM-LENOVO/Lenovo%20laptop%20motherboard%20schematic%20diagram.pdf

Alimentazione



da USB (5V)

da V-in (6 - 20V)

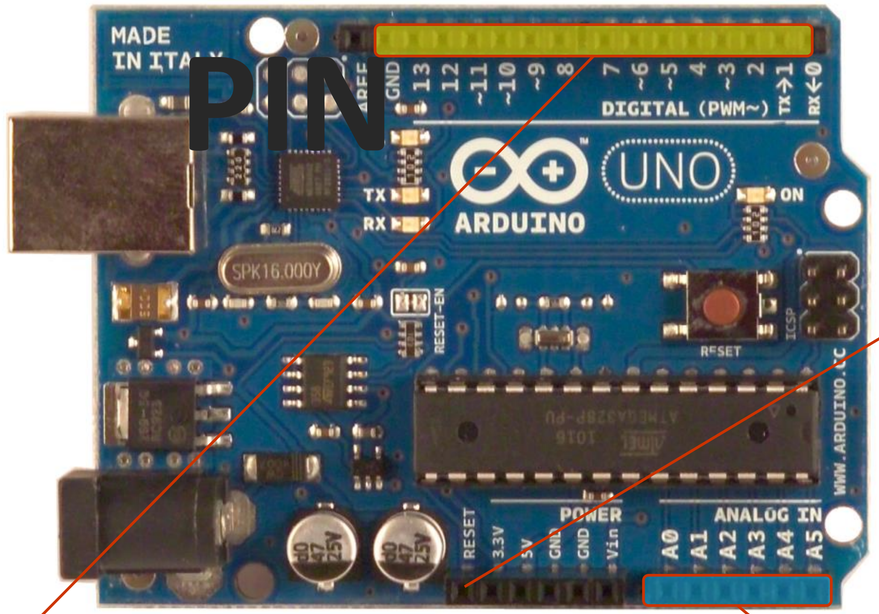


da Jack (6 - 20V)

Si consiglia una tensione tra 7 e 12V

con alimentatore esterno, sul pin Vin si ha la tensione <u>non regolata</u>

PIN



Reset permette di resettare lo stato

14 Ingressi/Uscite digitali

Possono essere utilizzati come input o output
operano ad una tensione di 5V e possono fornire fino a 40mA di corrente (ma…)

6 Ingressi analogici

risoluzione a 10bit (2^10 = 1024 intervalli di tensione differenti)

Pin Digitali

(SERIAL) pin 0 (RX) e pin 1 (TX) trasmissione e ricezione seriale. Sono connessi con l'USB e possono essere utilizzati per shield specifiche.

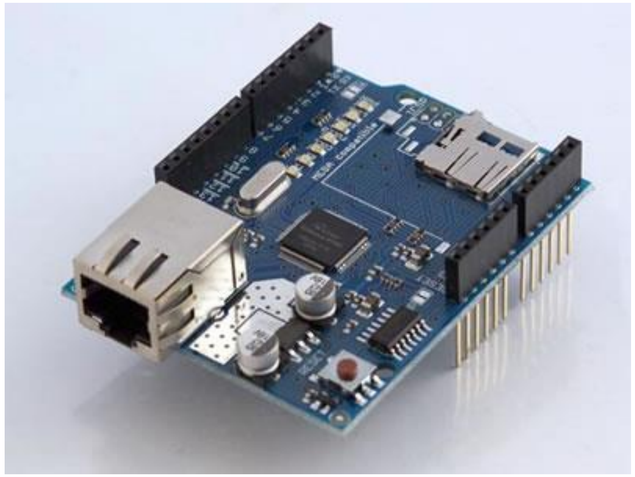
(IRQ) pin 2 e 3. Questi pin possono essere configurati per la generazione di interrupt (vedremo…)

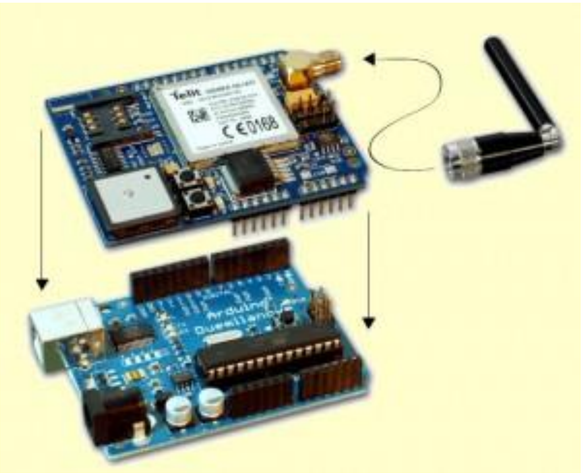
(PWM) 3, 5, 6, 9, 10, e 11. Possono essere utilizzati come PWM a 8 bit (vedremo…)

(SPI) 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). comunicazione SPI (SDCARD, Ethernet shield, GSM)

(LED) 13. Il pin 13 è collegato ad un led SMD presente sulla scheda

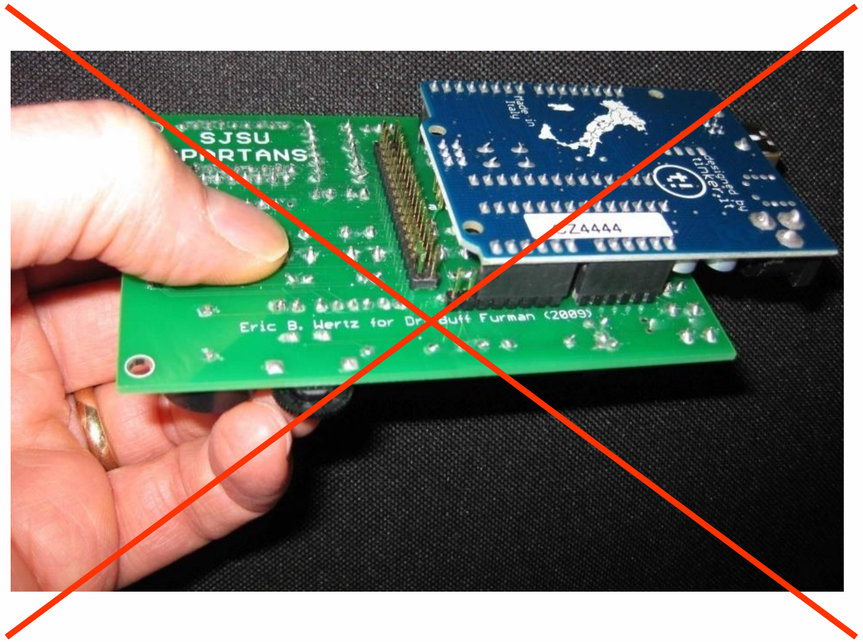
(I^{2}C) 4 (SDA) e 5 (SCL)

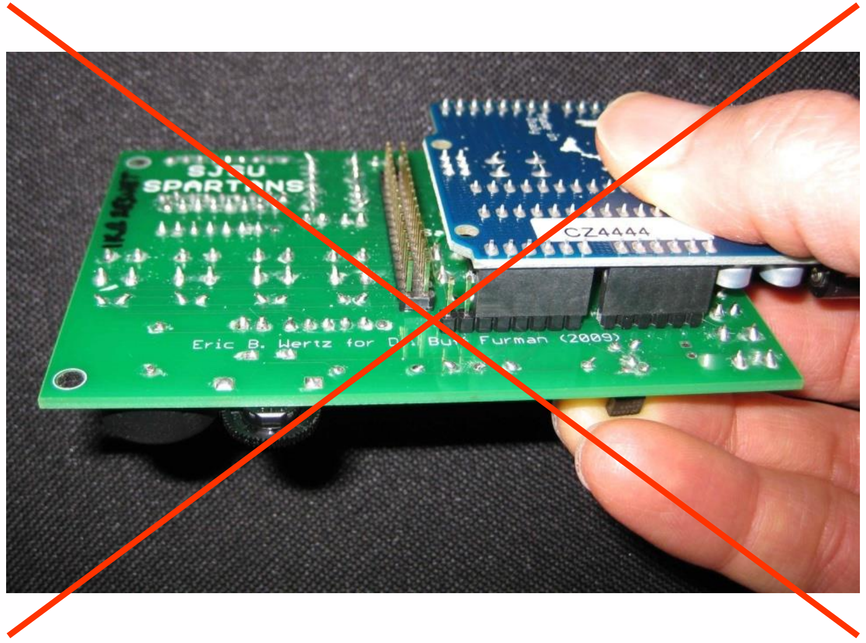




Premessa: come NON prendere I circuiti

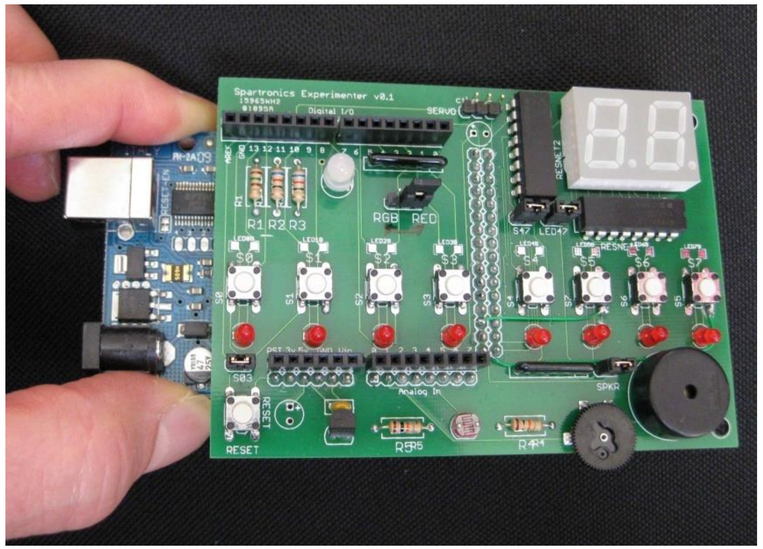
MAI!!!

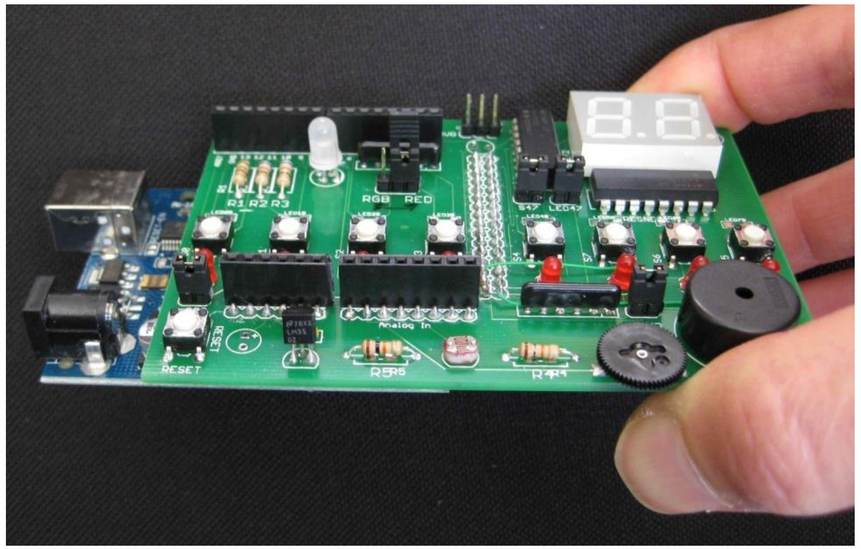




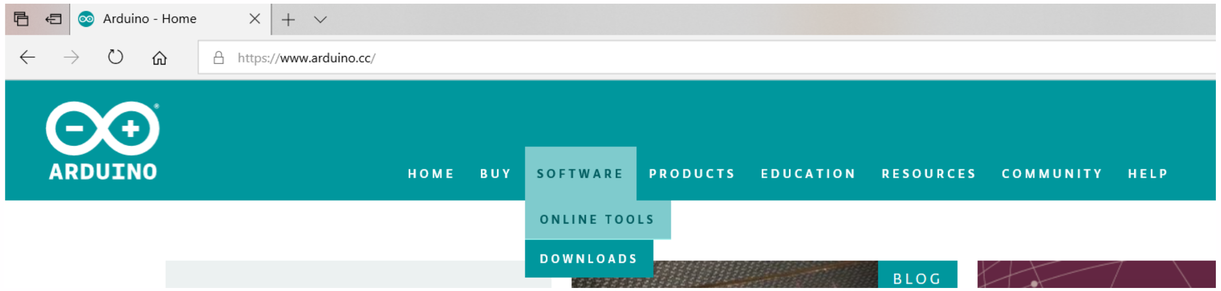
Premessa: Il metodo corretto

Sempre dai lati !!!





Installazione



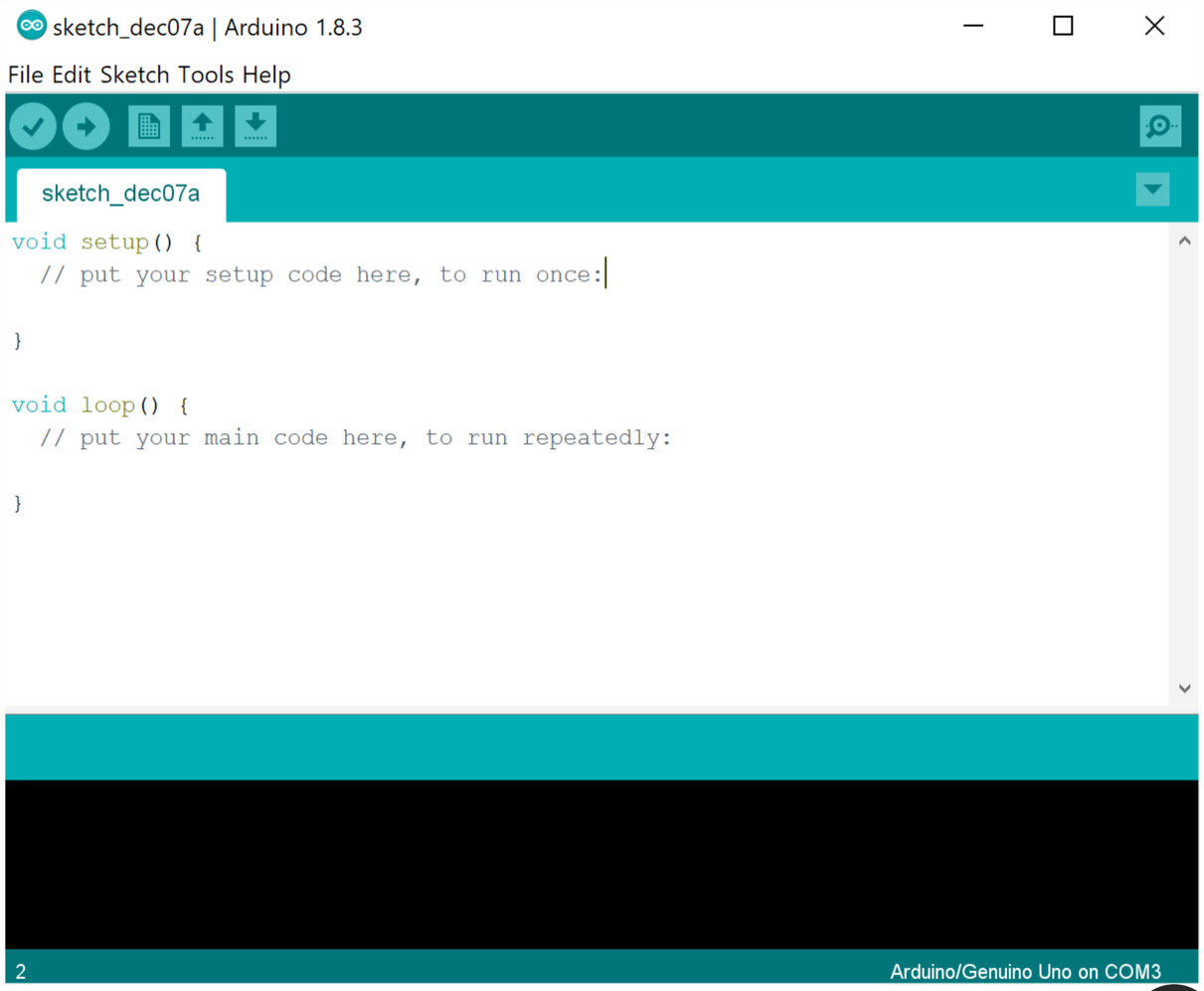
1) www.arduino.cc
2) SOFTWARE
3) DOWNLOAD
4) Selezionare la versione per il sistema operativo in uso

Ambiente di sviluppo

L’ambiente di sviluppo (IDE) è scritto in Java e pertanto e quindi risulta identico su tutti i SO

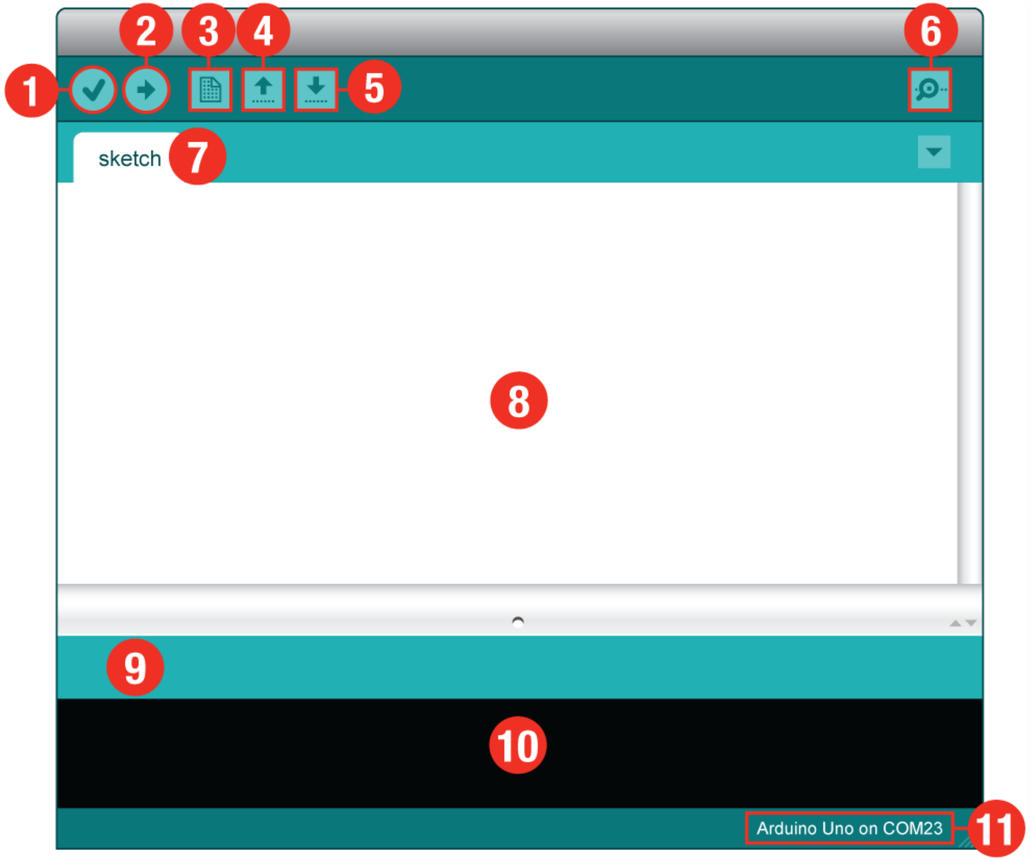
La programmazione avviene tramite linguaggio C/C++ ampliato con funzioni specifiche per le schede Arduino.

Non sono utilizzabili le comuni istruzioni di input/output su terminale (printf, scanf, cin, cout)



INTERFACCIA UTENTE

1. **Verifica:** compila il codice. Rileverà gli eventuali errori di sintassi. (CROSS COMPILAZIONE!)
2. **Carica:** invia il codice alla scheda.
3. **Nuovo:** apre una nuova scheda di codice.
4. **Apri:** consente di aprire uno sketch esistente.
5. **Salva:** salva lo sketch attivo.
6. **Monitor seriale:** si apre una finestra in cui vengono visualizzate tutte le informazioni seriali trasmesse dalla scheda. È molto utile per il debug.
7. **Nome sketch:** mostra il nome dello sketch su cui si lavora.
8. **Area Codice:** area in cui si scrive il codice del programma.
9. **Area messaggi:** area dell'IDE visualizza se ci sono errori nel codice.
10. **Console di testo:** mostra i messaggi di errore completi.
11. **Scheda e porta seriale:** mostra quale scheda e la porta seriale.



è necessario il collegamento tra PC e Arduino (solitamente via USB) per caricare lo sketch compilato

Le schede Arduino contengono un Bootloader che si occupa della comunicazione seriale e della scrittura dello sketch in memoria (vedremo che non è l'unico modo di programmare un SE)

Lo sketch caricato resta memorizzato fino alla programmazione di un altro sketch

Quando la scheda è alimentata, lo sketch viene eseguito ciclicamente. Possiamo alimentare Arduino via USB da un computer oppure tramite batteria / alimentatore (stand alone).

Lo sketch

L’ambiente di sviluppo di Arduino consente di scrivere i programmi chiamati SKETCH

I programmi sono scritti all’interno di un EDITOR, sono compilati e caricati (copiati) nella memoria della scheda di Arduino.

Funzioni principali

La funzione setup()

Chiamata all’inizio del nostro programma (sketch), serve per inizializzare le variabili, i vari pin e per iniziare ad usare le librerie. Questa funzione viene eseguita solo una volta.

```
int buttonPin = 3;

void setup()
{
  Serial.begin(9600);
  pinMode(buttonPin, INPUT);
}

void loop()
{
  // ...
}
```

La funzione loop()

Chiamata subito dopo la setup questa funzione, cicla costantemente eseguendo ad ogni ciclo le istruzioni presenti nel suo blocco.

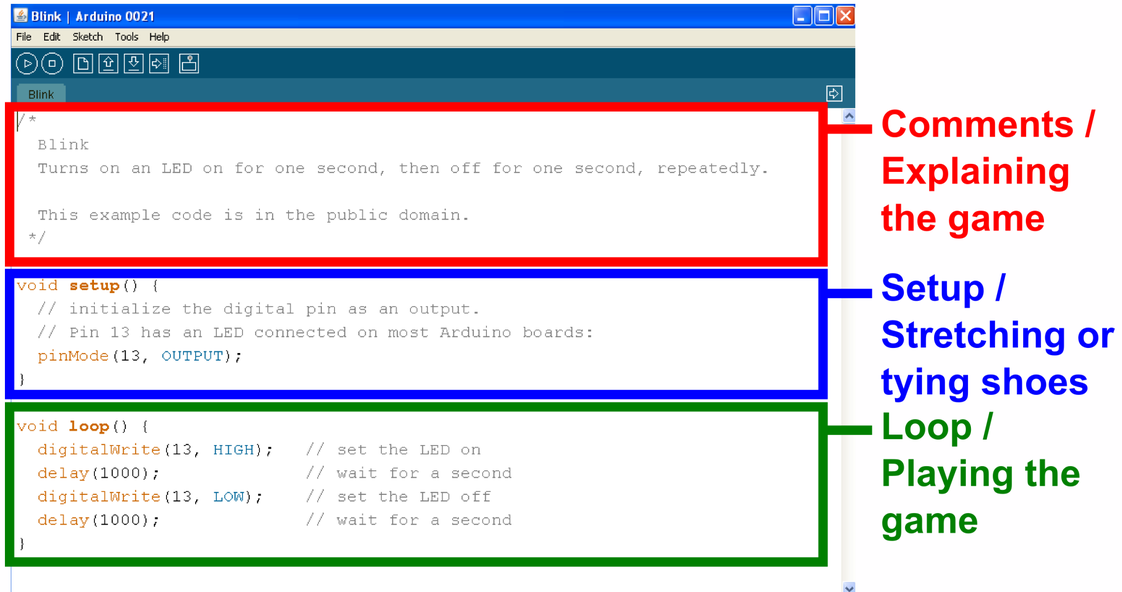
```
void loop()
{
  if (digitalRead(buttonPin) == HIGH)
    Serial.write('H');
  else
    Serial.write('L');

  delay(1000);
}
```

Sia la funzione setup() che la funzione loop() non ritornano alcun valore (void).

Tipica struttura di uno sketch

- commento che spiega cosa fa il programma, quando è stato scritto, come lo fa, chi ne è l'autore ;
- eventuale inclusione di librerie esterne;
- dichiarazione / inizializzazione delle costanti;
- prototipi delle eventuali proprie funzioni;
- dichiarazione / inizializzazione delle variabili globali;
- void setup() { - istruzioni di inizializzazione (porte d'ingresso / uscita, ecc…);
- void loop() { - istruzioni operative sui sensori e attuatori;



I passaggi dallo sketch all’esecuzione

Scrivere il codice del programma (sketch) con IDE Arduino

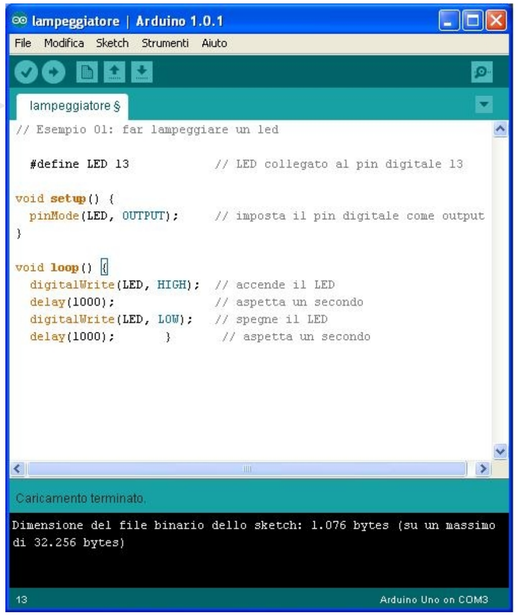
Salvare lo sketch con estensione .ino

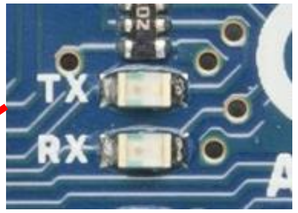
Compilare lo sketch

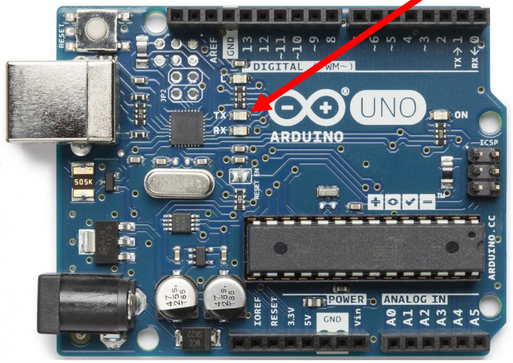
Caricare lo sketch sulla scheda Arduino

Durante il caricamento si accendono i LED TX/RX della scheda Arduino

Dopo pochi secondi viene eseguito lo sketch sulla scheda







Thank you.

Michele Martinelli – Senior R&D Engineer
Michele.martinelli@uniroma1.it

W • SENSE