Contents:

Tema d'esame del 14/6/2017

1. Descrizione

Introductory definitions

An *embedded system* is a computer system with a dedicated function (special purpose) within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts.

Embedded systems are electronic programmable sub-systems that are generally an integral part of a larger heterogeneous system.

- Componente fisica
- Contengono (uno o più) computer che elaborano informazione che verrà prodotta e/o consumata dal processo fisico

Il processo fisica influenza la computazione e viceversa (feedback). La computazione deve avvenire:

- In maniera **reattiva**: reazione a input del processo fisico
- In maniera **concorrente**: più fenomeni del processo fisico che scatenano computazione possono avvenire contemporaneamente
- **Real-time**: i risultati devono essere prodotti in un tempo compatibile con la dinamica del processo fisica.

Attuatori ~ Sensori (A<->D)

La complessità è dovuta alla varietà di domini applicativi.

Requisiti

- Affidabilità
- Efficienza
- Real-time constraints
- Flessibilità
- Grado di interazione con l'uomo
- Mercato

Affidabilità

Caratteristica di un sistema il cui funzionamento produce gli effetti voluti, o scostamenti accettabili da questi.

Business-critical: mette a rischio la riuscita del business.

Safety-critical: mette a rischio vite umane, affidibilità richiesta elevata, hanno impatto profondo sull'ambiente.

Attributi:

- Correttezza - rispetto dei requisiti;
- Robustezza - comportamento accettabile anche in situazioni impreviste/non specificate nei requisiti;
- Sicurezza (*security*) - il sistema impedisce usi non autorizzati;
- Innocuità (*safety*) - il sistema non ha comportamenti pericolosi.

Efficienza

Quella caratteristica di un sistema che porta a termine il proprio compito con un numero limitato di risorse.

Attributi:

- Peso
- Dimensioni fisiche
- Efficienza energetica
- Occupazione di memoria del software
- Grado di riuso
- Costo

Real-time constraints

Il **tempo di computazione** costituisce uno dei parametri dell'efficienza.

In alcuni domini applicativi, i vincoli real-time determinano l'affidabilità del sistema: una risposta esatta, in ritardo, è una risposta sbagliata.

2. Struttura

Introductory definitions

An *integrated circuit* or monolithic integrated circuit (also referred to as an **IC**, a chip, or a microchip) is a set of electronic circuits on one small flat piece (or "chip") of semiconductor material, normally silicon.

A *printed circuit board* (**PCB**) mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate. Components (e.g. capacitors, resistors or active devices) are generally soldered on the PCB. Advanced PCBs may contain components embedded in the substrate.

A *system on a chip* or system on chip (**SoC** or SOC) is an integrated circuit (also known as an "IC" or "chip") that integrates all components of a computer or other electronic systems. It may contain digital, analog, mixed-signal,

and often radio-frequency functions—all on a single substrate. SoCs are very common in the mobile computing market because of their low power-consumption. A typical application is in the area of embedded systems.

The contrast with a microcontroller, SoC integrates microcontroller (or microprocessor) with advanced peripherals like graphics processing unit (GPU), Wi-Fi module, or coprocessor.

Network on chip or network on a chip (**NoC** or NOC) is a communication subsystem on an integrated circuit (commonly called a "chip"), typically between intellectual property (IP) cores in a system on a chip (SoC). NoCs can span synchronous and asynchronous clock domains or use unclocked asynchronous logic. NoC technology applies networking theory and methods to on-chip communication and brings notable improvements over conventional bus and crossbar interconnections. NoC improves the scalability of SoCs, and the power efficiency of complex SoCs compared to other designs.

Architettura

The *Harvard architecture* is a computer architecture with physically separate storage and signal pathways for instructions and data. Today, most processors implement such separate signal pathways for performance reasons, but actually implement a modified Harvard architecture, so they can support tasks like loading a program from disk storage as data and then executing it.

The *von Neumann architecture* describes a design architecture for an electronic digital computer with parts consisting of a processing unit containing an arithmetic logic unit and processor registers; a control unit containing an instruction register and program counter; a memory to store both data and instructions; external mass storage; and input and output mechanisms. The meaning has evolved to be any stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus. This is referred to as the von Neumann bottleneck and often limits the performance of the system.

Nell'architettura di Von Neumann il programma e i dati sono nella stessa memoria.

Nell'architettura Harvard vengono usate memorie distinte per programmi e dati e bus distinti per accedervi.

Harvard rispetto a Von Neumann:

- Maggior parallelismo con pipelining;
- Possibilità di larghezza di bus diverse per dati e istruzioni;
- Maggior sicurezza (i dati non possono essere eseguiti).

Componenti

- Unità di elaborazione
  - General-purpose
    - Microprocessori (MPU) e microcontrollori (MCU)
    - Logiche programmabili (FPGA)
  - Special-purpose
    - Digital Signal Processors (DPS)
    - Graphic Processing Units (GPU)
    - Application-specific IC (ASIC)
- Memorie
  - RWM

        * SRAM
        * DRAM
    - NVRWM
    - ROM
- Sistemi di comunicazione
- Periferiche, sensori e attuatori
    - GPIO
    - PWM
    - Parallelo
    - RS-232
    - USB
    - I2C
    - SPI
    - CAN
    - JTAG

Unità di elaborazione

Microprocessori (MPU)

- `ADDR`, linee indirizzo, indicato quali dati leggere/scrivere
- `DATA`, comunicazione dati
- `R/W`, precisano l'operazione da eseguire
- `CLK`, segnale di clock, sincronizzazione

Tipologie

An *instruction set architecture* (**ISA**) is an abstract model of a computer. It is also referred to as architecture or computer architecture. A realization of an ISA is called an implementation. An ISA permits multiple implementations that may vary in performance, physical size, and monetary cost (among other things); because the ISA serves as the interface between software and hardware. Software that has been written for an ISA can run on different implementations of the same ISA. This has enabled binary compatibility between different generations of computers to be easily achieved, and the development of computer families. Both of these developments have helped to lower the cost of computers and to increase their applicability. For these reasons, the ISA is one of the most important abstractions in computing today.

An ISA defines everything a machine language programmer needs to know in order to program a computer. What an ISA defines differs between ISAs; in general, ISAs define the supported data types, what state there is (such as the main memory and registers) and their semantics (such as the memory consistency and addressing modes), the instruction set (the set of machine instructions that comprises a computer's machine language), and the input/output model.

Due *famiglie* di ISA: CISC e RISC

CISC

Esempi: x86, x86-64, 8051.

- Istruzioni che eseguono operazioni complesse, possibilimente con trasferimenti multipli da/a memoria;

- Lunghezza delle istruzioni variabile (più comuni, più corte);
- Pochi registri;
- Motivata dalla situazione fino alla fine anni '70:
  - Programmazione in assembler, necessità di istruzioni complesse per facilitarne il compito;
  - Memorie costose, istruzioni complesse a lunghezza variabile per ridurne l'utilizzo;
  - Velocità processore e memoria paragonabile, accessi multipli in memoria non problematici.

RISC

Esempi: AVR, ARM, MIPS, POWER.

- Diffusione compilatori e linguaggi ad alto livello: meno necessità di istruzioni asm complesse
- Aumenta la velocità del processore rispetto a quella della memoria, necessità di ridurre trasferimenti da/a memoria: molti registri;
- Minore costo nella memoria: lunghezza istruzioni fissa;
- Distingue istruzioni load/store da istruzioni di manipolazione che operano sui registri;
- Pipelining
- Più spazio per cache sul chip
- Architettura load/store

Logiche programmabili

Circuiti programmabili in hardware, composti da celle che realizzano varie funzioni. Si "programma" lo schema elettrico tramite interconnessioni tra celle e pin di I/O.

Le FPGA, ad esempio, sono logiche programmabili sul "campo", ovvero dopo che il chip è stato montato sul sistema. Sono matrici di celle logiche (milioni) collegate da interconnessioni programmabili. La programmazione avviene caricando una particolare configurazione in una RAM. Vengono utilizzati linguaggicome Verilog, VHDL e C.

Fasi dello sviluppo su FPGA:

- Compilazione per produrre TRL netlists
- Sintesi per mappare la netlist sulle unità della FPGA
- Placing and routing per stabilire *quali* unità nella matrice utilizzare ed interconnettere
- Output: bitfile che viene caricato nella memoria della FPGA

Special-purpose processors

Contengono istruzioni progettate per specifiche categorie di applicazioni. Tipicamente CISC e Harvard achitecture.

Esempi: Digital Signal Processors (DPS) tra cui Finite Impulse Response (FIR), Graphics Processing Units (GPU), Cryptoprocessors.

Memorie

RWM (volatili)

Il contenuto della memoria è ritenuto finchè questa è alimentata.

SRAM

Static RAM, utilizza circuiti bistabili (flip-flop)

- Elevatissima velocità
- Basso consumo energetico
- Bassa densità
- Costi elevati

DRAM

Dynamic RAM, utilizza carica in condensatore (necessita refresh)

- Elevata velocità
- Elevato consumo energetico
- Alta densità
- Costi bassi

NVRWM

Non volatili

- **ROM**
  - Mascherate o programmabili (PROM)
  - Altà densità
  - Basso costo
  - Affidabile
- **EPROM** (Electrically Programmable ROM): contenuto della memoria programmabile elettricamente e cancellabile con raggi UV
- **EEPROM** (Electrically Erasable Programmable ROM): contenuto cancellabile elettricamente a livello di byte
  - Bassa densità
  - Alto costo
  - Mediamente affidabile
- **FLASH**: EEPROM cancellabile a blocchi, tempi di cancellazione lunghi
  - Alte densità
  - Basso costo
  - Mediamente affidabile

Gerarchie di memoria

Le memorie più veloci sono le più costose e a minore densità, risulta pertanto utile organizzarle in maniera gerarchica: più *vicina* al processore la memoria più veloce e più piccola.

Se non trovo un dato in un livello gerarchico di memoria, accedo al livello successivo.

Sfrutta le proprietà di località del software, ma il tempo d'accesso diventa variabile.

[...][...] Mappe memoria [...] Configurazioni processore-memoria

Sistemi di comunicazione

Permettono di far dialogare la parte computazionale di un sistema embedded con altri sottosistemi digitali o con il mondo esterno.

Si distinguono per alcune caratteristiche:

- Digitale/Analogico
- Wired/Wireless
- Seriale/Parallelo
- Sincrono/Asincrono
- Point-to-point/Bus/Stella/...
- Campionato o event-triggered
- Bit rate
- Requisiti elettrici
- Controllo accesso, sicurezza, autenticazione
- Connettori fisici

General-purpose I/O (GPIO)

General-purpose input/output (GPIO) is a generic pin on an integrated circuit or computer board whose behavior—including whether it is an input or output pin—is controllable by the user at run time.

GPIO capabilities may include:

- GPIO pins can be configured to be input or output
- GPIO pins can be enabled/disabled
- Input values are readable (typically high or low)
- Output values are writable/readable
- Input values can often be used as IRQs (typically for wakeup events)

GPIO peripherals vary widely. In some cases, they are simple—a group of pins that can switch as a group to either input or output. In others, each pin can be set up to accept or source different logic voltages, with configurable drive strengths and pull ups/downs. Input and output voltages are typically—though not always—limited to the supply voltage of the device with the GPIOs, and may be damaged by greater voltages.

A GPIO pin's state may be exposed to the software developer through one of a number of different interfaces, such as a memory mapped peripheral, or through dedicated IO port instructions. Some GPIOs have 5 V tolerant inputs: even when the device has a low supply voltage (such as 2 V), the device can accept 5 V without damage.

Potrebbe essere necessario utilizzare più pin per lo stesso dispositivo, per esempio per un controllo con ridondanza (open drain o tristate configuration).

Pulse Width Modulation

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a modulation **technique** used to encode a message into a pulsing signal.

The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

Connessioni digitali cablate

Parallelo

1 linea per bit, per una certa ampiezza di parola.

(S)ATA, PCI, IEE1284

Un'interfaccia parallela richiede un elevato numero di pin:

- Poco adatto per un sistema embedded
- Solitamente half-duplex per dimezzare il numero di linee

Inoltre, le linee

- devono essere sincronizzate tra loro (diventa problematico con la distanza);
- fanno interferenza reciproca e sono più sensibili al rumore;
- devono essere pilotate più lentamente a causa della mutua induttanza/capacità.

Seriale

1 linea per direzione di flusso, singoli bit inviati in *sequenza*

RS232, USB, SATA, SPI, I2C

PCI express è invece un esempio di implementazione *mista*.

Tutte le interfacce piu recenti e veloci sono seriali, quelle parallele vengono spesso emulate per retrocompatibilità con i pin GPIO.

RS 232

In telecommunications, RS-232 is a standard for serial communication transmission of data. It formally defines the signals connecting between a DTE (data terminal equipment) such as a computer terminal, and a DCE (data circuit-terminating equipment or data communication equipment), such as a modem. The RS-232 standard is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors

- Versione a 9 pin: async point-to-point
- 20Kbit/sec a max 300 m di distanza
- Sincronizzazione con bit di start (1) e di stop (1 o 2)
- Trasmissione dei bit con livelli di tensione tra -15 e -3 (bit 1) e tra 3 e 15 (0)
- Durata bit a seconda della velocità di trasmissione negoziata

UART e USART

- UART: Universal Asynchronous Receiver/Transmitter
- USART: Universal Synchronous/Asynchronous Receiver/Transmitter

Componenti di un computer embedded (e di PC) che trasformano una o più parole di memoria in una sequenza di trasmissione su un link seriale, e viceversa. Usati per diversi tipi di protoccoli seriali (RS-232, RS422, RS-485).

USB

- Protocollo seriale, topologia ad albero con un controllore (master) e gli altri dispositivi sono controllati (slave) ma appare come bus.
- Max 25 m distanza, max 10 Gbit/sec
- Elettricamente più semplice e robusto di RS-232, richiede una logica di controllo sofisticata
- Consente di collegare una grandissima varietà di dispositivi

I2C

- Bus di comunicazione seriale tra circuiti integrati, con multi-master e arbitrazione
- Max 3.4 Mbit/sec, corta distanza
- Solo 2 fili
- Interfacciamento open drain (richiede resistenza di pull-up)
- Principalmente per comunicazione tra componenti a bassa velocità su stessa scheda, o su schede diversi

SPI

- Bus seriale
- Al contrario di I2C:
  - Non richiede resistenze di pull-up (meno consumi)
  - Numero di linee più elevato
  - Il troughput è molto più alto
  - Supporta un singolo master
- Più adatto per trasferire data streams, I2C è più adatto invece per trasferire parole singole.

CAN bus

- Controller Area Network, sviluppato da Bosch per connessione dispositivi in ambito industriale (elevato rumore elettromagnetico).
- Fino a 1 Mbit/s sotto i 40 m, max 500m a 125 Kbit/s
- Asincrono, multi-master basato su priorità

JTAG

- Protocollo hardware per il testing di circuiti integrati e stampati
- Originariamente creato per il boundary testing (test dei pin dei circuiti integrati presenti sulla scheda)
- Nelle implementazioni piu recenti viene utilizzata come porta per il debugging

Microcontrollori

- Microcontrollore = microprocessore + memoria + periferica su un chip singolo
- Diversi tipi di memoria (SRAM, EEPROM, FLASH) di dimensioni limitate
- Diversi tipi di periferiche on-chip (I/O, timer, convertitori ADC DAC, PWM, controller memoria esterna)
- Velocità di calcolo (KHz - MHz)
- Bassi consumi e costi (0,25 - 1 $ / unità)
- 4, 8, 16, 32 bit

Silicon Labs C8051F020

- Core 8051 (8 bit), 25 MHz
- SRAM: 4 kb, FLASH: 64 kb
- Devices: timer, digital I/O (8+4 bit), I2c, SPI, UART, DAC, ADC, memoria esterna, sensore temperatura

4. Periferiche

I **Comparatori** ricevono in ingresso due segnali analogici e ritornano un segnale alto o basso se il primo segnale è maggiore o minore del secondo.

Timer

Periferica che genera un segnale dopo un certo intervallo temporale. Funzionamento:

- Un segnale viene generato da un oscillatore al quarzo o ricevuto in input da un pin opportuno
- Il segnale può essere scalato secondo un divisore di frequenza configurabile
- Il segnale decrementa un registro contatore ad ogni colpo
- Quando il registro arriva a zero, viene generato il segnale di output

Real-time clock

- Usato per mantenere data e organizzarle
- Una batteria lo mantiene attivo quando il sistema è spento
- Oscillatore al quarzo per misura precisa del tempo

Watchdog timer

- Invece di segnalare la scadenza di un periodo di tempo, aspetta di ricevere un input entro il tempo configurato
- Un interrupt può essere innalzato tale segnale non arriva in tempo.
- is-alive check

Direct Memory Access

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (RAM), **independent** of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors.

Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology.

Funzionamento:

- CPU richiede al dispositivo I/O di effettuare una lettura
- Impostazione del controller DMA con l'indirizzo di destinazione dei dati
- `DMAREQ`: il dispositivo di I/O chiede al controller DMA di iniziare il trasferimento
- `BUSREQ`: il controller DMA chiede alla CPU di sospendere l'uso del bus
- Una volta ricevuto `BUSACK`, il controller asserisce l'indirizzo di destinazione, ed invia `DMAACK`
- Il dispositivo di I/O pone il dato sul bus dati
- Alternativa: dual address model (I/O -> registro DMA e poi MEM)

Modalità di trasferimento

Burst mode

An entire block of data is transferred in one contiguous sequence. Once the DMA controller is granted access to the system bus by the CPU, it transfers all bytes of data in the data block before releasing control of the system buses back to the CPU, but renders the CPU inactive for relatively long periods of time. The mode is also called "Block Transfer Mode". It is also used to stop unnecessary data.

Cycle stealing mode

The cycle stealing mode is used in systems in which the CPU should not be disabled for the length of time needed for burst transfer modes. In the cycle stealing mode, the DMA controller obtains access to the system bus the same way as in burst mode, using BR (Bus Request) and BG (Bus Grant) signals, which are the two signals controlling the interface between the CPU and the DMA controller. However, in cycle stealing mode, after one byte of data transfer, the control of the system bus is deasserted to the CPU via BG. It is then continually requested again via BR, transferring one byte of data per request, until the entire block of data has been transferred. By continually obtaining and releasing the control of the system bus, the DMA controller essentially interleaves instruction and data transfers. The CPU processes an instruction, then the DMA controller transfers one data value, and so on. On the one hand, the data block is not transferred as quickly in cycle stealing mode as in burst mode, but on the other hand the CPU is not idled for as long as in burst mode. Cycle stealing mode is useful for controllers that monitor data in real time.

Transparent mode

Transparent mode takes the most time to transfer a block of data, yet it is also the most efficient mode in terms of overall system performance. In transparent mode, the DMA controller transfers data only when the CPU is performing operations that do not use the system buses. The primary advantage of transparent mode is that the CPU never stops executing its programs and the DMA transfer is free in terms of time, while the disadvantage is that the hardware needs to determine when the CPU is not using the system buses, which can be complex.

Sensori

Comparatori

Ricevono due segnali analogici, ritornano un segnalo alto o basso se il primo dei due input è maggiore o minore del secondo.

Convertitori A/D

Convertono un segnale elettrico analogico in un *equivalente* segnale digitale, in due fasi:

- **Campionamento** segnale analogico continuo nel tempo viene discretizzato, osservando i suoi valori ad istanti regolari;
- **Quantizzazione** il segnale campionato viene approssimato con la sua misura intera rispetto ad una certa unità di misura.

L'intervallo di tempo che passa da una misurazione all'altra durante il campionamento equivale a `1/f`. `f` è detta *frequenza di campionamento*.

Numero di bit necessari = `Floor(log2(Vsw/LSB))`

Nyquist/Shannon

- Ogni segnale è rappresentabile come sovrapposizione di sinusoidi di frequenza e ampiezza variabile
- L'intervallo di frequenze [0, Fb] in cui il segnale possiede il 90-95% della sua energia è detto **banda di segnale**
- Nyquist: Se la frequenza di campionamento è >= 2 Fb, il segnale puo essere fedelmente ricostruito da un suo campionamento

Aliasing

- Se il campionamento viene effettuato ad una frequenza minore della 2Fb, il segnale ricostruito può risultare molto diverso da quello che codificava.
- Se campiono a bitrate basso, devo eliminare le frequenze fuori dalla banda Nyquist con un filtro antialiasing.

Rumore di quantizzazione

L'operazione di quantizzazione di un segnale analogico non è trasparente, ed introduce cambiamenti nel sistema che equivalgono alla sovrapposizione di un segnale spurio al segnale stesso.

Questo "rumore" è distribuito uniformemente nell'intervallo +- 1/2 LSB:

Assumendo 1 Ohm R:

- Valore medio nullo
- **Deviazione standard** = LSB / rad(12) = 0.29 LSB
- **Potenza** = varianza = LSB^2 / 12

Range dinamico

La precisione p di un sensore è la minima differenza tra due valori x(t) percepibili dal sensore. Il range dinamico D è il rapporto (H - L)/ p tra il range di un sensore e la sua precisione. Si misura normalmente in decibel: D(db) = 20log10 (H-L)/ p.

Rumore

Per "rumore" si intende qualsiasi sorgente che introduce imprecisione e inquina la misurazione (da non confondere con il caso particolare di quello relativo alla quantizzazione, intrinseco all'operazione stessa). Il sensore misura dunque $x'(t) = x(t) + n(t)$.

Il rapporto segnale/rumore (SNR) è dato dal rapporto tra potenza del segnale e quella del rumore. In decibel:

$SNR_{dB} = 20log_{10} = \frac{x_{RMS}}{n_{RMS}}$

Dove:

$g_{RMS} = \lim_{T\to\infty} \sqrt{\frac{1}{2T}\int_{-T}^{T} g^2 dt}$

Il **range dinamicoo** è, in questo caso, è il rapporto segnale/rumore tra il massimo segnale che non satura la misurazione e il minimo segnale individuabile (R di quantizzazione).

Tipi di convertitori A/D

- Approssimazioni successive: ricerca binaria negli intervalli di tensione
- Delta: ricerca lineare negli intervalli di tensione
- Rampa: confronto con una rampa
- Delta-Sigma
- Flash: meno precisi, minore ritardo di conversione, maggiore frequenza massima di campionamento, più complesso in generale (realizzati con molte resistenze)

Codec

Coppie di convertitori A/D -> D/A che rispettano un dato standard sono detti codec. La quantizzazione è un esempio di codec *lineare*.

Un altro esempio sono i codec A-law e u-law per i segnali audio:

- Sono più precisi intorno allo 0 (non linearità);
- riducono il rumore di quantizzazione con segnali a basso volume.

Accelerometri

Misurano l'accelerazione propria in una certa direzione. Funzionano misurando la posizione rispetto ad un intelaiatura fissa di una massa mobile fissata con una molla. Confrontando l'accelerazione misurata con quella di gravità si può misurare l'inclinazione rispetto alla verticale. Si possono utilizzare per stimare velocità e posizione integrando l'accelerazione nel tempo:

$p(t) = p(0) + \int_0^t v(t)dt$

$v(t) = v(0) + \int_0^t x(t)dt$

Attuatori

Un attuatore è un meccanismo attraverso cui un agente agisce su un ambiente, inoltre l'agente può essere o un agente intelligente artificiale o un qualsiasi altro essere autonomo (umano, animale). In senso lato, un attuatore è talvolta definito come un qualsiasi dispositivo che converte dell'energia da una forma ad un'altra, in modo che questa agisca nell'ambiente fisico al posto dell'uomo.

Anche un meccanismo che mette qualcosa in azione automaticamente è detto attuatore.

Solenoidi lineari

Attuatori di movimento: basati sul campo magnetico prodotto dalla corrente che attraversa un avvolgirmento e sposta una barra metallica. I solenoidi latching (bistabili) mantengono lo stato allo spegnimento dell'alimentazione.

Motori DC

Producono un momento meccanico proporzionale alla corrente che attraversa il motore, possono essere comandati via PWM se dotati di opportuna scheda di controllo.

TODO: formule?

5. Scheduling

TODO

6. Programmazione

La struttura del software determina *quanto* siamo in grado di controllare il tempo di risposta. Dipende da:

- Requisiti applicativi
- Velocità del sistema di elaborazione (processore)

In generale, maggiore controllo implica maggiore complessità architetturale.

Architetture Software

- Round-Robin
- Round-Robin con interrupts
- Function Queue Scheduling
- Real-time Operating System

Round-Robin

```
void main() {
    while (true) {
        if (/* device 1 needs service */) {
            // take care of device 1
            // handle data to/from device 1
        }
        ...
        if (/* device n needs service */) {
            // take care of device n
            // handle data to/from device n
        }
    }
}
```

Nessun interrupt, polling continuo sulle periferiche, il codice è strutturato come un ciclo che: - Legge lo stato dei dispositivi I/O a turno - Effettua le corrispondenti elaborazioni in funzione dello stato

Vantaggi:

- Molto semplice
- Facilmente analizzabile (tempo = un iterazione del ciclo)

Svantaggi: - La risposta al cambio di stato di un dispositivo non può essere più rapida di una iterazione - La durata di un'iterazione è data dalla somma delle singole elaborazioni - Fragilità nel caso vengano aggiunti dispositivi

Round-Robin con interrupt

Ogni routine serve un dispositivo ed imposta un flag a true, il ciclo principale elabora i dati relativamente ai flag che sono true.

```
volatile bool fDevice_1 = false;
...
volatile bool fDevice_n = false;
void interrupt vHandleDevice_1() {
    /* take care of device 1 */;
    fDevice_1 = true;
}
...
void interrupt vHandleDevice_n() {
    /* take care of device n */;
    fDevice_n = true;
}

void main() {
    while (true) {
        if (fDevice_1) {
            fDevice_1 = false;
            /* handle data to/from device 1*/;
```

```
        }
        ...
        if (fDevice_n) {
            fDevice_n = false;
            /* handle data to/from device n*/;
        }
    }
}
```

Vantaggi:

- La risposta al cambio di stato di un dispositivo è più rapida, al livello di priorità dell'IRQ
- È indipendente dall'elaboraizone del task ciclico, dipende solo dagli interrupt a priorità maggiore.

Svantaggi:

- Routine di interrupt e ciclo principale devono sincronizzarsi sulle risorse condivise
- Meno predicibile il tempo di elaboraizone del task ciclico
- Tutto il codice del task ciclico funziona allo stesso livello di priorità:
  - nel caso peggiore, ogni blocco del task ciclico deve aspettare la somma dei tempi di tutti gli altri blocchi prima di servire il corrispondente interrupt
  - Si potrebbe spostare il codice nelle routine di interrupt, ma questo aumenta la latenza nella gestione degli altri interrupt
  - Poco adatta se uno dei blocchi ha computazione lunga

Function-queue-scheduling

Architettura basata sull'uso di una coda di puntatori a funzione, le routine di interrupt aggiungono alla coda un puntatore alla funzione che effetua il calcolo associato all'interrupt, il main estrae i puntatori ed invoca le rispettive funzioni.

Vantaggi:

- Usando una coda di priorità l'ordine di esecuzione delle funzioni riflette le priorità degli interrupt
- La massimo attesa nell'esecuzione della funzione a priorità massimo è il tempo di esecuzione della funzione che ha il massimo tempo di esecuzione

Svantaggi: - L'esecuzione delle funzioni a priorità più basse può essere rimandata indefinitivamente - Se la funzione con il massimo tempo di esecuzione è lenta il tempo di attesa per la funzione a priorità massima potrebbe risultare comunque eccessivo.

```
QUEUE qFunc;
void interrupt vHandleDevice_1() {
    /* take care of device 1 */;
    enqueue(&func_1, PRI_1);
}
...
void interrupt vHandleDevice_n() {
    /* take care of device n */;
    enqueue(&func_n, PRI_n);
}
```

```
void main() {
    while (true) {
        while (empty(&qFunc))
            ;
    void (*func)() = dequeue(&qFunc);
    func();
    }
}

void func_1() {
    /* handle data to/from device 1 */
}
...

void func_n() {
    /* handle data to/from device 1 */
}
```

Real-time

Sfrutta l'utilizzo di un SO realtime, le routine di interrupt effettuano i compiti più urgenti di gestione dei dispositivi, ad ognuno di esso è inoltre associato un task del SO che effettua la computazione associata all'interrupt, le routine di interrupt segnalano al task associato che c'è del lavoro da fare utilizzando le primitive di segnalazione fornite dal SO.

```
OS_SIGNAL *sDevice_1;
...
OS_SIGNAL *sDevice_n;
void interrupt vHandleDevice_1() {
    /* take care of device 1 */;
    OS_notify(sDevice_1);
}
...
void interrupt vHandleDevice_n() {
    /* take care of device n */;
    OS_notify(sDevice_n);
}
void main() {
    sDevice_1 = OS_new_signal();
    ...
    sDevice_n = OS_new_signal();
    OS_add_task(&task_1, ...);
    ...
    OS_add_task(&task_n, ...);
    OS_start();
}
void task_1() {
    while (true) {
```

```
        OS_wait(sDevice_1);
        /* handle data to/from device 1 */
    }
}

...
void task_n() {
    while (true) {
        OS_wait(sDevice_n);
        /* handle data to/from device n */
    }
}
```

Vantaggi:

- Non è necessario gestire esplicitamente lo scheduling delle computazioni: se ne occupa il sistema operativo
- Se il SO è *preemptive* può sospendere l'esecuzione di una computazione a bassa priorità per eseguirne una a priorità più alta. (attesa $= 0$)
- Cambiamenti ai task a priorità più bassa non influiscono sul response itme dei task a priorità più alta: il design è generalmente più stabile ai cambiamenti

Svantaggi:

- Il SO ha un su overhead: il miglioramento dei tempi di risposta è ottenutoa fronte di una riduzione del thoughput
- Occorre sincronizzare le computazioni (non sono più eseguite in modo *run-to-completion*)
- la licenza di SO real-time commerciale costa