IoT

Marco Savarese

A.A 2023/2024

Indice

1 Introduzione del corso - 18/09/2023

IoT significa Digital Twin in cloud.

- **Digital twin**: un Digital Twin è una copia digitale di un oggetto fisico. Non deve seguire esattamente la struttura dell'oggetto di cui intende eseguire la replica (ad esempio, voglio creare il digital twin di una macchina con sensore GPS. Se della macchina necessito solo la posizione, il digital twin della macchina sarà solo il punto che descrive la sua posizione). Il digital twin, per essere considerato un'oggetto IoT, deve essere presente in cloud, dunque accessibile anche da fuori;
- **Arduino**: un Arduino non è un microcontroller. Arduino è una board che contiene un microcontroller e altri strumenti che gli permettono di interagire con l'esterno;
- **Microcontroller**: sistema che contiene, sullo stesso chip:
  - **Microprocessore**: un piccolo processore in grado di fare il fetch (prelevare) di un'istruzione ed eseguirla;
  - **RAM**: slot di memoria volatile(-> la memoria si svuota post spegnimento) per l'esecuzione delle operazioni;
  - **Program Memory**: slot di memoria non volatile per il salvataggio dei dati. Può essere read-only. Può essere accompagnata da una EEPROM;
  - **Convertitore A/D**;
  - **Clock**.

  Tra le altre presenta delle porte per comunicare con l'esterno.

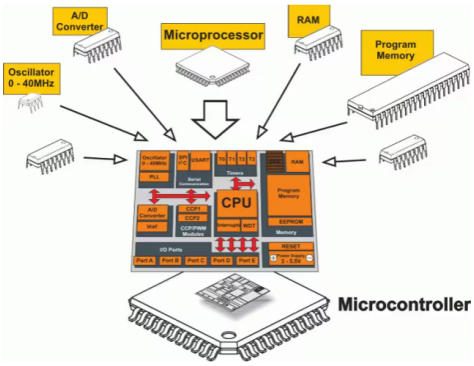


Figura 1: Struttura microcontroller

Vi sono delle differenze tra microcontroller e microprocessore.

Microcontroller:	Microprocessor
Freq. of the clock: 1 GHz	Over 3 GHz
Parallelism: 8/16/32	Parallelism: 32/64
RISC	CISC / RISC + extensions
Power: 0,001W	Power: 50W
Price/Unit (USD): 0,5	Price/Unit (USD): 50

Figura 2: Un confronto tra Microcontroller e Microprocessore

Si noti lato microcontroller la CPU limitata, con parallelismo limitato, che porta però a consumi e prezzi molto limitati (50cent a fronte di 50 se non 100€ di spesa).

1.1 Piccola panoramica su Arduino

- Non presenta un sistema operativo;
- Si compone di due funzioni principali, `void setup()`, l'operazione che avviene al boot del sistema e `void loop()`, subito dopo lo startup, esegue le operazioni all'interno fino al termine dell'esecuzione;

- Non presenta FPU (floating point unit);
- Non può eseguire Python, in quanto necessita di un sistema operativo al di sotto di esso.

2 Introduzione ad Arduino - 19/09/2023

2.1 Componenti di una scheda Arduino

2.1.1 Microcontroller

Arduino, come detto precedentemente è una piattaforma che presenta al suo interno un microcontroller.

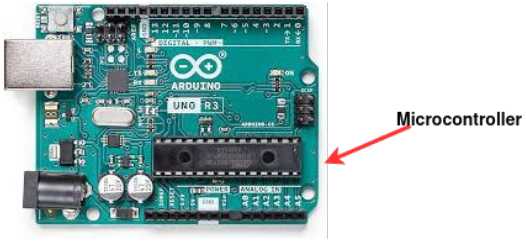


Figura 3: Microcontroller su una scheda Arduino

2.1.2 Porte di alimentazione

Normalmente, per avviare un microcontroller, bisognerebbe agire sui pin di alimentazione. Arduino invece aiuta con l'alimentazione del microcontroller, mettendo a disposizione due porte, una per l'alimentazione con connettore USB (in rosso), una per l'alimentazione con batteria (in blu).

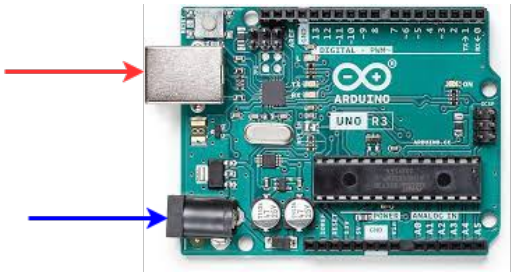


Figura 4: Connettori per l'alimentazione del microcontroller

Il filo di alimentazione USB viene utilizzato per la programmazione della board, in quanto sono presenti dei chip che permettono alla board di vedere il cavo USB come comunicazione seriale.

2.1.3 Clock

Normalmente in un microcontroller è presente un clock integrato, fatto attraverso cricuiti RC. Questa tecnica non permette di avere tempi precisi. Si potrebbe usare il quarzo, ma questi non è annegabile in un chip.
È per questo motivo che Arduino integra un clock esterno di quarzo, per rendere precise le operazioni.

2.1.4 Bottone di reset

Pulsante che, se premuto, resetta la board Arduino, eliminando il codice installato e ritornando alle impostazioni ”di fabbrica”.

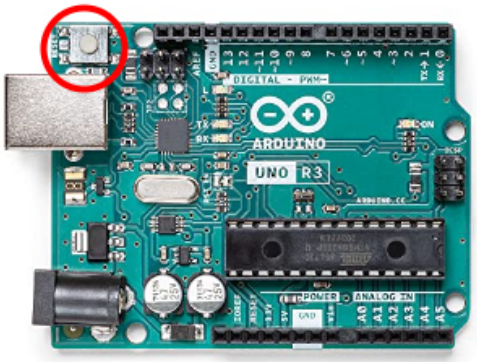


Figura 5: Bottone di reset

2.1.5 I/O

Arduino mette a disposizione 3 blocchi collegati ai pin del microcontroller per inserimento di periferiche I/O, senza che ci sia bisogno di saldarle.
Le possibilità:

- Analog In: pin per dispositivi di input analogici;

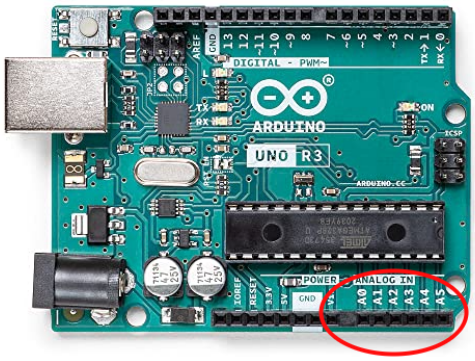


Figura 6: Porte di Analog In

- Porte per l'alimentazione: nello specifico, un pin che fornisce un output di 5V, uno per un output di 3.3V e due pin per la messa a terra;

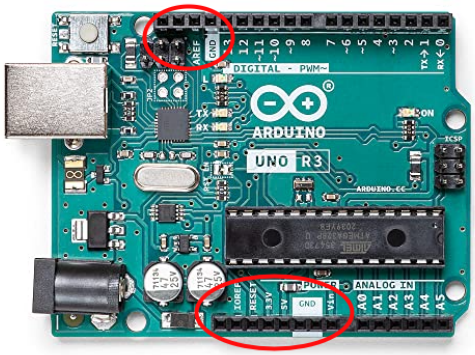


Figura 7: Porte per l'alimentazione della board (roba da elettronici)

- Porte numerate: utilizzate per input e output digitali.

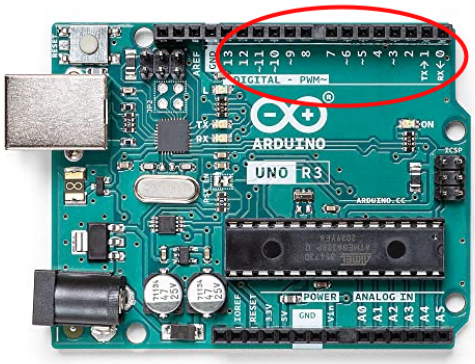


Figura 8: Porte I/O digital

Arduino fornisce solo output digitali, di base, ma ci sono 5 porte (3, 5, 6, 10, 11) dette porte PWM che possono simulare un output analogico, più avanti questo aspetto sarà esplorato.

2.2 Programmare un microcontroller

Cosa vuol dire programmare un microcontroller?
A livello di codice, vuol dire eseguire codice scritto da noi in aggiunta a librerie che vengono linkate di base da Arduino per il funzionamento della board.
Quello che scrive di base il programmatore sono due funzioni:

- `void setup()`: funzione che imposta valori di base per il programma. Viene eseguito una sola volta;
- `void loop()`: funzione che viene eseguita di continuo dal microcontroller.

Queste due funzioni sono le uniche presenti di base e definite all'inizio, ma nulla vieta di fare utilizzo di funzioni ausiliarie per il calcolo di risultati utili.
La funzione di `loop()` viene invocata a una frequenza costante (solitamente $30Hz$). Varia la sua durata dipendentemente dalla lunghezza del codice.
Un'altra funzione utile, presente già di base nell'ambiente di sviluppo Arduino, è la funzione `pinMode(PIN, MODE)`, funzione che va a configurare

ogni pin necessario dandogli una funzione, che sia o meno la funzione di default del pin.
ESEMPIO:

```
#define PINNR 13
pinMode(PINNR, OUTPUT)
```

In questo rapido esempio, si imposta il pin 13 come pin di output.
Altra funzione, `delay(mstime)` che attende un certo quanto di tempo in millisecondi. NON VERRÀ USATO IN QUESTO CORSO.

2.2.1 Modalità di programmazione di un microcontroller

Sono di particolare importanza due modalità:

- Programmazione event-based: il microcontroller rimane ”buono e fermo” fino alla presenza di un evento (INTERRUPT). A questo punto il microcontroller risponde con una routine predefinita. Il problema di questo approccio è che quest’interruzione può avvenire una volta sola. Le interruzioni funzonano bene solo se la risposta è corta ed è collegata ad un solo evento;
- Programmazione a tempo: i microcontroller hanno un framerate(tempo costante). Quando scatta il clock aggiornano gli input, li elaborano e aggiornano gli output. L’esecuzione di questo loop è il più veloce possibile.

2.3 Breadboard

Piattaforma con tanti buchi in cui si possono inserire i pin di vari sensori e attuatori, senza la necessità di saldarli.

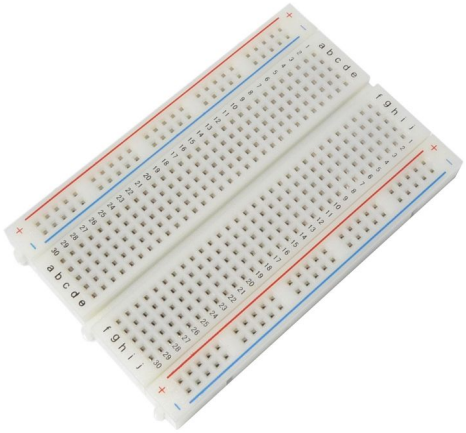


Figura 9: Breadboard

Sono collegate le file orizzontali, anche se ogni colonna funziona a sé.

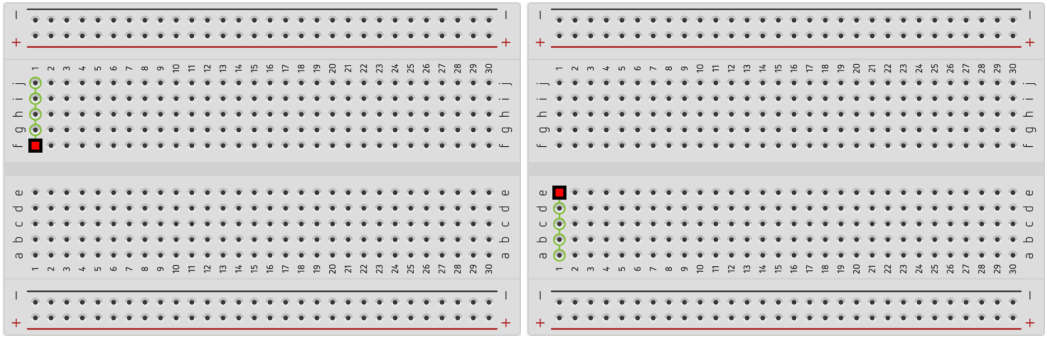


Figura 10: Collegamento file orizzontali (chiaramente viste dal basso)

Fanno eccezione le due colonne di + e - in cui a essere connessa a un certo input sarà l'intera colonna.

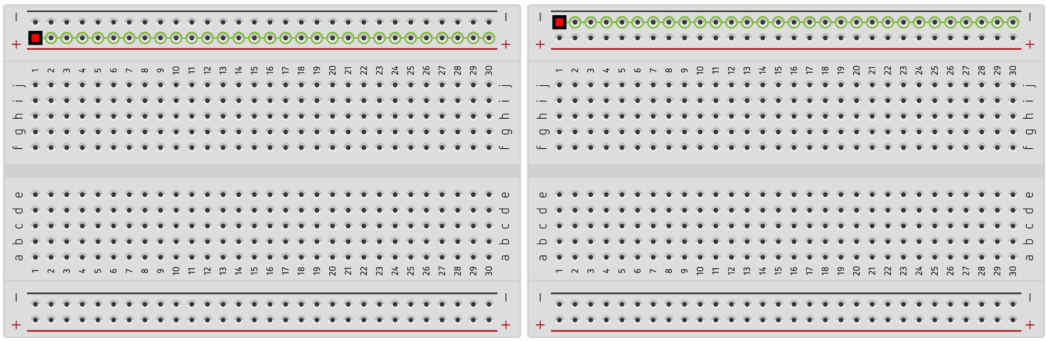
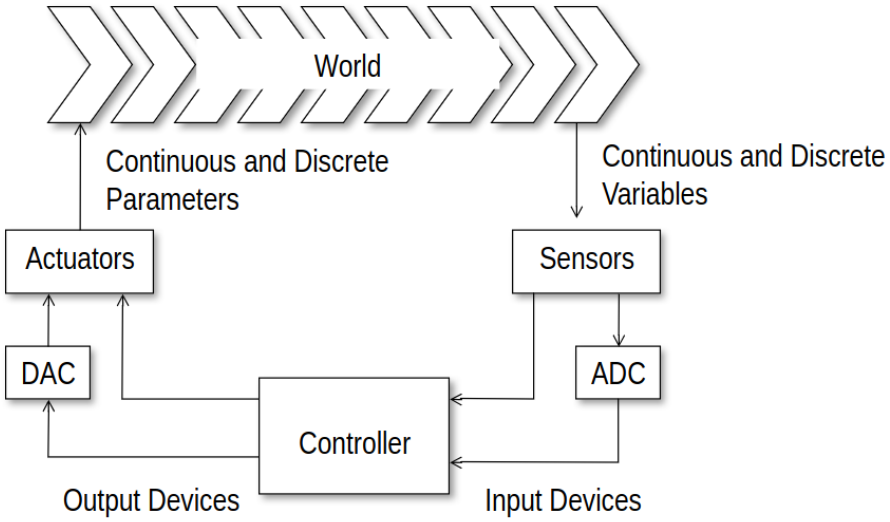


Figura 11: Collegamento file verticali + e -

Il discorso si estende anche all'altra fila di + e -.

3 Sensori - 20/09/2023

Un sensore è un dispositivo, un modulo o un sottosistema in grado di catturare le informazioni dell'ambiente che lo circonda.
Queste informazioni possono poi essere interpretate come variabili continue o discrete.
I sensori possono essere digitali o analogici.
Sono necessari, in alcuni casi, convertitori analogico-digitali per l'interpretazione dei valori.



3.1 Da segnale analogico a segnale discreto

I sensori digitali lavorano in un dominio discreto, sia dal punto di vista del tempo che dal punto di vista del valore assunto in un determinato istante di tempo t, ove invece il mondo esterno lavora in un dominio continuo.
Per rilevare e prelevare valori dal mondo esterno e utilizzarli per il nostro software è necessario **quantizzarli**.
Si possono reperire ulteriori informazioni sulla quantizzazione al seguente link.

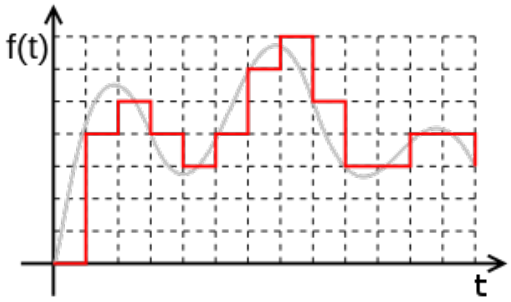


Figura 12: Processo di quantizzazione

Come è facilmente intuibile dalla figura 12, il processo di quantizzazione porta inesorabilmente alla perdita di alcune informazioni, introducendo il cosiddetto *rumore*. La perdita di informazioni è dovuta alla **frequenza di quantizzazione**, ovvero ogni quanto il valore viene prelevato.
Per evitare la perdita di dati essenziali, si può ricorrere al teorema di Nyquist-Shannon, che grazie alle sue considerazioni, fornisce un valore di frequenza di campionamento che garantisce di non perdere dati in sede di quantizzazione. Questo valore è

$$F_s \geq 2 * Fmax$$

ove:

- F_s: frequenza di campionamento;
- F_{max}: frequenza a cui sono limitati i valori in entrata.

3.2 Input di Arduino

Arduino può ”catturare” tre tipi di dati:

- **Dati analogici** (grazie ai pin da A0 ad A5);
- **Input binari** (dal pin 0 al 13, se questi sono programmati in input. LOW & HIGH);
- **Input digitali**

3.3 Conversione A/D in Arduino

La conversione analogico digitale in Arduino può essere effettuata tramite:

- **Flash converters**

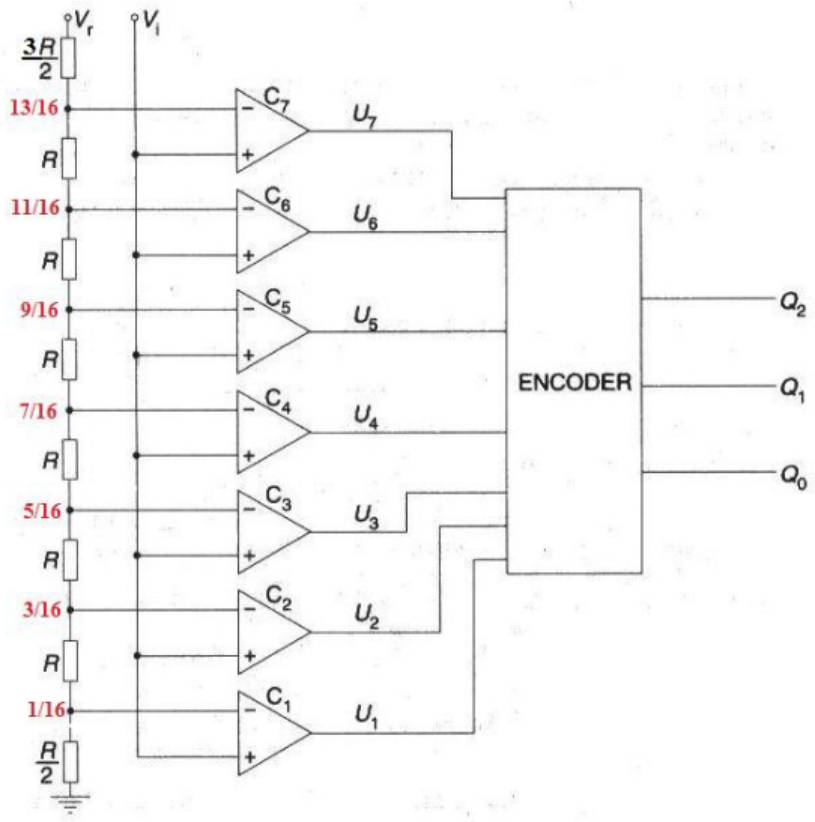


Figura 13: Struttura dei flash converters

- **Successive approximation ADC**: convertitore in grado di convertire una forma d'onda analogica continua in una rappresentazione discreta usando la ricerca binaria, ovvero prendendo in considerazione tutti i possibili livelli di quantizzazione fino a raggiungere un valore ottimale. È un approccio che sfrutta meno energia del precedente

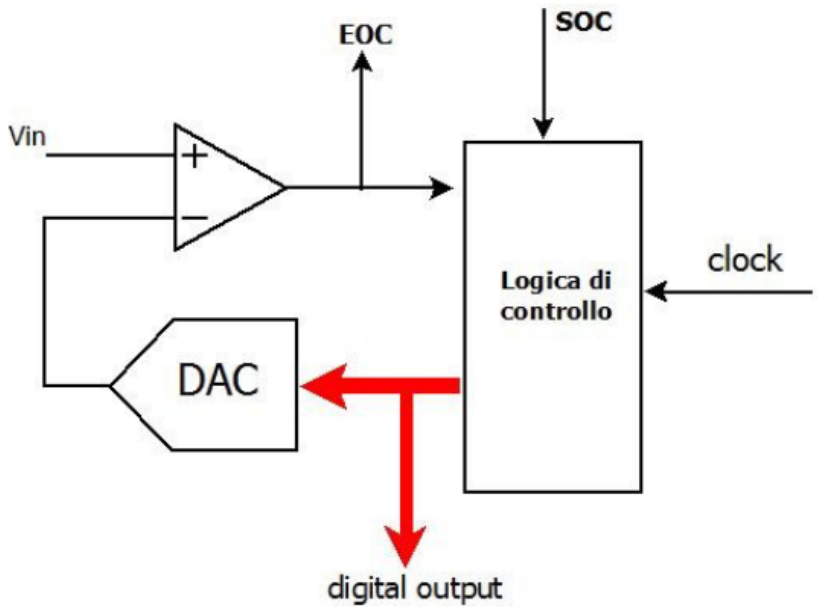


Figura 14: Successive approximation ADC

3.4 Dispositivi Analogici

In Arduino sono solo dispositivi di input.
Accettano un voltaggio in input, che varia in un certo range di valori, a differenza dei dispositivi digitali che accettano due tipi di input (HIGH - 1 - 5V, LOW - 0 - GND).
Per Arduino questo range va da 0V a 5V, per altri PLC industriali anche 10V.
Contando la larga diffusione di questo tipo di dispositivi, diventa importantissima la conversione analogico-digitale, considerata ad oggi essenziale per l'IoT e i sistemi embedded.

3.5 Caratteristiche dei sensori

Un sensore può essere classificato in base a vari parametri:

- **Precisione**: precisione del valore mostrato rispetto al valore reale. Questa caratteristica può essere enfatizzata da una corretta calibra-

zione, da un confronto tra le misurazioni nel corso del tempo, dalla linearità tra la misurazione e l'output. Si potrebbe incorrere, nell'uso del sensore, nel cosiddetto errore di isteresi, ovvero che il valore che leggiamo dal sensore potrebbe cambiare dipendentemente dal fatto che stiamo passando da un valore basso a un valore molto più alto in poco tempo e viceversa;

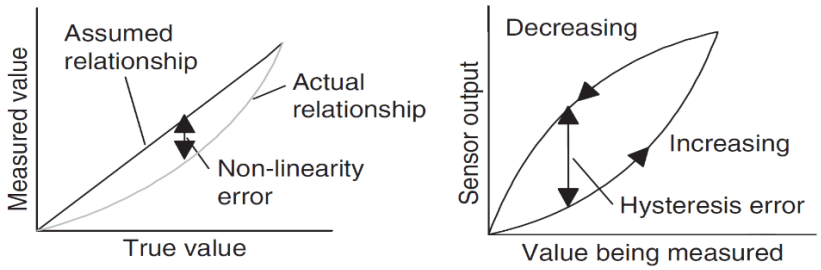


Figura 15: Precisione e errore di isteresi

- **Range**: ovvero il range di misurazione del sensore (e.g. da -200°C a 800°C);
- **Tempo di risposta**: tempo necessario al sensore per mostrare il valore misurato;
- **Sensibilità**: relazione tra la variazione dell'input e la variazione dell'output del sensore;
- **Stabilità**: abilità del sensore a mantenere una correttà correlazione tra input e output nel tempo;
- **Ripetibilità**: abilità del sensore di riportare lo stesso output a input uguali;
- **Affidabilità**: mantenere un certo livello di performance nel tempo.

3.6 Qualche sensore

3.6.1 Switch

Da informazioni sul dispositivo al mondo esterno.
Presenta:

- Due stati (aperto/chiuso);
- Due tipi (normalmente aperto/normalmente chiuso).

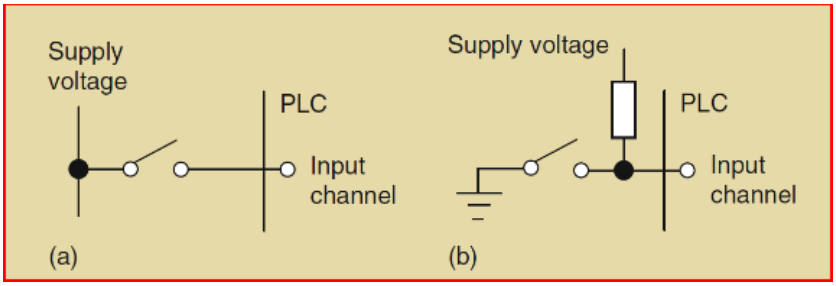


Figura 16: Switch

Sono sensori digitali.

3.6.2 Fotoresistenza

Sensori di luce con funzione di resistenza.
Dipendentemente dalla quantità di luce che li investe, aumentano la quantità di Ω di resistenza.
Sono sensori analogici.

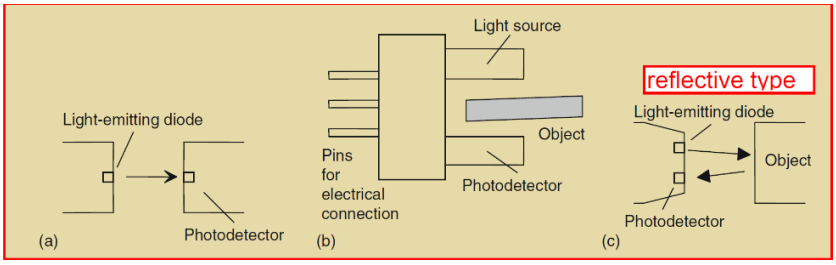


Figura 17: Fotoresistenza

3.6.3 Encoder

Si utilizzano per conoscere la posizione del sensore.
Producono un output digitale in risposta a un de-posizionamento lineare o angolare (in breve rileva il tipo di angolazione a cui si mette).

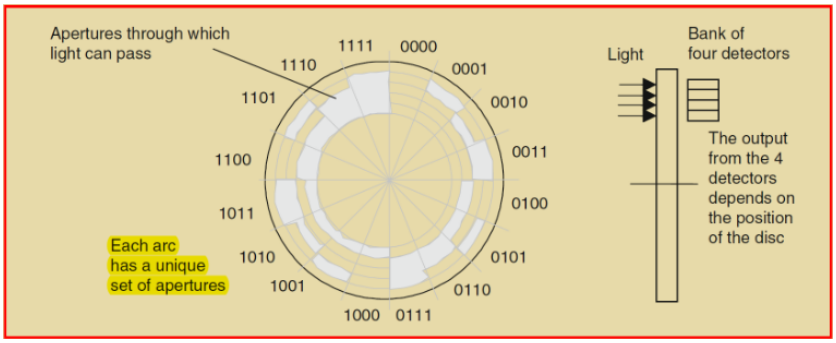


Figura 18: Encoder

Possono essere di due tipi:

- Absolute;
- Concerning.

(Non è stato spiegato cosa siano)

3.6.4 Sensori di temperatura

Di due categorie:

- Binario (on/off in base a valori limite prestabiliti);
- Resistenze termoregolate (traduzione migliore che ho potuto trovare per resistive sensors), aumentano la resistenza in base alla temperatura rilevata.

3.6.5 Altri sensori

- **Sensore piezoresistivo**: sensore che varia in resistenza se allungato;

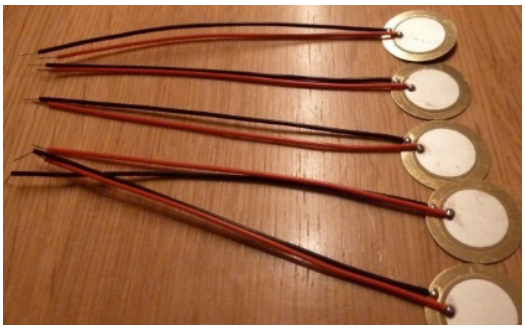


Figura 19: Sensore piezoresistivo

- **Sensore di prossimità a ultrasuoni**: misura il tempo dall'emissione di un onda a ultrasuoni fino al suo ritorno sul sensore stesso;

Figura 20: Sensore di prossimità a ultrasuoni

- **Sensori smart**: sensori provvisti di un microcontroller e di un'interfaccia di comunicazione digitale.

3.7 Per riassumere

Un **sensore digitale** è un sensore che fornisce o 0 (LOW) o 1 (HIGH) e utilizza i pin di input digitale.
Un **sensore analogico** è un sensore in grado di tradurre informazioni del

mondo esterno in resistenze variabili (e.g. fotoresistenza), utilizza i pin di input analogico. Info extra: per arduino leggono informazioni tra 0 e 1023.
Un **sensore smart** non è un sensore né analogico né digitale, in quanto fornisce un flusso di informazioni, comprese grazie all'utilizzo di protocolli di comunicazione ad hoc.

4 Attuatori - 25/09/2023

Passano informazioni dal microcontroller al mondo esterno.

4.1 Output digitale

4.1.1 LED

La quantità di luce non dipende dalla differenza di potenziale, ma dalla corrente. Se scorre abbastanza corrente attraverso il LED ci sarà una caduta di potenziale fissata dal tipo di led. La caduta di potenziale non dipende dal flusso di corrente.

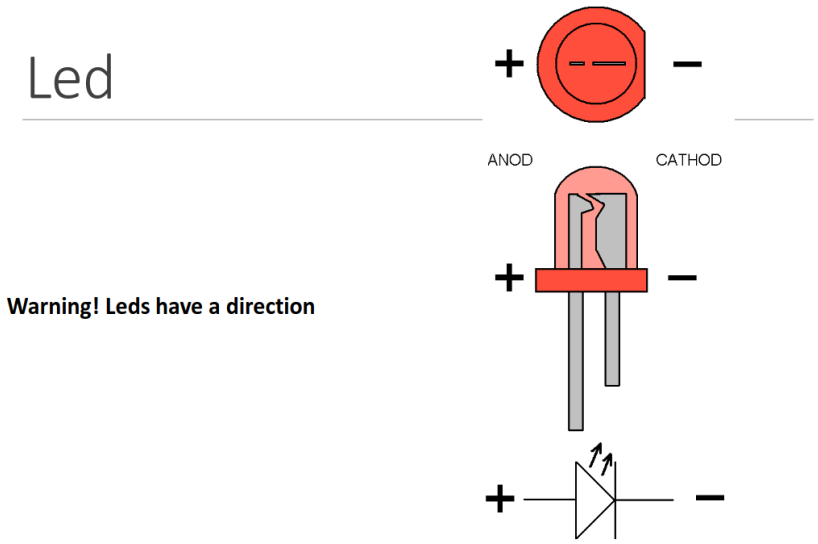


Figura 21: Direzione del LED

Color of LED	Voltage Drop (Volt)
Red	1.63 ~ 2.03
Yellow	2.10 ~ 2.18
Orange	2.03 ~ 2.10
Blue	2.48 ~ 3.7
Green	1.9 ~ 4.0
Violet	2.76 ~ 4.0
UV	3.1 ~ 4.4
White	3.2 to 3.6

Figura 22: Tabella della caduta di potenziale

Per controllare il flusso di corrente attraverso un led si può fare uso di una resistenza, calcolata in base alla legge di Ohm ($V = RI$) e alla caduta di potenziale sul led.
Come già detto, Arduino presenta un LED integrato sulla board, che interagisce con il pin 13.
Caso particolare: RGB LED, ha 4 pin, 1 normale, 3 che controllano i valori R G e B.

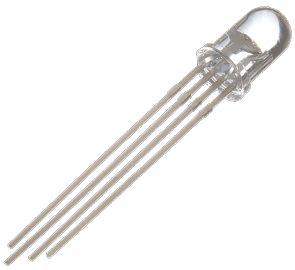
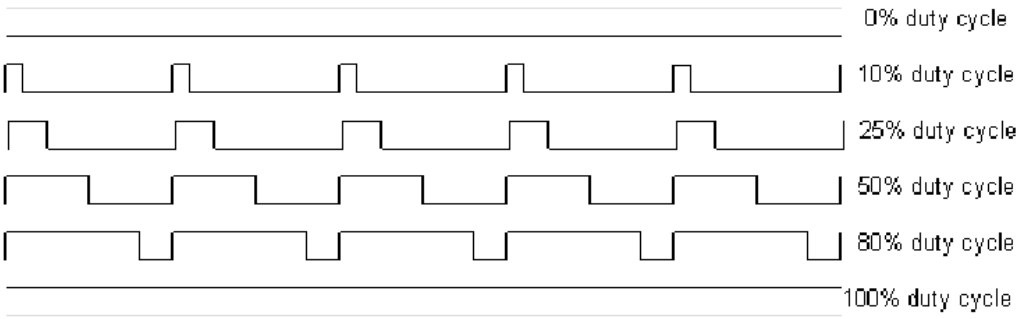


Figura 23: LED RGB

4.2 Output analogico

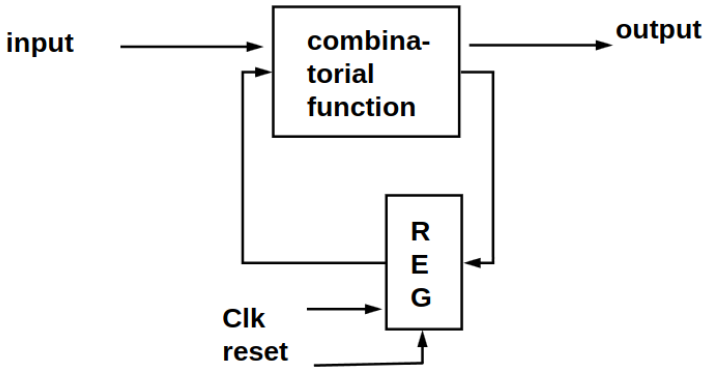
Sebbene Arduino non sia in grado di fornire un output analogico, esso può essere simulato grazie alla pulse width modulation (PWM).
Ciò che viene fatto, per fornire un valore mediano (dunque che non sia 0 o 1), è di valutare l'output in base al duty cicle dell'output fornito.



Per ottenere in uscita dai pin il segnale PWM desiderato, si utilizza la funzione `analogWrite(pin, value)` dove pin il primo argomento indica appunto il pin utilizzato, mentre value indica il valore da 0 a 255 del nostro duty cycle, proporzionato in una scala percentuale. Quindi un valore di 255 rappresenta un duty cycle al 100%, un valore di 126 indica un duty cycle al 50% e così via [Alecce, 2020].

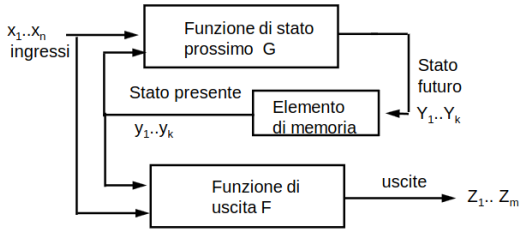
5 Macchine/Automi a stati finiti - 26/09/2023

Le macchine a stati finiti sono sistemi sequenziali che rappresentano in maniera schematica e semplificata il funzionamento di un sistema.
L'output di un sistema a un dato istante di tempo t dipende dall'input allo stesso istante di tempo, oltre che dagli input precedenti (da 0 a t).

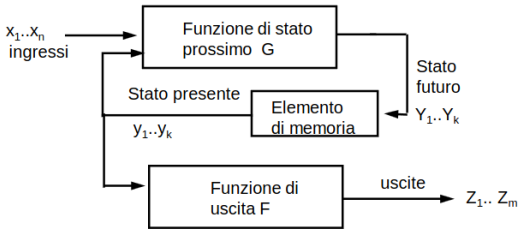


Ci sono due possibili paradigmi per la definizione di un automa a stati finiti:

- **FSM di Mealy**: l'output è dipendente dagli input;



- **FSM di Moore**: l'output dipende solo dallo stato corrente (più facile da sviluppare)



ATTENZIONE!
Non utilizzare le macchine a stati finiti quando non servono. Questa rappresentazione nasce soprattutto per semplificare la comprensione di un software. Se il diagramma inizia a essere più complesso del codice, allora esso basta per la comprensione dell'applicazione.

5.1 Come migliorare un FSM

- **Riduzioni delle transizioni di default**: per ridurre il numero di archi e migliorare la leggibiità del diagramma. Invece di aggiungere archi per la transizione in stati di default, si può aggiungere una nota e rimanere nello stesso stato;
- **Condizioni OnEntry**: azioni/output addizionali che avvengono una sola volta prima dello stato corrente;
- **Condizioni OnExit**: azioni/output addizionali che avvengono una sola volta dopo lo stato corrente.

6 Protocolli di comunicazione seriale - 28/09/2023

La comunicazione seriale è un tipo di comunicazione tra due dispositivi che avviene in maniera sequenziale, trasmettendo un bit dopo l'altro.
La comunicazione tra i due dispositivi può avvenire in maniera:

- **Assoluta**: i due dispositivi presentano un'unica linea fisica che li collega. All'inizio della comunicazione i due dispositivi dovranno concordare sui livelli di 0 e 1;
- **Differenziale**: si usano due linee di comunicazione per trasferire le informazioni. Questo tipo di comunicazione è molto utile per connessioni molto lunghe, in quanto il segnale è molto meno sensibile al rumore.

Per quanto riguarda la sincronizzazione, essa può essere effettuata in maniera:

- **Sincrona**: oltre i dati viene inviato un segnale di sincronizzazione (clock). Viene utilizzato per trasferimenti ad alta velocità. In caso di comunicazione differenziale, si può utilizzare la linea di clock differenziata;
- **Asincrona**: il clock è serializzato nei dati stessi, dunque trasmettitore e ricevitore utilizzano i dati per sincronizzarsi.

6.1 Trasmissione seriale asincrona - Il protocollo UART

Nella trasmissione asincrona i due sistemi hanno due frequenze di clock distinte. Hanno una frequenza di campionamento molto simile, ma fuori fase.
Per sincronizzare i due clock, il ricevitore deve riconoscere un evento specifico, codificato come l'inizio della trasmissione dei dati.
Il protocollo UART prevede, per l'invio dei dati:

- **Start bit**: bit che indica l'inizio di un pacchetto e consente la sincronizzazione dei dispositivi (il segnale prima di cominciare è perennemente sull'1, quando il trasmettitore comincia la trasmissione il segnale passa a 0 per un certo Δ_t);

- **Pacchetto di 8 bit**: informazione da trasmettere;
- **Bit di parità**: facoltativo, permette di verificare se il pacchetto è stato trasmesso in maniera corretta;
- **Stop bit**: al contrario dello start bit, il segnale passa a 1 per un certo Δ_t per indicare la fine dell'attuale trasmissione (e magari l'inizio della successiva).

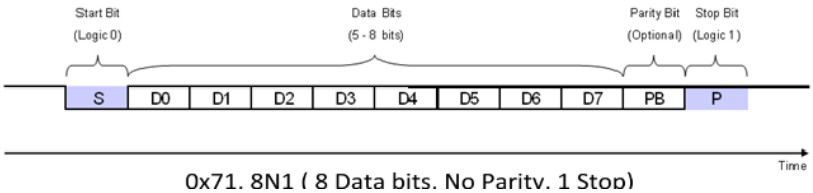


Figura 24: Pacchetto UART

6.1.1 Problematiche di UART

Per trasmettere 8 bit con UART si può notare come siano necessari 11 bit, tra start end e parity bit. Questo introduce dunque un overhead di poco più di $1/3$.
Il ricevitore, inoltre, non può confermare l'arrivo del pacchetto, dunque, anche in caso di errore di trasmissione, il trasmettitore continuerà a inviare dati.

6.1.2 Ricevitore

Lato ricevitore, per minimizzare gli errori di trasmissione è necessario che i dati vengano prelevati nel mezzo di ogni periodo di clock, dunque che la frequenza di clock del ricevitore sia un multiplo della frequenza di clock del trasmettitore.

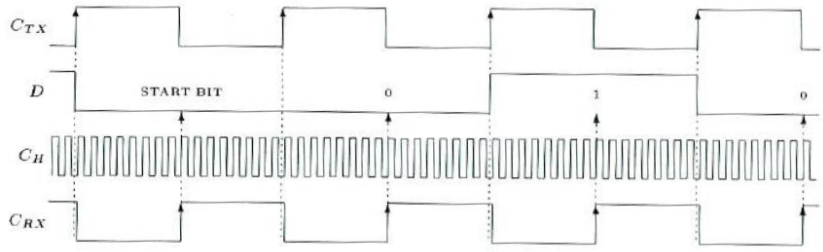


Figura 25: Esempio comunicazione tra trasmettitore e ricevitore UART

6.2 Altri standard di comunicazione seriale

6.2.1 I^2C

Inter-IC è stato sviluppato dalla Philips negli anni 80. Viene utilizzato per dispositivi non troppo lontani tra loro.
Il protocollo introduce 7 bit per la selezione del ricevitore (dunque un trasmettitore può inviare fino a 128 ricevitori, di cui 112 raggiungibili e 16 riservati).
Sono presenti due linee di comunicazione, una per i dati (SDA - Serial Data Line) e una per il clock (SCL - Serial Clock Line). Il protocollo è a tutti gli effetti un protocollo asincrono, con SCL che è il segnale di start che avvia la comunicazione.

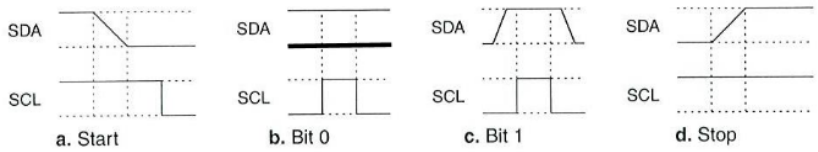


Figura 26: Funzionamento comunicazione Inter-IC

In assenza di trasmissione, SCL è perennemente a 1, la comunicazione comincia nel momento che SCL passa da 1 a 0, in cui vengono mandati i dati presenti sulla SDA. La comunicazione termina quando SCL torna a 1 e ci rimane, fino alla prossima comunicazione.

6.2.2 SPI

Basato sulla trasmissione a due vie asincrona, è un protocollo che prevede la presenza di un dispositivo master e uno o più dispositivi slaves.
La novità di questo protocollo è che gli slaves (analoghi dei ricevitori) possono rispondere al master, in quanto la comunicazione prevede almeno 4 linee:

- **Clock seriale**;
- **MOSI**: Master Output Slave Input, per comunicazioni da master a slave;
- **MISO**: Master Input Slave Output, per comunicazioni da slave a master;
- **SSX**: Slave Select, la X è sostituita dal numero dello slave. Abbiamo uno Slave Select per ogni slave con cui il master può comunicare, come si vede nella figura 27.

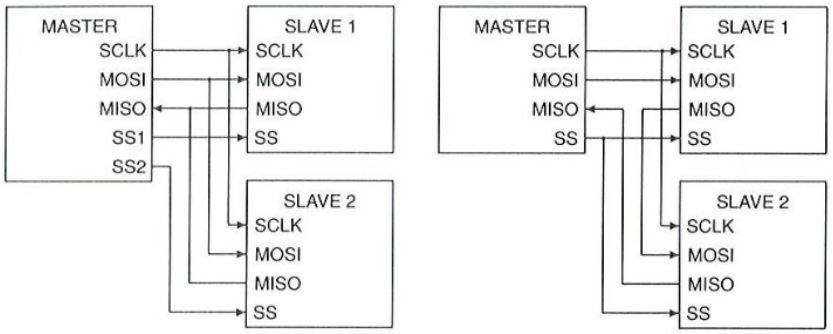


Figura 27: SPI

La comunicazione parte dal master, facendo passare lo SS specifico dello slave con cui vogliamo comunicare da 1 a 0. Il master manda un bit tramite il MOSI, lo slave risponde al master tramite il MISO, in modo da confermare l'arrivo del messaggio. Questo avviene fino al completo invio di un pacchetto da 8 bit.

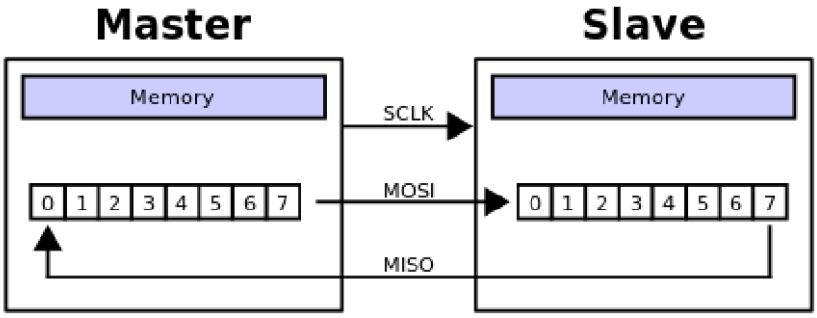


Figura 28: Comunicazione Master-Slave

6.2.3 RS-232

Protocollo seriale molto utilizzato in passato per stampanti e mouse.
Presenta 3 cavi:

- **Trasmissione**;
- **Ricezione**;
- **GND**.

Fa utilizzo del tipo di trasmissione *unbalanced*, per cui c'è una singola linea di trasmissione e il valore logico trasferito è codificato in base al valore di voltaggio rispetto alla messa a terra. I ivelli logici di 0 e 1 sono definiti 0 a 15V e 1 a -15V, mantenendosi a 1 quando è in idle.

6.2.4 RS-485

Alternativa a RS-232.
Prevede una comunicazione tra al massimo 32 trasmetittori e 32 ricevitori. Fa utilizzo della trasmissione differenziale, dunque usando due linee per la comunicazione.

6.2.5 CAN

Control Area Network. Non sono state date ulteriori informazioni, se non che è molto utilizzato nell'ambito automotive.

6.3 Conclusioni

Tra i protocolli riportati, non ce n'è uno migliore dell'altro, ma ognuno ha i propri punti di forza (e di debolezza) rispetto agli altri. La scelta del protocollo dipende dunque da diversi parametri del progetto, tra cui:

- Velocità di trasmissione necessaria;
- Rilevamento e correzione degli errori;
- Numero di ricevitori necessari;
- Modalità di scelta del ricevitore
- Sincronizzazione.

In alcuni casi, dunque, potrebbe essere necessario la creazione di un protocollo ad hoc.

Riferimenti bibliografici

[Alecce, 2020] Alecce, A. (2020). Tutorial – Arduino spiegato facile – PWM (Pulse Width Modulation). https://antima.it/tutorial-arduino-spiegato-facile-pwm-pulse-width-modulation/.