SISTEMI EMBEDDED AND INTERNET OF THINGS

Prof. Alessandro Ricci

AA 2017-2018

Appunti di Greta Bucciarelli

Sommario

Modulo 1.1 INTODUZIONE AI SISTEMI EMBEDDED

Definizione

Sistemi Embedded: sono sistemi di elaborazione special-purpose (cioè che svologono una specifica funzione/compito). Essi sono incorporati (Embedded) in sistemi/dispositivi di diverse dimensioni

→Interagiscono con il mondo fisico tramite sensori e attuatori

→Hanno una parte HW e una parte SW→software embedded (questa alcune volte non è presente)

Diffusione → elettronica di consumo, home appliances, office automation, business equipment, automibili …

Caratteristiche

Ogni sistema embedded tipicamente esegue uno specifico programma ripetutamente → eseguire un'applicazione specifica (minimizzare le risorse da utilizzare, massimizzare la robustezza)

→user interface dedicata

- Risorse Limitate (tightly constrained) e Efficienza
  - Vincoli progettuali in merito alle risorse (CPU, memoria...)
- Design metrics
  - Misura di una carateristica dellimplementazione/progettazione
    →Costo (inclusi i costi Run TIme)
    →Dimensioni, performance, consumo di energie
- Orientamento all'efficienza
  - Energy, cost-size, run-time, weight, cost efficent
- Affidabilità → dependability, reliability, availability
  - Inclusione di sistemi critici → ambito bio-medicale, trasporti ecc...

  → Quindi si richiede un'alta probabilità di corretto e continuo funzionamento, safety (non causare danni agli utenti/ambiente) e security (norme di sicurezza e privacy).
- Reattività e Real-time
  → Devono reagire prontamente agli stimoli ed eseguire elaborazioni/azioni in real-time
  → Hard real-time, violazione di deadline può causare problemi critici al sistema

CPS (sistemi cyber-physical)

Sono sistemi che integrano computazione con processi fisici

→ aspetti specifici rispetto ai sistemi di pura elaborazione delle informazioni

- Gestire tempo → correttezza oltre alle performance
- Concorrenza → processi fisici tipicamente in parallelo
- Reattività → eventi sincroni (reattivi VS trasformazionali)

Ci sono 3 sotto sistemi

- Parte fisica → include dispositivi/sistemi meccanici, biologici o processi chimici, o operatori umani (utenti)
- Parte computazionale/embedded →sono date più piattaforme ognuna contenente sensori, attuatori o computer/s
- Parte di rete →fornisce meccanismi/supporti per fare in modo che i pc comunichino

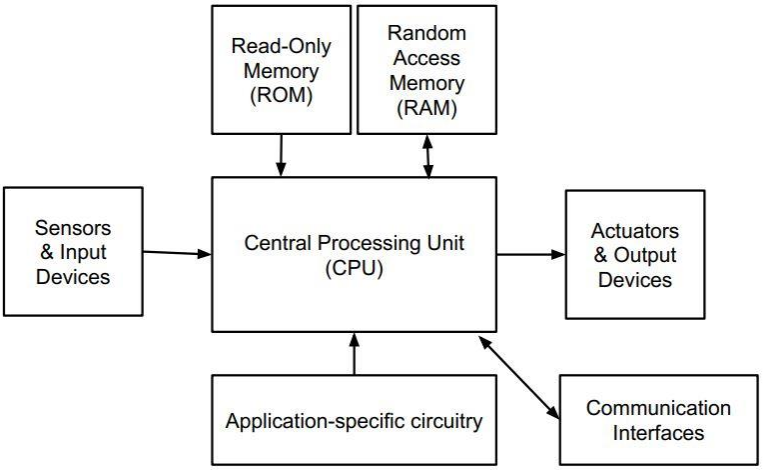
Industria 4.0 → quarta rivoluzione industriale → sfruttano IoT, Cloud e sistemi cyber physical

Physical computing →indica sistemi computazionali che interagiscono con il mondo fisico →framewok per interazione human-machine

Wearable computing → sistemi indossabili come smart watch, smart glasses, sistemi handfree e sistemi di realtà aumentata

ARCHITETTURA SISTEMA EMBEDDED

ARCHITETTURA HARDWARE



PROCESSORI

Tre tipologie

- **Processori general purpose** → processori che hanno un architettura e un insieme di istruzioni predefinito. E uno specifico comportamento dal software in esecuzione
- **Processori single purpose** → circuiti digitali per i mplementare una specifica funzionalità.
  ASIC= application-specific integrated circuit
- **Application Specific processor (ASIP)** → via intermedia, processori programmabili, logica specificata a livello SW (ES: microcontrollori)

Single Purpose → ci sono diversi tipi di implementazione

- Full-custom/VLSI
  - L'implementazione avviene progettando ottimizzando tutti i livelli (modo custom)
- Semi-custom
  - ASIC (application specific IC), in questo caso si parte da un certo insieme di livelli parzialmente/totalmente costruiti (implementandovi sopra il processore)
- PLD – programmable logic device
  - Tutti i livelli sono già presenti → programmare un IC opportunamente
  - I livelli implementano circuiti programmabili (la programmabilità è data dal creare/distruggere connessioni

PLA → programmable logic array → array programmabili da porte AND e OR

PAL →programmable array logic → solo un tipo di array

FPGA → field programmable gate array

- Circuiti integrati con funzionalità programmabili via sw → più generali rispetto a PLA e PAL
- Implentazioni logiche complesse
- Linguaggi di programmazione → Verilog e **VHDL**
- Modalità di programmazione visuale → **labView**

Micro-controllori

→ Nati come evoluzione dei micro-processori, integrano in un solo chip un sistema di componenti

→Massima autosufficenza funzionali per funzioni embedded

→Composto da processore, memoria permanente, memoria volatile e canali di I/O

→**CPU CISC** con architettura di Von Neumann (normalmente), ora stanno uscendo CPU ad architettura RISC esempio Texas Instruments)

→ Esistono a 8, 16, 32 bit

CONFRONTO MICROCONTROLLORI E MICROPROCESSORI

Caratteristica	Microcontrollore	Microprocessore
Velocità massima di clock	200 MhZ	4GHz
Capacità elaborativa massima in MegaFLOPS	200	5000
Potenza minima dissipata in watt (in stato di elaborazione)	1	50
Prezzo minimo per singola unità in USD	0.5	50
Numero pezzi venduti annualmente (in milioni)	11000	1000

Single-board micro-controller→ es: arduino

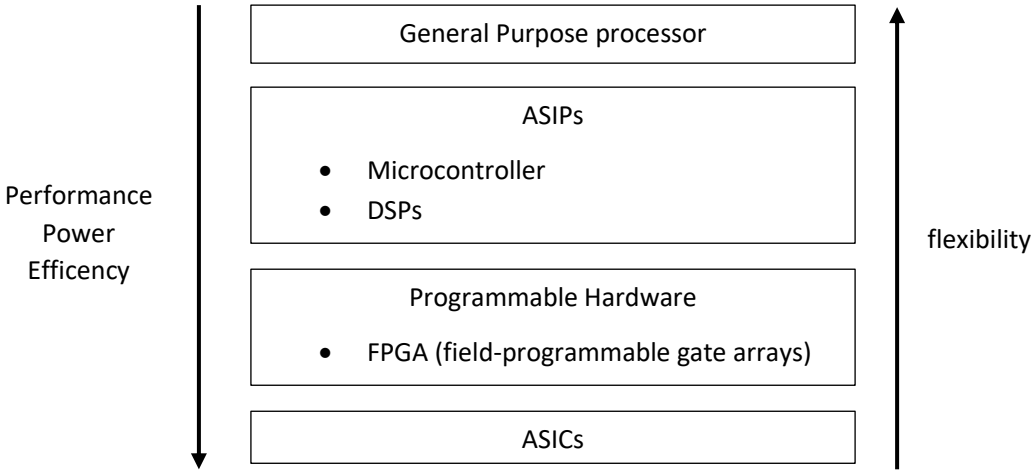
Incorporano in un'unica scheda il microcontrollore e la circuiteria

→ Così lo sviluppatore può usare subito la scheda →evitando la parte di progettazione e di sviluppo HW di contr

SOC E SINGLE-BOARD CPU

- In questo caso è un chip stesso che incorpora un sistema completo, che include CPI, memoria, controllori, etc
  - tipicamente usati per creare delle single-board CPU
- Esempi:
  - BROADCOM BCM2837 64bit ARMv8 Cortex A53 Quad Core (1.2Ghz)
    - usato in Raspberry Pi 3
  - ARM Sitara AM335x SoC - including ARM Cortex-A8 processor
    - usato in BeagleBone, Arduino Tre
  - ARM Cortex A7
    - usato in Raspberry Pi 2
  - AMD Geode
  - Texas Instruments OMAP3530

Quale tecnologia scegliere



MCU e SOC

→ La scelta dipende dai requisiti del sistema ed è sempre trade off

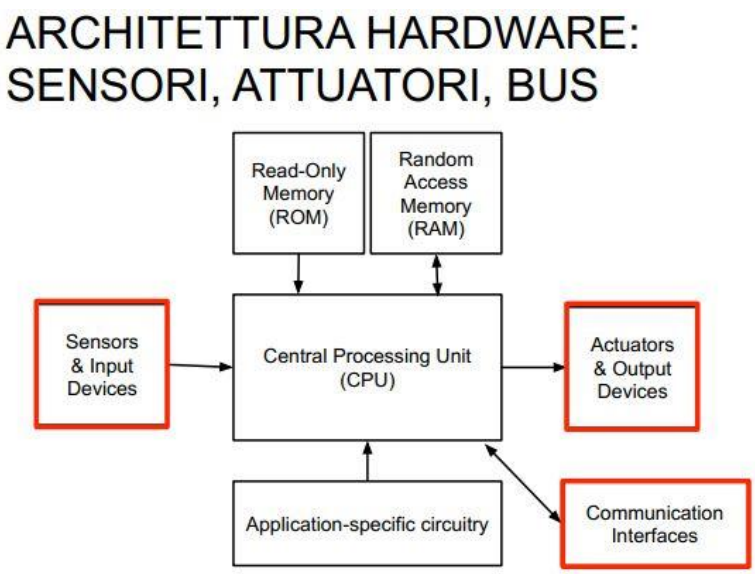
→ Oggi l'utilizzo di microprocessori e microcontrollori è sempre più frequente

→ Facilita lo sviluppo per la creazione di famiglie di prodotti

→ Processori con set predefinito di istruzioni = rapidità di implementazione

→ La programmabilità di microcontrollori e microprocessori porta benefici sostanziali a tutto il processo di progettazione e sviluppo

Sensori e Attuatori



Un sistema embedded interagisce con il sistema circostante tramite sensori ed attuatori

Sensori

- Dispositivi *trasduttori* che permettono di misurare un fenomeno fisico (es temperatura)
- Fornisce una rappresentazione misurabile
- Possono essere:
    - *Analogici*: la grandezza elettrica prodotta in uscita varia con continuità
    - *Digitali*: due soli valori o un insieme discreto di valori

Attuatori

- Dispositivi che producono un qualche effetto misurabile sull'ambiente da una specifica condizione o un evento chiamato *trigger*

Bus e protocolli di comunicazione

→ La comunicazione con i sensori e attuatori avviene tramite canali di comunicazione (BUS) con specifici protocolli di comunicazione

→ Protocolli più usati, implementano una **trasmissione seriale → parole multi-bit che vengono inviate sequenzialmente**

UART (universal asynchronous receiver-trasmiffer → ricevitore/trasmettitore asincrono universale)

- Più adatto per protocolli seriali
- Convertire flussi di bit di dati da un formato parallelo a uno seriale asincrono o viceversa
- Non genera o riceve direttamente i dati → ci pensano i dispositivi di interfacciamento (es. USB, bluetooth, ecc..)

USART (universal synchronous/asynchronous receiver-trasmiffer → ricevitore/trasmettitore sincrono/asincrono universale)

- Estende il protocollo con la trasmissione di un segnale di clock per la sincronizzazione

I2C → (inter-integrated circuit) protocollo/bus seriale sincrono, facile da usare e più espandibile

SPI → (serial peripheral interface) protocollo/bus seriale sincrono, più veloce di I2C, ma meno facile da usare

JTAG → protocollo std per il test dei dispositivi, da usare in accoppiata con strumenti di debug

CAN-BUS (controller area network bus)

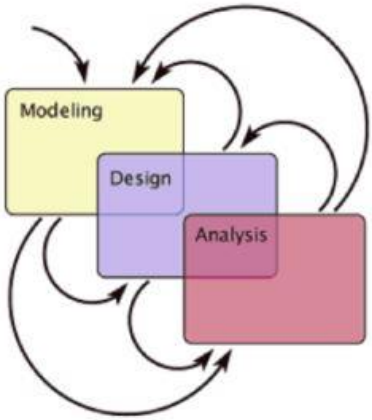
- Protocollo std seriale per bus usati nell'ambito automotiveed in contesti industriali
- Pensato per collegare unità di controllo
- Funzionare in ambienti disturbati dalla presenza di onde elettromagnetiche
- Scambio di msg multi-cast

INTERFACCE E TECNOLOGIE DI COMUNICAZIONE

- Tecnologie e standard che permettono la comunicazione wired e wireless con altri sistemi (embedded e non)
- Wireless in particolare
  - Bluetooth and Bluetooth Low-Energy (BLE)
  - ZigBee
  - Z-Wave
  - Wifi
  - …

Progettazione di sistemi embedded: uno sguardo introduttivo

Processo iterattivo: *modeling*, *design*, *analysis*



Modeling / Modellazione

→ Processo per approfondire e comprendere il sistema da costruire

→ Rappresentazione dei modelli che illustrano il processo

→ Rappresenta cosa deve fare il sistema

Design / Progettazione

→ processo per la creazione degli artefatti tecnologici

→ rappresenta come deve fare quello che deve fare il sistema

- La parte di modellazione è importante, perché da come si è descritto in generale quello che fa il sistema, ne viene anche la parte più specifica

Analysis / Analisi

→ processo per ottenere un'approfondita comprensione del sistema

→specifica perché un sistema fa quello che fa

Modellazione → il modello è la descrizione degli aspetti più rilevanti del sistema

- I sistemi embedded e cps sono composti da sottoinsiemi fisici integrati, di computazione e di rete
    - Modello della parte fisica
        - Modellazione di componenti dinamici **tempo-continui** (es: modelli matematici, teoria del controllo...)
    - Modello della parte logica
        - Modellazione di comportamenti dinamici **tempo-discreti** (es: macchine a stati, modelli concorrenti…)
    - Modelli ibridi → modelli ad attori

Progettazione di un sistema embedded

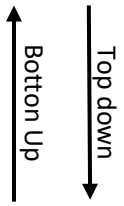
Scegliere la tecnologia e l'architettura HW + Scegliere Architettura SW

Analysis

- Sistema progettato per certi requisiti
- Requisiti di un sistema espressi in termini/proprietà specifiche
- Strumenti → linguaggi formali, tecnica per confrontare specifiche, tecniche per analizzare le specifiche

Top Down/Bottom-Up Design

- Requisiti
- Specifica
- Architettura
- Componenti
- Integrazione di sistema



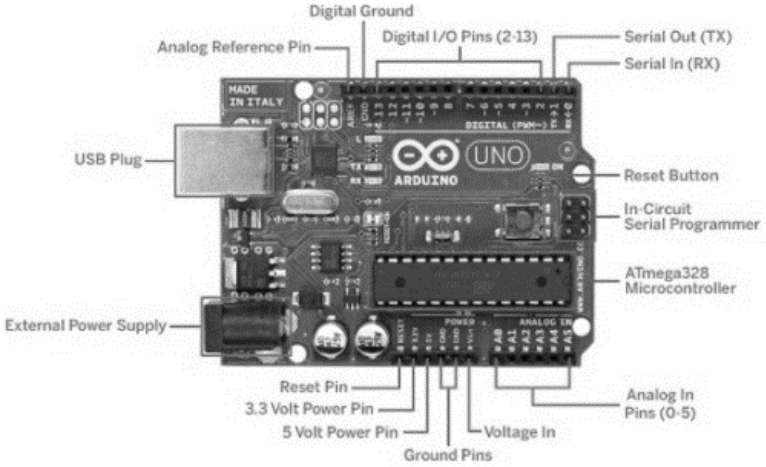
- **Requisiti** → descrizione informale del sistema e cosa si suppone faccia
  - Requisiti funzionali
    - Funzioni che il sistema svolge
  - Requisiti non-funzionali
    - Performance, costo, dimensioni fisiche, peso …
  - Requirement forms →nome, scopo, inputs, outputs, funzioni, performance …
- **Specifiche**
  - Descrizione più precisa dei requisiti (contatto tra cliente e progettista) → linguaggio UML
- **Progettazione Architettura →** come il sistema implementa le funzioni descritte
  - Architettura → struttura complessiva del sistema, usata per guidare lo sviluppo dei componenti, quali componenti servono e come interagiscono
  - Diagrammi a blocchi → HW e SW
- **Progettazione componenti**
  - Sviluppo dei singoli componenti, componenti ready-made
- **Integrazione di sistema**
  - Integrazione dei componenti seguendo l'architettura
- **Sviluppo e progettazione**
  - Programmazione e sviluppo del sistema
  - Dopo la creazione su un computer host, viene mandato in esecuzione sul sistema embedded
  - Linguaggi/piattaforme di alto livello
    - Linguaggio c/c++ e librerie di sistema
    - Diffuso utilizzo di linguaggi/piattaforme di alto livello con architettura run-time
- **Deployment, debugging, testing**
  - Sviluppo SW → avviene su host
  - Supporti per debuging → collegamento seriale per scrittura di msg relativi al stdout sul computer host
- **Importante**
  - Ogni sistema richiede una fase di modellazione una di progettazione
  - Principi dell'ingegneria del software
  - Mantenere il livello di astrazione individuato nella modellazione

Modulo 1.2 SISTEMI BASATI SU MICROCONTROLLORE

Arduino Uno

- Componenti principali
  - MCU ATMega 328P
    - 8 bit, 16 MhZ
    - Flash memory: 32 KB
    - SRAM memory: 2 KB, EEPROM: 1 KB
  - 14 pid digitali input/output
    - 6 possono essere usati come uscite PWM
  - 6 input analogici
  - connettore USB
  - power jack
  - ICSP header
  - pulsante di reset

- Altre specifiche
  - Operating voltage: 5V
  - Input voltage (recom.): 7-12 V
  - DC current per I/O pin: 40 mA
  - DC current per 3.3V: 50mA



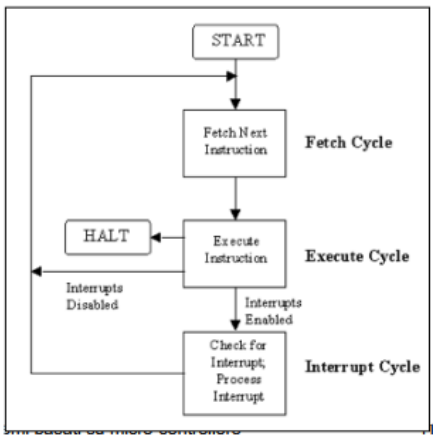
Programmazione di un micro-controllore

- Tipicamente la programmazione Si fa su un pc e poi dopo la compilazione viene creato l'eseguibile in codice binario, che trasferito sul microcontrollore tramite seriale. Nel microcontrollore non c'è nessun sistema operativo→ eseguibile traferito in memoria ed eseguito dal processore direttamente
- Trasferimento tramite USB ad opera di un bootloader, viene creato in ambiente wiring (framework opensource) e Arduino IDE (tool per editing con compilatore avr-gcc)Elementi di una mcu

CPU e Unità di memoria

- Il funzionamento di una CPU è definito a livello di modello dalla **macchina di Von Neumann**
  - ciclo di esecuzione *fetch-decode-execute*

- L'**instruction-execution cycle** di un computer in quanto macchina di Von Neumann consiste nel:
  - caricamento dalla memoria (**fetch**) dell'istruzione da eseguire, depositata in un apposito registro (**instruction register**)
  - **decodifica** dell'istruzione, che può comportare il caricamento di altri operandi dalla memoria
  - **esecuzione** dell'istruzione, che può portare all'aggiornamento dei registri e alla scrittura di dati in memoria



ARCHITETTURA HARVARD

- Variante dell'architettura di Von Neumann in cui *istruzioni e dati sono memorizzate in memorie fisicamente separate*, con bus distinti
  - nome deriva dal Harvard Mark I relay-based computer
    - istruzioni memorizzate in schede perforate e dati in dispositivi elettro-meccanici
  - architettura tipicamente usata in micro-controllori e DSP, con estensioni e varianti
    - es: Atmel AVR
- Aspetti di rilievo:
  - le due memorie utilizzate possono avere caratteristiche molto diverse, ad esempio:
    - **codice in FLASH memory**
      - accesso veloce in lettura, lento in scrittura
    - **dati in SRAM**
      - accesso veloce sia in lettura sia scrittura, consuma di più
- Codice e dati possono essere letti/scritti parallelamente
  - dal momento che usano bus distinti

Confronto fra memorie

Flash

- Memoria non volatile, veloce in lettura, lenta in scrittura, bassi costumi e costi

SRAM

- Volatile, veloce sia in scrittura che in lettura, maggior consumo rispetto a flash

EEPROM

- Memoria volatile, accesso meno performante, lettura/scrittura con libreria arduino

Architettura e funzionamento CPU → esegue le istruzioni codificate in linguaggio macchina

ISA → instruction set architecture (possibili istruzioni/registri riconosciuti dalla cpu) → GPR, SFR

I registri general-purpose del processore, sia quelli relativi all'IO sono mappati in memoria

Tempo di esecuzione dei programmi facilmente individuabile grazie alle informazioni che ci forniscono il numero di giri di clock (per un'istruzione, nella maggior parte dei casi, ci mettono 1 o 2 giri) → importante per il real-time

Da **wiring** all' **eseguibile**

- Programmi scritti in C/C++
- Wiring specifica com'è organizzata l'esecuzione (loop() e setup())
- Avr-gcc compila e linka il codice

Porte di I/O e convertitori analogico/digitali

GPIO → pin che gestiscono direttamente gli input e output

Possono essere analogici (può assumere qualsiasi valore all'interno di un certo range) o digitali (può assumere più valori)

Parametri di funzionamento per i pin → tensione (Volt) e Corrente (Ampere)

Presenza o meno di circuiti resistori di pull-up (internamente i pin possono essere equipaggiati di circuiti pull up, per fissare il valore della tensione VCC)

Porte di Input e Output → le porte sono ciò che permette al microcontrollore di interagire con i dispositivi esterni tramite i pin → SRF (registri special purpose) trasportano i segnali digitali e analogici, esso mantiene lo stato dei vari pin e a ogni porta è associato un SRF

Importare, scrivere, leggere un pin

- pinMode() → setta la direzione I/O di un pin
- digitalWrite() → imposta il valore di un pin in OUT
- digitalRead() → imposta il valore di un pin in IN

funzioni specifiche

- 0, 1 → interfaccia seriale
- 2, 3 → interruzioni
- 3, 5, 6, 9, 10, 11 →PWD
- 10, 11, 12, 13 → comunicazione SPI
- 13 → Builtin led

Pin analogici (A4 e A5 → I2C)

PULSE-WITH-MODULATION (PWM)

- Tecnica di pilotaggio di dispositivi di output che permette di **emulare in uscita un segnale analogico** a partire dalla generazione di segnali digitali detti PWM
- Il segnale analogico di un certo valore V su un pin è "emulato" mediante un segnale periodico che si ottiene modulando il *duty cycle* di un'onda quadra
  - ovvero un segnale che passa ripetutamente da 0 ad 1
  - per *duty cycle* si intende la percentuale di tempo che il segnale è ad 1 rispetto a quella in cui è a 0
- Sul pin si ottiene in tal modo un segnale equivalente analogico il cui valore (tensione) risultante medio dipende dal duty cycle
  - ad esempio, avendo duty cycle pari a 50% otteniamo una tensione media pari alla metà del valore massimo (es. 2.5V)
  - con duty cycle pari a 100% => 5 V, duty cycle 80% => 4 V
- Non funziona per pilotare qualsiasi dispositivo analogico di output
  - funziona bene ad esempio per pilotare LED con intensità variabile, motori in continua (per variare la velocità)

Segnali analogici

- Segnale digitale può assumere solo due valori (1, 0), mentre quelli analogici possono assumerne all'interno di un certo range. Quindi per poter lavorare con un segnale analogico bisogna convertire i valori digitali con cui il sistema lavora. La conversione avviene tramite il componenete ADC (Analog to Digital), che mappa il valore continuo in un valore discreto in un certo range

Lettura di Pin analogici

→Arduino uno può gestire fino a 6 segnali di pin analogici → A0, …, A5

Int analogRead(int PIN) → funzione della libreria per leggere il segnale

Con la funzione **map()** si possono mappare i valori

Logging via seriale

- Se si usa la porta seriale come interfaccia per gli output, allora utilizzando il serial Monitor di Arduino IDE

I/O e interruzioni

Meccanismo delle interruzioni permette al microcontrollore di reagire agli eventi evitando il *polling.*

In generale le CPU mettono a disposizione uno o più pin – chiamati IRQ (Interrupt Request) dove ricevere i segnali → quando riceve una richiesta di interruzione, sospende l'esecuzione della sequenza, salva sullo stack l'indirizzo

dell'istruzione che stava per eseguire, e trasferisce il controllo alla interrupt routine corrispondente (ISR o interrupt headler)

Programmare Interrupt Routine in wiring

Wiring è possibile implementare una ISR mediante la routine: attachInterrupt (intNum, ISR, mode);

È disponibile la funzione digitalPinToInterrupt(numPin) che recupera il numero dell'interruzione associata al pin specificato

Disabilitazione interruzioni

Esiste la possibilità di disabilitare/abilitare le interruzioni → STI = set interrupt (abilita) e

CLI = Clear Interrupt (disabilita) oppure in Arduino interrupts() e noInterrupts()

Quando disabilitiamo le interruzioni, il sistema non è più reattivo ad eventi esterni → questo impone alcuni vincoli sul design degli interrupt handler:

- Devono essere eseguiti in tempi brevi (non possono mai bloccarsi o eseguire loop infiniti)
- Bassa interrupt latency (quantità di tempo che impiega il sistema per reagire ad una interruzione)
    - Interruzione disabilitate per tempi brevi
    - Usare strategie per compiti computazioni onerosi

Note: interrupt su Arduino

- Su AVR/Arduino viene eseguito l'interrupt handler viene eseguito con le interruzioni disabilitate
- Non tutte le primitive funzionano se chiamate all'interno di un interrupt handler
    - Es delay() e millis() non funzionano
    - delayMilliseconds() funziona, perché non si basa su un timer

Timers

- ruolo fondamentale per i comportamenti time-oriented (misurare intervalli di tempo, segnali pwm, ecc..)
- possiamo pensarlo come un contatore che viene incrementato a livello HW a una certa frequenza, da cui si può configurare la frequenza, accedere/leggere un valore corrente del contatore, …

ATMega328

- timer interni (timer 0, timer 2 che sono a 8 bit, mentre il timer 1 è a 16 bit)
- Timer utilizzato per le funzioni di wiring come millis() e analogWrite() per i segnali PWM
- Modalità di gestione del timer come CTC (clear timer on compare match), in essa le interruzioni del timer sono generate quando il contatore raggiunge un certo valore e successivamente resetta il clock successivo e riparte (inserendo il valore nel registro compare match si specifica la frequenza delle interruzioni)

TIMER E INTERRUZIONI SU ATMega328: DETTAGLIO

- I contatori sono incrementati a 16 MHz
  - quindi ogni tick avviene ogni 1/16,000,000 di un secondo (~63ns),
- I timer0 e timer2 sono a 8 bit => impiegano 256*63 = 16128 ns = 16.1 us ad andare in overflow (e ripartire da 0)
- Il timer1 è a 16 bit => impiega 65536*63 = 4128768 ns ~ 4129 us ~ 4.1 ms
- Questa frequenza può essere modulata specificando un valore detto *prescaler*, che sostanzialmente funge da divisore della frequenza originaria con cui viene incrementato il timer (ovvero 16 MHz):

```
timer speed (Hz) = (Arduino clock speed (16MHz)) / prescaler
```

- Quindi specificando un prescaler di 1 => incrementa il contatore a 16MHz, un valore di 8 lo incrementa a 2 MHz, .. un valore di 1024 lo incrementa 16 KHz.
- I valori di prescaler possono essere 1, 8, 64, 256, and 1024.

- Considerando il prescaler, allora la frequenza con cui vengono generate le interruzioni è dato dalla frequenza di incremento diviso il valore specificato nel compare match register:

```
desired interrupt frequency (Hz) =
   16,000,000Hz / (prescaler * (compare match register + 1))
```

  - c'è +1 dal momento che il valore 0 rappresenta il primo valore significativo del registro
- Esprimendo l'equazione rispetto al compare match register:

```
compare match register =
  ( 16,000,000Hz/ (prescaler * desired interrupt freq)) - 1
```

dobbiamo controllare che questo valore sia inferiore a 256 se usiamo i timer 0 e 2, inferiore a 65526 se usiamo il timer1. Se non lo è dobbiamo aumentare il prescaler

- Esempio: supponiamo di volere una interruzione al secondo, ovvero alla frequenza di 1 Hz. Allora:

```
compare match register = (16,000,000 / (prescaler * 1)) -1
```

- Se usiamo un prescaler da 1024 otteniamo:

```
compare match register = (16,000,000 / (1024 * 1)) - 1 = 15,624
```

- Siccome il valore è maggiore di 256 però inferiore a 65536, possiamo usare il timer1.

Watch dog timer →timer che è in grado si eseguire il conteggio fino a un certo valore, dopo il quale genera un segnale output con cui resetta il circuito (se riceve in tempo il valore va bene, se no significa che il microprocessore è entrato in situazione critica e quindi resetta)

Bus e comunicazione seriale

- La complessità dei protocolli di scambio varia a seconda dei dispositivi I/O che si usano
- Le varie interfacce possono essere classificate come **seriali** (stream sequenziale bit per bit usando solamente un filo) **parallele** (trasferire più bit alla volta)
- I bus seriali si possono dividere in **ASINCRONI** (non viene usato nessun clock, ci sono due linee: trasmissione, ricezione. Es: USB e TTL) e **SINCRONI** (per la sincronizzazione delle parti si utilizza un segnale di clock, questo permette trasferimenti più rapidi. Es: I2C e SPI)

Seriale Asincrona

Dal momento che non c'è il clock a sincronizzare la comunicazione deve essere stabilito un protocollo (configurabile → le parti devono usare lo stesso protocollo), per assicurare che sia error-free.

- Baud Rate
  - Esso specifica quanto velocemente i dati sono inviati sulla linea seriale (espresso in bps)
  - Il tipico boud rate della seriale è 9600 bps, più è alto più i dati sono trasferiti velocemente. Il lklimite è 115200, legato alla capacità dei microcontrollori. → dopo possono saltare fuori errori
- Data Frame
  - Ogni blocco di dati è inviato sotto forma di pacchetto di bit. Essi sono creati aggiungendo un bit di sincronizzazione ai dati inviati. In ogni pacchetto si possono trovare dai 5 ai 9 bit, ma il valore tipico è 8.
  - I bit di sincronizzazione includono bit di fine e inizio pacchetto, mentre ci sono i bit di parità per avere una forma low-level.
- Hardware della seriale
  - Ci sono i due fili, uno per riceve (RX) e uno per trasmettere (TX). Il TX di un dispositivo va collegato al RX dell'altro dispositivo, e viceversa.
- UART (Universa asynchronous receiver/transmitter)
  - Componente fondamentale della seriale, compito di convertire i dati verso/da l'interfaccia seriale
- Seriale su Arduino
  - Arduino UNO ha una porta seriale HW TTL, multiplex sui GPIO 0 e 1.

- Pin0 = RX, pin 1 = TX
- C'è un chip che traduce da USB a TTL e viceversa
- Libreria Serial → completo controllo e gestione della porta seriale, utilizzabile per comunicare messaggi via USB
  - Alcuni metodi → begin(), end(), read(), parseFloat(), write(), flush(), print(), println()

Seriale Sincrona

La propria linea la accoppia sempre con un segnale di clock per sincronizzare → trasferimenti più rapidi

- I2C
  - È un bus/protocollo standard
  - Comunicazione veloce e robusta a due vie, usando il minor numero di bit
  - Utilizza due vie per la comunicazione → clock, data
  - Utilizza 7-10 bit di spazio di indirizzamento
  - Architettura Master Slave →contiene uno o più slave che ricevono info dal master
  - Due linee di comunicazione
    - SCL → Serial Clock Line. Serve per sincronizzare la comunicazione
    - SDA → Serial Data Line. Linea bidirezionale per inviare/ricevere messaggi
- Master Slave
  - La comunicazione parte sempre dal master → gli slave rispondono e basta
  - Tutti i comandi inviati dal master sono ricevuti da tutti i dispositivi sul bus
    - Ogni slave ha un suo ID di 7 nit che decide il master → solo la slave target reagisce al messaggio
  - Schema di comunicazione tipico:
    - Master invia start bit
    - Master invia l'id del dispositivo con cui vuole comunicare
    - Invia 0 o 1 (read o write)
    - Slave risponde con un ack
    - Se la comunicazione finisce, il master invia lo stop bit
- SPI
  - Implementa un protocollo seriale full-duplex che consente la comunicazione simultanea bidirezionale tra un master e slave
  - Non segue un vero e proprio standard formale → ci possono essere differenze implementative
  - SPI è Sincrono e usa un clock condiviso
  - È basato su master slave → controllato da master device
  - 3 linee di comunicazione:
    - **SCK**: Shared Serial Clock. Uno shared clock signal
    - **MOSI**: Master Out Slave In → inviare da master a slave
    - **MISO**: Master In Slave Out → inviare da slave a master
    - SS: Slave Select → selezionare lo slave
  - Specificando un buffer, dato che è full duplex, si possono inviare più byte
- CONFRONTO I2C E SPI
  - Ci sono molti dispositivi che sono disponibili sia in I2C, sia in SPI
  - I2C → richiede solo due linee, modalità indirizzamento slave più agile
  - SPI → velocità più elevate, semplice da usare, non richiede pull-up

Modulo 1.3 SENSORI E ATTUATORI

Panoramica Sensori e Attuatori

Sistema embedded interagisce con l'esterno tramite Sensori e attuatori

Sensori

- Dispositivi traduttori che permettono di misurare un certo fenomeno fisico e di fornirne una rappresentazione misurabile
- Possono essere analogici (se la grandezza in uscita varia con continuità) o digitali (insieme discreto di valori)

Attuatori

- Dispositivi che producono un qualche effetto misurabile sull'ambiente (analogici o digitali)

Misura

- Grandezza fisica = confronto di due quantità appartenenti a grandezza della stessa specie
- SI: sistema internazionale di unità di misura più diffuso

Incertezza della misura

- Quando le misure ripetute di un certo parametro non forniscono lo stesso valore
- Questo concetto esprime la dispersione dei valori (potenziale e reale)
- Il risultato di misura è costituito sempre da un numero e da un'incertezza (simbolo ±)

Errore

- Errore=valore misura-valore corretto
- Errori accidentali o errori sistematici
- Risultato della misura= fascia di valori tutti ugualmente validi a rappresentare il parametro (misurandolo)

Caratteristiche di un sensore

- Statiche
  - **Accuratezza**
  - **Precisione**
  - Linearità
  - Sensibilità
  - Risoluzione
  - Ripetibilità
  - Riproducibilità
  - Stabilità
  - …
- Dinamiche
  - Tempo di risposta (Risposta in frequenza)

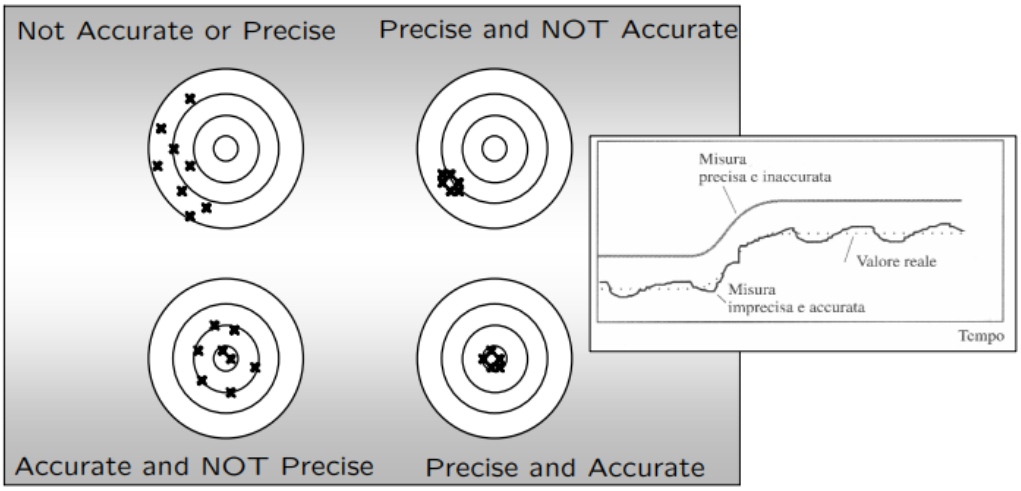
Accuratezza

- È la misura d quanto il valore letto dal sensore si discosti dal valore corretto
- Calcolo: È il massimo scostamento che c'è tra la misura fornita dal sensore e il valore vero del segnale misurato

Precisione

- Descrive quanto un sensore sia soggetto o meno ad errori accidentali (random errors)
- Legata alla ripetibilità
- Calcolo: deviazione dei valori letti rispetto al valor medio

Accuratezza vs precisione



Calibrazione

- aggiustamento dei parametri del sensore per far corrispondere l'uscita a valori rilevati accuratamente con un altro strumento

Taratura

- misurazione della grandezza di uscita per valori noti della grandezza di ingresso al trasduttore stesso

Parametri statici

- **Sensibilità**
  - rapporto fra variazione della grandezza (segnale) di uscita e corrispondente variazione (segnale) d'ingresso
- **Risoluzione**
  - capacità dello strumento (sensore) di risolvere stati (livelli) diversi del misurando
- **Ripetibilità**
  - attitudine dello strumento a fornire per uno stesso misurando valori di lettura vicini tra loro, in letture consecutive eseguite in breve intervallo di tempo
- **Riproducibilità**
  - vicinanza di risultati ottenuti sullo stesso misurando in diverse specifiche condizioni di misura
- **Stabilità**
  - attitudine di uno strumento a fornire valori di lettura poco differenti tra loro in letture eseguite indipendentemente sullo stesso misurando in un intervallo di tempo definito

Linearità

- Molti sensori possono essere modellati (con opportune approssimazioni) da *funzioni affini*
  - sia
    - x(t) la funzione che rappresenta la quantità fisica da misurare
    - f: R → R la funzione che rappresenta il valore misurato dal sensore, per cui f(x(t)) è il valore misurato al tempo t.
  - allora f è ***lineare*** se esiste una costante di proporzionalità a per cui, per ogni x(t) in R:

    f(x(t)) = a*x(t)

- E' una funzione ***affine*** se esistono due costanti a e b (detto *bias*) per cui:

  f(x(t)) = a*x(t) + b

- La costante di proporzionalità rappresenta la **sensitività** *(sensitivity)* del sensore
- Anche gli attuatori possono essere modellati da funzioni affini

Range

- ogni sensore realizza sempre un'approssimazione
- viene definito un intervallo di funzionamento come insieme dei valori che il sensore può misurare (i valori che escono si dice che saturano restituendo sempre o il massimo o il minimo)

Comportamento dinamico

- ogni sensore ha la propria dinamica
  - risposta non istantanea
  - la risposta è spesso trascurabile rispetto alle dinamiche del processo

- Quando in ingresso al trasduttore applichiamo una sollecitazione *a gradino* (cioè un gradino della grandezza da misurare) l'uscita (risposta) varierà fino a raggiungere, dopo un certo tempo, un nuovo valore.
- Si definisce:
  - **tempo di salita**
    - tempo impiegato per passare dal 10% al 90% del valore finale
  - **tempo di risposta**
    - tempo impiegato per raggiungere una percentuale prefissata del valore finale.

Alcune tipologie di sensori

Prossimità

- sensori in grado di rilevare la presenza di oggetti nelle immediate vicinanze (portata nominale)
- affidabilità elevata data l'assenza di meccanismi meccanici per attivarlo
- segnale d'uscita: rilevano solo la presenza o l'assenza di un oggetto, quindi di tipo on/off

La routine pulseIn

- routine efficiente della libreria wiring
- monitora il pin di input dove deve ricevere l'impulso
- tempi piccoli → ordine delle decine di microsecondi

Prossimità ottici

- si basano sulla riflessione di un fascio luminoso da parte di un oggetto rilevato
- normalmente fascio di raggio infrarossi, perché difficilmente si confonde con i disturbi generali da fonti luminose ambientali
- la quantità di luce dipende da composizione ed orientamento della superfice dell'oggetto → quindi si possono creare problemi per oggetti fuori dai 10 ai 100 cm

Sensori di movimento

- PIR (sensore infrarosso passivo)
    - Misura raggi infrarossi irradiati dagl'oggetti nel suo campo di vista
    - Non rileva autonomamente un movimento, ma rileva le brusche variazioni di temperatura. Infatti, c'è una fase di stallo in cui il pir rileva la normale temperatura così da capire quando cambia e quindi quando rileva un oggetto

Trasduttori di posizione angolare

Due tipi:

- Rileva la posizione senza avere un vincolo meccanico → inclinometro
- Rileva la posizione sfruttando un movimento meccanico → encoder

Sensori Accelerazione

Accelerometro → sensore che misura l'accelerazione a cui è soggetto l'oggetto a cui è connesso il sensore

Giroscopio → dispositivo fisico rotante, mantiene il suo asse di rotazione orientato in una direzione fissa

Tecnologia MEMS → accelerometri e giroscopi elettronici si basano su questa tecnologia

- Micro Electro-Mechanical System
- Insieme di dispositivi di varia natura integrati in forma miniaturizzata su uno stesso substrato di materiale semiconduttore (silicio)
- Coniugano le proprietà elettriche degli integrati con proprietà opto-dinamiche
- Fondamentali per lo sviluppo di sistemi embedded moderni

Sensori di contatto

- Pulsanti tattili
- Potenziometri
    - Cambiare in modo meccanico la resistenza del componente
- Potenziometro lineare
    - Tensione di uscita legata linearmente alla tensione di ingresso

Sensori di pressione

- Permettono di misurare la forza esercitata su una superfice

Sensori ottici

- Dispositivo che converte un'immagine in un segnale elettrico (usati da fotocamere digitali, telecamere ecc.)
- Funzionamento
    - Immagine viene focalizzata su una griglia composta da sensori puntiformi, che convertono la luminosità rilevata. Sensore in bianco e nero più semplice.
- Tipi:
    - In base a caratteristiche come rilevamento dei colori, sensibilità, ecc.
    - Secondo una classificazione base, come riprese a colori o in bianco e nero

Foto-rilevatori

- Dispositivo in grado di rilevare la radiazione elettromagnetica, fornendo in uscita un segnale.
- Tipi:

    - Differiscono per la porzione di spettro elettromagnetico che sono in grado di rilevare →fotocellula
- Tecnologia:
    - Fotoresistenza → basato su cariche fotogenerate
    - Fotodiodo → basato su cariche fotorenerate in una giunzione p-n
    - CCD → charge coupled device. Circuiti basati su cariche fotogenerate

Elettricità e magnetismo

- Rilevatori di tensione d corrente
    - Misurare la tensione fra due punti di un circuito
    - Applicazione che devono misurare il consumo di un certo apparecchio
- Magnetometro
    - Misura il campo magnetico
    - Può essere scalare o vettoriale
- Bussola (compass)
    - Strumento che indica il nord magnetico

Sensori di posizione

- Posizionamento globale, sistema di posizionamento e navigazione satellitare civile che attraverso una rete artificiale fornire a un terminale informazioni sull cordinate geografiche ed orario.
- Grado di accuratezza nell'ordine dei metri
- Funzionamento:
    - Metodo di posizionamento sferico
    - Che parte dalla misura del tempo impiegato da un segnale radio a percorrere la distanza satellite/ricevitore
    - Localizzazione avviene tramite la trasmissione di un segnale radio da parte di ciascun satellite

Sensori per identificazione

- RFID
    - Tecnologia che permette di avere piccoli tag contenti dati e di essere rilevate, letti e scritti wireless
- NFC
    - Analoghi a RFID, avviene a contatto tra lettore e tag
- iBeacon
    - tecnologia BLE (Bluetooth low energy) per trasmettere un certo UUID nel raggio di una certa località e rilevabile da lettori Bluetooth 4.0

Attuatori e dispositivi di output

Trasduttori di output

- i trasduttori convertono una forma di energia in un'altra
- I trasduttori di output permettono di realizzare sistemi embedded che eseguono e controllano azioni sull'ambiente fisico in cui sono emessi

Interfacciamento

Due casi:

- È sufficiente la corrente/tensione ai GPIO (ad esempio led)
- In caso contrario, il dispositivo deve essere alimentato da un circuito separato. Che, ad esempio in questo caso, mediante una GPIO si apre/chiude il circuito e con relè/transistor si fanno gli interruttori

Carichi resistivi e induttivi

- Dispositivi pilotabili possono essere:
    - Resistivi
        - Possono essere assimilati ad un componente che al passaggio della corrente oppone una resistenza

    - Induttivi
        - Operano inducendo corrente su un filo mediante la corrente su un altro filo (es: motori, solenoidi)
        - Generano una corrente/tensione inversa

Led (light emitting diode)

- Diodo ad emissione luminosa
- Sfrutta proprietà ottiche di alcuni materiali semiconduttori
- Elevata efficienza, lunga durata, affidabilità, basso consumo

Lcd

- Display basato sulle proprietà ottiche dei cristalli liquidi
- Campo elettrico applicato al liquido che è contenuto tra due superfici vetrose
- Ogni contatto manda una porzione del pannello (PIXEL)
- Basso consumo di potenza, quindi particolarmente indicati per l'embedded

Motori elettrici

Macchine elettriche di cui la potenza in entrata è di tipo elettrico e quella in uscita è di tipo meccanico (funzione di attutaore). E' composto da: statore e rotore (componenti che generano campo magnetico e che determinano il movimento del rotore)

- motori in corrente continua (CC)
    - indicati per progetti che hanno bisogno di rotazione continua a 360°, elevata velocità e controllabile
    - es: route di robot, slider per telecamera, ecc.
    - ci sono due tipi:
        - senza spazzole (passo-passo)
        - con spazzole → meno costosi e durano di meno
            - crea campo magnetico e tramite le spazzole inverte la polarità creando il movimento angolare
    - mediante il medesimo principio si può usare il motore per generare energia
    - controllo sulla velocità
    - invertendo la tensione si può invertire il senso di rotazione (ponte H)
- motori passo-passo
    - detto anche stepper, è un motore a corrente continua senza spazzole
    - la posizione del motore si può controllare accuratamente
    - usato nei progetti in cui devi controllare la posizione dell'albero accuratamente
    - esempio: stampanti 3D
- servo-motori
    - permette di pilotare l'angolo del rotore (come nel passo-passo)
    - la differenza è che la specifica posizione è assoluta, non relativa alla posizione corrente
    - escursione limitata a 180° (-90° a 90°)
    - 3 fili: GND, +5V, segnale controllo digitale
    - Controllo avviene inviando uno stream di impulsi al segnale di controllo ad una specifica frequenza → la lunghezza dell'impulso determina l'angolo (più lungo, più grande)

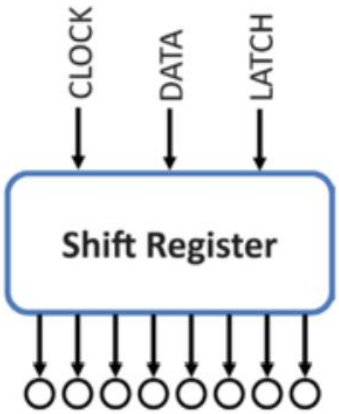
uso di transistor come switch

- Transistor si usano per tanti scopi (amplificatori, porte logiche, interruttori, ecc.)
- Modulando il pin base con un segnale pwm si può controllare la velocità del motore accendendo e spegnendo il transistor
- Duty circle del segnale pwm determina la velocità del motore (100% = velocità massima, 0%=velocità nulla)

Quando i Pin non bastano

È un problema ricorrente in progetti in cui il numero di pin è insufficiente per interfacciare tutti i dispositivi, o si utilizzano Arduini con più PIN, tipo Arduino Mega oppure dei **registri a scorrimento** (shift register)

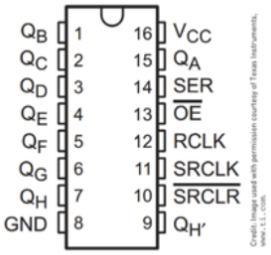
Shift register → dispositivi utili per convertire input sequenziale/parallelo in output sequenziale/parallelo



Registri SIPO (Serial in parallel Output) → la conversione (Seriale → parallelo) viene fatta tramite questi registri

- Dispositivi che accettano in ingresso uno stream sequenziale di bit
- Numero di bit in uscita può variare
- Possono essere concatenati facilmente in un circuito
- Segnali di Input
  - CLOCK → segnale che sincronizza l'acquisizione dei dati
  - DATA → bit dove insiste lo stream di bit in input
  - LATCH → segnale che campiona l'output

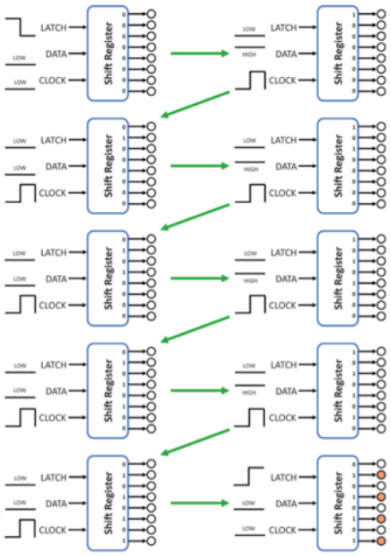
- Esempio concreto: shift register 74HC595



- Pin out:
  - SER è la linea dati,
  - SRCLK è il segnale CLOCK
  - QA - QH - output parallelo rappresentato da 8 bit/pin
  - RCLK è il segnale LATCH
- Gli altri:
  - OE (output enable) segnato, attivo basso - attiva o disattiva l'ou
  - SRCLK segnato - serial clear pin: quando attivo (basso), resetta il contenuto del registro.

FUNZIONAMENTO

- Device sincrono, che acquisisce il valore dei bit in input al fronte di salita del segnale del clock
  - ad ogni fronte, tutti i valori al momenti presenti nel registro vengono fatti scorrere a sinistra di una posizione e viene introdotto il nuovo bit
  - il pin LATCH è attivato alla fine dell'acquisizione
  - esempio
    - caricamento valore 10101010



Collegamenti in Daisy-Chain

- Per pilotare un numero maggiore di uscite usando più shift register è possibile mediante *collegamenti daisy-chain*
- Il collegamento in daisy chain si ottiene:
  - collegando il pin Q_H' al pin DATA di un altro shift register
    - il pin Q_H' contiene il valore del bit più significativo che ad ogni shift "esce", facendo posto a quello meno significativo
  - facendo in modo che tutti gli shift register coinvolti condividano i segnali LATCH e CLOCK
- Nel caso di un numero elevato di led è necessario disporre di alimentazione supplementare, oltre a quella fornita da Arduino

Interfacciamento dei dispositivi

Problemi dei livelli di tensione

I sensori/attuatori possono funzionare con livelli di tensione diversi → 5V Arduino e 3.3V Rasp

Nel caso si debbano utilizzare livelli di tensione inferiori rispetto a quelli forniti dalla scheda dei sistemi, bisogna stare attenti e prendere delle precauzioni se non si vogliono danneggiare i componenti. Per esempio nel collegamento seriale diretto Arduino-Raspberry:

- TX (Rasp) → RX (Arduino) Nessun problema. Perché il 3.3 è sufficiente per indicare il valore logico alto di arduino
- TX (Arduino) → RX (Rasp) Problema. Perché i 5V su Rasp possono provocare danni

→Partitore di Tensione, serve per ridurre in uscita i volti per ottenere quella che si vuole. Da 5V a 3.3V

Modulo 2.1 SISTEMI EMBEDDED E MODELLAZIONE OO

Progettazione e sviluppo di sistemi software embedded

- Approccio top-down (quello che vediamo ora)
  - Partiamo dal dominio e dalla modellazione con paradigmi di alto livello (es: OO)
- Approccio bottom-up
  - Partiamo dalle caratteristiche di livello hardware e dal comportamento che il sistema deve avere da quel livello

Modellazione e programmazione

- I paradigmi di programmazione sono paradigmi di modellazione
  - Come concettualizzo un sistema
  - Come analizzo e formulo la soluzione di un problema
- Modello
  - Aspetti salienti del sistema
  - Essi concernono la struttura, il comportamento e le azioni

Importanza

- Analizzare il problema e definire soluzioni più astratte
- Vantaggi
  - Miglior comprensione
  - Riusabilità
  - Portabilità
  - Estendibilità
  - …

Dimensioni fondamentali

- Struttura: come sono organizzate le parti
- Comportamento: comportamento delle parti
- Interazione: interazione delle parti

Paradigmi e linguaggi

- Paradigmi di modellazione:
  - Insieme coerente di concetti e i principi
- Linguaggi di modellazione:
  - Modo rigoroso non ambiguo per rappresentare i modelli → UML

Paradigma OO → modellazione e progettazione software (livello di astrazione fondamentale per catturare i concetti e aspetti)

- Proprietà importanti: incapsulamento, modularità, …
- Aspetti non direttamente catturati
  - Concorrenza, iterazioni asincrone, distribuzione …

Modellazione software su microcontrollore

- Controllore
  - Incapsula la logica per lo svolgimento del compito
  - Usa/osserva/gestisce risorse come attuatori/sensori
  - Modello attivo
  - Modello a loop di controllo (super loop → semplice, architetture a loop articolate)
- Elementi controllanti
  - Modellano le risorse gestite e utilizzate dal controllore
  - Incapsulano le funzionalità utili
  - Modello passivo
  - Interfaccia ben definita, stato/eventi osservabili, modello OO

ButtonLed

Modello a Loop e macchine a stati finiti

Modello a loop→ ogni ciclo il controller legge gli input di cui ha bisogno e sceglie le azioni da compiere

- La scelta stato in cui si trova
- Azione può cambiare lo stato interno

- **Efficienza**
  - negli esempi visti, nonostante a livello logico il controllore sia in idle in attesa di percepire input, con questo approccio continua ad eseguire cicli, anche quando l'input non è cambiato
- **Reattività**
  - la reattività del controllore dipende da quanto rapidamente viene completato un ciclo
  - se durante l'esecuzione del ciclo avvengono ripetute variazioni di stato nei sensori, tali variazioni non vengono rilevate..

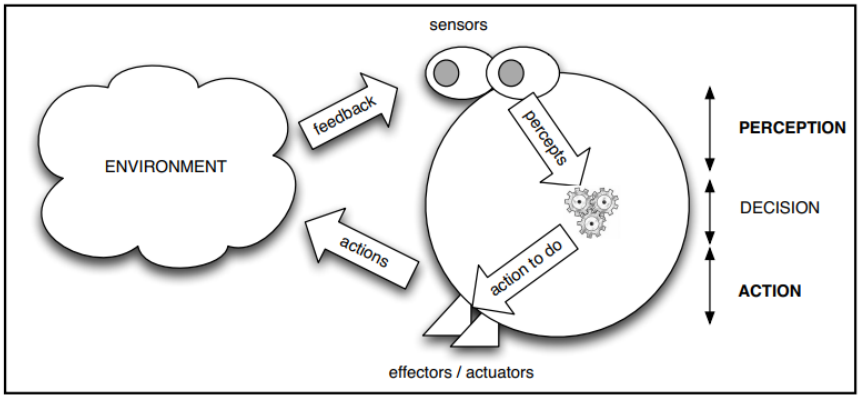
Controller come entità attiva

- Esso è dotato di un flusso di controllo autonomo
- Led/Luci/pulsanti sono considerati passivi. In quanto cambiano stato (es acceso o spento) e hanno specifiche interfacce che permettono al controller di integrarvi o er eseguire azioni o per leggerne lo stato

Modello ad Agenti

- Entità diversa dal punto di vista concettuale degli oggetti (è attivo, no interfaccia, ecc.)
- Agente: entità attiva, progettata per svolgere più task che richiedono di elaborare informazioni che provengono in input da sensori ed agre su attuatori in output (*task-oriented*)

- Concetto introdotto e utilizzato in vari contesti
  - Intelligenza Artificiale e DAI
  - modellazione e simulazione di sistemi complessi
  - software engineering (Agent-Oriented Software Engineering)
  - comunità di ricerca su Agenti e Sistemi Multi-Agente [WOO,JEN]



Modello a loop: modularizzazione a task

- Necessità di introdurre principi di tecniche di decomposizione e modularizzazione, quando i controllori e i loop di controllo sono complessi e articolati
- Decomposizione per task
  - approccio per agenti (esecutori di uno o più task)
  - integrazione di macchine a stati (task descrivibile con una macchina a stati finiti)

Sistemi di sistemi

- divisione di sistemi complessi in sottosistemi che interagiscono e cooperano
  - quindi interazione distribuita, come i sistemi multi-agente → paradigmi di modellazione

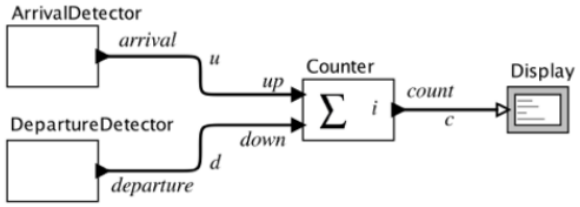
Modulo 2.2 MODELLI BASATI SU MACCHINE A STATI FINITI

Modelli per sistemi embedded

- le macchine a stati finiti sono il modello nel discreto più utilizzato per i sistemi embedded (essa stessa è un sistema discreto → cioè lavora con una sequenza di passi) ed è caratterizzata da sequenze di eventi

ESEMPIO: PARKING GARAGE

- Si consideri un sistema che deve effettuare il conteggio delle auto in un parcheggio, per tener traccia del numero di auto che ci sono in un qualsiasi istante (esempio tratto dal capitolo 3 di [IES], p. 42)
- Si consideri un modello di sistema costituito dai componenti:
  - *ArrivalDetector*, sottosistema che rileva l'arrivo di una nuova auto
  - *DepartureDetector*, sottosistema che rileva la dipartita di un auto
  - *Counter*, sottosistema che tiene traccia del conteggio delle auto



- In questo sistema ogni arrivo o dipartita di un auto sono modellati in modo naturale come eventi discreti

Counter → sistema che reagisce in sequenza agli eventi di input che si presentano e produce un output

(Quando un evento è presente alla porta up/down, il Counter incr/decr il conteggio e produce il valore in uscita.)

Dinamica: reazioni → Essa Ogni passo si una sequenza di step (reazione). Ognuno di essi si presuppone che sia istantaneo (Cioè durata nulla). Queste reazioni sono scatenate (triggered) dall'ambiente in cui opera il sistema.

Notazione di stato

→lo stato di un sistema è la condizione in cui un sistema si trova in un certo istante e rappresenta tutto ciò che è successo nel passato che ha effetto nel determinare la reazione per il futuro

Macchine a stati finiti **FSM → è un modello di sistema a dinamica discreta, in cui ogni input viene mappato in un output. È una macchina a stati in cui il numero di stati è finito. (può essere rappresentato tramite diagramma)**

Transizioni → ciò che rappresenta l'evoluzione dinamica discreta e quindi il comportamento della macchina, essa coinvolge sempre due stati e può essere rappresentata da **guardie** (funzione bool sulle variabili di input e specifica le condizioni per cui la transazione può essere eseguita) e **azione** (valore che devono assumerei le variabili di output come risultato).

NOTA: una FSM non specifica di per sé quando una reazione può avvenire (ovvero quando deve essere operata a far partire le transizioni). Ha due possibilità:

Event-triggered → la reazione avviene a fronte di un evento di input, ed è l'ambiente in cui è la macchina che stabilisce quando effettuare la transazione. Prendono il nome di **FSM asincrone.**

Time-triggered → le reazioni avvengono a intervalli temporali regolari, quindi è definito un periodo è una frequenza. Prendono il nome di **FSM sincrone**.

Modellazione I/O

- I/O è modello in termini di variabili a cui la macchina può accedere.
  - Variabili di input → modificate dall'ambiente in cui opera la macchina
  - Variabili di output → modificate dalla macchina stessa mediante azioni
  - Le variabili possono essere globali e quindi accessibili da tutte le azioni e condizioni

Disciplina

- Corretto funzionamento: ogni azione deve sempre terminare e non ci devono essere loop infiniti e la valutazione di una condizione non deve cambiare lo stato delle variabili

Macchine deterministiche

- Una macchina si dice:
  - Deterministica
    - Per ogni stato c'è al più una trasmissione abilitata per ogni configurazione di input
  - Ricettiva
    - Per ogni stato c'è una trasmissione abilitata per ogni configurazione di input
  - Anche se normalmente si definisce reattiva

Extended FSM

- Quando ci sono molti stati da rappresentare e quindi la macchina è molto articolata.
  - Quindi si considera una estensione utilizzano le variabili come parte dello stato

Macchine a stati finiti sincrone

- Molti comportamenti sono time-oriented nei sistemi embedded
  - Ordinamento temporale delle azioni e intervalli temporali (es bliking ogni tot)
  - Le PES (macchine a stati sincrone) sono macchine a stati finiti estese per il trascorrere del tempo → usano un certo periodo e utilizza un clock interno

Mapping Macchine Sincrone

- Si utilizza l'interruzione di timer programmabili per i linguaggi procedurali come il C

Campionamento (lettura periodica di un sensore ad una data frequenza)

- Caratterizzato da un certo periodo di campionamento → sufficientemente piccolo da evitare la perdita di eventi
- Per frequenza si intende il valore inverso nel periodo
- Maggiore è la frequenza (minore il periodo), maggiore è la reattività e maggiore è il consumo di energia
  - Quindi in generale il periodo deve essere più grande possibile

MEST (minimum event separation time) → intervallo più piccolo che può esserci fra due eventi di input

- TEOREMA: in una macchina a stati sincrona, scegliendo un periodo inferiore al mest si garantisce che tutti gli eventi saranno rilevati
  - Il MEST di eventi di input come pulsanti non include solo gli eventi in sé di pressione del pulsante, ma l'intervallo di tempo di tali eventi

Latenza → intervallo che scorre tra l'evento input e la generazione dell'output

- Obiettivo è **minimizzare** le latenze

Condizionamento dell'Input

- Sensori possono presentare imperfezioni che richiedono aggiustamenti (condizionamento del'input) del valore letto dal microcontrollore

Tecniche di filtraggio

- Sono utili per ignorare la presenza di input impuri
- Es: riduzione del periodo di campionamento (non elimina la possibilità, ma la riduce)

Modulo 2.3 ORGANIZZAZIONE A TASK CONCORRENTI

Organizzazione a task concorrenti

- Per approccio utilizzato nella programmazione concorrente è LA DECOMPOSIZIONE dei vari compiti del sistema in più task. Ogni compito è ben definito e può essere descritto con una macchina a stati finiti. Di cui l'insieme è il comportamento complessivo

Decomposizione in task: vantaggi

- Modularità
- Chiarezza
- Manutenibilità
- Estendibilità
- riusabilità

Decomposizione in task: svantaggi

- I task sono concorrenti (quindi la loro esecuzione si sovrappone nel tempo)
- possono avere dipendenze tra loro e bisogna coordinarli
    - **cooperative** → comunicazione, sincronizzazione
    - **competitive** → mutua esclusione, zone critiche

Gestione di periodi diversi

- → È necessario
    - Tenere traccia per ogni task, nel periodo specifico
    - Far funzionare la macchina con il periodo pari al massimo comune divisore
    - Tener traccia di ogni task del tempo trascorso

Multi-tasking cooperativo → scheduling round robin

- → Vantaggi
    - Non c'è interleaving (disporre in maniera non continua i dati, e non in maniera ordinata. Per evitare errori ed aumentare le prestazioni) tra le istruzioni di task diversi → tick() eseguita atomicamente
    - Non ci possono essere corse critiche
- → Svantaggi
    - Comportamento errato di altri task può compromettere quello di altri (es: loop infinito di tick())

Creazione di un semplice scheduler cooperativo

- → Definizione di un vero e proprio scheduler, in cui si tiene traccia mediante un'opportuna struttura dati (come una coda) la lista dei task da eseguire

Dipendenze tra task

- → Ovviamente non è sempre possibile suddividere i task in sotto task totalmente indipendenti.
- → Le varie dipendenze possono essere:
    - Temporale (un task non può essere eseguito finché un altro task non ha finito)
    - Produttore/consumatore (un task ha bisogno di un'informazione che sta in un altro task)
    - Relative ai dati (due task devono condividere dati per lettura e scrittura)
- → Mediante variabili condivise è il modello più semplice per rappresentare le dipendenze

Vantaggi

- → Migliore separation of concerns (task focalizzato su un compito ben preciso)
- → Migliore comprensibilità (la suddivisione rende il tutto meno complesso)
- → Migliore supporto al debugging (localizzazione rapida di errori)
- → Migliore supporto per modificabilità, estensione, riuso

Variabili condivise, atomicità azioni e corse critiche

- → PROBLEMA: accesso concorrente
    - non danno problemi se non è in lettura, ma se è in scrittura (corse critiche)

➔ SOLUZIONE: presupporre che le transizioni di ogni macchina a stati siano eseguite **atomicamente. (non deve esserci l'interleaving delle azioni di cui si possono comporre transizioni)**

Comunicazione tra task

➔ i task comunicano comunemente mediante l'uso di variabili globali condivise, nelle macchine stati sincrone
➔ alternativa. Scambio di messaggi asincrono
  - ogni task ha la sua coda di msg
  - approccio critico in termini di risorse di memoria

Interrupt-driven cooperative scheduler

➔ lo scheduler può essere invocato direttamente dall'interrupt handler del timer → verisione reattiva del sistema
➔ sfruttare facilmente le modalità di funzionamento a **basso consumo** del microcontrollore
  - sleep nel loop del microcontrollore
➔ consumo di energia nettamente inferiore, ma potrebbe inserire latenze che non sono compatibili con il periodo e i task che devono essere eseguiti

Parametro di utilizzo cpu e scheduling

➔ abbiamo fatto l'assunzione che le azioni eseguite avessero un tempo di esecuzione nullo o comunque inferiore
➔ questa assunzione non può essere plausibile per i sistemi che hanno molte azioni negli stati e nelle transazioni

Overrun del periodo col timer

➔ quando il tempo di esecuzione oltrepassa il periodo, è eccezione di overrun

In caso di overrun → quando il parametro di utilizzo è > 100%, allora si verificherà un overrun.

Possibilità:

➔ macchine a stati con periodi più lunghi
➔ sequenza di azioni più lunga
➔ spezzare sequenze lunghe fra più stati
➔ microcontrollore più veloce

WCET (worst-case-execution-time) → tempo di esecuzione nel caso peggiore in termini di numero di istruzioni eseguite ad ogni periodo

Jitter → ritardo che intercorre dal momento che un task è pronto per essere eseguito e il momento in cui viene effettivamente eseguito

➔ di solito, dare la priorità al task con periodo più piccolo, minimizza il jitter medio

Deadline → scadenza. È l'intervallo di tempo entro il quale un task deve essere eseguito dopo esser diventato ready

➔ se un task non viene eseguito entro la deadline, si ha un eccezione che può portare a malfunzionamenti. Se non viene specificata la deadline, allora di default è data dal periodo.

Scheduling a priorità statica e dinamica

➔ priorità = ordinamento con cui eseguire i task
  - quando più di un task è ready e quindi lo scheduler sceglie chi è il più prioritario
➔ possono essere:

Scheduling a priorità statica

➔ La priorità viene assegnata a ogni task prima che siano eseguiti e queste non cambiano durante l'esecuzione.
➔ APPROCCIO: Più la deadline è piccola, più la priorità è alta
➔ Ma se tutte le deadline sono uguali al proprio periodo, allora si assegna la priorità maggiore a quello con il periodo più piccolo
➔ Questo scheduling si chiamano RMs (rate monotonic scheduling) → cioè con le priorità basate sul rate del task

Scheduling a priorità dinamica

➔ La priorità del task man mano che il programma viene eseguito, per cui possono cambiare nel tempo
➔ APPROCCIO: EDF (earliest-deadline-first). Quindi quelli che hanno la deadline più vicina hanno la priorità maggiore

➔ Questo scheduler riduce jitter e missed deadline → ma è più complesso e costoso
➔ Più usato quando i task sono event-oriented e non time-oriented

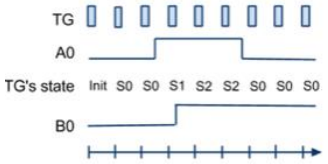
Preemptive e cooperativi

Uno scheduler può essere:

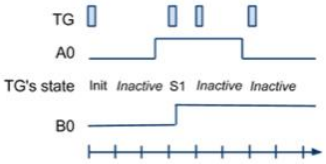
- Preeemptive:
  - è possibile togliere il processore ad un task in esecuzione prima che abbia completato → sono più difficili da implementare
  - più usato quando i task sono organizzati in termini di processi
- coperativo
  - questo viene eseguito fino alla fine del task → semantica run-to-completion
  - organizzazione a stati (es macchine a stati finiti sincrone) → ad ogni tick() la macchina esegue le azioni associate

Macchine a stati sincrone event-triggered

- nelle time-oriented, macchine a stati sincrone per rilevare cambiamenti sull'input eseguono campionamenti periodici (polling)
- ma nelle event-triggered, viene ridotto l'utilizzo della cpu, perché può essere sprecato in attese attive, e utilizza la rilevazione dei cambiamenti su pin di input e può generare interruzioni e chiama un interrupt. E il micro viene fatto transitare in uno stato di inattività ("inactive") finchè non deve transitare in uno stato attivo, quando viene ricevuto un evento da un pin di ingresso.



(synchSM con polling)



(triggered synchSM)

Macchine a stati event-driven

- modello alternativo per integrare modello a stati ed interruzioni/eventi
- si assume il modello asincrono, dove le trasizioni di stato sono determinate da guardie eventi/condizioni

Modulo 2.4 ARCHITETTURE AD EVENTI

Dalle interruzioni ad architetture di alto livello

- a basso livello le interruzioni di input rappresentano eventi che richiedono l'attenzione del processore
    - permettono di gestire i sensori evitando il polling
    - evitano che sia il programma/processore a interrogare continuamento il sensore per sapere il valore corrente e in particolare i cambiamenti
- sfruttare le interruzioni per realizzare architetture ad eventi di alto livello
    - pattern observer
        - elementi:
            - sorgente: mette a disposizione l'interfaccia per registrare gli osservatori
            - eventi generati: l'azione (es: pulsante premuto)
            - osservatori: mettono a disposizione un'interfaccia per la notifica dell'evento
    - reazioni
        - approccio alternativo che permetta di migliorare l'incapsulamento del comportamento reattivo
        - l'architettura ad eventi organizzata in eventi (rappresentati da tipo, sorgente e contenuto) e reazioni (agente controllore esegue a fronte dell'occorrenza di eventi con un certo tipo da una certa sorgente)
        - ad ogni occorrenza il gestore della reazione in prenderà controllo e verificherà quali reazioni sono applicabili e possono essere eseguite
    - event loop
        - ogni controllore ha una coda dove vengono aggiunti eventi di interesse
        - ciclo di controllo ha un loop infinito dove si attende che ci sia un evento in coda e elaborare la coda
        - ASPETTI ESSENZIALI: si disaccoppia il flusso di controllo che genera gli eventi e quello che esegue gli hadler

Note relative ad Arduino uno

- Limiti per ciò che concerne le interruzioni → solo 2 pin possono generare le interruzioni, quindi possiamo gestire al più due sorgenti
- Deve essere una procedura con un signature ben definita (0 parametri, no valore di ritorno

Note importanti

- Differenze di buttonImpl rispetto al pattern oberserver
    - Tipicamente nel pattern observer il flusso di controllo che esegue il listener non è l'osservatore
    - In questo caso invece è il flusso dell'osservatore
- Nonostante non ci siano flussi di controllo che possono infierire, corse critiche possono insorgere dal momento che il flusso di controllo principale viene interrotto mentre accede alle informazioni che vengono aggiornate dal listener
- Per evitare corse critiche è necessario rendere atomica l'esecuzione dei blocchi di codice del flusso principale → disabilitando e abilitando le interruzioni
- Il codice dei listener viene mandato direttamente in esecuzione dall' interrupt hendler

Modulo 3.1 SISTEMI EMBEDDED BASATI SU SOC

Dai microcontrollori ai soc

- Nei microncontrollori non c'è il sistema operativo
- Mentre nei sistemi embedded embedded basati su SoC c'è SO, memoria e potenza CPU

Soc e single-board cpu

- System-on-a-Chip, è un sistema completo che include cpu, memoria, ecc..

Sistemi operativi embedded e real time

- ROTS (sistemi operativi embedded e real time) → sono sistemi operativi progettati per venire usati nei Sistemi Embedded
- Compattezza, gestione estremamente efficiente delle risorse, affidabilità
- Sono spesso progettati per mandare in esecuzione una sola applicazione alla volta

Modulo 3.2 SISTEMI OPERATIVI EMBEDDED E REAL TIME

Sistemi embedded real-time

- Real-time: devono essere in grado di gestire/rispondere ad eventi e input entro i limiti di tempo prestabiliti
- Deadline: termine entro il quale un task deve essere terminato
- Tipologie:
    - HARD real-time: deadline devono essere rispettate sempre (es: safety-critical)
    - SOFT real-time: deadline possono rispettare condizioni normali e ci sono casi in cui possono non essere rispettate (es: non safety-critical)

determinismo → proprietà che indica che un sistema deve essere prevedibile, come il tempo per svolgere un task oppure il tempo per una certa azione

Benefici RTOS (real-time operating system)

- Migliorare la responsiveness
- Diminurire overhead
- Aumentare la portabilità
- Semplificare la condivisione delle risorse

Tipi di scheduling

- Big Loop → c'è un singolo loop di controllo dove ogni task viene polled per vedere se deve essere eseguito, controllo sequenziale o con priorità. Inefficiente e non c'è responsiveness
- Round-Robin → utilizza i turni per i task ready, thread eseguiti fino al completamento o al blocco, oppure si imposta una fetta di tempo per ogni thread così da usare la preemption
- Priority-Based → esegue sempre il thread con priorità maggiore, e se hanno la stessa priorità RR
- I sistemi RTOS supportano la priority based itegrata con la RR, per garantire una massima responsiveness, ma ha un costo e c'è la possibilità di starvation

Comunicazione scambio messaggi → permettono la comunicazione di messaggi tra task utilizzando delle code condivisibili tra più task

Sistemi real-time sincroni→ clock utilizzato per suddividere il tempo del processore in intervalli chiamati frame

- Il programma deve essere suddiviso in task in modo che ogni task possa essere completamente eseguito nel caso peggiore in un singolo frame
- Tabella di scheduling per assegnare i task ai frame, così da riuscire ad eseguire tutti i task all'interno di un frame
- Ad ogni clock lo scheduler assegna di nuovo i task ai frame
- Se un task è troppo grande per il frame, allora si suddivide in sotto-task
- Questo approccio è utilizzato nei sistemi di controllo hard-time

Sistemi real-time asincroni → non si richiede che i task completino l'esecuzione prima del frame

- Ogni task prosegue la propria esecuzione e lo scheduler invoca il prossimo task
- Pro: molto efficiente
- Contro: deadline più complessa da soddisfare

Modulo 3.3 ARCHITETTURE DI CONTROLLO PER SISTEMI EMBEDDED BASATE SU MULTI THREADING

Sistemi embedded e multi-threading

➔ Nei sistemi embedded con sistema operativo tipicamente mandano in esecuzione una singola applicazione con la possibilità che ci siano più thread al loro interno

Task e thread

➔ Il termine task è spesso usato come sinonimo di thread, anche se fanno riferimento a livelli logici di concorrenza diversi:
  - Thread → meccanismo abilitante fornito dal sistema operativo
  - Task → astrazione con cui modularizzare/decomporre il comportamento di un sistema

Mapping

➔ Quindi è possibile fornire un mapping per avere a livello logico i task, mandati in esecuzione sul livello fisco dei th
➔ Ogni task ha un proprio thread – one-to-one, per evitare control loop con busy waiting
  - Un proprio loop
  - Concorrenza task → concorrenza thread
➔ Più task eseguiti dal medesimo pull thread

Task e agenti

➔ Agente come componente software attivo che funge le veci di esecutore e controllore del task
➔ Agenti come astrazione/componente software
  - Entità attive → progettate per svolgere uno o più TASK
    - Flusso di controllo autonomo
  - Interazione con ambiente → elaborano le informazioni che arrivano in input dall'ambiente dai sensori
  - Interazione con altri agenti → comunicazione basata sullo scambio dei messaggi

Interazione coordinazione tra task

➔ Sistemi articolati e complessi beneficiano di una progettazione/sviluppo in cui si assume la decomposizione in più task
➔ Più task significa anche avere più thread in esecuzione concorrente (anche in parallelo su più processori)
➔ I task devono essere totalmente indipendenti, oppure essere dipendenti ma con opportuni meccanismi di gestione (es: semafori)
  - Quando si adottano schemi implementativi con stati ed elaborazione di eventi, non ci possono essere cicli infiniti o operazioni bloccanti in attesa di eventi di sincronizzazione
  - Semafori e monitor devono essere usati solo per la mutua esclusione e sezioni critiche
    - Il blocco dovrebbe essere per brevi periodi
  - Forma di sincronizzazione: scambio di messaggi o approcci asincroni → si potrebbe usare solo lo scambio di messaggi e non semafori/monitor

Scambio di messaggi

➔ Modello asincrono
  - Send non bloccante
  - Integrazione con modello ad eventi
➔ Ricezione messaggi: due macro-approcci
  - Receive esplicita: blocca il flusso di controllo in attesa che disponibile un messaggio
  - Reiceve implicita: la ricezione del messaggio viene gestita come un evento

Modulo 4.1 DAI SISTEMI EMBEDDED AD IOT

Reti di sistemi embedded: M2M (machine to machine)

- ➔ Insieme di tecnologie che supportano la comunicazione wired e wireless fra dispositivi al fine di permettere scambio di informazioni locali
- ➔ Usato soprattutto Monitoraggio e controllo

Sistemi SCADA

- ➔ È un sistema M2M molto diffuso
- ➔ Sistemi di supervisione che operano con codifiche di segnali trasmesse per canali → controllo remoto di sistemi

IoT

- ➔ M2M è una parte integrante dell'IoT (internet of things)
- ➔ Nel 199 è stato introdotto e l'obbiettivo iniziale era automatizzare l'inserimento di dati real-time in rete

- ➔ Sono sistemi costituiti da reti di oggetti fisici (embedded) che interagiscono mediante la rete internet.
- ➔ Infrastruttura che supporta la connettività ad ampia scala di dispositivi e dati
- ➔ Integrazione di sistemi embedded con sensori software e sistemi di comunicazione
- ➔ RFID (radio-Frequency Identification), è una tecnologia usata in origini per l'identificazione e/o memorizzazione automatica di informazioni inerenti oggetti tramite delle etichette elettriche (tag) e tramite queste possono rispondere a interrogazioni a distanza da parte di appositi apparati fissi e portatili
- ➔ Things → sono i sistemi embedded e i sensori in particolare
- ➔ Connettività e comunicazione
    - Studio di protocolli di comunicazione
    - Interoperabilità
    - Sicurezza → autenticazione, autorizzazioni, ecc.
- ➔ Collezionare/acquisire e trasmettere dati, ogni oggetto è generatori di dati

Internet e cloud

- ➔ Capacità di comunicare direttamente o indirettamente con la rete internet
- ➔ Gestire grandi volumi di dati generati dai sensori
- ➔ Queste informazioni diventano accessibili mediante opportuni servizi Internet o web che ne permettono lo scambio con altre applicazioni
- ➔ Cloud: paradigma di erogazione di risorse informatiche (archiviazione, elaborazione, trasmissione dati).
- ➔ I servizi cloud vengono primariamente utilizzati per **raccolta** dati dai dispositivi e **accesso** a dati inviati dai dispositivi

IoT + Web = web-of-things → WOT

- ➔ Consiste in sistemi che incorporano gli oggetti fisici di uso quotidiano nel www dando loro una API
- ➔ Facilita la creazione di profili virtuali degli oggi e la loro integrazione e riuso in tipi diversi di applicazioni
- ➔ Evoluzione dell'IoT in cui gli oggetti comunicano mediante livello di rete (ZigBee, Bluetooth, ecc.)

Modulo 4.2 TECNOLOGIE E PROTOCOLLI PER LA COMUNICAZIONE DI SISTEMI EMBEDDED

"Things" in IoT

- ➔ Dispositivi che usiamo tutti i giorni, di qualsiasi dimensione, dai piccoli (orologio da polso, ecc.) ai grandi (robot, auto, ecc.)
- ➔ Questi dispositivi interagiscono con gli utenti, generando e acquisendo informazioni nell'/dall' ambiente dove si trovano

Interazione con internet

- ➔ Capacità di comunicare direttamente con la rete internet
- ➔ Grandi volumi e stream di dati generati dai sensori
- ➔ Queste informazioni diventano accessibili mediante opportuni servizi internet o web. Ne permettono lo scambio, soprattuto con app mobile.

Comunicazione in sistemi M2M e IoT

- ➔ La comunicazione può essere wired o wireless
  - Wired: sono sheild/schede con porta ethernet e protocollo 802.11
  - Wireless: possono essere sheild oppure doungle USB contenente modulo ricevitore/trasmettitore e antenna
- ➔ La comunicazione tra microcontrollore e moduloi di comunicazione, di solito avviene tramite seriale

Connessione ad internet

- ➔ Ci sono due modi principali per connettere un dispositivo embedded alla rete internet in un ottica IoT:
  - Indirettamente:
    - Mediante la comunicazione di un nodo intermedio (tipo gateway) → come per tecnologia wireless come BT e ZigBee
  - Direttamente
    - Montando un modulo di comunicazione opportuno (es: moduli 3G/4G, wifi)
    - Protocollo a cui fare riferimento è quello di internet, l'IP (TCP o UDP)

Comunicazione wireless

- ➔ Si intende senza fili e sono forme di radiocomunicazione
- ➔ Fanno uso del canale radio o del mezzo, diffuse nell'etere per trasportare a distanza l'informazione tra utenti attraverso segnali elettromagnetici, appartenenti alle frequenze radio o microonde dello spettro elettromagnetico (banda radio)
- ➔ Una radio comunicazione può essere:
  - Terrestre → se si appoggia almeno in parte alle infrastrutture sulla terra
  - Satellite → se si appoggia almeno in parte alle infrastrutture in orbita

Bluetooth

- ➔ Trasmissione dati tramite reti personali senza fili, si basa sulla scoperta dinamica di dispositivi coperti dal segnale radio entro un raggio di qualche decina di metri
- ➔ Esempio: cellulare, stampanti, portatili, ecc.
- ➔ Standard progettato per avere:
  - Bassi consumi
  - Corto raggio d'azione
  - Basso costo di produzione
- ➔ Tipologie di rete
  - La rete di base del BT è chiamata **piconet**, ed è basata su master-slave
  - Ogni dispositivo bluetooth è in grado di collegarsi simultaneamente la comunicazione con altri 7 dispositivi slave
  - Un dispositivo per volta può comunicare con il master
  - Il master ha un clock che sincronizza la comunicazione
- ➔ Tipi di connessioni
  - I collegamenti che possono essere stabiliti tra i diversi dispositivi sono di due tipi:
    - orientati alla connessione
      - richiede di stabilire una connessione tra i dispositivi prima di inviare i dati
    - senza connessione
      - non richiede alcuna connessione prima di inviare i pacchetti, purchè conosca l'indirizzo del destinatario
- ➔ evoluzione dello standard
  - dalle 1.0 (1999) alla 4.0 (2010)
  - la versione 4.0 contiene i protocolli:
    - classic bluetooth
    - bluetooth high speed
    - bluetooth low energie → pensato appositamente per l'IoT (consumo di energia ridotto ed è più reattivo)

Zigbee

- ➔ una delle tecnologie più recenti ed utilizzate nei sistemi embedded
- ➔ utilizza un insieme di protcolli di comunicazione ad alto livello che utilizzano piccole antenne digitali a bassa potenza
- ➔ ideato per IoT
  - open global standard
  - basso consumo, basso costo

Zigbee VS Bluetooth

- ➔ miglior efficienza, range più ampio, creazione di reti P2P più semplice e agile
- ➔ soluzione migliore per IoT
- ➔ meno utilizzata a livello consumer

Tecnologie radio per identificazione → **RFID** viene sostituito sempre di più da NFC e iBeacon

- ➔ **NFC**: Near Field Comunication
  - Combinazione tra RFID e altre tecnologie
  - È bidirezionale e ha un raggio di 10 cm
  - Viene creata una rete P2P quando si avvicinano due dispositivi tra i due per inviare e ricevere dati
- ➔ **BEACON**
  - Utilizzate per localizzazione/identificazione wireless tipicamente in ambienti indoor
  - È un piccolo trasmettitore a bassissimo consumo che emette sempre un segnale (il proprio ID), così da farsi rilevare da un BLE (Bluetooth low energy)

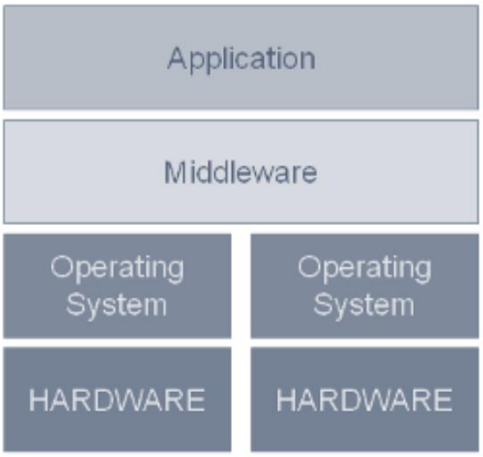
Modulo 4.3 PROTOCOLLI E MIDDLEWARE PER IOT

Device e servizi/server-side

- ➔ Due parti fondamentali dei sistemi IoT
  - Dispositivi stessi
  - Parte server-side che li supporta
  - Terzo livello che spesso è opportuno considerare → low powe gateway che risiedono tra device e internte
- ➔ In entrambi i casi, i device sono collegati in modo intermittente (comunicazione assa, esaurimento batterie, ecc.)

Middleware

- ➔ Livello software che risiede tra il sistema operativo e le applicazioni
- ➔ Fattorizza servizi e funzionalità di alto livello e le rende disponibili ad ogni applicazione che ne ha bisogno (comunicazione, coordinazione, sicurezza, ecc.)



Connettività, comunicazione, interoperabilità

- ➔ Gestione/comunicazione di dispositivi e applicazioni di tipo diverso
- ➔ Supporto protocolli di alto livello, tutta via con implementazione che tiene conto delle risorse limitate dei dispositivi
- ➔ Gestione di connessioni dirette o mediante gateway

Modulo 4.4 MODELLI DI COMUNICAZIONE E SCAMBIO DI MESSAGGI

Modelli di comunicazione

- ➔ La progettazione e programmazione di sistemi software embedded di rete richiede l'introduzione di opportuni modelli e meccanismi di consumazione di alto livello e i relativi supporti tecnologici

Modello a scambio di messaggi

- ➔ Comunicazione mediante invio e ricezione di messaggi
  - Modello alternativo alla comunicazione mediante memoria condivisa e modello di riferimento per i sistemi distribuiti
- ➔ Primitive: send e receive
- ➔ Tipologie:
  - Diretta/Indiretta
  - Sincrona/Asincrona
  - Linguaggio descrizione messaggi → interoperabilità

Diretta VS indiretta

- ➔ Diretta
  - Comunicazione avviene direttamente tra i processi (thread)
  - Primitive → send(ProcId, Msg) , receive(): Msg
- ➔ Indiretta
  - Si usano canali che fungono da mezzi di comunicazione indiretta
  - Invio e recezione sono tramite questi canali. Send(Channel, Msg), receive(Channel): Msg

Sincrona vs Asincrona

- ➔ Sincrona
  - La send ha successo quando il messaggio specificato è ricevuto dal destinatario mediante una receive
- ➔ Asincrona
  - La send ha successo quando il messaggio è stato inviato (ma non necessariamente ricevuto tramite receive)

Bufferizzazione

- ➔ Aspetto importante è la strategia del buffer
- ➔ È fondamentale nel caso asincrono, sia nel indiretto per i canali, sia per il diretto, per la coda che i processi usano.

Primitive: vari tipi di semantica

- ➔ Send(DestID, Msg)
  - Ha successo quando:
    1. Il msg è stato inviato (completo caso asincrono) → la verifica è implementata a livello di protocollo (errore se destinatario non esiste)
    2. Il msg è arrivato, ma non necessariamente ricevuto (errore se destinatario non esiste)
    3. Il msg è stato ricevuto (completo caso sincrono)
- ➔ Receive(): Msg
  - Semantica Bloccante (si blocca quando non è disponibile almeno un messaggio)

Buffer full

- ➔ caso asincrono, caso in cui il buffer di ricezione si riempia
- ➔ semantica per la send
  - se fallisce il msg viene scartato, si blocca in attesa che il buffer non sia pieno
  - se ha successo viene riscritto il messaggio del buffer

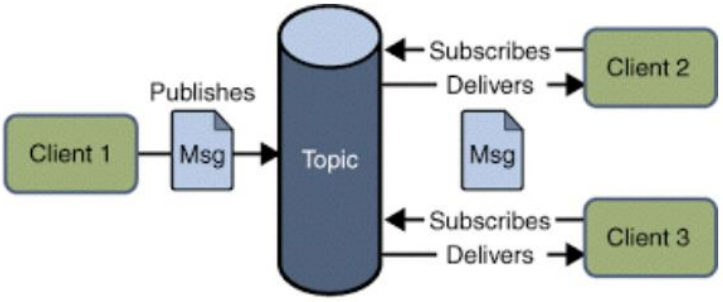
Rappresentazione dei messaggi

- ➔ aspetto fondamentale della comunicazione è il lingiuaggio con cui è rappresentato il messaggio. Che deve essere lo stesso per ogni partecipante alla comunicazione → interoperabilità

- ➔ oltre a questo bisogna mettersi d'accordo con il formato dei messaggi e la semantica dei protocolli (nel caso ci siano)
- ➔ può essere JSON o XML

Modello publish/subscribe

- ➔ questo modello di scambio messaggi è tipicamente usato per realizzare architetture ad eventi di alto livello
- ➔ simile a pattern observer
- ➔ usato in ambito IoT
- ➔ primitive: publish, subscribe



Modelli di comunicazione e architetture di controllo

- ➔ integrazione modelli di comunicazione con i modelli e architetture software di controllo visti per i sistemi embedded (loop controllo semplici, macchine a stati finiti, ecc.)

Integrazione nei sistemi embedded organizzati a loop

- ➔ problema: receive **bloccante → comporta il blocco del loop, non è usabile nell'implemnetazione degli stati finiti**

Estensione del modello

- ➔ Receive **non bloccante**
  - ReceiveMsg(): { Msg, errore/null} → nel caso in cui non ci siano messaggi, la receive non si blocca (restituisce errore o genera eccezione)
  - Primitiva per testare i messaggi disponibili isMsgAvaible(): boolean

Integrazione del modello FSM

- Una transazione può essere scatenata dalla presenza di un messaggio → uso della primitiva di test delle guardie
- Nell'insieme delle azioni associate alla transizione o allo stato c'è anche la ricezione (non bloccante) e l'invio di messaggi

Problema selezione messaggi

- ➔ Supponiamo che nel medesimo stato possano essere ricevuti più messaggi, di tipo diverso, e che a seconda del messaggio ricevuto la macchina deve transitare in uno stato o in un altro
- ➔ Le primitive fornite non sono pronte a questo
- ➔ Quindi possiamo usare primitive che permettano di esplicitare quali messaggi riceve
- ➔ Soluzione: Msg Pattern
  - isMsgAvaible(Pattern pattern): boolean
    - è true se è presente un messaggio che corrisponde al pattern specificato
  - receiveMsg(Pattern pattern): Msg
    - recupera i messaggi corrispondenti al pattern specificato

Integrazione nel modello task

- Un pattern può essere rappresentato o da un predicato, o analogamente da una interfaccia con metodo match implementata da classi concrete

```
interface Pattern {
  boolean match(Msg msg);
}
```

- Nelle primitive:
  - isMsgAvailable(Pattern pattern): boolean
    - è true se è presente almeno un messaggio che soddisfa la funzione match del pattern
  - receiveMsg(Pattern pattern): Msg
    - seleziona un messaggio (se presente) che soddisfi la funzione match del pattern

Integrazione nel modello a task

➔ lo scambio di messaggi può essere usato tra i task stessi o nel completamento dell'utilizzo di una memoria condivisa

Integrazione nel modello ad event loop

➔ il modello di comunicazione deve essere necessariamente asincrono con receive implicita o non bloccante

➔ l'arrivo di un messaggio è modellato come evento