Sistemi embedded, microcontrollori, PIC

Riferimenti:

- Ken Arnold, “Embedded Controller Hardware Design”, LLH Technology Publishing, 2001
- Roland Dubois, “Microprocessori, Computer Handbooks 662”, Gruppo Editoriale Jackson
- www.wikipedia.org

Autore: © 2008 Dott. Ing. Mario Buffardo
mistersnike@gmail.com

PARTE PRIMA: Controlli embedded: generalità e concetti fondamentali

Introduzione

Sebbene si dia per scontata la presenza di microprocessori nella nostra vita (basti pensare agli ormai diffusissimi PC), molti ignorano che l'impatto dei microcontrollori nelle abitudini quotidiane sia ancora maggiore: le cosiddette *applicazioni embedded* sono sistemi progettati per una sola applicazione specifica (spesso con una piattaforma hardware ad hoc) e diventano col tempo sempre più diffuse: calcolatrici, sportelli bancomat, cellulari (esclusi quelli di nuovissima generazione che sono dei veri e propri PC), stampanti, fotocopiatrici, forni a microonde, lavatrici…

Proprio per questo è necessario conoscere profondamente cosa significa progettare un sistema embedded, quali ne sono i limiti e quali i grandi vantaggi.

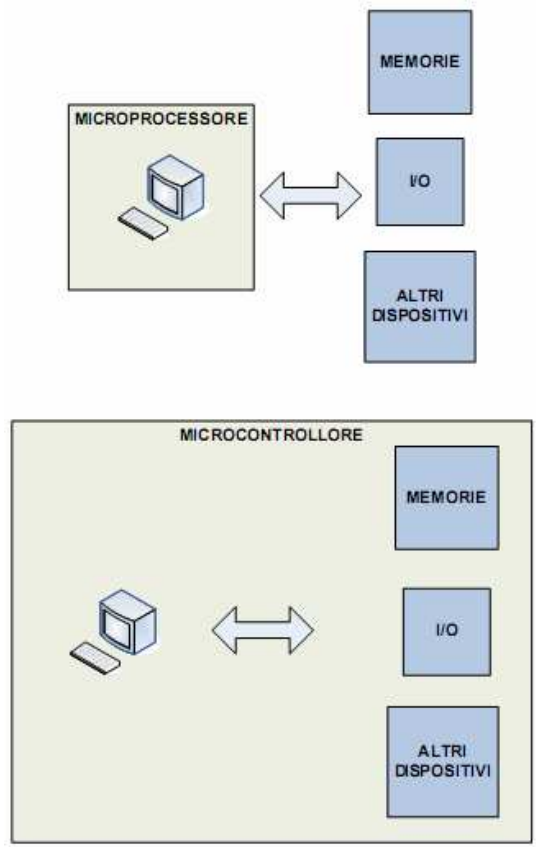
Questo lavoro ha l'obiettivo, per nulla facile, di aggiungere ulteriori elementi a questa discussione. L'autore spera di riuscirci, almeno in parte.

Microprocessori e microcontrollori

I *microprocessori* sono unità che contengono unicamente la Central Processing Unit (CPU). I *microcontrollori*, invece, hanno in aggiunta almeno anche una memoria interna (registri RAM, memorie ROM, EEPROM, flash…) e linee di I/O.

Ciò è dovuto al fatto che i microcontrollori sono usati per applicazioni dedicate e quindi hanno bisogno di essere più "autonomi" nel loro funzionamento. I microprocessori sono, invece, usati per raggiungere alte performance, dove sono richieste compatibilità e flessibilità. Infatti, avendo lo spazio del core occupato unicamente dalla CPU, è possibile realizzare architetture con alte velocità e un'elevata potenza di elaborazione. Tuttavia, proprio per questo, richiedono memorie esterne ed hardware aggiuntivo di I/O: è ciò che rende il progetto di sistemi basati su microprocessori più costoso. D'altra parte, avendo i microcontrollori integrati dispositivi timer, porte I/O, convertitori ed altro, risultano scarsamente modificabili e, quindi, dalle capacità ridotte in senso assoluto.

Applicazioni implementate o con microprocessori o con microcontrollori sono indicate con il termine generale di *microcomputer*. I microcomputer sono circuiti integrati in grado di elaborare informazioni. Essi esistono dal 1972 e la loro architettura, quindi, si ispira a quella dei grandi sistemi informatici. Essi, infatti, sono in grado di effettuare sia operazioni aritmetiche che logiche.



Un generico microcomputer è costituito essenzialmente da tre parti principali:

- *Unità di controllo*: decodifica le istruzioni prelevate dalla memoria di programma ed elabora i segnali di comando necessari all'esecuzione di un'istruzione.
- *Unità aritmetico-logica*: esegue le operazioni aritmetiche e logiche.
- *Registri*: alcuni sono accessibili dal programma dell'utente, altri no. Solo i primi interessano il programmatore e si dividono in: *registri dati* (consentono la memorizzazione temporanea delle informazioni, essendo utilizzati come RAM), *registri indirizzo* (sono i puntatori che contengono gli indirizzi di memoria), *Program Counter* (PC, contiene l'indirizzo della prossima istruzione da eseguire) e *registro di stato* (dalla cui lettura è possibile identificare lo stato del dispositivo in quel momento).

In definitiva, i microcomputer sono circuiti, la cui funzione è definita da un programma che si trova in memoria e che corrisponde a una sequenza di comandi. Questo *set di istruzioni* viene definito una volta per tutte dalla casa produttrice e non può essere modificato dall'utente, che lo può usare solo per scrivere il programma. Un set di istruzioni generico comprende le seguenti famiglie:

- Istruzioni di trasferimento e scambio;

- Istruzioni aritmetiche;
- Istruzioni logiche;
- Istruzioni di salto e ritorno al programma principale;
- Istruzioni di I/O;
- Istruzioni di controllo.

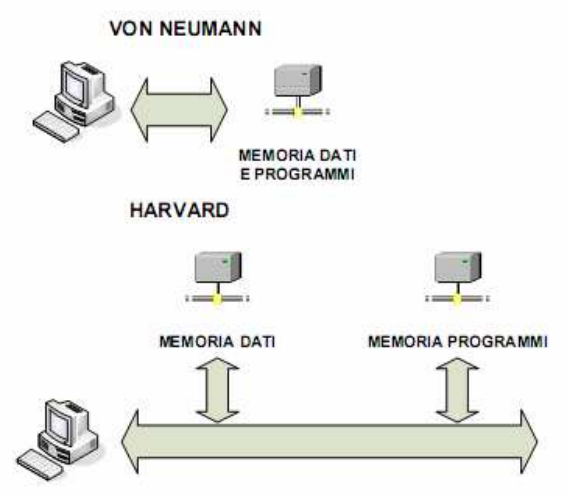
Architettura di von Neumann e architettura Harvard

La differenza tra microprocessori e microcontrollori è un riflesso della differente architettura interna tra i due dispositivi.

I microprocessori sono dispositivi basati su l'*architettura di von Neumann*, che è caratterizzata da una singola memoria sia per i dati che per i programmi, consentendo una maggiore flessibilità nell'allocazione della memoria.

I microcontrollori, invece, si basano su un'*architettura Harvard*, che ha memorie separate per i dati e per i programmi. In particolare, è usata una memoria ROM non volatile per memorizzare il programma da eseguire, mentre una memoria RAM volatile conserva le variabili di lavoro (che saranno, quindi, perse allo spegnimento del dispositivo).

Avere le due memorie separate consente di implementare il *parallelismo* durante il funzionamento del microcontrollore: le istruzioni possono essere indirizzate parallelamente ai dati. Purtroppo, molti microcontrollori hanno la CPU interna collegata alle memorie con un singolo bus per problemi realizzativi, quindi il parallelismo è limitato e resta solo il vantaggio di avere due memorie separate: una volatile (per i dati) e una non volatile (per le istruzioni da eseguire). Una ulteriore differenza dei microcontrollori rispetto ai microprocessori è che questi ultimi usano la RAM anche per memorizzare i programmi, giacché eseguono più programmi insieme e non un solo programma dedicato.



Bus

Ci sono tre tipi di bus che connettono la CPU agli altri componenti interni: bus dati, bus indirizzi e bus di controllo.

Il *bus dati* è quello usato per indirizzare i dati con cui il programma lavora. La larghezza di tale bus è il numero di bit che può essere trasferito nello stesso momento, in parallelo; tale parametro definisce la *word size* del microcontrollore. Molte architetture, per ottimizzare lo spazio, usano il data bus anche per indirizzare gli indirizzi.

Il *bus indirizzi* è usato per puntare alla memoria o alle linee di I/O.

Il *bus di controllo* ospita segnali che indicano il tipo di informazioni che viaggiano sul bus dati e determinano la destinazione di questi dati, in unione con le informazioni presenti congiuntamente sul bus indirizzi. I segnali su questo bus devono avere una tempificazione estremamente precisa e seguire protocolli prestabiliti.

Nell'architettura Harvard sarebbe ideale avere il bus dati e quello indirizzi separati, in modo da utilizzare a pieno il parallelismo interno: il primo comunicherebbe con la memoria volatile RAM (per i dati), il secondo con quella non volatile ROM (per le istruzioni).

Le connessioni tra CPU, memoria e linee di I/O sono in genere uno a uno sul bus. I quattro più comuni tipi di controlli della CPU sono:

- la CPU sta leggendo dati o istruzioni dalla memoria (*memory read*): il processore posiziona un indirizzo sul bus indirizzi e attiva il segnale di lettura della memoria (ciò provoca che la corrispondente locazione di memoria è messa sul bus dati);
- la CPU sta scrivendo dati nella memoria (*memory write*): il processore posiziona un indirizzo sul bus indirizzi e il dato da scrivere sul bus dati, dopo

di ché attiva il segnale di scrittura della memoria (ciò provoca che nella corrispondente locazione di memoria è caricato il contenuto del bus dati);

- la CPU sta leggendo dati da un dispositivo di I/O (*I/O read*);
- la CPU sta scrivendo dati su un dispositivo di I/O (*I/O write*).

Si userà la convenzione di definire la direzione di trasferimento rispetto la CPU, quindi i termini input e lettura saranno sinonimi, così come i termini output e scrittura.

Un componente cruciale per l'ottimizzazione dei bus è il *decodificatore*, che identifica i diversi stati che assume una parola binaria: ad esempio, un decodificatore a 8 bit consente il riconoscimento di 256 stati. Quindi, il trasferimento di un'informazione codificata in binario consente di diminuire il numero di collegamenti: ad esempio, non occorrono 256 per identificare un'informazione, ma solo 8.

Temporizzazione e diagrammi temporali

Si intende per *bus cycle* una singola transazione tra la memoria e la CPU. Lo scopo di un'analisi temporale è determinare la sequenza di eventi in ciascun bus cycle, in maniera da poter stabilire il tempo che ogni componente necessita per rispondere al cambiamento. Questi tempi devono poi essere paragonati ai tempi di risposta dei componenti (forniti dal costruttore), in modo da poter verificarne la compatibilità. Ad esempio, i fronti di discesa e di salita di un segnale non saranno mai netti, ma più o meno graduali: il tempo che il segnale impiega a portarsi a un livello alto o basso è un tempo di transizione, di cui il progettista deve tenere conto.

Le specifiche temporali più importanti per interfacciare i componenti tra di loro sono:

- *tempi di discesa e di salita;*
- *ritardi di risposta*, dovuto alla propagazione all'interno di un componente;
- *tempo di setup*;
- tempo necessario affinché un segnale arrivi al *valore desiderato*;
- ritardo necessario per abilitare o disabilitare il *tri-state*;
- *durata minima* di un impulso;
- *frequenza di clock*.

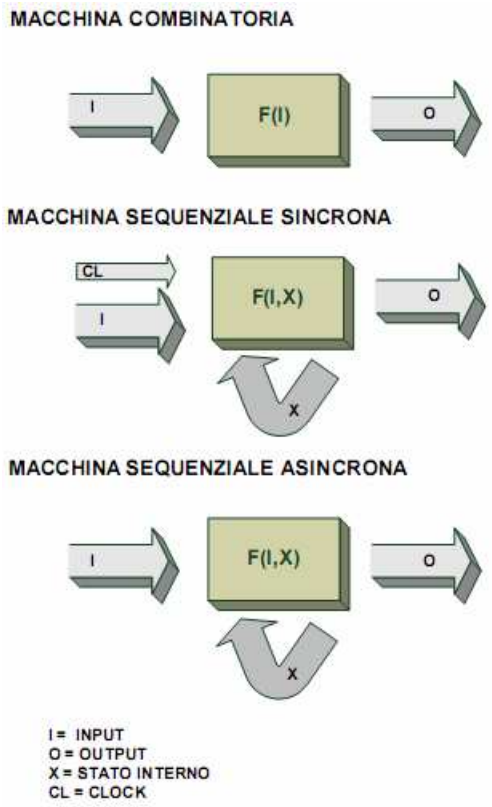
Logica combinatoria e logica sequenziale

La logica delle *macchine combinatorie* non ha memoria e le uscite che genera sono funzioni logiche degli ingressi presenti in quel momento, dopo un certo tempo di ritardo

(*ritardo di risposta*). Esempi di queste macchine sono le porte logiche (AND, NOT, OR…), buffer, invertitori, multiplexer, decodificatori.

La logica delle *macchine sequenziali*, invece, ha memoria. Ciò significa che le uscite sono funzione degli ingressi presenti e passati. Esempi di queste macchine sono flip-flop, registri di memoria, microprocessori, contatori.

Inoltre, esistono due tipi di logiche sequenziali: la *logica sincrona*, per cui c'è un cambiamento solo quando c'è la transizione di un segnale di clock, e la *logica asincrona*, che non usa alcun segnale di clock. Esempi di logica sincrona sono i contatori, i microprocessori, elementi di memoria; esempi di logica asincrona sono porte logiche e decodificatori. In un microcomputer sono, ovviamente, presenti componenti che usano sia una logica che l'altra.



Parametri temporali

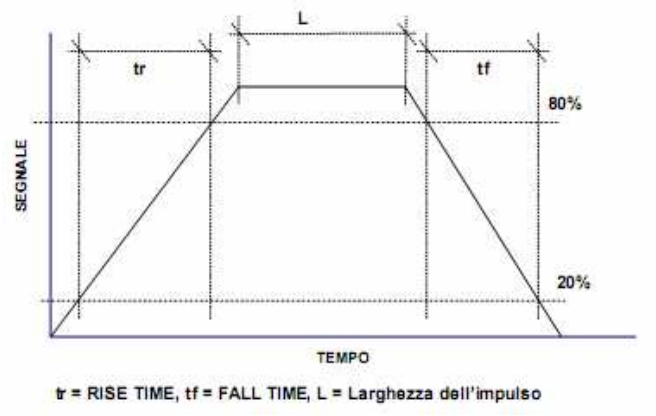
Per *tempo di salita* (*rise time*) di un segnale si intende il tempo richiesto a un segnale logico in tensione per portarsi dal 20% all'80% del suo valore massimo. Per *tempo di discesa* (*fall time*) di un segnale si intende il tempo richiesto a un segnale logico in tensione per portarsi dall'80% al 20% del suo valore massimo. Alcuni produttori danno questi tempi riferiti anche al 10% e al 90%.

Il *ritardo di propagazione* (*propagation delay*) è il tempo che un ingresso di un circuito richiede per provocare un cambiamento in uscita. Ogni dispositivo, anche i fili, esibiscono un ritardo di propagazione. Inoltre, in genere, i ritardi di propagazione per transizioni alto/basso (τ_{PHL}) sono più brevi di quelli relativi a transizioni basso/alto (τ_{PLH}). Questo ritardo "asimmetrico" è comune per la logica TTL, per uscite open-drain e open-collector, in quanto assorbono corrente meglio di quanto ne generino. Quindi, la capacità di carico all'uscita di questi circuiti si carica più lentamente quando questi devono fornire corrente (quando, cioè, devono potarsi a un livello logico alto), mentre si scarica più velocemente (quando si portano a un livello logico basso). In definitiva: $\tau_{PHL}<\tau_{PHL}$. Inoltre, è bene ricordare che questi tempi sono in genere misurati riferendosi al 50% delle transizioni: si inizia a misurare in corrispondenza del 50% della transizione in ingresso e si finisce di misurare al 50% della transizione in uscita.

Il *tempo di setup* (*setup time*) è il tempo che impiega un segnale a mantenere stabile il suo valore prima che intervenga una nuova variazione sul clock. Il *tempo di mantenimento* (*hold time*) è il tempo che impiega un segnale a mantenere stabile il suo valore dopo che è avvenuta una variazione sul clock. Per tempi minori di questi si ha incertezza sulla stabilità del segnale, che potrebbe oscillare tra diversi valori.

Quando più dispositivi possono pilotare la stessa linea, c'è la possibilità che due o più dispositivi tentino di pilotare quella linea in modo opposto (*contesa del bus*). Lo stato del bus non è prevedibile in questo tempo. In particolare, poiché alcuni dispositivi assorbono corrente (*sink*) e altri la erogano (*source*), in una situazione di contesa del bus si possono verificare picchi di corrente che possono essere anche molto pericolosi. Si tenta, quindi, di introdurre dei tempi morti tra l'azione di un dispositivo e quella di una altro dispositivo, in modo da evitare accavallamenti così pericolosi.

La *larghezza di un impulso* è l'intervallo temporale tra il suo fronte di salita e il suo fronte di discesa. Questo parametro è molto importante nei segnali di clock. Il *periodo* (T) di un segnale è la somma del tempo di salita, del tempo in cui il segnale è alto, del tempo di discesa e del tempo in cui il segnale è basso. La *frequenza* di un segnale (f=1/T) può essere data sia come limite inferiore che come limite superiore in fase di specifiche; ciò dipende dall'architettura usata e dai componenti in gioco.



Fan-out

Alle uscite dei circuiti sono collegati, il più delle volte, diversi carichi. Se un'uscita non è sufficiente a pilotare il carico totale, tale carico non garantisce il corretto funzionamento del circuito. Il *fan-out* è il numero di input che può essere pilotato da un'uscita.

Il fan-out dipende dalla sorgente che pilota e dalle caratteristiche del carico. In particolare sono importanti le caratteristiche DC: la minima corrente che può essere prodotta da un output e la massima corrente necessaria per pilotare un input (*DC fan-out*).

Valori che sono dati come specifica sull'output sono la I_{OLmin} (*minimum output low*, sink), minima corrente necessaria per avere uscita bassa, e la I_{OHmin} (*minimum output high*, source), minima corrente necessaria per avere uscita alta.

Valori che sono dati come specifica sull'input sono la I_{ILmax} (maximum input low), massima corrente necessaria per avere un ingresso basso, e la I_{IHmax} (maximum input high), massima corrente necessaria per avere un ingresso alto.

Un antro tipo di fan-out è la capacità di un'uscita di pilotare la capacità equivalente dei carichi; tale situazione è conosciuta come *AC fan-out*.

La capacità che un'uscita si trova a dovere pilotare è data dalla capacità equivalente del carico e, in parallelo, dalla capacità dei fili (queste due capacità, quindi, si sommano). I produttori di componenti usano una capacità di test (C_L, capacità di carico) per fornire le specifiche dei loro componenti, vale a dire come si comportano in quella particolare situazione di test, assunta come situazione "realistica" per il normale funzionamento del componente stesso (carico nominale). Tale valore di C_L è fornito, poi, nel datasheet del componente. Se quel componete è usato in una situazione in cui la capacità di carico che vede è maggiore di C_L, allora le specifiche (soprattutto quelle temporali del componente) non è garantito che siano rispettate: una capacità addizionale incrementa i tempi di salita e di discesa dei segnali sulla linea in questione. Molti componenti forniscono la loro capacità

in ingresso (C_{in}) in modo da permettere al progettista di verificarne la compatibilità nel circuito in cui sono inseriti. Le capacità di dispersione e quelle dei fili su un circuito sono stimate in genere nell'ordine di 1pF o 2pF per pollice, su una scheda standard (non progettata apposta, cioè, per alte velocità). Tipici valori per la capacità di carico sono invece dell'ordine di 50-150pF

Effetti sulla linea di trasmissione

Quando si usano alte velocità, i tempi di salita e discesa dei segnali sono generalmente dello stesso ordine della propagazione del segnale; gli effetti sulle linee di trasmissione diventano, quindi, significativi.

Se un segnale non è assorbito completamente quando arriva alla sua destinazione, esso può essere riflesso lungo il collegamento. A basse velocità questo effetto può essere ignorato, ma – ormai – alle attuali velocità dei processori è una omissione ingiustificabile.

Le tecniche di progettazione di circuiti ad alte velocità costituiscono un argomento molto complesso, al di fuori degli obiettivi del presente lavoro. Tuttavia è necessario affrontare il discorso almeno in modo qualitativo. Una *linea di trasmissione* è un conduttore abbastanza lungo tale che il segnale arrivi all'estremità di destinazione significativamente diverso da come è partito; ciò è dovuto al tempo di propagazione necessario per il segnale.

In questa sede, si assumerà che le interconnessioni tra componenti non siano così lunghe da provocare tale comportamento. Questa semplificazione non è sbagliata nel caso dei microcontroller, che hanno velocità mediamente inferiori a quelle dei microprocessori. Per questo è importante non usare componenti più veloci di quanto l'applicazione richiede; in tal modo non ci si imbatte nei problemi dovuti alle alte velocità.

Ground bounce

Il *ground bounce* (letteralmente, rimbalzo della massa) è un altro effetto che si verifica ad alte velocità, quando si crea un picco di corrente attraverso il pin di massa: uno o più pin di un chip cambiano livello logico contemporaneamente e scaricano tutti attraverso il pin di massa (per questo, tale fenomeno è più frequente sulle transizioni alto/basso).

Anche se l'induttanza parassita (L) del pin di massa può sembrare insignificante (dell'ordine dei nH), transizioni molto veloci (usuali nel cambio degli stati logici) possono causare tensioni elevate. Infatti, la tensione *v* sul pin del ground è data da

$$v = L\frac{di}{dt},$$

dove *L* è l'induttanza parassita del pin, *i* è la corrente di scarica che fluisce durante le transizioni, *t* è il tempo di scarica. Da una valutazione sommaria, *L* è dell'ordine dei nH, *i* dell'ordine degli A, *t* dell'ordine dei ns; *v* risulta, quindi, dell'ordine dei V, per niente trascurabili. Questo effetto è affrontato riducendo le lunghezze dei collegamenti (ad esempio, usando componenti SMD i pin hanno dimensioni ridotte), costruendo i componenti con più pin di massa (in modo da ripartire le correnti delle scariche) e usando piani di massa diffusi che disperdono più velocemente le correnti elevate. Da ciò si capisce che distribuire i piani di massa e quelli di alimentazione su un circuito stampato non è per niente una cosa immediata e semplice.

Anche l'alimentazione di un chip è soggetta a un fenomeno analogo al ground bounce.

Questi fenomeni possono provocare un'oscillazione del livello logico durante piccoli impulsi. Per tale motivo è fondamentale, in fase di progettazione, l'analisi dei margini di rumore.

Famiglie logiche

Le tre famiglie logiche più diffuse sono le seguenti:

- *TTL* (*Transistor-Transistor Logic* o *logica bipolare*);
- *NMOS* (*N-channel Metal Oxide Semicondutor field effect transistor logic*);
- *CMOS* (*Complementary n- and p-channel MOS logic*).

La famiglia TTL è la più diffusa, seguita dalla successiva NMOS e poi dalla CMOS.

In particolare, la famiglia CMOS usa bassi livelli di potenza e permette un alto livello di integrazione; per questo è la più diffusa. Integrati che usano logiche TTL o CMOS appartengono, ad esempio, alla famiglia 74xxxx.

La *famiglia TTL* richiede correnti di pilotaggio che vanno dalle centinaia di microampere alle decine di milliampere, in base alla versione che si considera. I livelli di tensione sono i seguenti:

- input: 0-0.8V per il livello basso, 2.4-5V per il livello alto;
- output: 0-0.4V per il livello basso, 2.8-5V per il livello alto.

I *margini di rumore*, per i circuiti TTL sono in genere dell'ordine dei 0.4V.

Se si interfacciano logiche diverse tra di loro, è importante assicurare – per un corretto funzionamento – la compatibilità e la coerenza dei livelli logici.

La logica TTL è in grado di assorbire correnti elevate ed è usata, per questo, per pilotare bus veloci e con grossi carichi. Con la TTL si usano *pull-up* sia attivi (generalmente un altro transistor) che passivi (ad esempio, un comune resistore). Il pull-up *attivo* (o totem-pole output) usa un transistor per generare corrente ed un altro per assorbirla. Il pull-up *passivo* usa un transistor per assorbire corrente e un resistore per generarla (nel senso che il resistore costituisce il by-pass verso l'alimentazione). Se non c'è nessun pull-up collegato al collettore del transistor ed esso è collegato direttamente al carico si parla, allora, di uscita *open collector*. La logica TTL è in genere usata per applicazioni economiche, molto veloci, in cui non ci sono specifiche stringenti circa la potenza da assorbire in gioco (elevata, in questo caso).

La *logica NMOS* fu introdotta per il suo consumo minore di potenza. Le correnti in ingresso sono praticamente nulle, a causa dell'alta resistenza in ingresso al gate. Tuttavia, sono logiche più lente a causa dell'elevato valore delle capacità di ingresso. Per il pull-up, il discorso è analogo a quello fatto per la famiglia TTL (qui, ad esempio, si parlerà di *open drain*).

La *logica CMOS* ha il vantaggio di richiedere meno potenza della stessa NMOS in regime stazionario. Quasi tutta la potenza richiesta, infatti, è dovuta alle transizioni logiche, che implicano la scarica dei condensatori interni. Quindi, è facile rendersi conto che l'assorbimento di potenza è funzione della frequenza di clock della porta. Alcuni processori ottimizzano questa architettura, rallentando la frequenza di clock quando non è richiesta in nessuna applicazione. Per questi motivi, la logica CMOS lavora a voltaggi ridotti, con alimentazioni di 3V o meno. Inoltre, tale famiglia logica ha margini di rumore maggiori e risulta, quindi, meno sensibile ai rumori rispetto alla famiglia TTL e NMOS. La famiglia CMOS usa alimentazioni di valore nominale pari a 3.3V.

Memorie

Esistono numerosi tipi di memorie. In questa sede ci si concentrerà su quelle più utili per la progettazione di sistemi embedded e su quelle con tecnologia a semiconduttore a stato solido e non su quelle magnetiche o ottiche.

La distinzione più importante da fare, circa le memorie, è su come queste siano collegate alla CPU. Le *memorie primarie* lo sono direttamente, tramite bus. Le *memorie secondarie* sono, invece, collegate alla memoria attraverso un controller intermedio.

La CPU, quindi, accede direttamente alla memoria primaria. Un esempio di questa memoria è la RAM. In realtà, il termine RAM indica solo una modalità di accesso e non un

tipo di memoria. Tuttavia, le memorie primarie hanno molto spesso un accesso RAM e contengono le istruzioni e i dati di cui la CPU necessita per lavorare. Giacché la CPU deve accedere a istruzioni e dati velocemente, la memoria primaria deve avere un tempo di accesso minore della secondaria, dell'ordine dei ns. Le memorie secondarie, invece, hanno in genere tempi di accesso dell'ordine dei ms.

I semiconduttori usati per le memorie primarie, che garantiscono queste alte prestazioni, sono sfortunatamente molto costosi e richiedono più potenza. Per questo motivo, le memorie primarie si adoperano solo per memorizzare istruzioni e dati, mentre le secondarie sono usate per conservare programmi o applicazioni che non si usano spesso, per cui non è molto dannoso avere tempi di attesa più lunghi per l'accesso. Questi tempi maggiori sono dovuti all'interposizione tra CPU e memoria di un controller dedicato. Questo è il caso di supporti ottici o magnetici, che – è bene ricordarlo – occupano spazi maggiori rispetto le memorie a semiconduttore.

Le memorie secondarie hanno il vantaggio di poter conservare una quantità maggiore di dati, essere più economiche e non volatili (conservano i dati anche dopo che è stata tolta l'alimentazione). Tutto questo, ovviamente, a scapito di una maggiore lentezza.

Nelle applicazioni embedded è buona regola usare memorie non volatili per conservare programmi e dati costanti e memorie volatili per conservare variabili e dati temporanei.

È possibile classificare le memorie in base alla modalità di accesso. Per le memorie RAM (*Random Access Memory*, memoria ad *accesso casuale*) il tempo di accesso è essenzialmente indipendente da dove il dato è memorizzato. Ogni locazione di memoria (1 bit) è identificata da un numero di riga e di colonna. Queste memorie si contrappongono alle memorie ad *accesso sequenziale*, in cui il tempo di accesso dipende dalla locazione in cui il dato è stato memorizzato. L'esempio tipico è quello dei nastri magnetici. Le memorie ad *accesso diretto*, invece, sono una sorta di combinazione tra le RAM e le sequenziali. Sono usate sui dischi, in cui le informazioni sono disposte in anelli concentrici (*track*). L'informazione è memorizzata sequenzialmente su ogni track, ma ci si può spostare da una track ad un'altra in maniera random. I temi di accesso sono, così, intermedi tra quelli delle RAM e quelli delle memorie sequenziali.

A questo punto è bene accennare ai modi di *indirizzamento in memoria*:

- *Indirizzamento esteso*: si usa direttamente l'indirizzo desiderato;

- *Indirizzamento indiretto tramite registri*: si usano puntatori che contengono l'indirizzo desiderato;
- *Indirizzamento indicizzato/relativo*: si usano puntatori indicizzati e paginazione della memoria;
- *Indirizzamento implicito*: usato per modificare il contenuto di un registro interno del microcontrollore;
- *Indirizzamento immediato*: nell'istruzione non appare l'indirizzo, ma direttamente il dato.

Un altro modo per classificare le memorie è considerando come le informazioni sono scritte in memoria. Ad esempio, le memorie *read/write* sono memorie che possono essere scritte tanto facilmente quanto possono essere lette dal processore.

Memorie read/write

Le RAM *statiche* (SRAM) sono caratterizzate dal fatto che ogni bit è memorizzato in un flip-flop. Tali flip-flop sono organizzati in righe e colonne e ognuno richiede quattro transistor per ogni bit. Quindi, le SRAM sono generalmente quattro volte meno dense delle DRAM, che usano un solo transistor per bit. Le SRAM sono volatili: mantengono il dato finché sono alimentate; le DRAM, invece, devono subire un refresh.

Le RAM *dinamiche* (DRAM) usano condensatori per memorizzare il dato: se il condensatore è scarico si ha uno "zero" memorizzato, se è carico un "uno". I condensatori, però, perdono la loro carica nel tempo (in qualche ms), perdendo anche l'informazione che memorizzano. C'è quindi bisogno di un continuo refresh, ricaricando il condensatore prima che sia scarico completamente. In realtà, la carica è immagazzinata grazie alla capacità interna di un MOSFET; in tal modo, è richiesto solo un transistor per ogni cella.

Memorie a sola lettura (Read Only Memory, ROM)

Le memorie a sola lettura ROM (*Read Only Memory*, ROM) sono una classe di memorie che non possono essere cancellate o modificate dal processore.

La *mask* ROM è una memoria che può essere programmata al momento della fabbricazione grazie a una maschera fotografica e non può più essere cambiata.

Le PROM sono *programmabili* (anche più di una volta) dall'utente, usando apparecchiature dedicate, genericamente indicate come PROM programmer o PROM burner (infatti la programmazione avviene bruciando opportuni fusibili). Molto spesso, ad esempio, si utilizza un bus seriale per inviare i dati da inserire in memoria.

Le EPROM, *erasable* PROM, sono ROM elettricamente programmabili con raggi ultravioletti (con apparecchiature chiamate UV EPROM), attraverso una finestra trasparente nel package. La cancellazione della memoria con questa tecnica è totale e non selettiva.

Le EPROM *flash* sono varianti delle EPROM standard, che non necessitano di raggi UV per la cancellazione: un'inversione di una tensione cancella tutta la memoria in un colpo.

Le EEPROM o E^2PROM, *electrically erasable* PROM, sono elettricamente cancellabili, un byte alla volta. Sono possibili dai 10000 ai 100000 cicli di scrittura.

Altri tipi di memoria

Le CMOS RAM o *non-volatile* RAM (NVRAM) sono RAM non volatili: una batteria conserva il dato anche quando è tolta l'alimentazione. In questo modo si ha una memoria non volatile con la stessa facilità di lettura e scrittura di una RAM. Ovviamente, ci sono limitazioni imposte dalla presenza di una batteria.

La RAM *ferro-elettrica* è una memoria a semiconduttore non volatile con un accesso veloce e cicli di scrittura illimitati. I costi di questa tecnologia sono superiori a quelli per altri tipi di memoria.

Lo standard JEDEC per il pin-out delle memorie

Le memorie RAM, ROM, EPROM e EEPROM hanno un pin-out che rispetta lo standard JEDEC (*Joint Electronic Device Engineering Committee*), in maniera da rendere facile l'interfacciamento di diverse memorie sullo stesso circuito (provvedendo, al limite, a modificare opportuni jumpers).

Questo standard prevede memorie a 24, 28 e 32 pin (JEDEC 24-, 28-, 32-*pin memory footprint*).

Considerazioni sull'organizzazione della memoria e sulle prestazioni

Le memorie sono organizzate in un fissato numero N di locazioni, ciascuna di un fissato numero M di bit. Ad esempio, una memoria di 128Kx8 ha 128000 locazioni di 8 bit ciascuna (128KByte).

Sono già stati analizzati i parametri temporali critici per i chip. In particolare, per le memorie è opportuno soffermarsi su alcuni parametri in particolare:

- il *cycle time* è il tempo necessario per leggere o scrivere una generica locazione della memoria;
- il *tempo di refresh* (per memorie dinamiche) è il tempo minimo che deve intercorrere tra un'operazione di lettura o di scrittura e la successiva.

In genere, per velocizzare l'accesso alla memoria è tipico usare, soprattutto in applicazioni spinte, *memorie cache*: sono buffer di memoria con accesso più veloce (ma con dimensioni minori rispetto alla memoria principale), che conservano i dati che si ritengono più usati. Esistono numerosi algoritmi per identificare i dati sottoposti a caching e per allineare la cache alla memoria principale (ovviamente, il riallineamento tra le due memorie è cruciale quando si scrive in cache, e non quando vi si legge).

Quantificare la velocità di una memoria è fondamentale in quanto il prezzo delle memorie è inversamente proporzionale alla velocità di accesso delle stesse.

Una delle caratteristiche in DC che riveste un ulteriore interesse è poi il *consumo energetico*, soprattutto in applicazioni basate su batteria.

Le *memorie asincrone* non richiedono segnali di clock per le loro operazioni e rendono disponibile il dato con un ritardo di un tempo di accesso (*tempo di propagazione interna*), dopo che le linee di controllo si sono stabilizzate. Le DRAM, invece, sono – ad esempio – *sincrone*, giacchè richiedono dei segnali (RAS e CAS) per caricare i dati. Le memorie asincrone sono, ovviamente, più facili da progettare e più facilmente compatibili con i bus dei processori.

Un altro elemento importante per le prestazioni di una memoria è la loro dimensione. Fissata la memoria da utilizzare, è possibile emulare una memoria più grande con la tecnica della *memoria virtuale*: in pratica, si basa su un continuo scambio tra la memoria principale e quella secondaria del disco. Questo complesso meccanismo di rilocazione continua (effettuabile solo con particolari software che lo gestiscono) fa apparire all'utilizzatore, quando ne ha necessità, una memoria più ampia. Quando si tenta – in pratica – di accedere a una locazione di memoria che non esiste, il software reindirizza ad un'altra porzione di memoria, dopo che l'hardware abbia preventivamente salvato il contenuto nella memoria (secondaria) del disco.

Sistemi complessi, come i PC, effettuano un controllo sui dati da leggere o scrivere in memoria (*error detection*); se c'è un errore si procede eventualmente con la ricostruzione del dato, secondo particolari algoritmi (*error correction*). Ad esempio, un controllo tipico è quello effettuato sul *bit di parità*: un bit di controllo è aggiunto alla stringa di bit di dati, tale bit è funzione del numero di bit alti contenuti nel dato. Il controllo di parità può essere

orizzontale se si riferisce al dato in esame, o *verticale* se si riferisce al controllo di una posizione in più dati in esame. Il solo controllo orizzontale, infatti, non rileva errori che corrompono bit a due alla volta. Un altro controllo ancora, anche questo tipico, è la *checksum*: si sommano tutte le word dei dati e si considerano i bit meno significativi di questa somma. Tale controllo non rileva errori dovuti alla memorizzazione di dati in un ordine non esatto. Per ovviare a tale inconveniente si usa la CRC (*cyclic redundancy code*): si effettua la XOR logica tra i diversi dati, nell'ordine in cui devono essere memorizzati.

Ci sono, poi, due tipi di errori che si verificano per la maggiore:

- *soft errors*: si verificano solo una volta, a causa di rumore sulle linee;
- *hard errors*: sono errori che si verificano sempre (ad esempio, impossibilità di scrivere o leggere una locazione di memoria), alla cui origine ci sono difetti di tipo hardware.

Infine, si ricorda che il trasferimento di dati tra una periferica e le memorie avviene in genere attraverso il microcontrollore; quando, però, le periferiche richiedono una maggiore velocità, esse possono accedere alla memoria direttamente, senza passare per il microcontrollore. Questo è quello che si chiama accesso diretto alla memoria (*Direct Memory Access*, DMA).

Lettura dalla memoria

La CPU effettua le seguenti operazioni per *leggere un dato dalla memoria*:

- selezione della locazione da leggere, scrivendo l'indirizzo sull'address bus;
- opportuna impostazione delle linee di controllo, a seconda del protocollo e del tipo di memoria;
- linea di controllo per il read attivata, in modo che la memoria risponda scrivendo il dato sul data bus;
- la memoria scrive il dato sul data bus;
- disattivazione delle linee di controllo e indirizzo.

Scrittura in memoria

La CPU effettua le seguenti operazioni per *scrivere un dato nella memoria*:

- selezione della locazione da scrivere, scrivendo l'indirizzo sull'address bus;
- opportuna impostazione delle linee di controllo, a seconda del protocollo e del tipo di memoria;

- occupazione del data bus con il dato da scrivere;
- linea di controllo per il write attivata, in modo che la memoria scrivi il dato sul data bus all'indirizzo dell'address bus;
- disattivazione delle linee di controllo e indirizzo.

Introduzione alle logiche programmabili

Gli ASIC (*Application Specific Integrated Circuits*) sono *circuiti integrati* (IC) progettati o programmati per soddisfare le specifiche esigenze del sistema in cui i chip sono inseriti. Sono, quindi, differenti da altri IC general purpose, che hanno costi più alti essendo rivolti ad un numero maggioritario di applicazioni con funzionalità del tutto generiche.

I *gate array* sono dispositivi generici costituiti da un insieme di porte logiche, i cui collegamenti esterni sono stabiliti dall'utente al momento dell'utilizzo.

I dispositivi con logica programmabile (*Programmable Logic Devices*, PLD) sono una famiglia di dispositivi costruiti tutti nello stesso modo, che possono essere personalizzati usando processi speciali di programmazione, il più delle volte in EPROM. Una EPROM può implementare funzioni logiche arbitrarie usando le linee indirizzo come input e le linee dati come output: la tabella di verità è programmata nella EPROM in modo che ad ogni combinazione di ingressi corrisponda in maniera univoca un'uscita. In genere ci si riferisce a queste logiche con il nome di *Programmable Logic Array* (PLA) o *Programmable Array Logic* (PAL, marchio registrato dalla AMD Inc.). Alcuni PLD sono programmabili bruciando alcuni fusibili interni (fuse-link) per aprire o meno determinate connessioni. Tale approccio, ovviamente, è irreversibile. Altre tecnologie (come le EPROM, le EEPROM o le RAM) rendono, invece, i PLD riutilizzabili.

I *Field Programmable Gate Array* (FPGA) sono una via di mezzo tra i gate array e i PLD: sono una array di porte logiche programmabili via software dall'utente.

Interfacciamento con altri dispositivi

Spesso i pin di un microcontrollore possono essere collegati direttamente ad altri dispositivi (come gli switch dei tasti o i led). In altri casi, invece, c'è bisogno di un circuito di interfacciamento, per adattare i livelli di tensioni e le correnti in gioco.

In genere, gli stessi pin sono usati sia come input che come output, la circuiteria interna dei microcontrollori provvede a queste operazioni.

Le interfacce più comuni per la comunicazione con altri dispositivi sono: porte parallele, porte seriali, timer, contatori, ADC, DAC, …

Interruzioni

Con il *polling* si ha un controllo ciclico – in istanti di tempo ben definiti – dei registri dell'interfaccia, dalla cui interpretazione è possibile capire se l'interfaccia stessa deve essere servita o meno. Tale tecnica non è il massimo in termini di prestazioni e velocità. Una valida alternativa, qualora l'hardware del microcontrollore le supporta, sono le *interruzioni*.

Le *interruzioni* (o *eccezioni*) possono essere sia hardware che software.

Le interruzioni *software* sono delle istruzioni usate per chiamare delle subroutine, allocate in aree stabilite della memoria. Sono sostanzialmente degli eventi sincroni: accadono nella maniera stabilita dal sistema operativo, sempre allo stesso punto del programma.

Le interruzioni più comuni sono quelle *hardware*, legate ad un evento fisico, che causa l'esecuzione di una specifica subroutine (ISR, *Interrupt Sub Routine*). Esse sono degli eventi asincroni, che possono verificarsi in qualsiasi momento dell'esecuzione del programma. Molte applicazioni, infatti, sono guidate da eventi (nessuna azione avviene se non come conseguenza di un determinato evento). Un sistema che risponde agli eventi è detto sistema *real-time*.

Quando avviene un'interruzione (e se esse sono abilitate), il programma ferma la normale esecuzione (salvando in un'opportuna zona della memoria il program counter) e va all'indirizzo indicato da un particolare buffer, l'*interrupt vector table*. A tale indirizzo è allocata la ISR da eseguire (per ogni interruzione si può prevedere una particolare ISR). Alla fine della ISR, il processore ritorna alla normale esecuzione, recuperando il valore del *program counter*. In presenza di più interrupt è possibile fissare una *priorità*, in modo da decidere quale interruzione deve essere servita.

Metodi di costruzione

I controlli embedded sono implementati, per lo più, su PCB (*Printed Circuit Board*). Il PCB è costruito di materiale isolante, laminato con un sottile foglio di rame. Strati multipli di materiale isolante e di rame vanno, poi, a costituire PCB *multi-layer* (a cui si può accedere con opportuni fori sulla struttura, detti *jumper*).

Nel progetto del layout del PCB ci sono innumerevoli elementi da considerare. Mentre per circuiti a bassa velocità gli effetti parassiti possono essere ignorati, ciò non è vero per circuiti con velocità spinte, in cui problemi di EMC (dovuti a segnali ad alte

frequenze) sono in agguato. Per essi bisogna prestare molto attenzione alle forme delle piste, al loro spessore, al posizionamento dei componenti,… Tutti questi elementi, infatti, possono modificare le prestazione della scheda.

Power planes e ground planes

Quando è possibile, è buona norma usare PCB con almeno 4 layer, di cui uno dedicato all'alimentazione (power, V_{cc}) e uno di massa (ground): *power planes* e *ground planes*. Un vantaggio di questo tipo di soluzione è che ad alte frequenze c'è un disaccoppiamento capacitivo sui disturbi dell'alimentazione, riducendo – quindi – il rumore sull'alimentazione stessa. I power planes, inoltre, riducono sia le emissioni irradiate EMI che la suscettibilità del circuito verso i disturbi EMI che si ricevono: essi, cioè, agiscono come uno schermo (*shield*) rispetto ai rumori EMI.

Problemi di grounding

Avere un'unica massa su un circuito è una problematica molto importante, più complicata di quanto non sembri. Si possono verificare problemi che vanno da interferenze sui segnali a eventi molto più pericolosi.

Tali situazioni possono essere dovute a diverse motivazioni:

- loop di massa (*ground loops*) che introducono un'induttanza o una resistenza eccessiva sul circuito di massa;
- mancanza di masse differenti in un sistema o isolamento insufficiente tra le masse: ground di terra, ground di sicurezza, ground analogico, ground digitale;
- piani di massa non ideali, che implicano il fatto che la corrente che circola in un circuito induce tensione in un altro circuito.

Le soluzioni dipendono dal tipo di problema e dalle frequenze in gioco. In genere è bene usare piani differenti per alimentazione e ground o usare un'unica massa, onde evitare isolamento insufficiente tra più masse.

Compatibilità elettromagnetica (EMC)

Le *interferenze elettromagnetiche* (EMI) diventano un problema sempre più importante, soprattutto negli ultimi anni, in cui numerosi dispositivi elettronici emettono segnali nello stesso range di frequenze (basti pensare ai cellulari). Per questo motivo, gli Enti preposti hanno definito limiti per le emissioni, che dipendono – ovviamente – anche

dalla tipologia dei dispositivi. Lo spirito di queste normative è che i dispositivi devono avere dei limiti per le loro emissioni (disturbi irradiati) e devono avere una bassa suscettibilità rispetto alle emissioni provenienti da altri dispositivi, che possono indurre correnti nei conduttori del circuito (disturbi condotti).

Per risolvere i problemi EMC, il primo passo è identificare la sorgente di rumore, il percorso interessato e la destinazione finale, in cui il problema si manifesta. Una volta identificati questi tre punti, il progettista può decidere se eliminare la sorgente del rumore, interrompere il percorso usando shield o tecniche simili, ridurre la sensibilità del circuito circa gli effetti.

Non esistono algoritmi generali, ma solo molta esperienza da acquisire e molto tempo da dedicare ai test.

Scarica elettrostatica (ElectroStatic Discharge, ESD)

Scariche elettrostatiche – di migliaia di Volt – si manifestano sui dispositivi elettronici, in genere, quando qualcuno (magari dopo aver camminato in un posto con bassa umidità) tocca poi un dispositivo. Infatti, solitamente il posto dove più spesso si hanno scariche sono le tastiere di interfaccia.

Le scariche elettrostatiche possono provocare danni anche molto seri.

Le ESD possono essere ridotte con tecniche di shielding e di grounding. In genere sono gli stessi provvedimenti che si prendono per problemi EMC.

Tolleranza all'errore

È possibile dividere gli errori (*fault*) in hard (irreversibili) e soft (reversibili).

I sistemi di autodiagnosi non sempre, ovviamente, rilevano tutti gli errori. Ad esempio è possibile usare sistemi di verifica per i dati memorizzati su ROM, come ad esempio: il controllo di parità, la checksum o il controllo di ridondanza ciclico (Cyclic Redundancy Check, CRC).

Sistemi di sviluppo hardware

Esistono due categorie di sistemi di sviluppo:

- i sistemi per l'*analisi passiva* permettono all'operatore di monitorare lo stato del sistema (oscilloscopi, analizzatori di stati logici, probe logici);
- i sistemi per l'*analisi attiva*, invece, consentono all'operatore di interagire con il sistema mentre esso sta in running, a volte anche di fare cambiamenti run

time (ad esempio sistemi In-Circuit Emulator, ICE, che si sostituiscono alla CPU, emulandola).

Il sistema che si sta analizzando, solitamente, è chiamato *target device*, il computer che si utilizza per sviluppare, editare compilare, assemblare e scaricare il codice nel target è chiamato sistema *host*.

Sistemi di sviluppo software

Esistono tre famiglie di software che servono per lo sviluppo e i test delle applicazioni:

- *Translator*: assembler, compiler. linker, interpreter (gli assemblatori traducono linguaggio assembler o esadecimale in linguaggio macchina binario, i compilatori effettuano la stessa operazione con linguaggi di alto livello);
- *Debugging*: software/firmware monitor, ICE;
- *Utility*: PROM programming, misura delle performance, istogrammi della frequenza di esecuzione.

Analisi termica del progetto

La temperatura di un componente a semiconduttore è un parametro critico durante l'operatività del componente stesso. Infatti, l'affidabilità di un componente diminuisce esponenzialmente con l'aumento della temperatura.

Con semplici circuiti in analogica, per fortuna, è possibile calcolare facilmente la *temperatura operativa di un componente*. Sussistono, infatti, le seguenti equivalenze:

- Temperatura ⇔ Tensione;
- Potenza dissipata ⇔ Corrente;
- Resistenza termica ⇔ Resistenza elettrica.

Quindi, è possibile calcolare:

AUMENTO DI TEMPERATURA (°C) = POTENZA (W) X RESISTENZA TERMICA (°C/W).

La resistenza termica di più componenti sistemati uno sull'altro è la somma delle singole resistenze (come se fossero resistenze elettriche in serie).

I valori tipici di resistenza termica sono ampiamente documentati e tabellati.

Misura della performance di un processore

Gli indici più comuni usati per il paragone di più processori sono:

- IPS (*Instructions Per Second*) o MIPS (*Millions Instructions Per Second*) o BIPS (*Billions Instructions Per Second*). È un indice poco utile: infatti dice fa frequenza con cui l'istruzione più veloce è eseguita su una macchina. Spesso questa istruzione è la NOP (No OPeration): 500MIPS può, quindi, significare anche che il processore non fa niente 500 milioni di volte per secondo.
- OPS (*Operations Per Second*) o MOPS o BOPS sono i tempi di esecuzione dell'istruzione, basati su un mix di differenti istruzioni. I FLOPS (MegaFLOPS, GigaFLOPS, …) sono simili, ma si riferiscono ad operazioni in virgola mobile, in modo da quantificare le operazioni più stressanti per un processore. Il problema con questi indici è che il valore risultante dipende dal mix di istruzioni scelto per la misura.
- *Benchmarks* sono piccoli programmi che vengono eseguiti per testare funzionalità critiche: ad esempio, testare la stessa funzionalità su processori diversi. Ovviamente, quella che si misura non è solo la performance del processore, ma quella di tutto il sistema.

Nella scelta del processore da usare per una produzione, tuttavia, diversi sono gli elementi da considerare: non solo la velocità di tale processore ma anche – e soprattutto – la disponibilità di un secondo fornitore.

Altre interfacce

Ci sono molti sensori ed attuatori che possono facilmente essere connessi al microcontroller. I sensori più comuni misurano temperatura, pressione, posizione, velocità, campo magnetico,… Ormai il numero dei sensori e la loro qualità, nonché il prezzo, ha raggiunto livelli enormemente competitivi. Lo stesso dicasi per attuatori o per altri dispositivi, come ad esempio i led.

Conversione analogica dei segnali

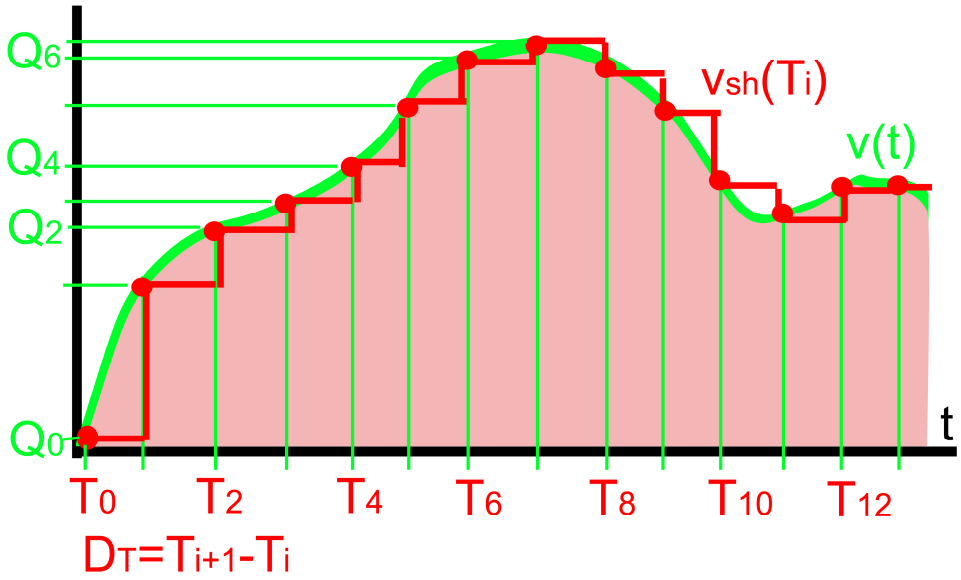
In molte applicazioni embedded, i processori hanno a che fare con segnali che non sono digitali per loro natura: i segnali provenienti dal mondo reale sono analogici (variano nel tempo in maniera continua). Essi devono essere discretizzati, in modo da poter essere processati dalle CPU. Si usano per questo i *convertitori analogico-digitale* (ADC o A/D). Prima della conversione in digitale vera e propria (*quantizzazione*), c'è bisogno che i

campioni del segnale siano mantenuti per un certo periodo di tempo (*Sample and Hold*, SAH o S/H). Molti microcontroller hanno integrato almeno un A/D.

Dualmente, esistono dispositivi che realizzano la conversione opposta: da *digitale in analogico* (DAC o D/A), ogni volta che la CPU deve inviare un segnale al mondo esterno. In genere, i microprocessori non hanno un D/A interno; invece hanno un modulo PWM (*Pulse Width Modulated*), che può essere usato al posto del D/A convenzionale. Il segnale PWM è in genere un treno di impulsi con duty cycle variabile da 0% a 100%: la media di questo segnale (ottenibile con un filtro in uscita) costituisce il valore analogico desiderato.

Un'altra conversione tipica è la *conversione dei livelli logici*. Ad esempio, ciò è richiesto per l'I/O seriale su standard RS232: i livelli di +24V e -24V devono essere convertiti in logica TTL, 0V e 5V.

Analogamente, la conversione dei livelli logici è richiesta per interfacciare logiche TTL e ECL.



ESEMPIO DI QUANTIZZAZIONE. IL SEGNALE TEMPO-CONTINUO V(T) È PRIMA CAMPIONATO. $V_{SH}(T_I)$ È IL SEGNALE OTTENUTO DA UN CAMPIONAMENTO IN AVANTI. TALE CAMPIONAMENTO È A PASSO UNIFORME (D_T È COSTANTE), MA NON SEMPRE È COSÌ, DIPENDE – INFATTI – DAL SEGNALE DI PARTENZA. L'INSIEME DISCRETO $\{Q_0, Q_1, \ldots, Q_N\}$ È IL SEGNALE QUANTIZZATO.

Speciali interfacce seriali sincrone proprietarie

Si ricorda che, mentre nei protocolli seriali asincroni la trasmissione dei dati avviene carattere per carattere con opportuni caratteri di start e di stop, nei protocolli seriali sincroni i dati vengono trasmessi come un continuo flusso di bit coerenti con un segnale a frequenza fissata, detto *clock*.

Particolari interfacce seriali sincrone proprietarie nascono dalla necessità di usare un numero ridotto di pin. Esistono, ad esempio, due noti standard proprietari che rispondono a questi requisiti:

- il bus seriale della Philips, I^2C (*Inter-Integrated Circuit bus*);
- il bus seriale della National, *MicroWire*.

Il bus I^2C è più flessibile perchè consente a più periferiche di condividere lo stesso bus. È anche più complesso, giacchè gestisce più periferiche e più master. Il MicroWire richiede pin in più se ci sono più periferiche sullo stesso bus.

Suggerimenti per la progettazione dell'hardware

La realizzazione degli schematici è una parte importante in ogni progetto hardware. Alcuni suggerimenti di massima sono:

- schematici in multi-pages possono essere strutturati gerarchicamente;
- un foglio al top-level dovrebbe mostrare le interconnessioni tra gli altri fogli;
- in ogni disegno gli input dovrebbero stare sulla sinistra del foglio e gli output sulla destra del foglio.

È buona norma produrre una lista (*check list*) del progetto che includa:

- tensioni di alimentazione presenti (con rispettive tolleranze);
- componenti passivi utilizzati;
- componenti attivi utilizzati (con le rispettive tensioni di alimentazione e con l'assorbimento);
- altri chip utilizzati (con le rispettive tensioni di alimentazione e con l'assorbimento).

Molti CAD forniscono spreadsheet con queste informazioni. È importante verificare che la somma dell'assorbimento nel caso peggiore sia compatibile (non superiore) con l'alimentazione di tutto il sistema.

Un punto cruciale nella progettazione è verificare la compatibilità dei livelli di tensione. Ciò è importante per due motivi:

- per una corretta interpretazione dei livelli logici (0 e 1);
- per evitare di danneggiare gli input dei diversi dispositivi coinvolti.

La capacità di tollerare le tensioni in ingresso senza danni da parte di un chip è detta *absolute maximum rating*, i livelli logici sono definiti nelle caratteristiche in DC. In particolare il parametro V_{IH} indica la massima tensione in ingresso sostenibile dal chip. In quest'ottica è importante analizzare i margini di rumore durante il progetto dello schema.

Altro elemento da monitorare è il fan-out in DC, vale a dire la capacità che un chip ha di alimentare altri chip ad esso collegati. Esso è strettamente legato alla corrente che il chip può erogare. In tale ottica, le resistenze di pull-up devono essere scelte in modo tale da minimizzare i consumi.

Inoltre, è bene ricordare che le specifiche di timing dei dispositivi digitali si riferiscono a una particolare condizione di carico. Se il carico capacitivo sulle uscite è maggiore di quello indicato nelle specifiche, allora le specifiche sul timing, indicate nel datasheet del dispositivo, non saranno più valide. I ritardi dipendono, infatti, dalla corrente di scarica della capacità di carico. Un carico generico, infatti, può essere – in prima approssimazione – modellato come una resistenza opportuna (che determina la corrente che il chip deve fornire a regime per il corretto funzionamento) e da un'opportuna capacità (che modella i ritardi e le dinamiche dei segnali). Quindi, sarebbe utile testare le specifiche sempre nel *worst case*. Un ritardo tipico da avere su un PCB standard è tra i 100 e i 200 picosecondi per pollice. Questi ritardi, che possono essere visti come distorsioni in frequenza, hanno effetti molto seri specialmente se interessano i segnali di clock.

Nella stessa ottica, i piani di alimentazione e massa sono importanti perché evitano le impedenze di connessione, fornendo – invece – un ottimo disaccoppiamento. I collegamenti verso massa dovrebbero essere i più corti possibili per prevenire il ground bounce. I pin di alimentazione e massa devono essere forniti di un *condensatore di bypass* a frequenza bassa, pari alla frequenza più altra di interesse del circuito (bisogna riferirci al rise time del clock, non alla frequenza del clock stesso). Un valore tipico per i condensatori di bypass è 0.1 microfarad; per circuiti con velocità più spinte (rise time non superiori a 5ns), invece, conviene usare anche 0.01 o 0.001 microfarad (o qualche centinaio di picofarad). I condensatori ceramici (multilayer) lavorano meglio ad alte frequenze rispetto ad altre tecnologie (a filo, a tantalio o elettrolitici). I PCB, è buona norma, dovrebbero poi avere anche condensatori a tantalio o elettrolitici, per proteggersi dai picchi di corrente a medie frequenze (bypass dei picchi di corrente). Inoltre, per i collegamenti verso l'alimentazione e la massa, è importante evitare piste che creano loop (*power e ground loop*), i quali innescano fenomeni elettromagnetici indesiderati.

L'alimentazione dell'elettronica analogica deve essere – poi – separata da quella dell'elettronica digitale: le alte frequenze della digitale potrebbero indurre rumore sull'analogica. Poi anche la massa dell'elettronica digitale deve essere separata dalla massa dell'analogica e dall'alimentazione dell'analogica (soprattutto per evitare problemi alle *radio frequenze*, RF). Inoltre, i piani per l'alimentazione analogica non devono

sovrapporsi ai piani digitali. La massa digitale e quella analogica devono essere connesse solo in un punto (usando, magari, un jumper), in genere vicino al convertitore analogico/digitale. I segnali analogici possono essere separati da quelli digitali usando alte impedenze. Comunemente, poi, si usano diodi Schottky per il *clamping* (limitazione) dei segnali, in modo da non avere livelli logici più alti dei desiderati.

Ovviamente, è buona norma separare l'elettronica di potenza da quella di segnale e riportare la logica del sistema, dopo un reset, ad uno stato noto.

Conclusioni

Finisce qui questa sintetica, ma si spera esaustiva, panoramica sui sistemi embedded.

Nella seconda parte, di prossima pubblicazione, di analizzerà un caso pratico: il *microcontrollore PIC* della *Microchip Inc*.