Progettazione Elettronica

Alberto Tibaldi

16 ottobre 2010

Indice

Capitolo 1

Lezione 01

1.1 Protocollo One-Wire

Un protocollo spesso utilizzato in ambito di elettronica è il cosiddetto "protocollo one-wire", circuitalmente implementabile mediante il seguente stratagemma:

In un protocollo di trasmissione digitale si possono trasmettere sostanzialmente due segnali: un '1' logico e uno '0' logico. Il concetto di questo tipo di protocolli è il seguente: utilizzando il segnale logico affermato sia come informazione sia come alimentazione per gli inverter, è possibile risparmiare in termini di alimentazione. Quando "non si parla", la capacità si carica; il fatto che questa si carichi permette di mantenere l'alimentazione anche per gli istanti di tempo in cui si trasmettono segnali negati, ossia '0' logico. Naturalmente, se non vi sono lunghe sequenze di zeri, questo sistema potrà funzionare, esclusivamente su di un filo.

1.2 Materiali Piezoelettrici

Su materiali in genere fortemente anisotropici quali quarzi o ceramiche, atomi caricati positivamente e negativamente sono posti in una struttura irregolare, che tende a generare, a livello macroscopico, effetti abbastanza particolari. Un classico esempio è quello dei materiali piezoelettrici, materiali che, di fatto, rappresentano un'unione tra il dominio meccanico e quello elettrico. In altre parole, a causa delle grosse anisotropie dei materiali, una sollecitazione meccanica del materiale comporta il generarsi di una carica elettrica; dualmente, la presenza di una differenza di potenziale ai capi del circuito provoca variazioni nelle dimensioni meccaniche del materiale.

Materiali di questo tipo genericamente vengono classificati come isolanti, ma in questi casi, il fatto che venga su di essi sollecitata la suddetta forza, provoca la generazione di cariche. Si può dire che la piezoelettricità sia modellizzabile mediante un coefficiente che lega la carica Q presente sulla superficie del materiale con la forza F impressa su di esso:

Si è detto che:

$$Q \propto F \Longrightarrow V = \frac{Q}{C} \propto F$$

Si può tradurre la forza in differenza di potenziale V, considerando la capacità del dispositivo piezoelettrico, modellizzabile come un generatore di corrente in parallelo a una capacità (data dalle proprietà dielettriche del materiale) e una resistenza che modellizza i fenomeni di perdite, che dissipano immediatamente la carica nel materiale.

La prima applicazione che si può dunque pensare per blocchi basati su questo materiale è la trasduzione meccanico-elettrica. Applicazioni in cui si possono trovare quarzi sono ad esempio trasduttori di rugosità di un materiale, o microfoni (poichè le onde sonore tendono a deformare il quarzo per via delle oscillazioni meccaniche che inducono, oscillazioni che poi verranno amplificate mediante stadi di amplificazione). Tendenzialmente, tensioni di $50 \div 100$ V producono spostamenti di qualche μm. Altra applicazione è quella di "cicalino", "buzzer": dal momento che l'applicazione di una tensione provoca variazioni nelle dimensioni fisiche, riuscendo a farlo "suonare" si può ottenere quello che è il buzzer presente in ciascun computer (specialmente fisso):

In questo modo si può usare quindi la deformazione meccanica del quarzo per far muovere una membrana.

Di fatto, un piezoelettrico è modellizzabile mediante una capacità non lineare. Per pilotarlo, esistono diverse strategie, diversi trucchi comunemente utilizzati, per farlo ad esempio oscillare o farlo comportare in diverse maniere:

- Un esempio potrebbe essere quello di usare un circuito logico sequenziale instabile, costituito anche di semplici porte quali NOT logici:

  Polarizzando, il segnale continua a invertirsi, ottenendo di fatto un'onda quadra che pilota il piezoelettrico. Volendo è possibile variare questo fenomeno introducendo, al posto delle porte logiche, amplificatori di potenza.

- Variante dello schema precedente è il cosiddetto "pilotaggio a ponte" o a "H":

In questo caso si parla di piezoelettrici, ma un pilotaggio di questo genere potrebbe essere utilizzato con carichi di tipo più svariato, quali ad esempio motori elettrici.

- Avendo una matrice, un blocco in grado di trasdurre in maniera idonea le variazioni del quarzo, è possibile fare ancora qualcosa di interessante, quale un oscillatore. Utilizzando uno schema concettualmente di questo tipo:

  Si può pensare che esista la seguente equivalenza:

  Utilizzando dunque una struttura di questo tipo, è possibile ottenere un oscillatore basato sul quarzo. Sostanzialmente, si può vedere che la frequenza di oscillazione di un quarzo è abbastanza elevata:

  $$f_{quarzo} \sim 2^{15}\,\text{Hz}$$

  Utilizzando un blocco di amplificazione o una porta logica o elementi più complessi quali una FPGA, è possibile ottenere un oscillatore digitale al quarzo:

  Al fine di far lavorare il quarzo in questa maniera, tendenzialmente lo si af lavorare al centro della caratteristica, ottenendo una frequenza di lavoro del tipo:

  $$f = \frac{1}{2\pi\sqrt{LC_1 \oplus C_2}}$$

  Dove C_1 e C_2 sono capacità introdotte dall'utente. In realtà è possibile sfruttare gli effetti capacitivi parassiti contenuti all'interno del quarzo, ottenendo ottimi risultati.

L'uso come "oscillatore" è probabilmente uno dei più interessanti per il quarzo. Esso è così frequentemente utilizzato che spesso si trovano dei bundle pre-montati sotto il nome di XCO. Di frequente, nei sistemi a microprocessore, si tende addirittura a inserire il quarzo assieme alla CPU, in modo da sfruttare direttamente un segnale di temporizzazione prodotto all'interno del sistema per la gestione delle attività di elaborazione.

1.3 Pilotaggio di schermi a cristalli liquidi (LCD)

Uno schermo a cristalli liquidi (LCD : Liquid Crystal Display) è sostanzialmente un sistema di visione basato su di una matrice di elementi allineati

tra due elettrodi trasparenti, e due filtri polarizzanti (al fine di visualizzare correttamente l'immagine prodotta dai cristalli). Questi "liquidi" hanno una sorta di "testa polarizzata": introdotte in un condensatore, rappresentato dai due elettrodi, è possibile sfruttare la polarizzazione di queste "teste" per orientare in diverse maniere ciascuno di questi liquidi, ottenendo immagini. Il campo elettrico riesce infatti a muovere e orientare questi cristalli.

Prima di introdurre il campo questi cristalli in fase liquida si organizzano in modo da essere complessivamente neutri, ma in maniera casuale; questo tipo di fenomeno provoca l'introduzione dell'opacità lattiginosa, "grigia", propria degli LCD. Introducendo questi famosi elettrodi e applicando il campo è possibile orientare e produrre luce, dal momento che il campo elettrico risulta essere perpendicolare a esse. Il fatto che gli elettrodi siano trasparenti deve essere ovvio: ciò che si intende ottenere è un dispositivo che produce immagini, dunque emette luce; utilizzando una metallizzazione tra i cristalli si potrebbe comunque introdurre una polarizzazione, ma non si potrebbe visualizzare la luce. Per questo motivo si utilizza di solito un conduttore trasparente basato sull'indio-stagno (ITO).

Per ciascuna cella di cristallo liquido, dunque, è sostanzialmente necessario un pilotaggio mediante questo tipo di "condensatori trasparenti"; questi condensatori, a causa della loro costituzione (materiale + dielettrico "particolare"), risulterebbero essere, oltretutto, implementanti capacità di tipo "non lineare". L'idea, nella fattispecie, potrebbe essere la seguente:

Introducendo una topologia a righe e colonne si può ottenere un array bidimensionale; pilotando con varie tensioni i cristalli si riesce dunque ad introdurre un colore.

Quali caratteristiche aggiuntive deve avere questa tensione di polarizzazione? Bisogna prestare molta attenzione perchè, introducendo una normale tensione continua (DC), lo schermo non funzionerà di certo (e anzi probabilmente si danneggerà): gli elettrodi che si utilizzano per la polarizzazione sono di fatti immersi nel liquido; questo liquido tuttavia presenta proprietà particolari, tra cui proprietà elettrolitiche: nel caso si utilizzi una DC, si avrebbe un fenomeno di elettrolisi che provocherebbe un'elettromigrazione nel liquido. Ciò che bisogna fare dunque, per polarizzare e dunque pilotare uno schermo LCD, è utilizzare rigorosamente tensioni/correnti alternate, a valor medio nullo! Anche solo una componente continua nella AC porterebbe al fenomeno di elettromigrazione, danneggiando il dispositivo. Continuando a invertire la polarizzazione, in AC, si continua a "girare" e "girare" le molecole, in modo da non far migrare gli elettroni ed evitare i danni.

Si noti che esistono problemi di banda passante: come qualsiasi sistema, se il segnale di eccitazione è troppo rapido si ha il rischio di non far proprio

muovere le molecole, nel caso la loro costante di tempo sia troppo alta rispetto all'eccitazione. Bisogna per questo motivo documentarsi sulle informazioni dei cristalli, in modo da polarizzarli opportunamente.

1.4 Condensatori di bypass

Una cosa che potrebbe capitare, in circuiti stampati, è l'avere a che fare con piste estremamente lunghe. Ciascuna pista tuttavia potrebbe presentare fenomeni di diverso tipo, uno su tutti di tipo induttivo:

Il fatto che si abbia a che fare dunque con piste di questo genere potrebbe provocare fenomeni negativi, quali perdite o lentezza nel raggiungimento dell'alimentazione. Ciò che viene fatto, dunque, è introdurre delle capacità (mediante condensatori) in modo da ottenere un effetto "serbatoio": dal momento che l'induttanza insita nella pista rallenta la propagazione dell'alimentazione, introdurre una capacità molto vicina al punto in cui l'alimentazione è necessaria fornisce, nel caso di lentezza di propagazione, un "rifornimento temporaneo" continuo di carica. Ciò che si fa dunque è dimensionare una piccola capacità, in modo da un lato da introdurre una piccola caduta di tensione, dall'altro comunque la possibilità di compensare perdite dovute alle piste induttive. Supponendo di aver a che fare con circuiti alimentati con 5 V, si possono tollerare tranquillamente perdite di tensione di 0,2 V; volendo compensare una caduta di carica dell'ordine di 1 μC, dunque, è possibile, mediante la definizione di capacità (lineare), ottenere qualcosa del tipo:

$$C = \frac{Q}{V} = \frac{1\,\mu\mathrm{C}}{0.2\,\mathrm{V}} = 50\,\mu\mathrm{F}$$

Naturalmente, buona cosa potrebbe essere quella di sovradimensionare queste capacità, introducendo la possibilità di compensare perdite anche superiori alle massime. Queste capacità, realizzate mediante condensatori, vengono comunemente dette "condensatori di bypass".

Capitolo 2

Lezione 02

2.1 Isolamento galvanico

Uno dei problemi principali che si hanno utilizzando processori o altra elettronica se montata in sistemi esterni di vario tipo è il fatto che si possa avere una suscettibilità di diversa entità ai problemi esterni. La carica elettrostatica nei blocchi esterni nella fattispecie potrebbe provocare problemi di vario tipo, sotto questo punto di vista. Ciò che si può fare per evitare il propagarsi di questi problemi è dunque utilizzare sistemi di protezione per le scariche elettrostatiche, onde evitare fenomeni di folgorazione. Una delle idee alla base di questi meccanismi di protezione potrebbe essere l'isolamento galvanico: volendo trasferire informazione ad esempio a un calcolatore senza dover prestare troppe attenzioni, un'idea potrebbe essere quella di disaccoppiare i riferimenti dei circuiti esterni e di quelli interni, ossia separare le masse e il potenziale di riferimento del processore.

2.1.1 Trasformatore Induttivo

Esistono diversi modi per ottenere un isolamento di tipo galvanico; il più classico senza dubbio è il trasformatore induttivo:

Se vi fosse una tensione di disturbo tra i due, ossia se uno tra primario e secondario per esempio si trovasse ad alte tensioni a cause di variazioni dei riferimenti, non è detto che anche l'altro subisca qualcosa di simile, dal momento che i riferimenti sono tra loro disaccoppiati. Primario e secondario possono tra loro "flottare", dal momento che tra primario e secondario non esiste una continuità metallica che possa in qualche modo collegare i due riferimenti. Il trasformatore sostanzialmente è un filtro passa-alto: le tensioni di polarizzazione, principali cause di scariche in sistemi non isolati, non possono passare, dal momento che un trasformatore di questo tipo isola la

continua, la filtra. Ciò che può passare tuttavia potrebbe essere una componente alternata: la capacità modellizza infatti il fatto che tensioni a frequenze sufficientemente elevate (si spera siano segnali) possano tranquillamente passare per il trasformatore. Dal momento che spesso tuttavia i disturbi possono essere di tipo impulsivo, almeno una buona parte delle armoniche che costituirà l'impulso (segnale a larghissimo contenuto spettrale) potrà passare, quindi fare danni. Ciò che si può fare è migliorare il trasformatore mediante uno schermo elettrostatico tra i due avvolgimenti, ottenendo un circuito equivalente di questo tipo:

Le due capacità permettono sostanzialmente di trasformare il sistema filtrante da passa alto a passa banda: un segnale statico come prima non può passare, ma neanche un segnale ad alta frequenza, a causa dell'effetto passabasso introdotto dallo schermo; tutto ciò che potrà passare di un segnale, dunque, è sostanzialmente il contributo armonico contenuto tra f_{min} e f_{max}.

Ci si potrebbe a questo punto porre una domanda: e se si intendesse monitorare una continua? Beh, nei segnali digitali, di fatto, si dovrebbe fare qualcosa del genere: un segnale digitale è un segnale tendenzialmente costante, che può assumere due valori. Dal momento che il segnale statico non può passare, questo sistema potrebbe essere inadatto. Ciò che si potrebbe fare è modulare la parte continua mediante una portante a frequenza idonea, in modo da rendere "passabile" il segnale e poterlo quindi interpretare "dall'altra parte".

Esempio di uso di trasformatori induttivi: RJ45

Il tipo di spinotto RJ45 (tipico delle connessioni via ethernet), è costituita da tre trasformatori piccolissimi, trasformatori che trasportano informazioni relative a tre elementi:

- Dati da trasmettere da una parte all'altra della connessione;
- Segnale di clock, ossia di temporizzazione per il protocollo;
- Informazioni sulle collisioni avvenute: il protocollo ethernet richiede controlli di questo tipo.

Mediante queste tre informazioni è possibile ottenere un dialogo tra monte e valle; se la linea da una parte o da un'altra subisse cambiamenti di riferimento per i motivi più svariati, il fatto di comunicare mediante questi microtrasformatori permette di isolare galvanicamente tutto.

Si noti che, per ora, non è possibile inviare una trasmissione con molti zeri o molti uni consecutivi: introdurre sempre lo stesso dato rende il segnale

sostanzialmente prossimo ad una continua, dunque verrà filtrato; ciò che si può fare è tuttavia utilizzare un modulatore, al fine di evitare questo problema. Si usa di solito una modulazione di tipo "Manchester", ossia una modulazione digitale di fase, al fine di fare ciò. Volendo andare ad alta frequenza, serve inoltre un trasformatore a banda larga, ma quindi sostanzialmente utilizzare delle linee di trasmissione, linee accoppiate, ma quindi trasformatori di tipo elettromagnetico (sostanzialmente a parametri distribuiti). Mediante trasformatori normali si può ottenere una banda dai 20 ai 100 kHz, mediante trasformatori elettromagnetici dai 100 ai 100 MHz.

Oltre a modulare, con un trasformatore galvanico, è necessario demodulare, cosa che richiede dunque l'introduzione di una certa circuiteria anche a valle del sistema; dopo aver demodulato, si deve poter rendere il dato "interpretabile" dal calcolatore in uscita dal pin "a valle"; per far ciò si utilizzano convertitori basati su comparatori di soglia, mediante amplificatori operazionali; un'alternativa potrebbe essere quella di utilizzare inoltre operazionali dotati di uscita open collector: considerando come due possibili stati la saturazione o l'interdizione, si può introdurre il significato di "0" o "1" logico.

Si noti che anche parlando di questi tipi di sistemi, di conversione, è necessario introdurre protezioni. Una protezione spesso utilizzata è la seguente (circuito di clamping o di clipping):

Questo sistema è molto utilizzato e funziona, ma ha un grosso difetto: il fatto che si introducano dei diodi nel sistema, diodi tipicamente a giunzione, restringe la banda passante del sistema, rendendolo inevitabilmente lento. Ciò che si dovrebbe inoltre fare è "proteggere la protezione": essa non è assoluta, per quanto comunque ottima.

Una protezione un po' più "moderna" della precedente potrebbe essere la seguente:

Ciò che si può fare è mettere un circuito come questo a valle del trasformatore: se si hanno picchi di tensione il fusibile si fonde, danneggiandosi ma non facendo danneggiare tutto il resto: meglio sostituire un fusibile che l'intera CPU. Il dispositivo introdotto è detto "TVSD" (Transient Voltage Suppression Diode). Esso si può trovare anche sotto il nome "tranzorb", o "transil", a seconda della ditta che lo distribuisce; si tratta in sostanza di una coppia di diodi zener in antiserie: quando essi vanno in rottura elettrostatica, si crea un corto circuito verso il potenziale di riferimento del sistema, che annulla dunque le sovratensioni/sovracorrenti. Questo dispositivo ha una transcaratteristica di questo tipo:

Questo tipo di schema è molto interessante nel caso si abbiano scariche poco frequenti ma ingenti; nel caso le scariche abbiano basse entità ma siano molto frequenti, la protezione con il circuito di clipping è più indicata.

Cenni al disaccoppiamento capacitivo

Una tecnica alternativa per l'isolamento galvanico è basata sull'introduzione di due capacità nel circuito, ottenendo sostanzialmente tra "primario" e "secondario" qualcosa del genere:

Questo tipo di disaccoppiamento, basato sulle due capacità, permette di non far passare le frequenze basse sui segnali di "andata" e "ritorno" dal circuito. Un sistema di questo tipo è riscontrabile in alcuni amplificatori operazionali della Analog Device, spesso rappresentati mediante il seguente simbolo:

In questo modo si riesce a disaccoppiare in maniera ottimale le due parti dell'amplificatore operazionale.

Altro modo per disaccoppiare potrebbe essere quello di usare dei relè; non si approfondisce l'argomento.

2.2 Optoisolamento

Come abbiamo detto l'isolamento galvanico è assolutamente d'obbligo. Ciò che abbiamo introdotto, tuttavia, non è molto "semplice". Si può fare di meglio, sotto il punto di vista della semplicità almeno? Beh, modulare, abbiamo capito, è brutto: serve una sorgente elettromagnetica che permetta di essere isolata in modo diverso, per esempio "otticamente". Un "modulatore naturale", ossia un dispositivo in grado "naturalmente" di passare da una continua ai THz, è il LED: accendendo/pilotando un led è possibile generare infatti un segnale elettromagnetico a frequenza elevatissima. Ciò che si può fare dunque è utilizzare, come isolamento galvanico, una semplice guida d'onda ottica: mediante uno spazio libero e delle lenti, si ottiene qualcosa del genere.

Ciò che si fa in sostanza è utilizzare una coppia LED + Fotodiodo di questo tipo:

Quando nel diodo LED scorrono pochi microampere esso sostanzialmente è spento, dunque nella guida ottica vi è il buio; se si invia nel LED un flusso di cariche, si arriva a correnti anche dell'ordine dei mA, dunque si ottiene una luminosità intensa. Il fotodiodo "catturerà" fotoni emessi dall'optoisolatore e li trasformerà in tensione/corrente, ottenendo da un lato trasmissione di segnale, dall'altro isolamento ottico e dunque galvanico (poich il collegamento tra i due segnali non è elettrico bensì esclusivamente ottico). In uscita dal fotodiodo spesso si introducono comparatori di soglia, in modo da discretizzare l'uscita. Questo è sostanzialmente il principio di funzionamento degli optoisolatori.

Come si fanno in pratica? Beh, da un lato esistono moduli con optoisolamento incorporato, mediante fotodiodi+comparatori integrati; l'optoisolamento intrinsicamente contiene modulazione, dunque non è di solito necessario introdurre ulteriori sistemi di modulazione.

Un esempio di funzionamento per qualcosa, ad esempio un lanciamissili, potrebbe essere il seguente:

A seconda della resistenza introdotta si può ottenere oggetti più "veloci" o più "sensibili": la velocità e la sensibilità sono sostanzialmente dipendenti dal punto di funzionamento in cui si utilizzano questi blocchi:

Mediante la resistenza si può scegliere quanta corrente/tensione introdurre al LED, ma dunque sostanzialmente quanto renderlo luminoso. Introducendo una resistenza bassa, si fa correre molta corrente, ottenendo dunque una reattività molto elevata, dal momento che il diodo è molto acceso; avendolo molto acceso tuttavia la sensibilità è bassa, cosa positiva sotto il punto di vista della reiezione dei disturbi in ingresso, dal momento che distinguere le variazioni per quanto concerne un LED molto acceso è più complicato; al contrario, con una resistenza elevata, si può ottenere un LED che si accenderà molto di meno, rendendo dunque sensibili le variazioni di luminosità in funzione dei disturbi, ma molto più lento il circuito. La resistenza di pull-up per il LED si può dimensionare in questo modo:

$$R_{pu,1} \Longrightarrow \frac{V_{AL} - V_{MIN}}{R_1} \leq I_{max}$$

Dati dal datasheet i vari dati della formula si può ricavare semplicemente R_1.

Per quanto riguarda invece il fotodiodo, come si vede dallo schema esso viene sostanzialmente polarizzato inversamente, dunque la transcaratteristica da considerare sarà fondamentalmente ribaltata rispetto a quella appena vista:

Mediante la resistenza si può scegliere quanta corrente/tensione introdurre al fotodiodo, ma dunque sostanzialmente quanto amplificare la percezione della luce in ingresso, e la relativa trasformazione in corrente. Introducendo una resistenza bassa, si fa correre molta corrente, ottenendo dunque una reattività molto elevata, dal momento che il diodo è molto acceso; avendolo molto acceso tuttavia la sensibilità è bassa, dal momento che distinguere le variazioni per quanto concerne un fotodiodo molto acceso è più complicato; al contrario, con una resistenza elevata, si può ottenere un fotodiodo che si accenderà molto di meno, rendendo dunque sensibili le variazioni di luminosità, ma molto più lento il circuito. La resistenza di pull-up per il fotodiodo si può dimensionare in questo modo:

$$10\,\mathrm{k\Omega} \leq R_{pu,2} \leq 100\,\mathrm{k\Omega}$$

Se la tensione varia troppo lentamente (in dipendenza del punto di lavoro scelto) allora non è neanche lontanamente prossima all'essere digitale; per questo motivo è necessario in uscita utilizzare un Trigger di Schmitt, in modo da avere condizioni di scatto brusche con isteresi.

Modificando la lunghezza della guida d'onda sarà possibile ottenere diverse tensioni:

Per quanto riguarda a questo punto l'interfaccia con il motore, si deve introdurre un'uscita in grado di interfacciare correttamente l'optoisolamento con il motore per il lanciamissili. In sostanza, considerando il trigger di Schmitt come pilotato da una logica standard (sui 3,3 V), si può fare qualcosa del genere:

Si ricordi che correnti troppo elevate sono sempre svantaggiose: correnti troppo elevate potrebbero in questo caso bruciare i contatti. Pilotando il circuito mediante un transistore, e il carico mediante un relè, è possibile fare ciò. Per pilotare il relè è necessario un traslatore di livello, in grado di trasformare un livello di tensione in un altro (i relè di solito si pilotano a tensioni alte, come 12 V). O si mette un MOS/BJT di potenza, in grado da fare da open collector e pilotare il carico, o si utilizzano interruttori elettronici di potenza, come gli SCR, i TRIAC, gli IGBT:

Questi sono interruttori non-meccanici, dunque possono commutare anche a frequenze molto più elevate, dal momento che non si hanno poli dettati dai limiti delle latenze meccaniche. Realizzando un demodulatore pilotato da un fotodiodo, si può causare un cambio di corrente nel fotodiodo e accendere il TRIAC ad esempio, ottenendo un "optorelè"! In questo modo, ottenendo la traslazione di livello, anche con una normale logica da 3,3 volt è possibile da un lato trasformare il segnale digitale in segnale discreto di comando, e pilotare un dispositivo di potenza.

Queste tecniche possono trovare applicazioni nei più svariati campi, quali la rilevazione di variazioni ottiche: spesso vengono da un lato usati come "fine corsa" nei sistemi meccanici, ma spesso vengono usati ad esempio come sensori di posizione:

Quando il fotodiodo non è più alimentato significa che c'è qualcosa in mezzo al supporto propagativo ottico. Utilizzando una batteria di LED/fotodiodi, si riesce a costruire un sistema di rilevamento per la presenza di oggetti in un dato spazio. Volendo usare per problemi di questo tipo, assolutamente scorrelati dall'isolamento galvanico, la presenza di metalli in un liquido per esempio, è sufficiente utilizzare un rilevatore Hall e un piccolo magnete, sfruttando il principio della magnetoresistenza e così trovando qualcosa.

2.3 Microprocessori e Sistemi Digitali

Normalmente un microprocessore viene fornito in schede in cui sono contenuti, assieme ad esso, alcuni altri blocchi:

Vi è un bus al quale son collegate tutte le strutture con cui il microprocessore deve interagire: memoria (nel caso di architettura Harvard, la memoria dati e quella codice sono separate), interfacce per periferiche di vario genere. Solitamente, il blocco di memoria considerato separatamente dal microprocessore contiene memoria dati, ossia programmi o dati che potrebbero essere eseguiti, una volta portati, per ordine del processore, nella memoria codice. Il bus, generalmente, trasporta tre tipi di dati:

- Indirizzi;
- Dati;
- Segnali di controllo.

Un sistema elettronico digitale completo potrebbe avere questa topologia:

Dato a disposizione un sistema del genere, bisogna tener conto di alcune cose:

- Controllare che ci sia memoria sufficiente per eseguire l'algoritmo e il programma (utilizzando margini larghi per la realizzazione: le implementazioni in linguaggio macchina potrebbero nascondere sorprese sotto questo punto di vista);
- Controllare che il microprocessore sia appropriato per le operazioni da svolgere (in termini di parallelismo dei dati da gestire: 8, 16, 32 bit di parallelismo a seconda dei casi);
- Guardare cosa c'è nelle periferiche standard a disposizione.

Le periferiche standard sono periferiche normalmente introdotte assieme a microprocessore e memoria al fine di eseguire del codice. Potrebbero non essere presenti tuttavia periferiche standard idonee allo svolgimento di particolari operazioni; in questo caso, un'idea potrebbe essere quella di simulare, mediante una descrizione VHDL, la periferica in questione, ottenendo dunque un comportamento più o meno equivalente del sistema finale.

Alcune altre osservazioni: sempre tenersi larghi, come già detto, sulla memoria ROM, per il codice; l'architettura Harvard è decisamente preferibile, al fine di evitare false interpretazioni. Dai dati, si manda il codice che si intende eseguire al microprocessore, il quale a sua volta lo carica su di

una memoria ROM, ma generalmente programmabile (dunque Flash o EEPROM). Mantenendo in zone diverse le memorie il program counter (PC) può incrementarsi tranquillamente, dunque il processore "fetchare" l'istruzione $(n+1)$-esima, ancora mentre esegue la n-esima, dal momento che non vi è trasporto di codice sul bus di dati: è possibile effettuare contemporaneamente i preparativi per la futura operazione e al contempo lavorare sul bus, dal momento che la memoria codice è collegata direttamente al processore.

Se poi la memoria di dato (non direttamente collegata al processore) è una SRAM, è pure possibile mandare in ibernazione il sistema elettronico: fermando il clock e abbassando la tensione, i dati elaborati dal processore fino ad un certo istante vengono mantenuti, il sistema viene congelato su di uno stato, dunque si può riprendere in seguito, mediante un qualche evento (in seguito descritto), l'elaborazione, esattamente da dove si è rimasti.

Molte schede hanno un oscillatore interno, in modo da poter generare un segnale di temporizzazione all'interno del sistema elettronico stesso. Spegnendo mediante comandi questo clock, la macchina a stati realizzata dal processore si ferma, dunque abbassando le tensioni è possibile fermare lo stato del circuito. Il cristallo al quarzo alla base dell'oscillatore in uso (generalmente, un oscillatore di Pierce) viene fatto oscillare alla minima frequenza che può generare (nel caso da noi analizzato, 32 kHz): una volta che si manda il sistema in ibernazione, in questo modo, esso ha un consumo ridotto. Quando il sistema è in uso normale, il cristallo continua a lavorare a questa frequenza, ma un PLL fa aumentare la frequenza:

Utilizzando una portante di 2,4 GHz, si riesce a far aumentare la frequenza dell'oscillatore, in modo da aumentare la frequenza di clock e dunque temporizzare il circuito con un segnale molto più veloce, facendolo operare molto più rapidamente.

In un sistema digitale, al fine di garantire un corretto funzionamento, è necessario seguire i seguenti passi:

1. Alimentare il circuito;
2. Temporizzare il circuito;
3. Tenere in reset il sistema finchè la frequenza di PLL non si è stabilizzata al valore corretto di funzionamento;
4. Fare un conteggio di prova, dunque "sganciare" il processore che dovrà lavorare correttamente (power-on reset);
5. Controllare che le informazioni siano presenti e corrette, aggiornare il sistema e caricare i condensatori di controllo.

Tra gli altri sotto-sistemi presenti generalmente in un sistema digitale del tipo descritto ce n'è anche di altri generi (watchdog, hard-reset, e altri meccanismi di power-on reset): è infatti necessario mantenere "fermo" il processore, affinchè non parta ad effettuare le elaborazioni da stati errati, producendo dunque risultati errati. Si aspetta per un certo numero di clock in uno stato di reset, dopo di che l'alimentatore riceve un segnale di "power good", segnale che informa l'alimentazione che il PLL converge a una certa frequenza. Lo stesso circuito (generalmente un comparatore) che controlla e invia il power good può generare un "power fail", ossia un segnale in cui si ordina di salvare tutte le informazioni salvabili su di una ROM, dunque su di una memoria non volatile, per poi "uccidere" il sistema.

2.3.1 Introduzione al Motorola MC9S12E128

Il processore che verrà descritto e utilizzato per il resto delle lezioni sarà un HCS12, nella fattispecie il MC9S12E128 della Motorola (ora Freescale). Si tratta di un processore a 16 bit, con 112 piedini di I/O. Esso è programmabile mediante l'introduzione di dati da un computer a una memoria flash, con un'interfaccia USB.

Una cosa che può essere molto importante è l'interfaccia elettrica tra un componente generico e il microprocessore in questione. Questa può essere fatta sostanzialmente in due maniere:

- Mediante software, introducendo dati e programmi in memoria che simulino una certa periferica, come già detto;
- Collegando periferiche vere e proprie al processore, una volta capito dove e come mettere qualcosa sull'uscita.

Uno degli elementi più interessanti del processore è il controllore degli interrupt: per qualsiasi tipo di operazione sarà sempre necessario effettuare diverse routine, sostanzialmente riconducibili però a due tipi:

- Routine legate ad eventi periodici, dunque in cui si campiona qualcosa ad intervalli regolari, in cui la frequenza tuttavia potrebbe non essere dettata dal clock (processi ripetitivi);
- Routine legate ad eventi aperiodici, dunque non prevedibili e non ripetitivi.

Ciò che è possibile è che sia necessario interrompere il PC (program counter) e far partire una procedura di interrupt, al fine di dare priorità

a un evento piuttosto che a un altro. Esistono porte sempre aperte e pronte che, anche a processore spento, sono in grado di ricevere segnali in grado di avviare una procedura di interrupt ed eventualmente risvegliare il processore e fargli eseguire qualcosa. Una cosa importante è essere in grado di scrivere processore per svolgere le varie funzioni dell'interrupt: se la macchina era ibernata in hardware, una routine può dunque svegliarla! Ciò sarà interessante in seguito.

Capitolo 3

Lezione 03

3.1 Aspetti del microprocessore MC9S12E128

3.1.1 Cenni sull'interrupt

Un parametro molto importante per quanto riguarda il processore è la cosiddetta "interrupt latency" (tempo di latenza dell'interrupt): da quando si invia un segnale per richiedere un interrupt a quando il processore si "sveglia" e lo esegue, vi è una certa latenza, parametro molto importante. Esistono diverse tecniche per la gestione dell'interrupt:

- MMC : Controllore mappatura di memoria: si tratta tutto sostanzialmente come memorie, accedendo dunque a registri, periferiche e altro sempre come si accede ad indirizzi di memoria. Attribuendo alle periferiche ad esempio indirizzi poco interessanti in quanto non utilizzati, si può utilizzare una mappatura a memoria per le suddette periferiche;
- BDM : Modulo di bus presente : è come avere un processo in parallelo al programma eseguito che osserva un registro di memoria. Esso permette di sostituire lavori più complicati ad esempio in fase di debug, quali osservare con l'oscilloscopio i segnali sulla scheda su porte di servizio.
- Bus Multiplexato (Multiplexed external bus interface): si legge la memoria e si scrive un'istruzione. Si avrebbe una struttura di questo genere:

  Ciascuna linea ha parallelismo 32, e si prevede di utilizzare quattro linee: una per il program counter, verso la memoria di codice; idem tra memoria di codice e istruction register; per la gestione di dato inoltre servono altri due registri e altre due linee da 32 bit: il Memory Address Register (MAR), ossia quello su cui scrive l'indirizzo per poi ritrovarsi

il dato nell'altro (Memory Data Register): si tratta in sostanza di due registri temporanei, in cui nel primo si "scrive" l'indirizzo da dove si vuol leggere/scrivere, nell'altro si legge/scrive il contenuto.

Con una struttura di questo tipo si dovrebbero saldare (a livello comunque alto, software) circa:

$$4 \times 32 = 128$$

Ossia 128 pins. Per un sistema piccolo un numero così elevato di interconnessioni può essere inutile. Ciò che si può fare con il multiplexing, è ciò:

Si sceglie in sostanza "cosa far interagire": si può scegliere se inviare nel bus il program counter piuttosto che il memory address register, e l'instruction register piuttosto che il memory data register: si tratta di scelte coerenti, dal momento che le prime due sono differenti sotto il punto di vista della scelta e dell'indirizzo codice da prelevare, le ultime due sotto il punto di vista del codice vero e proprio.

Nel processore in questione c'è una cosa di questo genere:

Si ha un sistema di multiplexing del bus tale da ridurre ulteriormente le linee di bus. Si usa concettualmente qualcosa di questo genere: si porta a 1000, dopo di che si porta indietro, mediante il comando "LOAD 5000" l'instruction register (IR), quindi si scrive nel MAR l'indirizzo di 5000; si preleva dalla memoria il dato ad esso corrispondente, e lo si scrive nell'MDR, dove sarà disponibile. Si noti che è sempre necessario passare dall'address bus. Se non si hanno problemi di velocità, anzichè inviare 4 bit per volta, se ne può inviare 1 per volta, risparmiando sul parallelismo dell'address bus, ma perdendo in velocità: al posto di 4 colpi di clock se ne impiegano 16, dal momento che si riduce il parallelismo della trasmissione di dati.

Si noti ancora una volta: si parla di interrupt, ma si vuole ribadire il seguente concetto, che comunque verrà ri-ribadito e ripreso: anche se il sistema sta dormendo, ossia se il clock è fermo, le tensioni basse, dunque la macchina a stati fissa ad uno stato e senza possibilità di riprendersi di "propria spontanea volontà", esistono alcune linee (16) sempre in attesa, sveglie, che possono svegliare il processore quando è necessario (un po' come nei pc, con lo stato di "stand by": quando si preme un tasto, il computer torna a funzionare).

3.2 Periferiche standard del sistema

Verranno a questo punto elencate le periferiche standard del sistema elettronico digitale in studio, ossia il MC9S12E128, della Freescale.

- ADC e DAC a 8 bit: dal momento che serve un buon compromesso tra il parallelismo di bit e la velocità del sistema sono stati scelti 8 bit per la suddetta periferica. Si hanno a disposizione 16 canali e 1 convertitore multiplexato per cui si usa un canale alla volta, trattato mediante un sistema di sample and hold. Generalmente la conversione D/A è poco utilizzata. Spesso questo tipo di periferica non è fondamentale, se si intende utilizzare la PWM (Pulse Width Modulation) per il sistema.

- Timer : per ogni applicazione si ha sostanzialmente un automa a stati finiti che va descritto basandosi su durate temporali. Avendo a disposizione solo il processore, si potrebbe utilizzare l'interrupt: il limite temporale sarebbe di fatto la latenza di interrupt, dunque l'unico tempo che separerebbe due operazioni sarebbe la latenza che trascorre a causa dell'interrupt. L'interrupt però non è la soluzione a tutti i problemi: se si usassero sempre e comunque solo interrupt, il processore sarebbe sostanzialmente in grado di portare avanti solo un'operazione per volta: supponendo di stampare un libro mediante interrupt, nel mentre non sarebbe possibile leggere la posta o ascoltare musica. Usando invece un hardware dedicato che conta il tempo, utilizzando come frequenza dedicata a ciascuna operazione una sottomultipla della frequenza dell'orologio quarzato (frequenza molto elevata), senza contatori esterni, si riesce a ottenere il T_{clk} del timer. A partire da esso si possono richiamare in maniera "ordinata" le chiamate ad interrupt, risolvendo dunque i problemi di non simultaneità delle operazioni, potendone gestire, mediante il timer, più di una per volta. I timer servono per definire:

  - Segnali periodici (atti a campionare ogni tot istanti, mediante una sorta di "sveglia periodica");
  - Sequenze di impulsi (in grado ad esempio di identificare gli stati di un qualcosa, come di un semaforo).

Capitolo 4

Lezione 04

Si propone a questo punto una piccola correzione del test effettuato in aula, in modo da fissare alcune idee riguardo gli argomenti precedentemente proposti. Si divide in 7 sezioni, una per ciascun quesito.

Esercizio 01

Volendo pilotare un LED accendendolo con 10 mA a $V_\gamma = 1,2$ V mediante l'uscita di un microprocessore con $V_{OL} = 0,4$ V. Come I_{OL} si hanno 6 mA.

Ciò che si deve fare è usare due porte logiche in parallelo. Dal momento che esse non si comportano tuttavia in modo ideale, nel senso che non sono in grado di autolimitare la corrente, è necessario usare delle resistenze supplementari; queste verranno dimensionate in modo da limitare la corrente sul diodo (LED).

Esercizio 02

Per quanto riguarda i condensatori di bypass, si considera il seguente esempio: si vuole valutare la costante di tempo del circuito, in modo da evitare problemi nella ricezione dell'alimentazione per la porta logica. Si puó stimare la costante di tempo τ come:

$$\tau = \frac{L}{R} = 100\,\text{ms}$$

Ció che si puó fare dunque é, data una tensione minima consentita V_{min}, utilizzare la seguente formula:

$$\Delta Q = (V_{nom} - V_{min}) \cdot C$$

Aumentando la capacitá, in modo da non farla mai scaricare del tutto, si ottimizza introducendo un margine. Ci si puó aspettare una capacitá nell'ordine dei nF.

Esercizio 03

Per XTAL si intende un generico cristallo; esso é realizzabile, a partire da un elemento piezoelettrico quale il quarzo, mediante un oscillatore, ad esempio quello di Pierce.

Esercizio 04

Dal momento che l'LCD non deve subire fenomeni di elettromigrazione, esso deve essere pilotato in modo che la continua sia nulla. Ció che si deve fare é usare un segnale in controfase a media nulla. Per spegnerlo, é sufficiente mandare il segnale in fase.

Ció si puó fare utilizzando un EXOR: esso é, come ben noto dalla teoria dei sistemi digitali, un inverter con selezione. Se il dato é 0 si ha il clock in fase, altrimenti si inverte, ottenendo ció che si vuole.

Esercizio 07

Sono giá stati spiegati due sostanziali metodi per la presenza di tensione: trasformatore e optoisolatore. Mediante entrambi si ottiene disaccoppiamento dal riferimento di rete, dunque si puó visualizzare la presenza di qualcosa. Nel secondo modo oltretutto si riesce a visualizzare otticamente la presenza della tensione.

Esercizio 06

Per quanto riguarda il power-on reset, quando la V_{AL} é a regime e i controlli per i colpi di clock scelti dal progettista sono positivi, quindi quando la macchina é ben cadenzata in modo da non far finire i tutto in stati non previsti, si puó sganciare il reset per entrare nello stato di funzionamento regolare.

Il condensatore permette di dare lo "sgancio", ricaricandosi mediante R, e scaricandosi mediante il diodo.

Esercizio 08

Il seguente circuito:

Sarebbe semplicemente un bistabile. Semplicemente, sarebbe una sorta di flip-flop che si autosostiene. Per poter imporre un valore in memoria, si puó utilizzare l'interruttore forzando un certo segnale, ad esempio uno "0" molto forte.

Questo circuito puó essere utilizzato per uno scopo ben preciso: come circuito "antirimbalzo". Per ogni contatto meccanico si ha un "rimbalzo" (click). Questo circuito non ha problemi (se si imposta una resistenza R "forte"), perché al primo colpo si forza il segnale, e si evita di avere problemi.

Esercizio 09-10

Ció di cui si parla é uno standard di comunicazione; nella fattispecie, si parla delle seriali del computer, per comunicare per l'appunto serialmente tra calcolatori o tra un calcolatore e periferiche. Al fine di utilizzarla é necessario specificare il protocollo, sia sotto il punto di vista meccanico, del connettore, sia dal punto di vista elettrico. Parlando di RS233, si considera un livello alto compreso tra i 3 e i 30 V, basso tra i -3 e i -30 V. Si possono dunque introdurre specifiche logiche di protocollo quali velocitá, ordine di trasmissione, e cosí via.

4.1 Introduzione al Laboratorio

Come è ben noto, quando si ha a che fare con un sistema elettronico le regole d'oro sono sempre le stesse:

- Alimentare il circuito;
- Cadenzarlo (temporizzarlo);
- Farlo funzionare.

Per alimentare un circuito bisogna già prestare alcune attenzioni: bisogna utilizzare il manuale e scoprire quali piedini è necessario utilizzare, possibilmente separando le alimentazioni "di potenza" da quelle "pulite", le piú critiche, quali oscillatori vari e PLL. Di solito, le V_{DD} e le frequenze dei PLL vengono generate automaticamente dal sistema; ciò che si fa quindi è mettere capacità di bypass, in modo da filtrare ed evitare i problemi riguardo le linee induttive. Si ha inoltre un certo numero di fusibili, ripristinabili, in modo

da poter evitare danni permanenti alle parti piú sensibili, quale ad esempio il processore.

Il potenziometro ha un comportamento di questo tipo:

Un convertitore ADC è collegato al trimmer, in modo da fornire, per la corrente al suo interno, un'uscita digitale. Il comportamento di questo può essere regolamentato mediante il calcolatore, con il software "Codewarrior": esso è sostanzialmente un compilatore C ad-hoc per la scheda col quale, caricando alcune librerie, si può impostare tutto. Le librerie hanno diverse utilità, quale ad esempio quella di mappare le porte, assegnando agli indirizzi fisici nomi simbolici, elencati sul manuale. Spesso si deve fare uso delle seguenti routine, per l'inizializzazione delle periferiche:

```
PTT = 0x00; /* Valore di default in uscita */
DDRT = 0xFF /* 8 vit della porta t diventano uscite e si usa sempre questo ordin
```

Si potrebbe ad esempio introdurre un codice di questo tipo:

```
main() {
PeriphInit(); /* Inizializzazione delle periferiche */
EnableInterrupts; /* Routine di abilitazione degli interrupt */
for( ; ; ) { /* For ever */
if(ATDSTAT_CCF0) /* Se tutto \‘e pronto */
PTT = ATDDR0H; /* Lo metto sull'uscita dei LED, in modo da visualizzare */
}
```

Questo codice andrà sulla flash e farà ciò che è stato detto. La porta T (PTT, Porta T) o comanda i led o fa da timer. Ciò che non si può fare è comandare direttamente i led con il timer: si passa per T.

Capitolo 5

Lezione 05

Questa lezione sarà prevalentemente sulla parte software del microprocessore. Verrà fornita un'infarinatura sullo stile da adottare per il codice, quindi verranno proposte alcune indicazioni per l'uso corretto del processore.

5.1 Programmare BENE in C

Il microprocessore in uso si programma mediante il linguaggio C. Il problema è che, se dai corsi di informatica si è abituati a programmare su processori dalle risorse immense, ciò ora non è piú possibile, dal momento che si parla di sistemi embedded, le cui risorse sono di fatto minime. Gli strumenti a disposizione sia per la programmazione sia per il debugging sono limitati: in un microprocessore non è possibile effettuare operazioni di debug mediante printf(), dal momento che non si hanno periferiche di output a terminale. Tutto ciò che si ha a disposizione, di fatto, sono 8 led, dunque si può dire che il debugging sia complesso. Usare un debugger a livello di calcolatore che effettua la programmazione inoltre è antipatico, in quanto la temporizzazione del debugger e quella del microprocessore non coincidono, dunque il programma potrebbe andare molto piú piano rispetto alla realtà. Troppa velocità potrebbe causare problemi che un debugger non sarebbe in grado di mettere in evidenza. Ciò che si deve fare, dunque, è imparare a scrivere un codice meno errato possibile. Esistono tecniche atte a permettere di scrivere programmi con una quantità minima di codice; esse sono basate sui seguenti due concetti:

- Utilizzare il C in maniera corretta;
- Capire cosa succede quando si compila, linka, carica su scheda il programma.

Il codice deve essere semplice, enorme, anche lento, però molto facile da usare. Assembly ha i problemi opposti: il codice non è portabile e aggiornabile, dunque per ogni upgrade andrebbe riscritto da 0. Per questo motivo, per programmare sistemi embedded, C è preferibile ad Assembly: a meno che non si debbano scrivere pochissime righe per implementazioni rapide, o che devono soddisfare alcune operazioni altrimenti non realizzabili.

Ci si potrebbe a questo punto chiedere: come si scrive in maniera decente il codice? Beh, cercando di ordinare il codice, utilizzando due fondamentali tipi di files sorgenti: i ".c" e i ".h", ossia i cosiddetti "header files". I ".c" son ben noti dunque è inutile parlarne; per quanto riguarda gli header files, in sostanza essi servono a contenere le dichiarazioni dei files omonimi .c , in modo da poter richiamare il loro contenuto. Si noti che negli header files non vanno definiti oggetti, ma solo dichiarati: le definizioni vanno contenute negli omonimi .c .

Ciò che si può fare è, in questi header file, definire anche le macro: esse sono sostanzialmente delle sostituzioni letterali a parti di codice, effettuate dal precompilatore: il precompilatore, tra le altre cose che fa, sostituisce ciò che sta dopo il "#define" dovunque sia usato nel codice, letteralmente, in modo da effettuare tutto ciò che sarebbe anche senza la macro. Le macro aumentano la leggibilità del codice, abbastanza gratuitamente. Esse vanno generalmente scritte in maiuscolo, e seguendo alcune regole tipografiche assolutamente non banali. Se ne fa un esempio:

```
#DEFINE SUM(A, B) ((A)+(B))
```

Come mai questa grande abbondanza di parentesi? Il motivo è abbastanza semplice: le parentesi evitano incomprensioni tra gli operatori: dal momento che gli operatori nel linguaggio C hanno una certa precedenza rispetto ad altri, quello che può capitare è che le istruzioni nelle macro siano male interpretate; abbondando con le parentesi, si evitano incomprensioni, poichè si fa in modo da "disaccoppiare" le varie espressioni letterali dagli operatori, rendendo insignificanti le precedenze. Altro esempio di macro potrebbe essere il seguente:

```
#DEFINE MAX(A, B) ((A)>(B) ? (A) : (B) )
```

Si propone a questo punto un esempio di header file, in modo da meglio permettere la comprensione.

```
#IFNDEF PIPPO_H
#DEFINE PIPPO_H

void f_1(int a);
void f_2(char *b);
floag g_xx(void);

#ENDIF
```

Come funziona ciò? Beh, se già non esiste (IFNDEF) un certo header PIPPO_H, lo si può definire; al suo interno saranno dichiarate tutte le varie define e funzioni interessanti, dopo di che si potrà chiudere la IFNDEF mediante una ENDIF. Questo header file, associato a un certo pippo.c , sarà richiamabile mediante il seguente comando:

```
#include pippo.h
```

Il che permetterà di includere nel file tutte le funzioni definite in pippo.c . Si noti che tra header file e c-file vi deve essere assoluta coerenza di tipi di funzioni e parametri passati, altrimenti potrebbe non funzionare piú niente.

5.2 Storage class

Quando da informatici si compila un file, si produce un file eseguibile, ad esempio ".exe" nel caso di sistemi Windows-like. Il contenuto di questo file è codice macchina, pronto per essere eseguito. In un calcolatore ordinario, ciò che si fa in pratica è caricare l'eseguibile nella memoria RAM (molto grossa), quindi eseguirlo. In un sistema embedded una cosa di questo tipo non è fattibile: la RAM in esso non è assolutamente grossa, di conseguenza bisogna muoversi in modo diverso. Ciò che si fa in pratica è utilizzare memorie ROM (piú lente in scrittura ma molto meno costose) non volatili per il codice, quindi memorizzare in RAM solo le variabili, per poi aggiornare la ROM solo quando e se necessario.

Date le piccole possibilità dell'embedded a volte si cambia qualcosa.

Osserviamo a questo punto un piccolo spezzone di codice, in modo da introdurre i principali tipi di variabili:

```
main(){
int g; /* Variabile globale */
void f(int x);
int a; /* Variabile locale */
static int b; /* Variabile locale static */
...
...
}
}
```

Sono state qua analizzati tre tipi di variabili:

- Variabili globali: sono in RAM fino alla fine per un programma (sempre, per un embedded);
- Variabili locali: la loro vita è limitata alla durata di una funzione, dunque la memoria assegnata per esse viene rilasciata al termine della funzione;
- Variabili locali "static": diventano globali, ma con visibilità limitata al file .c in uso. In un altro file .c le variabili globali possono essere lette, queste no.

Variabili globali e statiche sono inizializzate a zero, le variabili locali no. Le variabili globali occupano posizioni preferenziali in RAM, nella quale vengono introdotte per ordine di dichiarazione; quelle locali vengono messe in uno stack, interno alla RAM stessa, mediante funzioni di PUSH implementate nel microprocessore.

Vi è un altro problema implementativo, derivante dalle differenze filosofie hardware utilizzate nella storia: un tempo, nei processori piú vecchi, si utilizzava l'idea CISC, ossia avere processori in grado di eseguire operazioni estremamente complesse, a costo di avere poco spazio per i registri; nelle tecnologie piú nuove, basate sull'idea del RISC, si ha un insieme di operazioni ridotto, piú semplice, ma si hanno piú registri a disposizione, in modo da avere memoria veloce a disposizione piú rapidamente.

Le variabili globali vengono introdotte nei registri; dal momento che esse sono abbastanza antipatiche sotto il punto di vista dell'implementazione in hardware, esse sono tra le prime cose che il compilatore cerca di semplificare o ridurre, dunque ottimizzare. Ciò che si potrebbe fare tuttavia è introdurre nei registri direttamente variabili locali, mediante un'apposita parola chiave:

```
...
register int i;
...
```

Mediante "register" si impone il fatto che la variabile sia introdotta, salvata, proprio nei registri di sistema.

Si può desiderare il contrario, ossia che una certa variabile non venga messa nei registri. Ciò si può fare utilizzando, anzichè la variabile, il suo indirizzo, mediante il comando $\&i$. Una soluzione ulteriore, che può essere molto utile per un altro motivo, richiede l'introduzione di un altro tipo di variabile, per ora non nominato:

```
...
volatile int i;
...
```

Il tipo "volatile" identifica variabili che possono essere modificate da oggetti esterni. Ciò che in pratica si ottiene, parlando di sistemi del nostro tipo, è una variabile che non può essere ottimizzata dal compilatore C: dal momento che volatile è un tipo di variabile che può essere molto spesso modificata, ottimizzarla costringerebbe la perdita di informazione. Quando dunque si dichiara una variabile come volatile, si ha la garanzia che essa non subisca alcun tipo di ottimizzazione. Non dichiarando volatile, potrebbe capitare che, data la seguente istruzione:

```
...
int i;
...
i = 0x55; /* Non eseguita */
i = 0xAA; /* Eseguita */
```

Venga eseguita esclusivamente l'ultima, dal momento che la prima risulterebbe essere, per il compilatore, inutile.

5.3 Sistemi a microprocessore

Molto spesso, in un sistema a microprocessore, le periferiche sono "memory mapped": esse sono visualizzate e accessibili, di fatto, come se fossero celle di memoria, dunque è possibile recuperare il set di istruzioni assembly (e/o C) per utilizzarle. Per poter usare le varie porte/periferiche, dunque, può essere necessario usare l'algebra dei puntatori, e dichiarare puntatori, ad esempio come:

```
unsigned char *ptr = 0x1000;
```

Quello che si fa, come già accennato, è usare librerie che contengano l'associazione tra gli indirizzi di memoria (fittizi, poichè vi son collegate periferiche) in modo da trattarli mediante acronimi semplici.

Bisogna creare un livello di astrazione hardware in modo da aumentare la portabilità del software. Si ha un qualcosa del genere:

In pratica, l'HAL, ossia Hardware Abstraction Level, è quello che permette questa astrazione. Ciò che accade a questo livello, ad esempio, nel caso del convertitore A/D, potrebbe essere la lettura della corrente/tensione dal trimmer, per trasformarla in un segnale luminoso mediante i LED. Leggendo il manuale e osservando gli esempi si può capire come effettuare queste operazioni

5.3.1 Compilazione

A questo punto ci si può chiedere cosa capita al momento della compilazione, nell'ambito dei sistemi elettronici che si stanno trattando. Si ha in sostanza uno schema a blocchi di questo tipo:

Osserviamo passo-passo cosa capita:

1. Si parte dal progetto del file ".c";
2. Mediante il pre-processing, si eseguono tutte le direttive per il processore, ossia le #DEFINE, le #IFNDEF, le #INCLUDE, e le varie parenti; ciò che si fa, in pratica, è introdurre una serie di sostituzioni testuali dettate da queste operazioni;
3. Una volta effettuate le sostituzioni letterali, si prende (mediante il compilatore) il file c preprocessato, traducendolo in istruzioni elementari assembly, mediante un certo livello di ottimizzazione;

4. Una volta ottenuto un file ".s", ossia assembly, viene introdotto il link tra le operazioni assembly e il codice macchina, dettante le operazioni esatte che il processore deve effettuare per eseguire il programma. Da qui si avrà un certo numero di file ".o".

5. Dati i diversi file ".o", ossia file oggetto, è come se fossero costituiti da un certo insieme di blocchi, con dei "ganci" sul vuoto, scollegati; il compito del linker sarà quello di produrre un eseguibile finale, mediante l'unione tra loro di tutti questi "ganci"; l'eseguibile sarà dunque pronto per essere eseguito dall'hardware, dal processore.

Capitolo 6

Lezione 05

In questa lezione ci si occupa sostanzialmente di due argomenti, riguardo aspetti interni del microprocessore: i meccanismi di "reset" e di "interrupt", dunque i "timer".

6.1 Reset e Interrupt

Ci si concentrerà sostanzialmente, come processore, su quello utilizzato in laboratorio, ossia l'HC12. Come già detto, in un processore il POR (Power On Reset) è fondamentale, al fine di poter evitare che la macchina inizi a lavorare "in transitorio", ossia quando il cadenzamento non è ancora ben definito. Si introduce inoltre la possibilità di avere un reset esterno, in modo da riportare "manualmente" la macchina in uno stato predefinito dal progettista del sistema. In totale tuttavia non sono solo presenti sistemi di reset di questo tipo, nel sistema, dal momento che bisogna essere in grado di fronteggiare anche altri tipi di problemi; alcuni tipi di reset interni al sistema introdotti nell'HC12 sono ad esempio i seguenti:

- XTAL Reset (atto a resettare in caso di problemi nella temporizzazione del sistema: se il quarzo per qualche motivo subisse danni o smettesse di funzionare, è necessario interrompere l'elaborazione del sistema, dal momento che un cattivo cadenzamento del circuito provocherebbe l'ingresso in stati indesiderati, facendo nascere problemi nell'elaborazione);
- Low Voltage Reset (atto a resettare in caso di brown-out o black-out del sistema: se per qualche motivo la tensione va sotto la soglia, il reset potrebbe essere utile, al fine di salvare dati e per l'analisi "post-mortem", come in una scatola nera di un aereo. Il reset, con la memoria flash EEPROM, potrebbe essere utilissimo sotto questo punto di vista);

- COP : Correct Operating Properly: si ha una sorta di contatore che, se arriva a "0", fa partire il reset. Questo circuito si basa sostanzialmente sulla seguente idea:

  Si dimensiona la resistenza R in modo che se il condensatore non viene ricaricato entro un certo tempo scatta un allarme; ciò che deve capitare è qualcosa del genere:

  Se gli impulsi non partono, il condensatore non si ricarica, dunque significa che in qualche modo il sistema è in uno stato di malfunzionamento, e si deve introdurre il reset.

Questo ultimo esempio è molto simile a ciò che viene comunemente chiamato "watchdog", ossia "cane da guardia": si ha sostanzialmente un'idea di questo tipo:

Si ha TC, ossia la "soglia di riferimento" per quanto riguarda lo 0. Il registro è quello che "carica" il contatore, in modo da non farlo arrivare a "0", nel caso in cui il sistema funzioni correttamente; se ci sono dei problemi, dettati ad esempio dall'oscillatore, il LOAD non può avvenire, dunque non si "dà da mangiare" al cane, che dopo un po' si arrabbia, e fa partire un segnale di reset in modo da bloccare tutto e cercare di stabilizzare. Il problema è se neanche il watchdog funziona: a questo punto solo un buon reset esterno può mettere a posto tutto o quantomeno far uscire la macchina dallo stato, "uccidendo" il suo lavoro.

In termini di programmazione del circuito: se non si intende usare alcuna feature, si può anche evitare di preoccuparsi: comandano il reset esterno e il POR. Le altre modalità di reset di default non sono abilitate dunque, se si intende usarle nel proprio sistema, è necessario abilitarle mediante le funzioni C esplicate nel manuale del processore.

6.2 Organizzazione della memoria / Interrupt

La memoria nel processore che si intende utilizzare ha un'organizzazione di questo tipo:

All'inizio, da 0x0000 a 0x0400 si han indirizzi assegnati per le periferiche, gestite mediante la modalità "memory mapped"; da 2000 a 4000 si ha a che fare con la RAM, ossia la memoria volatile del sistema, utilizzata per la memorizzazione delle variabili da gestire. Da 4000 a 8000 si ha un banco di flash, nel quale si introduce il codice da eseguire; la flash non viene mai utilizzata per dati volatili, in quanto molto lenta. La gestione "a banchi" della flash (da 4000 a 8000, e poi via per diversi blocchi) è scelta in quanto si vuole disaccoppiare il codice dai dati salvati (fino a 8000 è codice, dopo dati). In

questo modo si può estendere l'indirizzamento. Dopo diversi banchi di dato flash, si ha la interrupt table. In una ROM vi è un programma, detto "boot loader", che permette di far riavviare la macchina anche dopo lo spegnimento dal codice introdotto dal progettista, dal momento che le Flash sono memorie non volatili (per quanto purtroppo meno potenti). I ".o" vanno tutti nella flash, in coda.

Come già accennato, l'ultima zona di indirizzi riguarda la interrupt vector table, la reset vector table e anche il boot loader: questa zona è off-limits, nel senso che si può solo utilizzare ciò che vi è contenuto, assolutamente senza scrivere. Negli ultimi due byte c'è anche l'indirizzo a cui andare quando si riceve un segnale di reset (0xFFFE , ossia il link da cui si riparte quando si resetta la macchina). Altro indirizzo notevole è il 0xFFFC: ossia il vettore a cui si salta nel caso in cui l'oscillatore (XTAL) smetta di funzionare. Parlando di reset e interrupt, si ha a che fare sostanzialmente con due tipi di ambienti:

- Ambiente aperto: si va ad una routine che è ritornata dopo l'operazione di interrupt;
- Ambiente chiuso: si blocca l'esecuzione della macchina.

Da qua si può notare quale sia la differenza tra interrupt e reset: una procedura di interrupt permette di salvare lo stato della macchina nello stack, effettuare l'operazione richiesta da chi ha richiesto l'interrupt, quindi ri-caricare lo stato dallo stack e proseguire l'esecuzione; il reset "blocca" violentemente l'esecuzione, non salvando nulla nello stack, dal momento che non si intende proseguire con le operazioni da fare.

Tra reset e interrupt vector table ci sono le cosiddette "TRAP": quando si richiede di effettuare operazioni inesistenti, illegali, quale ad esempio la divisione per 0, si carica un'istruzione che non si conosce, in modo da non impallare tutto.

Esistono diversi tipi di interrupt:

- IRQ : Interrupt Request Mascherabile : questa non fa accantonare l'operazione sul momento eseguita, ma viene eseguita in base alla sua priorità;
- XIRQ : Interrupt Request Non Mascherabile : ha la priorità su qualsiasi altra operazione, che fa bloccare;
- SWY : Routine software di interrupt, eseguita mediante una chiamata al sistema operativo.

Per le prime due si ha qualcosa di questo genere:

Si han due bit di controllo atti ad abilitare o meno l'interrupt nel registro di controllo, mediante un meccanismo di questo tipo. Questi due bit possono mascherare o meno il segnale di interrupt, quando si voglia.

L'interrupt piú interessante e importante, tuttavia, è il RTI (Real Time Interrupt) : data una main() che chiama una procedura all'infinito, se si ha un evento raro, quello sta nel loop e ogni tanto viene chiamato, mentre faccio altre cose; se è poco probabile, devo controllare molto spesso che ci sia, anche se questi controlli faranno sprecare tempo. Se invece uso il RTI, ho un timer che ogni tot dà un interrupt, quindi si blocca l'esecuzione del main() (nella quale non dovran esserci controlli, ma solo un'abilitazione all'inizio dell'interrupt) e fa saltare al processo indirizzato nel RTI.

Quello che può succedere è che, a causa della priorità, l'interrupt non funzioni mai; quello che si può fare è usare un RTK (Real Time Kernel): il main non esiste piú, dal momento che solo gli interrupt vengono man mano eseguiti e le loro procedure autochiamate.

6.3 Timer

Storicamente, i primi contatori utilizzati erano dei "down counters", ossia contatori alla rovescia. Uno schema poteva essere per esempio il seguente:

I registri contengono i numeri da cui si deve partire a contare; una volta introdotti i bit, si deve partire a contare. La partenza è legata dunque al fatto che alcuni bit vengano modificati; ciò che si deve fare, nella fattispecie, nel caso si carichino in diversi colpi, a causa di differenze tra i parallelismi, è controllare sul manuale quale parte del registro caricare all'inizio. Volendo un registro ripetitivo, si introduce l'OR, in modo da poter ripartire da capo al momento desiderato. Potrebbe essere d'altra parte necessario mantenere un'uscita stabile; ciò si può fare introducendo un flip flop prima del pad di uscita, in modo da mantenere stabile l'impulso in uscita dal contatore. Si può comparare l'uscita con uno "0", ottenendo un generatore di onda quadra. Un altro problema è: non è possibile utilizzare direttamente il clock come contatore, dal momento che ci si aspetta di avere frequenze molto elevate, dunque conti molto veloci e la conseguente necessità di registri molto grossi. Ciò che si usa a tal fine è un "prescaler": si tratta di un divisore di frequenza per divisore M.

Esistono diverse modalità piú "moderne" e che possono portare ad altre idee per quanto riguarda la realizzazione di timers o la loro applicazione; verranno ora esposte le idee fondamentali dietro i contatori piú moderni.

- Input capture : si tratta di una sorta di "fotografia" dell'ora (interna alla macchina) in cui avviene un certo evento esterno al sistema. Dato questo evento, si registra l'ora (timestamp) del medesimo, e si attiva una flag indicante il fatto che un'ora è stata registrata. Basandosi su di questi timestamp sarà possibile effettuare altre operazioni temporizzate in un certo modo;

- Output compare : una volta memorizzata una certa ora, e contato mediante un contatore un certo tempo a partire da essa, è possibile generare un evento, un segnale in uscita all'ora desiderata. Volendo ad esempio svegliare il computer dopo una certa ora, si "campiona" l'ora in cui si spegne, e si avvia un contatore in grado di arrivare ad una certa cifra, basandosi sulla base tempi interna al sistema; una volta che il contatore raggiunge una certa cifra calcolata internamente si invia un segnale all'esterno del timer in modo da riattivare il sistema intero.

Per evitare di leggere cose sbagliate, si freeza il counter in un registro, in modo da assicurarsi sull'informazione.

Basandosi sempre sui contatori, è possibile effettuare cose interessanti, quali contare impulsi (clockando il contatore mediante un qualche generatore di impulsi), o creare un frequenzimetro, clockando mediante una AND di un certo "tick" e di un treno di impulsi, in modo da garantire una certa frequenza.

Capitolo 7

Lezione 06

In questa lezione ci si occuperà nuovamente degli aspetti software della progettazione elettronica.

La scorsa volta si è parlato di linguaggio di programmazione C. Passare da C ad assembly è molto facile, anche se spesso (sempre quasi) questa operazione è effettuata mediante strumenti automatici (detti compilatori). Passare da assembly a C non è assolutamente facile, dal momento che bisogna "decompilare" (o "reversare") il codice. Ciò che si può fare, dato un programma, è ricavare le istruzioni assembly che effettua, semplicemente associando ai codici delle istruzioni un'istruzione assembly; passare a un linguaggio di alto livello non è assolutamente facile, poichè l'assembly è l'implementazione ultima del C; tornare al C significa semplicemente costruire un programma in grado di fare le stesse cose, a ri-ottenere lo stesso assembly, ma ciò si può fare in molti modi.

Un altro aspetto che può essere critico in sistemi embedded è avere un codice molto "generale", ossia "troppo portabile": una printf(), istruzione apparentemente banale, richiama in sè diverse librerie, diverse runtime; una funzione di questo genere da sola occuperebbe 500 k di roba, cosa assurda per degli embedded. Sarebbe molto meglio (ipotizzando che il processore usi di queste funzioni) usare una putf(), che semplicemente stampa stringhe. Si ricordi: "la generalizzazione ha un prezzo", e questo prezzo è il costo, in termini di spazio, in memoria.

7.1 Cross compilation

In un sistema embedded si ha un qualcosa di particolare, rispetto a ciò a cui si è abituati: la macchina su cui si lavora è diversa da quella in cui si compila l'applicativo. Si parla, in questo ambito, di "cross compilation":

La compilazione avviene sul computer, dunque mediante USB si programma il dispositivo, che eseguirà il codice.

La domanda interessante a questo punto è: dove e come si debugga? A domande di questo tipo si può rispondere in piú modi, ossia proporre diverse tecniche di debug. All'interno del targer, ossia del circuito programmabile, esiste un software per la comunicazione, mentre nel calcolatore il software che si usa per la compilazione contiene un sotto-software in grado di interpretare i dati elaborati dalla macchina per gestire il debug. Questo tipo di debug sfrutta dunque il raccoglimento dal processore di dati, elaborati, in modo assolutamente intrusivo: comunicare dati comporta fare operazioni, cosa che costa in termini di calcoli e operazioni da far eseguire al processore. Nei processori Motorola / Freescale esiste qualcosa di piú interessante, ossia il BDM (Background Debug Module): si tratta di un blocco hardware che si interfaccia alla memoria come il processore, aspettando che il BUS sia libero e leggendo, effettuando il debug indipendentemente dal processore e senza togliere tempo all'esecuzione. Il BDM è un altro sottoprocessore, che si interfaccia alla memoria come un processore normale, indipendentemente da esso, e solo a scopi di debug.

Questo tipo di debug è detto "non-intrusivo", dal momento che l'applicazione non viene interrotta, e il microprocessore principale continua a lavorare. Unico limite è che non si può leggere dai registri: essi variano troppo in fretta, dunque non è mai possibile "trovare il tempo per accedervi" in maniera consona. Solo in questo caso si blocca l'esecuzione per far parlare il vero processore.

Esistono sostanzialmente due modi per fare debug:

- Debug software: nella memoria codice, al posto di una certa istruzione, si manda un interrupt software, modificando la memoria codice (ottenendo dunque qualcosa di intrusivo); a questo punto si legge quel che si vuole;

- Debug hardware: il BDM permette di fare il "watch" di una variabile, osservando il BUS indirizzi. Quando sul bus si mette l'indirizzo dell'istruzione interessante, che modifica una certa variabile, il BDM blocca il processore, e permette la visualizzazione. Ciò è interessante anche al fine di rilevare bachi, dal momento che se ci sono dei blocchi del processore in istanti non richiesti, significa che c'è qualcosa che non funziona, ma senza BDM non si potrebbe scoprire.

A questo punto si ha un problema: su di un sistema embedded può capitare di non avere un loader. L'unico punto in cui si può mettere un programma è la memoria Flash, dal momento che la RAM è volatile. Ciò che si fa

è far eseguire il programma dal primo byte in poi, senza avere possibilità di scelta. Ciò che si deve fare come prima cosa è inizializzare lo stack, dunque la cosa migliore è mettere lo stack all'inizio della RAM.

Nella memoria ROM Flash si ha un salvataggio delle variabili da inizializzare, variabili che vengono mandate in RAM, e dello stack da inizializzare. Lavorare sulla Flash è impossibile, ma salvare i dati una sola volta per poter ripartire da lì a sistema riattivato dopo spegnimento è assolutamente possibile e anche molto furbo, a patto di ordinare immediatamente il ripristino dello stato del sistema. Esiste una funzione assembler, chiamata "map_data_section", che permette di copiare tutti i bytes interessanti dalla flash a dove si vuole. Poi conviene abilitare gli interrupt.

Potrebbe essere importante comunicare le dimensioni al linker, mediante n file di testo creato a priori. Esso contiene informazioni sulle dimensioni delle RAM/ROM, di questo tipo:

```
memory.x /* Nome del file, non \‘e parte del codice */

/* Inizio codice /*

memory {
page(0) (vwx) : origin = 0x0000 lenght = 0x0040
text(vx)              : origin = 0x7000 lenght = 0x4000 /* ROM da 16 kB */
data                   : origin = 0x2800 lenght = 0x4000 /* RAM */
```

Mediante files di questo tipo è possibile mappare il codice e i dati dove si vuole, specificando le caratteristiche e le dimensioni della memoria.

Capitolo 8

Lezione 07

Molto importante puó essere, come giá detto, attivare l'interrupt; uno dei motivi per cui esso puó essere importante, é la gestione del timer. Uno dei vari utilizzi che si puó fare di un timer é la generazione della modulazione a larghezza di impulso (PWM : Pulse Width Modulation), fondamentale per pilotare dispositivi di vario genere.

L'obiettivo che ci si pone in questa lezione é vedere come si puó pilotare un motore elettrico, brushless (la cui velocitá varia a seconda della frequenza del segnale di pilotaggio). I motori brushless sono molto interessanti per l'elettronica moderna, dal momento che:

- Sono molto piccoli, rispetto ai motori elettrici tradizionali;
- Hanno una coppia piú elevata a paritá di caratteristiche rispetto ai motori tradizionali (sono dunque piú efficienti);
- Raggiungono velocitá superiori.

Mediante un microprocessore é dunque possibile generare un segnale modulato in PWM atto a pilotare motori di questo genere. Trattandosi di motori trifase, l'idea fondamentale sará quella di avere qualcosa di simile a tre sinusoidi sfasate tra loro di 120°; unico svantaggio di questi motori é che spesso bisogna gestire l'elettronica di potenza atta a pilotarli, dunque si potrebbero avere complicazioni legate a ció. Nel nostro caso, si analizzeranno le idee atte a generare un segnale PWM utilizzando i canali "timer" del HCS12. Ci interessa definire il periodo dell'impulso T_p e il tempo in cui sta alto il segnale, T_{ON}, attribuendo importanza soprattutto a questo secondo aspetto:

Quello che si fa in pratica é quello che potremmo chiamare "dimmer": un regolatore elettronico in grado di attenuare a proprio piacimento il segnale di pilotaggio di un motore (in egual maniera, a dei LED). L'idea da sfruttare

per un dimmer é la seguente: se si definisce un periodo T_p complessivo breve abbastanza da avere una frequenza molto superiore rispetto al polo meccanico del motore (o se si vuole, nel caso di LED, al "polo dell'occhio"), il fatto di non poter percepire le variazioni é a nostro favore, dal momento che in questo modo si permette di considerare non tutte le variazioni, ma solo la "media", ottenendo una sorta di media integrale reale (l'idea alla base del televisore). Ogni volta che si arriva all'output compare, si manda un segnale di interrupt. Si usa uno schema di questo genere:

A partire dalle specifiche che si intende soddisfare, si ricava il valore di divisione per N per il prescaler, dunque si specificano gli estremi di variazione per il duty cycle. Si noti che sull'accensione e sullo spegnimento di un motore si possono avere problemi: il motore é sostanzialmente modellizzabile mediante un'induttanza, elemento circuitale che nasconde una triste caratteristica: il fatto che far variare la corrente troppo rapidamente comporta enormi variazioni di tensione. Bisogna stare attenti a queste, ai fini di non "bruciare tutto".

Servono per definire questo tipo di regolatore due registri: uno per avere informazioni su T_p, ossia sul periodo globale, e uno per T_{ON}, ossia per il periodo di accensione. Sull'HCS12 (sui relativi manuali e sulle application note) si legge come funziona il timer, dunque come si programma ai fini di utilizzarlo. I passi da seguire sono fondamentalmente i seguenti:

1. Abilitazione dell'interrupt e dei piedini utili;
2. Attribuzione di un significato ai piedini che si vogliono usare; Calcolare il tempo di interruzione e abilitare un interrupt.

A questo punto é necessario avere alcune informazioni: la frequenza di PWM e il duty cycle. Si immagini che:

$$f_{PWM} = 100\,\text{Hz}; \quad f_{CLK} = 16\,\text{MHz}$$

Supponendo di aver a disposizione un contatore da 16 bit per effettuare il timing, si avrá come massimo timer possibile (a partire ovviamente dal PWM per fare il timing):

$$f_{MAX,timer} = 100 \times 2^{16} = 6,55\,\text{MHz}$$

Sará dunque necessario usare un prescaler, atto a ridurre la frequenza alla massima consentita per il timer, in modo da poter usare il clock.

$$N_{prescaler} = \frac{f_{CLK}}{f_{MAX,timer}} = \frac{16\,\text{MHz}}{6,55\,\text{MHz}} = 2,4427$$

Fatto ció, quando si arriva a fine conteggio si introduce T_{ON} o T_{OFF} dopo un interrupt, a seconda che si fosse a 1 o a 0 nel colpo precedente.

Un punto interessante é come far partire o fermare questa procedura: il motore puó essere comandato in modo da "partire piano" o "fermarsi piano" ad esempio, per questioni di sicurezza. Fino a qua si usa il timer per fare il PWM, ma in microprocessori un po' sofisticati si usano hardware dedicati per gestire il PWM, hardware di questo tipo: si ha un contatore dedicato con prescaler e registri in cui si introducono T_{ON} e T_{OFF} desiderati; si utilizzano sei canali di questo tipo:

Maggiori informazioni si possono trovare sul manuale del microprocessore.

8.1 PWM con Fault Protection

Come giá detto, un motore elettrico é sostanzialmente assimilabile a un sistema trifase di tipo induttivo, dunque a qualcosa di questo tipo:

Per ogni induttore, si puó effettuare il pilotaggio mediante un circuito atto a far ció, un esempio su titti potrebbe essere il ponte H:

Ci sono diversi modi per realizzarlo. Un modo un po' costoso ma di sicuro efficiente é quello di utilizzare quattro transistori nMOS a effetto di campo, i primi due come interruttori high-side e gli ultimi due come interruttori low-side. I low-side sono facili da pilotare: dal momento che essi sono tra polarizzazione e massa, la loro gestione é banale. Per quanto riguarda gli high-side, la questione non é cosí facile: sará necessario utilizzare dei traslatori di livello, in modo da poterli gestire, dal momento che essi sono a tensioni elevate. Si accendono i transistori per far scorrere corrente e invertire il campo generato. Si noti che si punta a far scorrere la corrente da un lato a un altro, dunque i due transistori dello stesso lato non devono mai essere gestiti assieme.

Buona cosa é introdurre protezioni: dal momento che si ha a che fare con induttori, usando variazioni brusche si puó far salire troppo la tensione. Introducendo un diodo si puó bloccare la tensione, in caso di emergenza.

Altra cosa da evitare é il fatto che vi sia un "overlap" sui segnali di pilotaggio A e B: introducendo un margine temporale minimo che garantisca che i due segnali non assumano mai lo stesso valore, in modo da rovinare il ponte e il motore, si riesce a ottenere buoni risultati. Volendo fermare il motore, si possono accendere solo i transistori alti o solo quelli bassi, scaricando cosí l'energia su di essi.

Alcune migliorie effettuabili sul sistema sono ad esempio una resistenza di sense, R_s, atta a vedere quanta tensione/corrente vi sia su, mediante un comparatore di soglia. Altro posto su cui si puó usare il comparatore di

soglia, anche piú utile, é ai capi dell'induttanza. Questo ponte é realizzabile sia con FET sia con BJT; in questo secondo caso si faccia attenzione, dal momento che i BJT soffrono maggiormente per la thermal runaway.

8.2 Comunicazione Seriale

Una delle comunicazioni piú utilizzate in assoluto é la cosiddetta "comunicazione seriale". Oltre a mandare simboli di n bit, si deve inviare il dato (frame), in modo da dare e rispettare riferimenti temporali. Avendo a disposizione in sostanza un segnale binario, si devono avere due valori di riferimento per il segnale. Come livello "basso" si considerano -12 V, come livello "alto" +12 V.

Un primo problema per la comunicazione é l'avvio: quando si inizia a trasmettere si utilizzano i bit di start, in modo da permettere al ricevitore di rilevare l'evento di inizio. Si hanno dunque i tempi di segnalazione (di simbolo), e un tempo in cui si resta a zero.

Si trasmette partendo dal LSB, terminando verso l'MSB. Ora si usano, per la trasmissione, 8 bit, il che significa avere la possibilitá di trasmettere una tra 256 informazioni. Lo standard permette l'opzione di un bit di paritá, e di un segnale di stop. Da definire, sono i seguenti parametri:

1. Lunghezza di simbolo (appena discussa);
2. Tipo di paritá;
3. Numero di stop.

Definito tutto ció, si puó iniziare a parlare di trasmissione.

La trasmissione appena descritta viene detta "trasmissione asincrona", e costa due tempi di segnalazione in piú.

Al fine di realizzare il sistema di trasmissione, é sufficiente avere uno shift register con qualche controllo.

La ricezione é piú complcata da trattare: non si sa, di fatto, quando si deve campionare al fine di avere un'informazione corretta, dal momento che non si ha una base dei tempi in comune tra trasmettitore e ricevitore. Ció che si fa, é iniziare a clockare sull'ingresso di start, verificando tuttavia che esso non sia un disturbo, campionando tre o quattro volte per vedere che lo start "permane" e non é solo un impulso di linea. Ció che serve dunque é un orologio locale, la cui frequenza sia intorno alle 16 volte maggiore rispetto al clock del trasmettitore, in modo da avere la certezza di acquisire correttamente lo start; sapendo a questo punto a quanto si trasmette, si

sovracampiona e rileva, come detto, lo start. Una volta fatto ció si puó trovare la potenza, fasarsi, dunque contando fino a 8 di 16 (dove 16 il multiplo della frequenza del segnale) ci si puó trovare bene a metá. Si passa a questo punto il PL e il En del contatore che fa da orologio, in modo sincrono o quasi rispetto allo start. Si noti che una volta fasato tutto non si rimane sincroni per sempre, dal momento che gli oscillatori hanno una precisione finita. Se il simbolo (8 bit) fosse lungo, dopo un po' si avrebbe uno sfasamento.

Lo standard volendo prevede anche la possibilitá di trasmettere una quantitá infinita di simboli, per cui chi parla deve anche trasmettere il clock, proponendo dunque una trasmissione seriale sincrona.

Esistono alcune tecniche per la trasmissione:

- NRZ (Not Return to Zero) : non si ritorna, rimane mai a 0;
- RZ (Return to Zero) : dualmente a prima, si ritorna a zero.

In generale, le periferiche utilizzate per la realizzazione di porte di questo tipo, sono le USART (Universal Synchronous Asynchronous Receiver Transmitter), in grado di implementare questo standard. Un altro tipo di metodo é il RZI, in grado di comunicare con l'IRDA : il ritorno a zero invertito. Come detto, ció é perfetto per comunicare con le infrarosse: si dá l'impulso al LED quando si é a zero, e non si fa niente quando si é a "1". Questo protocollo a volte é utilizzato anche nei sistemi wireless.

Un problema é che spesso si ha un XCO (oscillatore a cristallo), e serve un prescaler in grado di fornire la frequenza alla velocitá del baudrate. Ció che si realizza in pratica, spesso, é il cosiddetto "baud rate generator":

Si ha un prescalamento per un N sconosciuto, in modo da ottenere una f_B, ossia frequenza di baud, tale da far rimanere tutto sincronizzato.

Per comunicare esistono in sostanza le seguenti modalitá:

- Full-duplex: si utilizzano sostanzialmente tre fili: un riferimento comune, un filo per l'invio e uno per la ricezione. Utilizzando tre canali separati per la trasmissione del segnale, é possibile inviare e ricevere contemporaneamente segnale.
- Half-duplex: si risparmia un filo (poiché si ha sempre filo di trasmissione e filo di riferimento) ma solo uno per volta puó trasmettere: si trasmette e riceve in tempi diversi.
- Interrupt: una modalitá é quella di effettuare una richiesta di interrupt ogni qual volta si intenda trasmettere qualcosa, bloccando le operazioni e trasmettendo.

Esistono alcune casistiche importanti da cosiderare: trasmettitore vuoto e ricevitore pieno. La cosa importante per i dispositivi é quella di essere interpellati dopo che l'intero dato é stato completamente assemblato, non solo per un pezzo, dunque si vuole trasmettere scrivendo nel trasmettitore solo quando il buffer é vuoto. Si fa qualcosa del genere:

La S viene introdotta subito dopo che A va nel trasmettitore, ossia nello shift register, dal momento che il REGTX in questo modo risulta essere libero. Bisogna quindi aspettare per un po', in modo che tutta la A trasmessa nello shift register esca, bit per bit.

Una pratica frequente per testare la comunicazione seriale é quella di partire con un loop back: trasmettendo e mettendo il sistema in loopback il ricevitore riceve quello che si manda fuori. In pratica "ci si ascolta da soli", vedendo che tutto effettivamente funzioni. Se ció funziona si puó trasmettere sul serio all'esterno, altrimenti si ferma tutto.

8.3 SPI

Quando si parla di SPI, ossia di seriale come mezzo di comunicazione per periferiche, ció che si deve sostanzialmente fare é attribuire prioritá alle varie periferiche, introducendo un master e degli slave; tutto verrá pilotato dallo stesso clock, clock sostanzialmente fornito dal master del sistema. Il master sará quello che avrá il diritto di scrivere nello shift, fermandosi. Il baud rate generator produrrá gli 8 impulsi, dunque la comunicazione si avvierá.

8.4 Inter Integrated Circuit (I2C)

Uno standard per le comunicazioni (della Philips ma molto utilizzato) é il cosiddetto I2C. Esso é utilizzato come standard per la comunicazione tra circuiti integrati. L'idea é quella di far parlare il master con memorie non in parallelo, bensí attraverso due fili: un segnale di riferimento (temporizzazione, un clock), e un segnale half-duplex. Il controllore di tutto ció é la periferica I2C.

Si ha un segnale di start e un segnale di stop, dopo di che si modula il dato in RZ. Sostanzialmente sono possibili quattro simboli, ossia "0", "1", "start", "stop".

8.5 EIA RS Standard

Un ultimo standard che si intende citare é questo: il clock é piú breve del dato, in modo da non dover usare flip flop, ma solo latch. Quando lo strobe é attivo, il dato deve stare fermo. Se il dato va a zero mentre il clock é fermo si ha lo start, e viceversa. Quando si manda via il pacchetto si dá lo start, gli indirizzi, quindi ci si mette in ascolto. Se poi dall'altra parte si riceve il segnale che dice che si é pronti, si continua con la trasmissione dell'informazione.

Capitolo 9

Lezione 08

Quasi tutte le interfacce utilizzate per sistemi di tipo embedded si basano sulla RS232, ossia su di un'interfaccia di tipo seriale, come quella che é stata finora descritta. Al momento di fare un nuovo progetto, con una nuova interfaccia, cambiare tutto potrebbe essere molto complicato; per questo motivo si punta ad utilizzare sempre la seriale, con dei moduli poco costosi (dai 10 ai 15 euro) in grado di interfacciare standard piú moderni (quali ethernet o USB) alla seriale. Per aggiornamenti di un certo prodotto, dunque, utilizzare la seriale potrebbe essere l'idea migliore.

Interfacce quali ethernet o USB permettono prestazioni molto maggiori rispetto alla seriale per quanto riguarda le comunicazioni; spesso tuttavia tutto ció non é necessario, dal momento che se il sistema funzionava bene prima, allora continuerá probabilmente a funzionare bene anche senza aggiungere velocitá di trasmissione per le comunicazioni. Ció che bisogna fare in pratica é capire a priori (prima del progetto) quali siano i requisiti del sistema, le specifiche, dunque utilizzare un protocollo fatto apposta: deve essere l'applicazione a dettare i vincoli sulla velocitá dello standard da utilizzare.

Per comunicare si ha uno schema di questo tipo:

Si hanno, per la trasmissione dati, le porte 57 e 58; da 59 a 60 si ha il canale 1. I segnali in uscita da 57 e 58 sono in uscita da delle porte CMOS, dunque saranno presenti tensioni definite tra -3 V e +3 V. Al fine di trasmettere correttamente, da -15 V a + 15 V, sará necessario progettare dei traslatori di livello.

Le porte hanno tre alimentatori in sostanza: V_{micro} (tensioni per l'alimentazione di circuiti logici, dunque le classiche 5 V, 3,3 V, 3 V...); per poter avere $\pm V_{RS}$, tuttavia, servono componenti elettronici in grado di fornirle. Per fare ció si usano componenti quali l'MC1488 o l'MC1489, circuiti normalmente in vendita sui cataloghi. Questi possono essere messi nel connettore a vaschetta.

Il ricevitore ha bisogno solo di un'alimentazione, in quanto il segnale che arriva é molto intenso; al fine di fornire una corretta alimentazione ed evitare problemi, si "tosa" il segnale in ingresso, mediante il solito circuito di clamp:

Un optoisolamento puó essere sempre ottimale.

In totale, l'interfaccia al micro é un qualcosa di questo genere:

Al fine di aumentare la tensione, si puó usare un qualcosa di questo genere:

La topoogia generale é quella di un doppio-condensatori commutati: quando il sistema va a regime, ad ogni switching si ha un raddoppiamento della tensione, ottenendo di fatto una pompa di carica. Un'implementazione a MOSFET é la seguente:

Nello schema generale si é introdotto un secondo condensatore, che fa da "volano": esso sostanzialmente serve per ricaricare molto rapidamente tutto il sistema, permettendo oltre a tensioni alte anche un'elevata rapiditá. Il ICL7690 fa "questo di mestiere".

Nelle flash, nei processori, al fine di effettuare la programmazione servono tensioni molto grosse; la pompa di carica é fondamentale dal momento che permette di tirare su le tensioni e permettere semplicemente la programmazione dei dispositivi. Le flash memories sono catalogate come ROM, e si leggono velocemente ma programmano lentamente: per ogni bit scritto bisogna aspettare parecchi μs , ma anche alcuni ms, dal momento che la pompa impiega molto tempo per generare la tensione e portare la carica.

Un'idea per avere altre velocitá potrebbe essere la seguente:

Si possono utilizzare transistori programmabili, per porte parallele, in grado di ricordare le configurazioni. Un esempio in cui si fa ció, é nei televisori moderni: si memorizzano nei transistori le informazioni sulla frequenza, e si mantengono in modo da recuperarle al cambio di ogni canale. I transistori vanno su di un DAC potenziometrico che fa da "potenziometro digitale".

Idea pratica per trasmettere

Un'idea per procedere potrebbe essere questa: si puó misurare la velocitá di un impulso e quella sará la velocitá standard del sistema (sistema di autobaud, ossia di sintonizzazione automatica su di una certa frequenza); ció é bello, se non ci sono disturbi. Nel caso ci fossero, si campiona il disturbo e la sua relativa velocitá, impostandosi su qualcosa di diverso. Una volta scelto il baud rate (SCIBR), si configura la lunghezza della parola (SCICR1), dunque si abilitano trasmissione Tx e ricezione Rx (SCICR2), e si trasmette in polling (SCISR1). Ció si puó implementare mediante le seguenti istruzioni assembly:

```
Txchar:

Loop : input status;
            and mask;
            jz loop;
            output char;
```

Vediamo a che serve:

1. Per trasmettere bisogna capire se il trasmettitore é pronto; visto che non si puó mettere di brutto ció che si vuole nello shift, bisogna leggere lo stato del periferico (mediante "input status" e vedere se il buffer é vuoto. SCRDRL dice cosa c'é nell'input;

2. Operazione di mascheratura: a seconda dei bit o si dá "0" o un numero che non é 0! Se si ha "0", il JZ torna al loop, altrimenti significa che il buffer é da svuotare, mediante "output char".

Idea pratica per ricevere

Dualmente a quanto detto per la trasmissione, si puó proporre ora un'idea pratica per la ricezione di caratteri. Un'idea potrebbe essere la seguente:

1. Leggere lo stato del periferico;

2. Se c'é qualcosa andare avanti, altrimenti tornare a leggere lo stato;

3. Prendere in ingresso ció che c'é nel data register.

Si avrá qualcosa di questo tipo:

```
Rxchar:

Loop : input SR;
            and ....;
            jz loop;
            input DR;
```

Si noti una piccola variante:

```
RxcharWithRet

Loop : input SR;
            and ....;
            jz loop;
            input DR;
            ...
            ...
            call Txchar;
            ret
```

Si introduce un "ritorno con eco": si "rimanda indietro" ció che é arrivato, in modo da dimostrare al trasmettitore che é arrivato correttamente.

9.1 Background per la porta seriale

Lo schema di alimentazione del sistema elettronico complessivo, dunque anche per la porta seriale, é un qualcosa di questo genere:

In sostanza si ha un primo circuito di filtraggio, mediante due condensatori e due induttori, per le basse frequenze; si introduce dunque un diodo, atto a proteggere il sistema da eventuali inversioni di alimentazione. Si introducono a questo punto due condensatori, uno elettrolitico (grosso) e uno piú ridotto. Il condensatore elettrolitico é una sorta di riserva per le "grosse" richieste, dal momento che un semplice condensatore ceramico non sarebbe in grado di fornire molta carica se necessario, mentre il secondo condensatore, ceramico, piú vicino all'uscita, semplicemente fornisce piccole cariche, nel caso ci sia questa necessitá. L'impedenza del sistema condensatore+filo é un qualcosa del tipo:

$$Z = \sqrt{\frac{L}{C}}$$

Dove L é l'induttanza dei fili, C la capacitá del condensatore. Questo schema ha il pregio di limitare la inrush current: gli induttori limitano la variazione di corrente, dunque non si possono avere picchi di corrente nel sistema.

Capitolo 10

Lezione 09

10.1 Touchpad

Ció che spesso si potrebbe dover fare é introdurre interfacce gradevoli per l'utenza, quali un touchpad, un touchscreen, o cose di moda di questo tipo. Ció che si puó pensare é che il sistema sia un qualcosa di questo genere:

Si puó riportare sostanzialmente tutto ad un insieme di resistenze equivalenti, o meglio di potenziometri equivalenti: a seconda di dove si posiziona il dito sullo schermo si ottiene sostanzialmente un doppio potenziometro (poiché il dominio dello schermo é planare), caratterizzato da due parametri α e β caratterizzanti il valore della resistenza ottenuta per una certa posizione del dito. Si ha a che fare dunque con due potenziometri, i cui valori di resistenza possono essere stimati mediante una misura di tensione, dunque passando per un amplificatore operazionale per il disaccoppiamento e dunque per un ADC, mediante il quale si ottiene un potenziometro digitale equivalente. Spesso questi ADC sono realizzati mediante condensatori commutati, dunque il buffer é buona cosa per disaccoppiare le impedenze.

Il senso della cosa si riporta al concetto di $R_\square$: dal momento che si schiaccia, della $R_\square$ (costante in ogni punto), si seleziona un punto particolare, in grado di evidenziare un valore di α e uno di β. I processori a questo punto possono valutare il valore della resistenza, e comportarsi in modo diverso a seconda del segnale che viene inviato per questo motivo.