NUMERO: 1061 DATA: 09/09/2014

A P P U N T I

STUDENTE: Cofano

MATERIA: Sistemi Digitali Integrati

Prof. Ruo Roch

Appunti di

Sistemi Digitali Integrati

Lezioni del prof. Ruo Roch

Mario Cofano, Giulia Santoro

7 agosto 2014

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 72

Capitolo 1

Classificazione delle memorie

Le memorie possono essere classificate secondo i seguenti parametri:

- Metodo d'accesso
  - Accesso casuale –> RAM
  - Accesso sequenziale –> FIFO, LIFO, ecc..
  - Accesso per contenuto –> CAM: content accessible memory, memorie che in funzione del dato cercato restituiscono l'indirizzo in cui tale data è presente (usate per internet e nel mondo delle telecomunicazioni)
- Volatilità
  - Volatile: l'informazione scompare se si stacca l'alimentazione
  - Non volatile: l'informazione non scompare se si stacca l'alimentazione
- Scrivilibiltà
  - RO (Read Only) / RW (Read and Write) : tipi oramai poco usati
  - Mostly Read: la lettura è l'attività principale e più eseguita mentre la scrittura è effettuata in modo limitato (tipologia usata, ad esempio, nei cellulari)
  - Write One, Read Many: come i vecchi CD in cui si scrive una volta e si legge infinite volte
- Staticità

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 72

Capitolo 2

ROM: Read Only Memory

2.1 Masked ROM

La memorizzazione avviene per la presenza di un contatto fisico

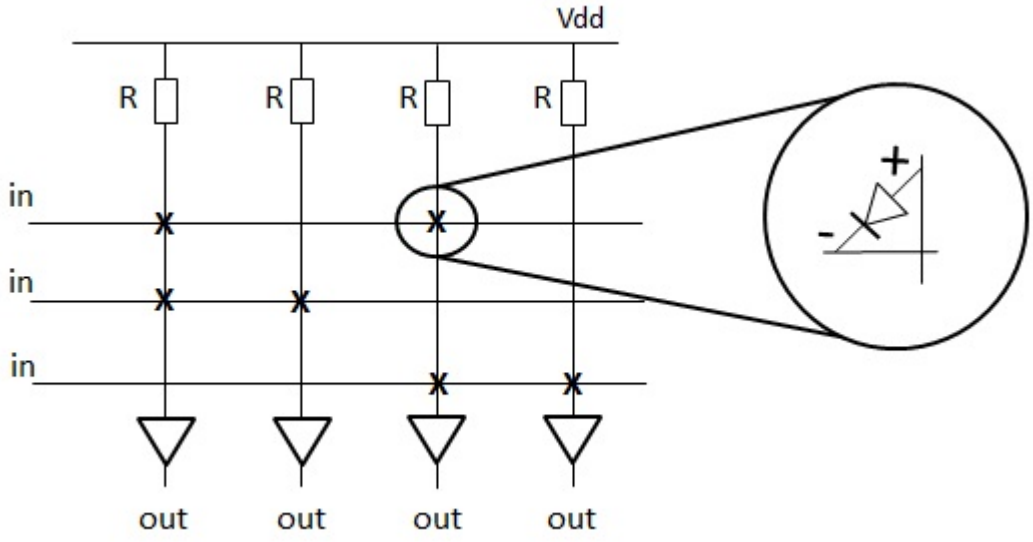

Figura 2.1

Le uscite sono fisse ad 1. Quando voglio selezionare una linea pongo in=0, in questo modo il diodo entra in conduzione portando out=0. I diodi sono connessi solo dove serve salvare degli zeri ed in questo modo si sono abbattuti i costi di produzione. Poichè il valore logico (0 o 1) da memorizare dipende dalla presenza o meno del contatto fisico, tali memorie sono non volatili ed hanno lo svantaggio che, come suggerisce il nome, possono essere programmate solo una volta per cui l'unica operazione disponibile all'utente è

2.2 PROM: Programmable Read Only Memory

La struttura è identica a quella vista prima con i diodi. L'unica, fondamentale, differenza sta nel fatto che i diodi sono posti in serie a dei fusibili e sono inseriti su tutti gli "incroci" tra Bit Line(ovvero la linea che va in output) e Word Line(ovvero l'input). Adesso è il cliente a programmare a suo piacimento la memoria semplicemente fondendo i fusibili nei punti in cui non si desidera contatto. Tuttavia ciò comporta degli svantaggi:

- Il chip non è testabile perchè testarlo equivalrebbe a programmarlo e verificarne il funzionamento (e quindi non sarebbe più una memoria programmabile dall'utente). Dunque vendere un prodotto senza testarlo e garantendo che funzioni significa che le aziende devono sfruttare processi tecnologici molto accurati in modo da ridurre quasi a zero la probabilità d'errore: questo avrebbe costi enormi

- Nel programmare la memoria, l'utente fonde il fusibile ed il metallo fuso cola chissà dove all'interno della memoria stessa: questo potrebbe provocare cortocircuiti indesiderati

2.3 EPROM: Eresable Programmable Read Only Memory

Rappresenza l'evoluzione delle PROM; infatti, invece dei diodi vengono usati i MOS con floaring gate visti prima: come sappiamo, tali MOS sono programmabili usando forti tensioni e possono essere de-programmati usando delle radiazioni ultraviolette che permettono all'elettrone di "tornare indietro" nel canale. Per poter usare tale radiazione, i package non potevano esser fatti in plastica ma in ceramica con una finestrella in quarzo con avidenti aumenti dei costi di produzione. Per questo motivo esistono anche le memorie *OTP: One Time Programmable*, cioè EPROM programmate precedentemente e vendute con i soliti package in plastica rendendo di fatto impossibile per l'utente programmare e de-programmare: i costi sono abbattuti notevolmente e l'utente perde la propria capacità di programmare ma questo poco importa perchè si è in realtà osservato che alla stragrande maggioranza degli utenti non interessa programmare delle memorie.

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 72

Capitolo 3

RAM: Random Access Memory

3.1 Static RAM

Come già sappiamo, la RAM è una memoria volatile e, in particolare, la SRAM oltre ad essere volatile, è statica: ciò significa che staccando l'alimentazione, il dato va perduto ma, se l'alimentazione non viene spenta, allora il dato sarà sempre in memoria. La memorizzazione avviene grazie a due inverter connessi in retroazione come nell'immagine:

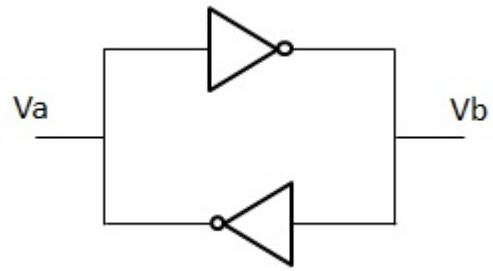

Figura 3.1

La cui caratteristica ingresso-uscita è la seguente:

Ovviamente, finchè la tensione si trova in un intorno di Vdd o 0V, il valore logico rimane stabile, mentre si parla di metastabilità nel momento in cui la tensione si discosta troppo avvicinandosi ad un intorno di Vdd/2. Il circuito di una cella della SRAM è il seguente:

Gli inverter sono rappresentati dalle coppie di transistor M1 e M2, M3 e M4, mentre M5 e M6 sono i *pass-transistor* ovvero dei MOS che, quando accesi, consentono il collegamento tra bit line (BL) e gli inverter. Come funziona la cella? In fase di lettura viene accesa la word line (WL) il che permette di

WL, il valore su BL forza il punto Q a passare da 0 ad 1 oppure da 1 a 0 a seconda della tensione su BL; è opportuno sottolineare che questo meccanismo di sovrascrivere un valore logico con un altro diverso valore è possbile solo dimensionando nella maniera oppurtuna tutti i transistor. In realtà, la cella è leggermente più complessa di quella qui rappresentata: infatti, bisogna tener conto del fatto che sulle BL sono connesse centinaia e migliaia di celle, per cui tutte le capacità parassite dei MOS si sommano generando quindi una capacità molto grossa che potrebbe rendere la lettura/scrittura molto lenta o che addirittura potrebbe "corrompere" il dato salvato generando quindi un errore. La soluzione sta nel precaricare entrambe le BL al valore Vdd/2 prima di ogni operazione in modo da caricare le capacità parassite impedendogli quindi di far cambiare il valore del dato salvato. Un altro problema è legato sempre alle capacità parassite e riguarda il tempo che la cella impiega a fornire il dato in uscita: infatti, maggiori sono i parametri parassiti e più tempo ci vorrà a leggere il dato. La soluzione sta nel collegare BL e $\overline{BL}$ ad un comparatore la cui uscita scatta ad 1 o 0 a seconda della differenza di tensione tra le due BL: essendo il valore di BL complementare a quello di $\overline{BL}$, bastano pochi millivolt per riuscire ad ottenere il dato in uscita.

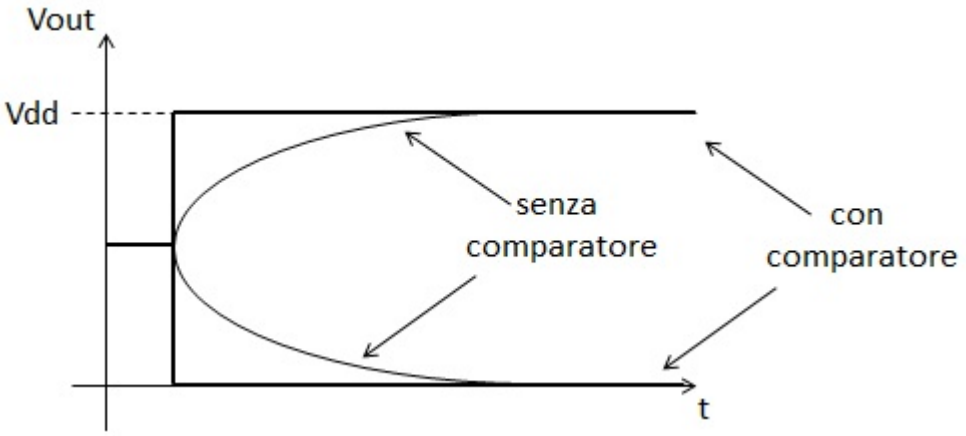

Figura 3.4

Le dimensioni di una memoria RAM sono indicate dal costruttore, ad esempio, come *128k x 8* cioè 128000 righe per 8 colonne. Ovviamente è impensabile fare una struttura del genere perchè risulterebbe essere lunghissima e stretta come un capello, ovvero molto fragile! Come fare? Dunque 128kx8=2^{17}x2^3 = 2^{20} celle di memoria, per cui si potrebbe pensare di ottenere una struttura quadrata con 2^{10} celle per lato. Tuttavia, avendo necessità di solo 8 colonne, posso usare un mux con 8 ingressi e 7 bit di selezione in modo

memorizzati i dati ed altre piccole info (come l'orario, la data, ecc..); tali RAM sono dette NVRAM, cioè Non-Volatile.

Per poter studiare una memoria di qualsiasi tipo è fondamentale comprenderne il funzionamento anche analizzando i timing diagrams, cioè i diagrammi temporali che mostrano il modo corretto in cui fornire segnali alla memoria per eseguire le operazioni disponibili. Tuttavia, prima di ciò, è importante imparare a "capire" i timing diagrams.

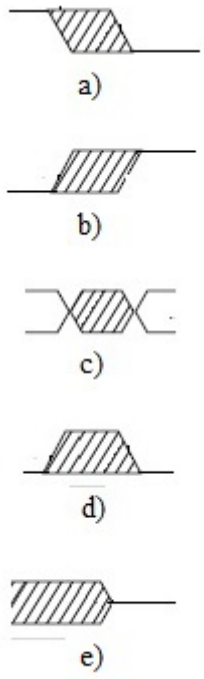

Figura 3.6

Le figure a) e b) rappresentano transizioni ripettivamente da 1 a 0 e da 0 a 1 che sono UNICHE e che avvengono in un punto qualsiasi dell'intervallo. Le figure c) e d) rappresentano il caso in cui su un bus o su una generica linea, per un certo intervallo di tempo, non è noto il valore presente. La figura e) reppresenta il passaggio ad alta impedenza (indicata come Hi-Z). Nei data sheet, tra le varie informazioni fornite, può essere indicato il *power sequencing* cioè l'ordine in cui applicare tensioni di alimentazione nei circuiti che ne presentano più d'una: sbagliando si rischia di bruciare il circuito.

Vediamo ora i timing diagram di una SRAM in caso di lettura e scrittura:

Dal timing si evince che si fornisce l'indirizzo di cui si vuol leggere il dato e contemporaneamente si attiva il CE. Solo quando viene attivato il OE parte effettivamente l'operazione di lettura. Ricordiamo infatti che affinchè

pilotare il bus dati. Anche in questo caso, per effettuare l'operazione devono essere attivi sia CE che WE e si può scegliere quale delle due usare come condizione dominante. Tipicamente le letture sono più lente delle scritture.

I segnali che abbiamo visto possono essere classificati in tre tipi:

- Dato: non ha informazione di tipo temporale; ha significato solo per il suo valore
- Clock: ha solo significato temporale
- Strobe: ha significato sia per il suo valore che di tipo temporale

Il WE, ad esempio, si comporta come un segnale di strobe poichè oltre ad abilitare la scrittura, il campionamento è effettuato sul suo fronte di salita.

SRAM Sincrone In presenza di circuiti asincroni, cioè senza un clock di sistema, è difficile realizzare un progetto soprattutto se ad alte frequenze; infatti il circuito deve essere progettato in modo da funzionare in condizioni di massimo ritardo delle porte logiche. Duqnue, per poter ridurre tale ritardo non bisogna avere catene combinatorie molto lunghe. In genere, se non diversamente specificato, la SRAM è asincrona (detta anche ASRAM); siccome ciò causava perdite in prestazioni, i produttori hanno permesso agli utenti di inserire all'estarno della RAM dei registri di sincronizzazione utili a rendere il processo di lettura/scrittura sincrono. Successivamente, invece di lasciare questo compito agli utenti, sono stati i produttori stessi ad inserire registri all'interno delle RAM creando, di fatto, delle RAM sincrone chiamate quindi SSRAM. Vengono quindi inseriti dei registri tra ogni compoente logica e sul confine realizzando ciò che viene chiamata *pipeline*. Pipelinando un sistema, cioè "spezzettando" le catene logiche in pezzi più piccoli separati dai registri, la frequenza di funzionamento cresce di molto poichè il tempo di propagazione è quello di un dato che si muove da un registro al successivo, quindi deicisamente piccolo. Ovviamente, inserendo molti registri nel cammino logico, si riscontra un aumento della latenza, ovvero del tempo che impiega il segnale da ingresso all'uscita. Possiamo quindi affermare che pipelinare un sistema comporta un leggero aumento della latenza ma un grosso aumento della frequenza di lavoro e, ovviamente, del throughput (cioè quanti MByte/s riescono a fluire attraverso il sistema).

Un sistema pipelinato è tanto più efficente quanto più si lavora in continuazione: se infatti si forniscono tanti indirizzi di seguito alla memoria, con un sistema pipelinato si ottengono i dati in uscita molto velocemente; ma se

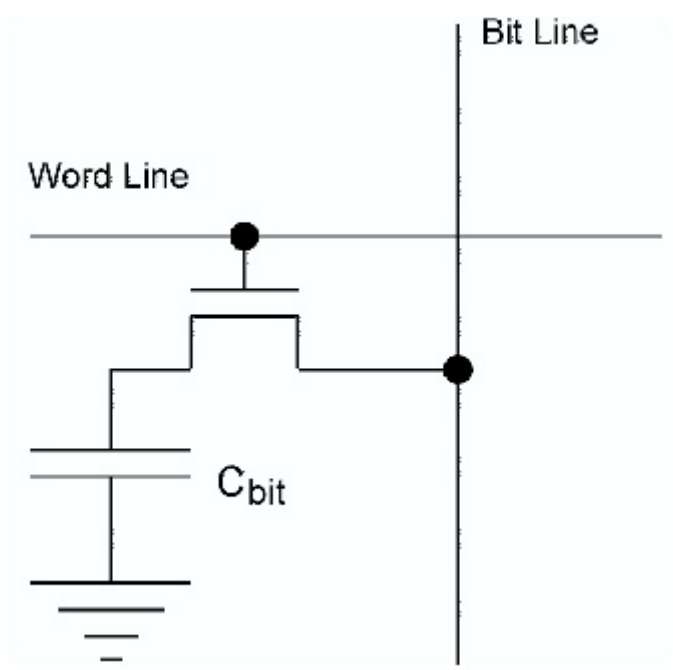

Figura 3.9

Per risovere entrambi i problemi, prima di effettuare la lettura vado a precaricare la bit line a $V_{CC}/2$:

Dopo aver precaricato la BL, viene attivato il pass transistor che quindi permette di connettere la C_{bit} alla BL: dunque la carica immagazzinata su C_{bit} finirà sulla BL che ora avrà una tensione pari a $V_{CC}/2 \pm \epsilon$, dove ϵ è il piccolo contributo di tensione fornito da C_{bit}. Allora in uscita dalla BL è posizionato un comparatore con soglia a $V_{CC}/2$:

- se su C_{bit} era salvato un 1, sulla BL vi sarà $V_{CC}/2 + \epsilon$ che, essendo maggiore di $V_{CC}/2$, porterà l'uscita del comparatore ad 1 logico;
- se su C_{bit} era salvato uno 0, sulla BL vi sarà $V_{CC}/2 - \epsilon$ che, essendo minore di $V_{CC}/2$, porterà l'uscita del comparatore allo 0 logico.

Dimostrazione Tutto si basa sul principio di conservazione della carica:

$$Q = CV$$

$$Q_{t=0^-} = Q_{bit} + Q_{parassita} = C_{bit}V_{CC} + C_{parassita}\frac{V_{CC}}{2}$$

sia in grado di rilevare ϵ, è necessario che sia affetto da un errore di offset piccolissimo.

Col procedimento descritto, dopo aver letto il dato, lo distruggo poichè su C_{bit}, dopo la lettura, ci sarà la stessa tensione della BL, cioè circa $V_{CC}/2$. Per evitare ciò, riporto l'uscita in ingresso per mezzo di un interruttore.

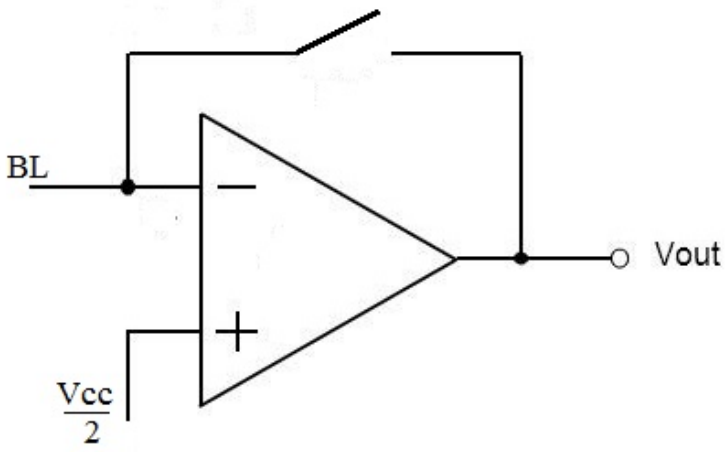

Figura 3.11

Sostanzialmente, ogni qual volta effettuo una lettura sono costretto ad effettuare una scrittura. Questo ciruito è lo stesso usato per il *refresh*, cioè per rinfrescare l'info salvata su C_{bit} che si deteriora a causa delle correnti di perdita del pass transistor: per rinfrescare l'info, basta leggerla. Quindi nelle DRAM la lettura è effettuata anche se non richiesta perchè è utile per l'operazione di refresh. Ovviamente, siccome sulla BL sono connesse molte celle, con una sola operazione di lettura posso rinfrescare il contenuto di tutte le celle connesse.

La struttura interna di una DRAM è molto simile a quella di una SRAM eccetto per il fatto che la densità è maggiore.

Il transistor su cui salvo l'informazione, cioè quello che si comporta come un condensatore, occupa un'area che è circa 1,5 volte quella di un MOS normale. Moderne tecnologie cercano di ridurre la superficie occupata da tale MOS aumentando l'area del substrato e quindi aumentando il numero di cariche immagazzinabili.

Nelle SRAM, il numero di bit di indirizzo necessari per indirizzare il dispositivo è pari a $\log_2(n.dicelle)$. Nelle DRAM viene invece sfruttato il concetto di *multiplexing address*: il bus degli indirizzi fornisce prima l'indirizzo della riga e poi, in sequenza, quelli delle colonne. In questo modo il bus degli indirizzi è ampio la metà e quindi posso usare la metà dei bit per indirizzare.

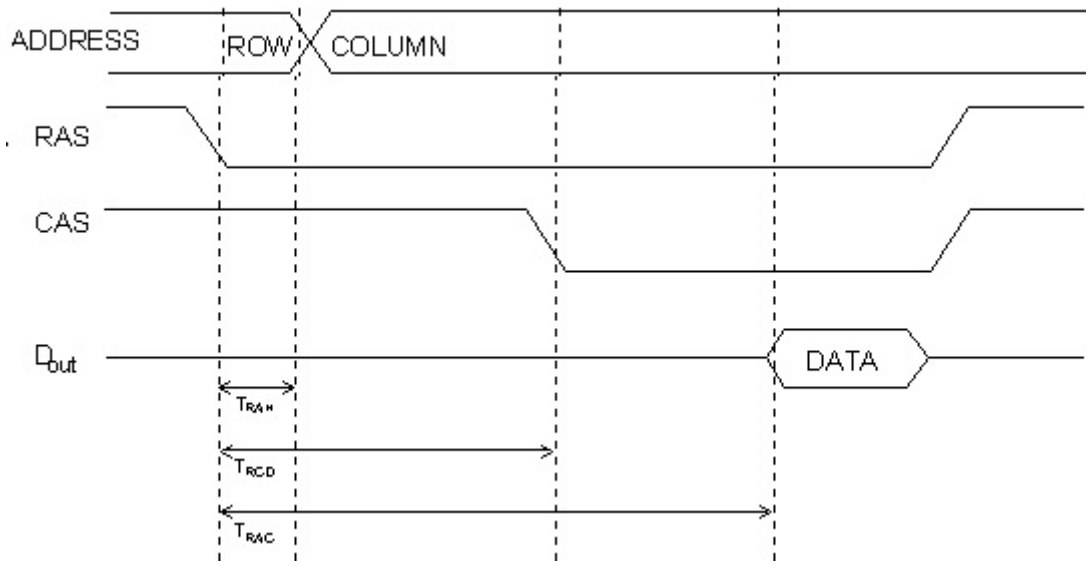

Figura 3.13

$\frac{1 refresh}{16\mu s}$. Per capire quanta banda spreco, devo capire ogni quanti cicli di lettura faccio un refresh. Suppongo che t_{read}=$t_{refresh}$=100ns (generalmente è anche più piccolo), per cui in $16\mu s$ faccio 160 letture; se devo fare un refresh vuol dire che ogni 160 letture "butto" un ciclo, cioè perdo 1/160, ovvero meno del 1% di prestazioni; sembrerebbe una perdita minima ma in realtà non è così, infatti:

$$t_{latenza_{max}} \approx 200ns(lettura + refresh)$$

$$t_{latenza_{tipico}} \approx 100ns(lettura)$$

quindi ciò vuol dire che ho un'incertezza del 100% sul tempo d'accesso! Questo può essere un problema se, ad esempio, voglio generare un'onda quadra usando delle istruzioni software (come il *loop*): in tal caso, infatti, l'onda non sarà perfettamente periodica e vi sarà del *jitter*.

DRAM Sincrone Analogamente a quanto visto per le SRAM, anche nel caso di DRAM, aumentando la velocità d'accesso alle memorie è stato necessario passare da DRAM asincrone a DRAM sincrone. Infatti, se non diversamente specificato, una DRAM è asincrona (ADRAM); altrimenti si parla di SDRAM. Per poterle renderle sincrone, si inseriscono registri di *pipeline* che, come detto, peggiorano di poco la latenza ma permettono un grosso miglioramento in termini di throughput e frequenza. In realtà, la latenza non è mai tanto bassa perchè, anche inserendo tantissimi registri, il

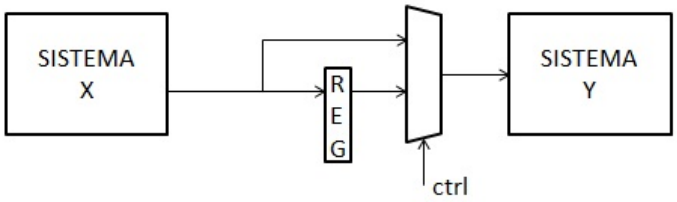

Figura 3.14

tuttavia, sul bus dati vi è un solo dato ogni colpo di clock e quindi ciò rappresenta uno spreco, ovvero "butto" il 50% della banda. Con le DDR si è risolto il problema: ogni colpo di clock vengono estratti due dati leggendoli sia sul fronte di salita che di discesa del clock:

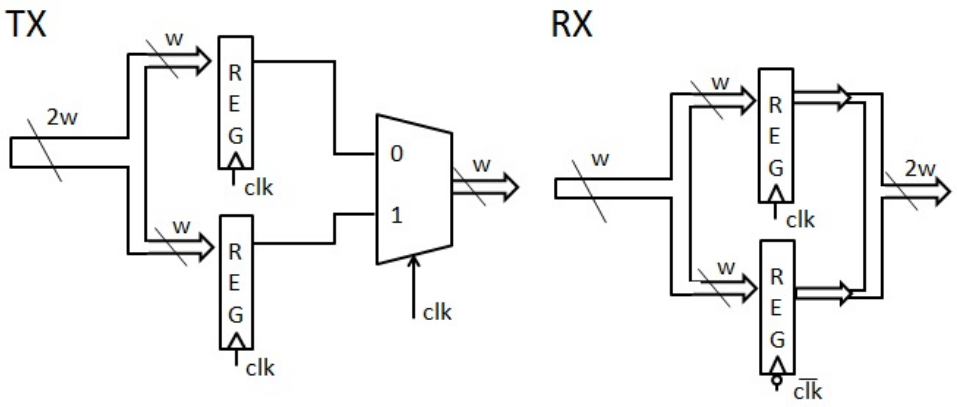

Figura 3.15

Le word sono divise in due registri e sono buttate fuori grazie ad un MUX controllato dal clock e che quindi fa uscire due dati ogni colpo di clock, ovvero un dato per livello logico (0 e 1). In ricezione si sfrutta lo stesso principio: l'uscita del MUX andrà in ingresso a due registri che campionano i dati uno sul fronte di salita e l'altro sul fronte di discesa del clock. Tutto questo funziona in teoria, ma in pratica accade che, a causa dei ritardi di propagazione, il clock non è in grado di campionare i dati nel modod corretto. Per superare questo problema, la RAM genera un segnale chiamato DQS che è in fase coi dati: all'interno dei chip, il clock della RAM e il DQS sono perfettamente allineati.

I dati e il DQS viaggiano insieme fuori dal chip in modo che siano ritardati della stessa quantità e per cui rimangono sempre in fase. Per assicurarsi di ciò, sui PCB si può notare che le piste hanno forme diverse perchè devono

Capitolo 4

Memorie FLASH

4.1 NOR FLASH

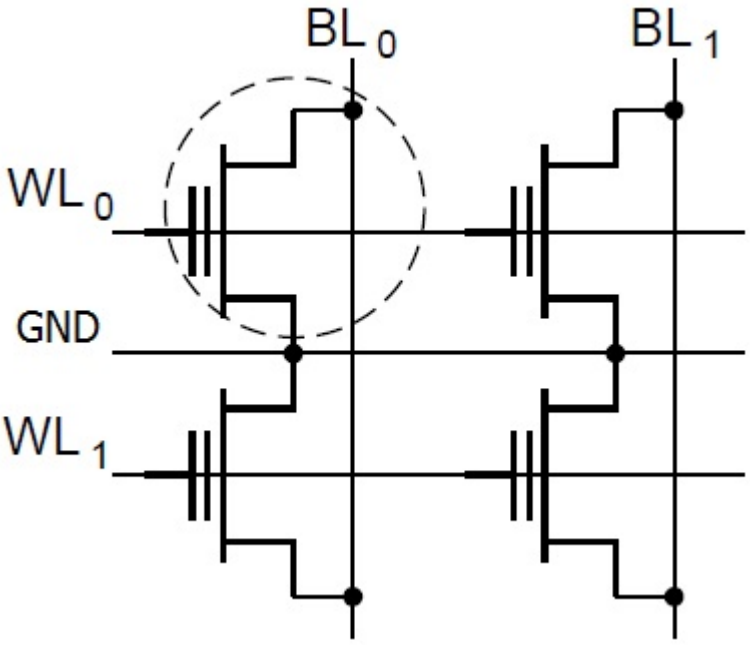

Figura 4.1

Le memorie flash sono indistinguibili da una memoria RAM per il fatto che l'interfaccia verso l'esterno è esattamente identica. A parte ciò, le due tipologie sono totalmente differenti; la prima differenza la si nota dall'immagine: la cella che memorizza i dati è completamente diversa ed è costituita da MOS con floating gate. Tale struttura è infatti la stessa di una E^2PROM.

struttura interna della memoria per effettuare operazioni di lettura/scrittura! Dunque l'interfaccia è svincolata dai dettagli tecnologici interni della flash. Nelle flash, l'operazione di cancellazione avviene per blocchi e non bit per bit. Inoltre alcuni produttori aggiungono delle caratteristiche che possono essere utili all'utente come, ad esempio, controlli di sicurezza che impediscono all'utente stesso di cancellare accidentalmente del codice. Tra questi vi sono due controlli in particolare:

- VPEN: se disattivato impedisce la scrittura; ad esempio è importante eni PC poichè cancellare, ad esempio, il BIOS, che gestisce e contiene tutte le info per poter avviare il PC, significherebbe buttare la scheda madre o rimandarla al costruttore per riprogrammarla. E' possibile anche "proteggere" solo alcuni blocchi.
- STATUS: ci informa riguardo lo stato in cui si trova una flash, cioè se ha, ad esempio, finito una scrittura.

In fase di progetto di un sistema bisgona evitare di incorrere nel fenomeno di *shortage*, ovvero bisogna evitare di inserire nel progetto un tipo di componente (come una memoria o altro) prodotto da una sola azienda al mondo; se per sfortuna tale azienda dovesse subire un incendio o altri danni e non riuscisse più a produrre tale componente, il progetto sarebbe da rifare. Per questo motivo si cerca di prendere in considerazione componenti simili tecnologicamente in modo da non stravolgere il progetto. Per evitare ciò, almeno per quanto riguarda le flash NOR, i più grossi produttori hanno creato un consorzio chiamato *CFI: Common Flash Interface*. All'interno di ogni flash prodotta con questo standard vi è una tabella (una specie di mini data sheet) che contiene tutte le info sul modello che si sta utilizzando. I comandi per leggere tale tabelle sono universalmente riconosciuti e identici per tutti i prodotti del CFI. I software accedono alla tabella, leggono gli algoritmi necessari per effettuare tutte le operazioni e da quel momento in poi iniziano ad operare. Ad esempio, per poter effettuare una scrittura in realtà bisogna effettuare due scritture: ovvero devo prima scrivere il comando che indica alla FSM l'operazione da svolgere (scrivere un bit, scrivere una word, ...) e poi scrivo effettivamente l'informazione che mi interessa. Una volta terminata l'operazione, la flash setterà il segnale di status per indicare che ha finito l'operazione. Nelle operazionei "pericolose", l'indirizzo viene fornito due volte per evitare che vi siano operazioni fatte per errore.

Esempio:

- 1 step: scrivo "40"
- 2 step: scrivo "40" + "0011"

Nelle flash più moderne vi sono dei DC-DC che convertono i 3,3V di alimentazione in 15-20V utili per effettuare operazioni di scrittura/lettura sui MOS con floating gate.

4.2 NAND FLASH

Sono più usate delle NOR flash perchè hanno una maggiore densità ma anche tanti svantaggi; tuttavia, se usate nel posto giusto, possono portare benefici. Tuttavia, minori sono le dimensioni fisiche e maggiori sono le probabilità d'errore. Le memorie NAND flash hanno la caratteristica di avere accesso sequenziale; per poter avere accesso di tipo casuale si effettua lo *shadowing*, ovvero il contenuto della FLASH viene copiato nella RAM e letto da li.

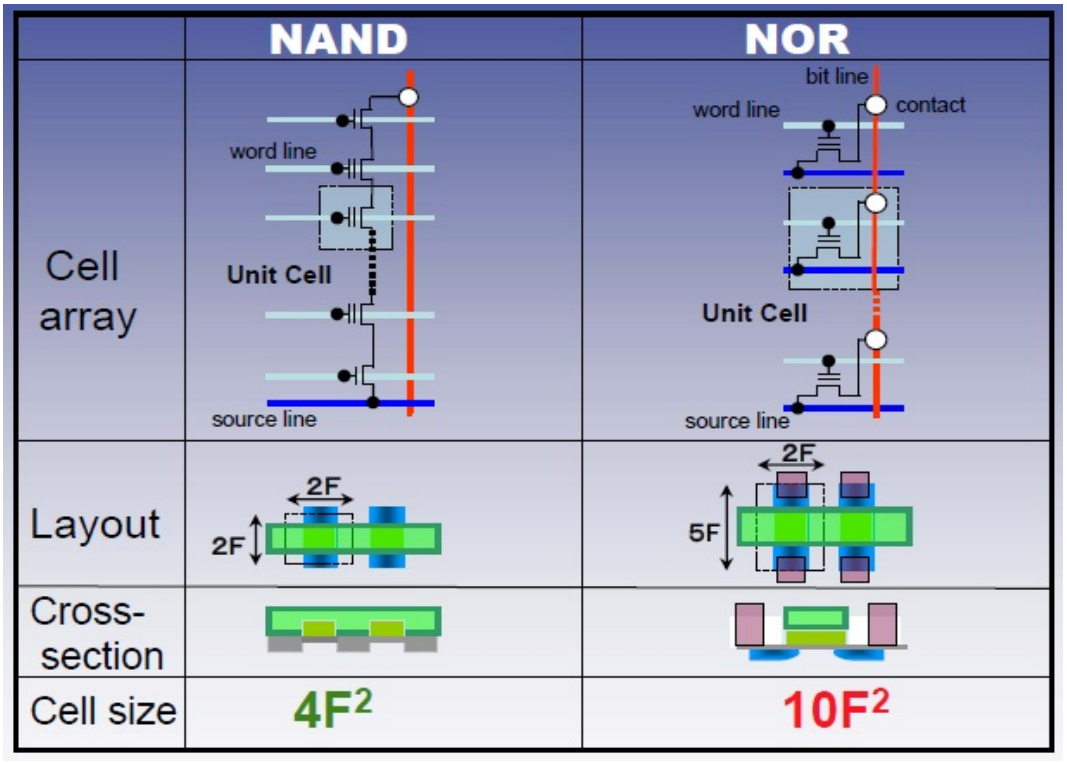

Figura 4.5

Le NAND flash nascono per poter ottimizzare lo spazio: infatti, rispetto alle NOR flash vi è un minore numero di connessioni tra MOSFET e ground e tra MOSFET e bit line. Infatti la dimensione di una cella NAND è $4F^2$ rispetto ai $10F^2$ di una NOR flash (dove F è un numero arbitrario che dipende dalla tecnologia).

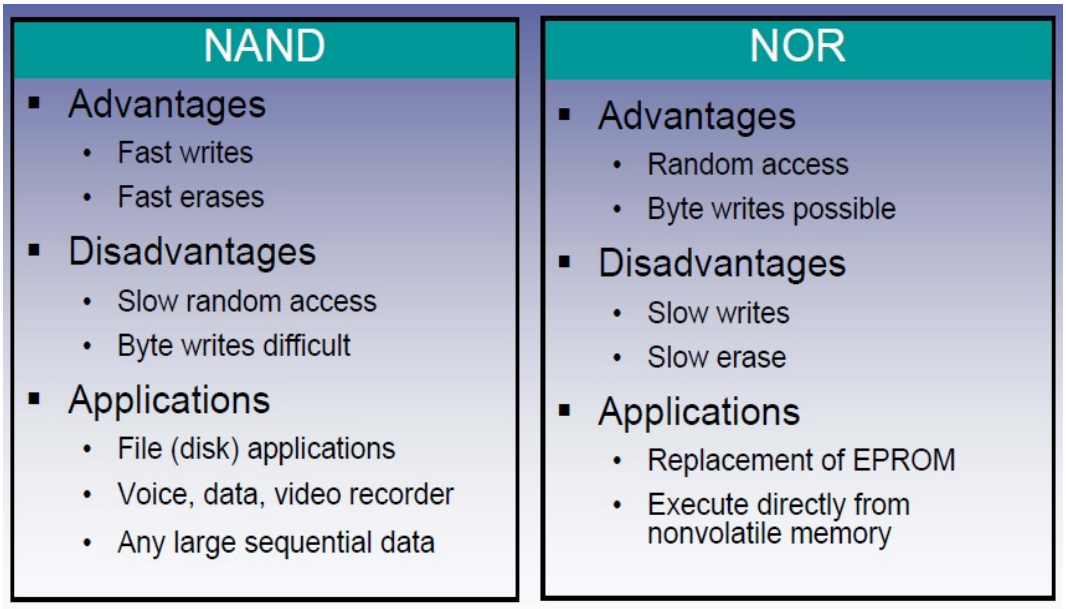

Figura 4.7

la struttura cristallina del MOS si danneggia e non si riesce più a scrivere/leggere la singola cella. Per evitare ciò, si esegue il *WARE LEVELING* ovvero si cerca di far "invecchiare" tutti i blocchi nello stesso modo cioè si evita che un blocco sia scritto mille volte ed un altro solo dieci. Come nel caso delle DRAM, l'interfaccia è di tipo multiplexato altrimenti il numero di piedini del dispositivio sarebbe eccessivo.

Nelle NAND flash le operazioni di lettura e scrittura sono migliori in termini di banda rispetto a quelle di una NOR flash; lo svantaggio è che l'accesso è sequenziale. Infatti, in caso di lettura, le prestazioni di una NAND e NOR flash sono simili ma in caso di scrittura è più veloce la NOR (180 μ s per 32 byte) della NAND (300 μ s per 2112 byte): quindi, se devo scrivere un singolo byte conviene usare una NOR mentre se voglio scrivere più byte conviene usare la NAND. Anche dal punto di vista delle cancellazioni, la NAND flash è più veloce. Attualemente le NOR flash sono limitate come memorie codice per salvarci all'interno il BIOS.

La piedinatura delle flash è sempre la stessa in modo da garantire compatibilità in aventi e indietro: ci sono infatti molti piedini NC (non collegati); in futuro, infatti, tali piedini poterbbero essere utili per avere più banda.

Tutte le operazioni di una flash sono eseguite a partire da un'unità di memoria fondamentale detta *pagina*. In caso di lettura, seleziono una riga intera e la carico in uno shift register e la lettura avviene sequenzialmente; nel frattempo, l'array di memoria può eseguire un'altra operazione.

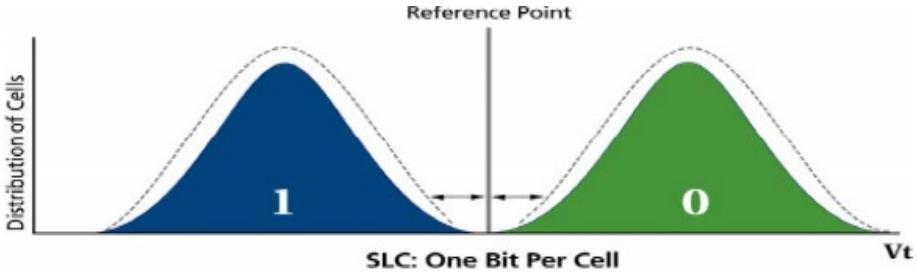

Figura 4.9

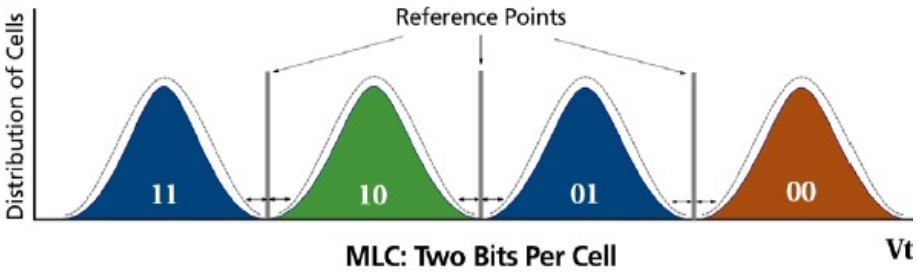

Figura 4.10

Più stati si desiderano e più precisione è richiesta; per cui avere più di 4 stati è quasi impossibile o per lo meno inconveniente. Le SLC danno la massima prestazione in termini di velocità e affidabilità e si usano nei dischi allo stato solido e in molti sistemi embedded in cui è importante l'informazione da salvare. Le MLC sono meno affidabili e per questo usate nelle applicazioni per i consumatori (memorie per cellulari, penne USB ed altro).

Le proprietà di una SLC sono le stesse di una "flash standard":

- 100.000 cicli di scrittura/cancellazione
- 1 bit di ridondanza ogni 512 byte per la ECC (error correction code)
- 25μs per lettura di una pagina
- 200-300μs per programmazione
- 1,5-2ms per cancellazione di un blocco

Le proprietà di una MLC sono:

- 10.000 cicli di scrittura/cancellazione
- 4 bit di ridondanza ogni 512 byte per la ECC (error correction code)

La NAND flash, rispetto alle NOR, ha una maggiore probabilità d'errore. In teoria, il floating gate è un isolante perfetto ma in realtà non è proprio così: infatti, dopo una decina d'anni l'informazione tende a sparire e questo rappresenta un problema per gli archivi storici.

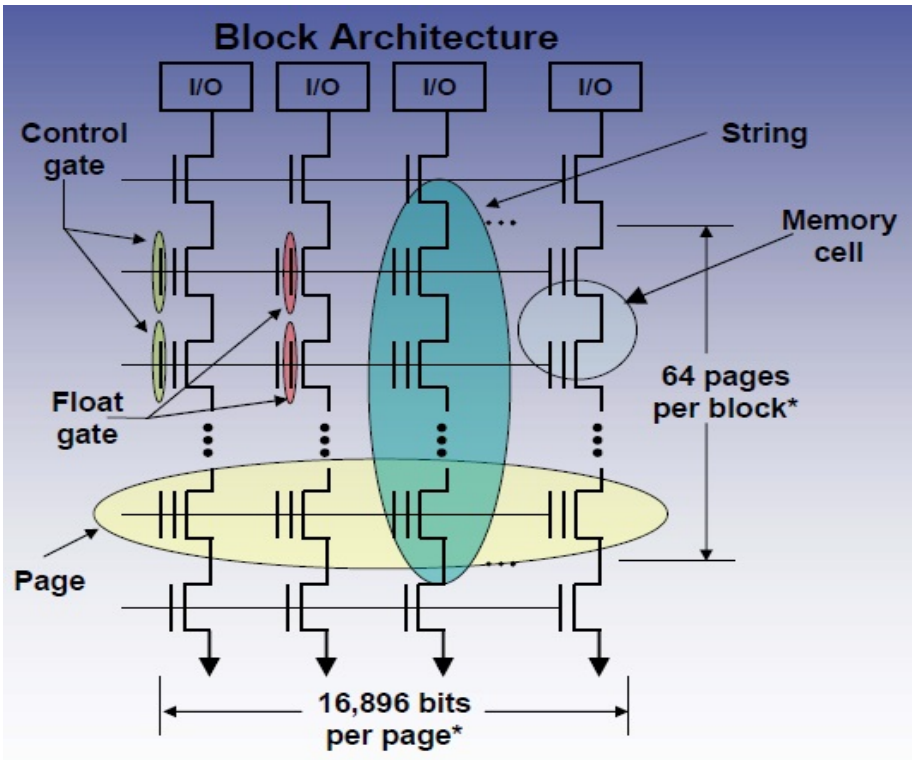

Figura 4.11

Altro problema è l'endurance, ovvero quanti cicli di scrittura/lettura si riescono a fare: il problema sorge quando, a causa delle deformazioni del reticolo cristallino, si sposta la tensione di soglia dei MOS ed è quindi impossibile scrivere o leggere. A questo punto o si butta la flash o si delimità il blocco danneggiato e non lo si utilizza più riducendo di fatto le dimensioni della memoria; esistono tuttavia tecniche per il recupero dei dati dai blocchi inutilizzabili. Purtroppo, durante la lettura o scittura di una cella, anche le celle vicine vengono danneggiate e stressate: infatti, per leggere attivo tutte le celle tranne quella di cui mi interessa il contenuto: questa condurrà oppure no a seconda dello stato. Se conduce, il bit sarà programmato. Per far ciò, devo fornire una tensione alta (circa 5V) in modo che tutti i transistor selezionati conducano anche se il loro stato è 0. Per questo motivo, quando leggo una pagina, scrivo debolmente anche le altre pagine.

non è detto che con una tensione inferiore a 20V nessuna cella si programmi; infatti anche con 10V, una cella può essere debolmente programmata (*weakly programmed*). Allora, dopo un gran numero di letture di un blocco, lo riscrivo. Tante letture incrementano la probabilità d'errore.

Quindi ogni volta che programmo un MOS, i 4 adiacenti vengono stressati: questo non danneggia le celle ma potrebbe modificare il dato salvato. Per evitare ciò si potrebbe riscrivere i dati adiacenti al bit modificato ma questo ridurrebbe di molto l'endurance. Per ciò esistono diverse soluzioni: scrivere non un bit per volta ma una pagina per volta sequenzialmente, così che ogni bit viene disturbato una volta sola. Nelle MLC si può solo scrivere una pagina per volta epr cui il problema non sussiste. Come già detto, per poter evitare che vi siano celle più "invecchiate" di altre, una μCU esterna si occupa di gestire il *ware leveling* (tale procedura non è obbligatoria per le SLC). In caso di errore interviene l'ECC (error correction code): il circuito di ECC deve essere progettato in base al numero di errori accettabili chiamato BER (*Bit Error Rate*). Ogni applicazione ha un *Target BER Zone* (es. il PC domestico può sbagliare una volta al giorno mentre un server può sbagliare una votla all'anno).

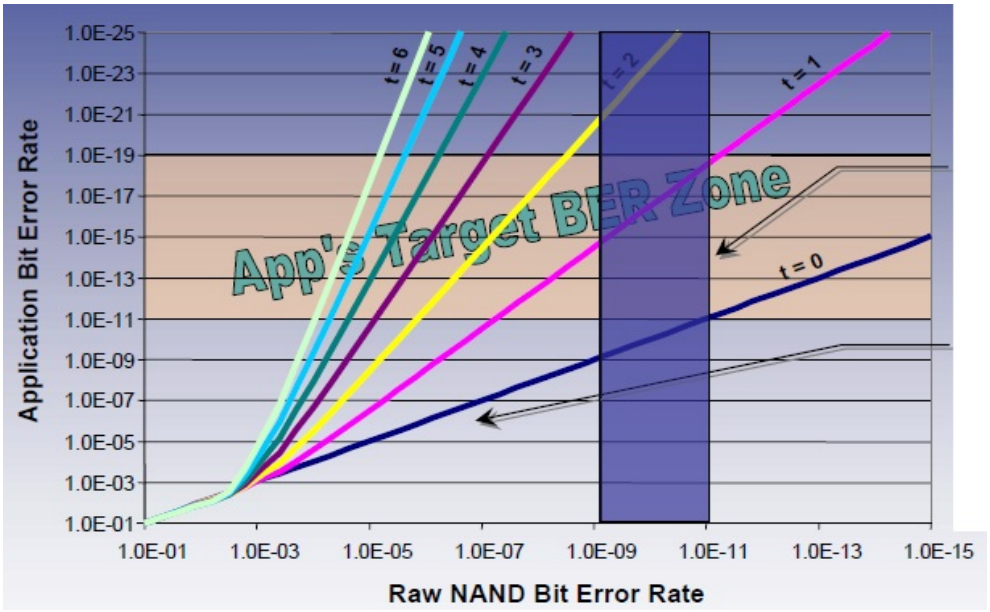

Figura 4.14

Lo step successivo alla creazione della flash è il e-MMC: non è altro che una flash con tassi d'errore bassissimi poichè dotato di un μcontrollore che si occupa di gestire il *ware leveling* e il ECC. In questo modo l'utente è

Capitolo 5

Il sistema di memoria

Un generico sistema di memoria è costituito da una memoria principale (veloce e costosa, posta vicino alla CPU e usata per salvare dati e programmi manipolati dalla CPU) e da una memoria secondaria (es. hard disk, lenta ed economica ma più grande di quella principale).

Il problema è che la CPU lavora a velocità maggiore rispetto alla memoria poichè lo sviluppo tecnologico dei due rami è differente (es. RAM dinamiche/CPU tempo d'accesso 20-30ms/1ns). Dagli anni 80 in poi, la velocità delle CPU è aumentata esponenzialmente mentre quella delle memorie non è cambiata molto in termini di latenza. Più una memoria è grande, maggiore è il percorso che la corrente deve compiere, maggiore è la latenza. Per cui si è pensato di usare memorie piccole e veloci da mettere vicino alla CPU e memorie più grandi e lente a distanza maggiore dalla CPU: a questo punto si è creata una gerarchia di memorie.

Come è possibile combinare una memoria grande e lenta con una piccola e veloce per far sembrare il tutto grande e veloce? Si usa una struttura piramidale!

L'intento di tutto ciò non è cambiare il comportamento della CPU poichè questa deve credere di comunicare con una sola memoria senza sapere che "sotto di essa" esiste tale piramide. Dato che la CPU deve credere di comunicare con una memoria piatta, la cache, oltre alle istruzioni, deve contenere una FSM (cache controller) ed un sistema hardaware che si occupa di prelevare dati dal livello di memoria inferiore se i dati richiesti dalla CPU non sono già contenuti nella cache.

I dati che leggo più spesso devono trovarsi in cima alla piramide. La CPU non deve sapere dell'esistenza della piramide per cui introduco un livello di astrazione.

Se nella cache non c'è il dato richiesto dalla CPU, la cache lo va a prendere dalla memoria principale e lo salva. Se dati e indirizzi sono presenti nella

Si parla di:

- HIT: il dato cercato da CPU è in cache
- MISS: il dato cercato dalla CPU non è presente in cache e deve esser preso dalla main memory con conseguente perdita di tempo

Dunque si definiscono:

- HIT RATE: accessi efficaci in cache/accessi totali
- MISS RATE: accessi mancati in cache/accessi totali

Se la cache è ben progettata, si ha che MISS RATE $\ll$ HIT RATE. Generalmente, il rapporto di grandezza tra cache e main memory è di tre ordini di grandezza.

Il principio di funzionamento della cache si basa sul concetto di *località*:

- Località temporale: se accedo ad una locazione di memoria in un istante "t" è molto probabile che vi riaccederò dopo "Δt"
- Località spaziale: se accedo ad un indirizzo è molto probabile che al prossimo accesso accederò ad una locazione vicina a quella precedente

La comunicazione cache-CPU è efficace poichè la statistica ci viene in aiuto

Generalmente la cache è divisa in cache-istruzione e cache-dati ma potrebbe anche non essere sdoppiata:

- Cache Unica
  - costo inferiore
  - si sfrutta tutta la memoria a disposizione; infatti se avessi la cache sdoppiata rischierei di avere cache istr. piena e cache-dati semi vuotacon conseguente spreco
- Cache Doppia
  - massima banda a disposizione

linee presenti è 64kByte/4 byte=2^{14} mentre il numero di blocks nella main è 16MByte/4 byte= 2^{22}.

Direct Mapping Si chiama direct mapping perchè ogni block della memoria è mappato in un'unica linea della cache:

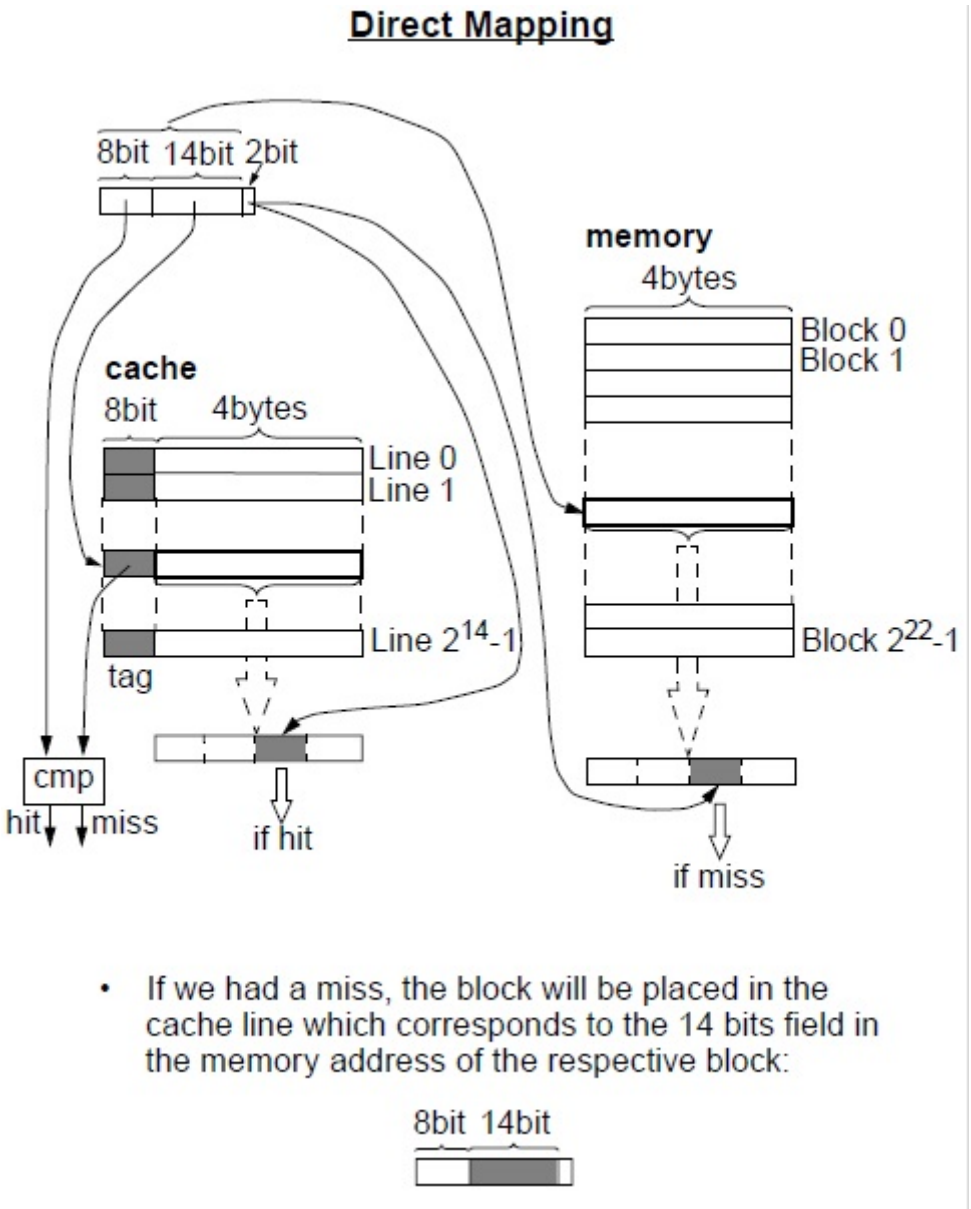

Figura 5.4

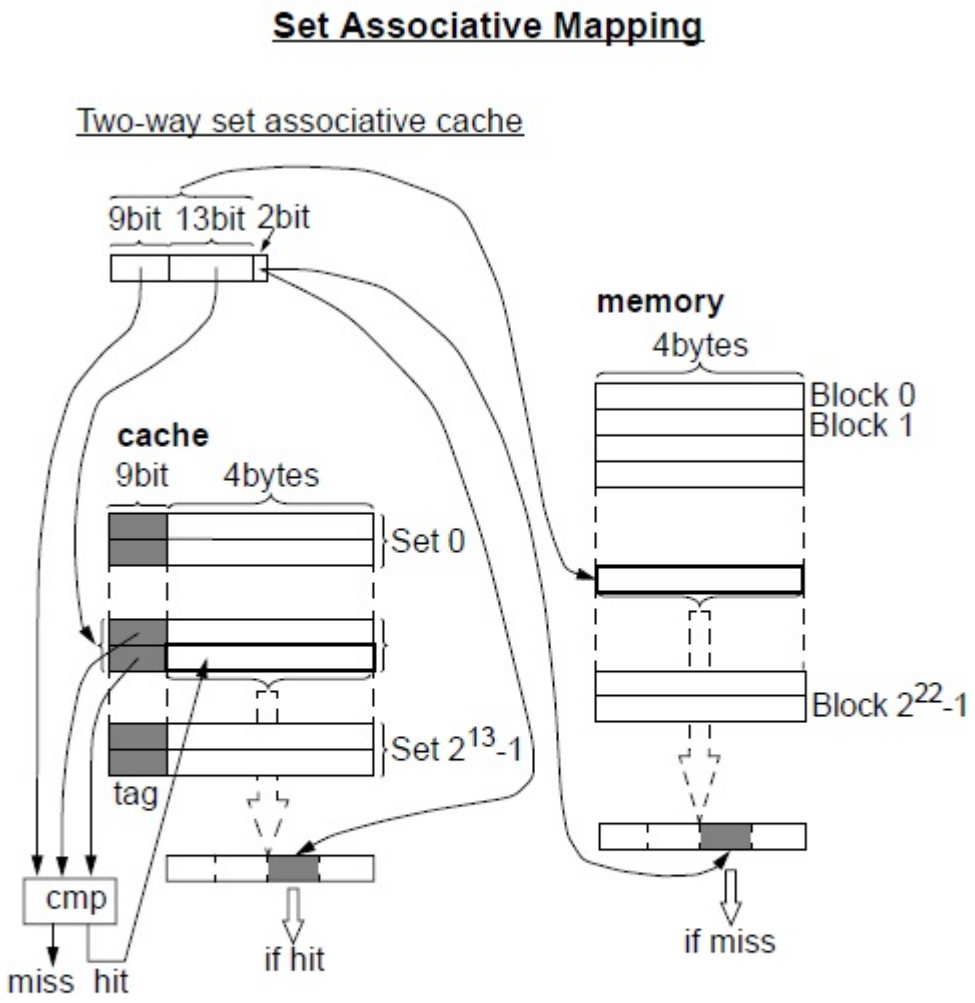

Figura 5.5

questo tipo di mapping è quello di garantire un po' di flessibilità ma si paga nel fatto che il numero di bit di tag aumenta. Nel momento in cui bisogna sostituire una linea con un nuovo block, come si fa a decidere in quale delle due possibili linee del set spostare il block? Vi sono tre possibilità:

1. 2 linee di set vuote

aggiorno la cache e scrivo il nuovo dato in un buffer posizionato all'interno del cache controller; sarà poi il cache controller ad effettuare la scrittura nella main senza che venga fermata la CPU. Tale strategia diventa fallimentare non momento in cui voglio eseguire due scritture di seguito poichè nel buffer può esserci solo un dato per volta: perderei in prestazioni! Tuttavia, anche in questo caso la statistica ci viene in aiuto: infatti, le operazioni di scrittura sono molto meno frequenti di quelle di lettura, dunque è abbastanza raro che vi siano due scritture consecuitive.

- Copy-Back: dopo una scrittura, invece di aggiornare tutta la cache, si aggiorna solo la linea della cache e con un bit viene "segnalata" la linea modificata; quando quella linea viene puntata per essere sostituita, allora viene aggiornato il contenuto nella main. Questo significa che si riesce a mantenere sempre coerenza senza però fermare ogni volta la CPU; quindi se si eseguono, ad esempio, 100 scritture sulla stessa linea, non c'è bisogno di aggiornare il contenuto della main tutte e 100 le volte, ma solo una volta, cioè quando quella linea viene puntata per essere sostituita.

Memoria virtuale Lo spazio di indirizzamento richesto e visto dai programmi è generalmente più grande di quello disponibile. Solo una parte del programma va nella main memory mentre tutto il resto è salvato nella memoria secondaria (Hard disk). Per poter essere eseguito, un "pezzo" di programma viene prima caricato nella main. Il processore fornisce l'indirizzo in cui prendere il dato e lo fornisce a livello logico (virtuale); altri componenti hardware si occuperanno di convertire l'indirizzo virtuale in locazione fisica (MMU: memory management unit).

In poche parole, lo stesso meccanismo che gestiva lo scambio di dati tra main memory e cache viene usato per scambiare dati tra memoria secondaria e main memory sempre per poter garantire che la CPU "veda" una memoria piatta quando invece sappiamo essere di tipo piramidale. Tuttavia, la grande differenza tra la comunicazione main-cache e main-memoria secondaria, è data dal fatto che nel primo caso si usa una FSM (cache controller), ovvero componenti hardware, che si occupa di spostare i dati tra main e cache; nel secondo caso, invece, è il *sistema operativo*, ovvero software, che si coccupa di spostare dati. Perchè questa differenza? Le dimensioni dei dati spostati tra memoria secondaria e main è molto maggiore di quelli spostati tra main e cache, per cui farlo in hardware non è agevole.

L'indirizzo virtuale fornito dalla CPU si riferisce ad una parte di programma che è contenuto nella memoria fisica (cache o main memory), e dunque, per potervi accedere, la *Memory Management Unit* (MMU) converte l'indirizzo virtuale in fisico. Se la CPU vuole accedere ad una pagina che non è presente nella main, si parla di *PAGE FAULT*.

Un programma è costituito da un grande numero di pagine salvate sul disco; solo alcune di queste possono essere salvate nella main e usate dalla CPU. E' il sistema operativo che si occupa della comunicazione tra main e disco e che quindi cerca di minimizzare il numero di page fault (che corrispondono ai "miss" visti nel caso di cache e main memory).

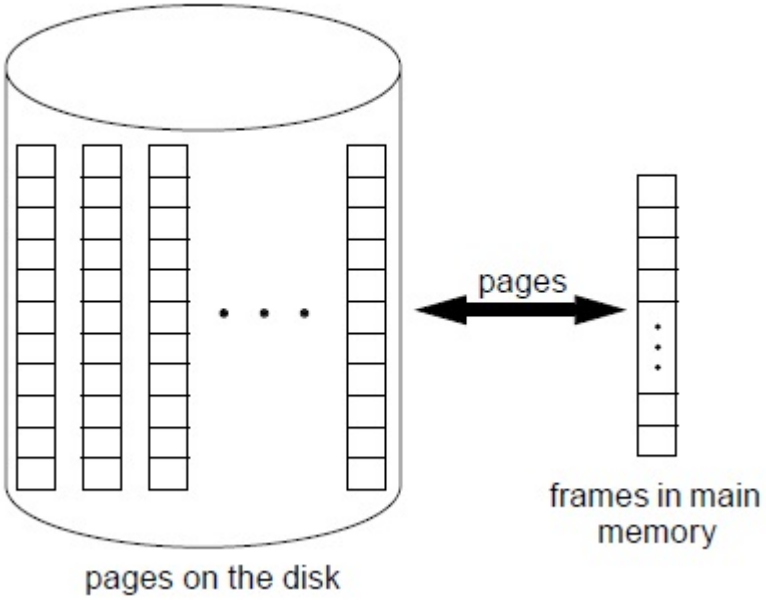

Figura 5.8

Per poter effettuare la conversione da indirizzo virtuale a fisico, la MMU usa la *page table*. L'indirizzo virtuale è costituito da *page number + offset* mentre l'indirizzo fisico è formato da *frame number + offset*.

Tuttavia se la page table è grande, si perde un sacco di tempo a scorrerla tutta per trovare il dato richiesto; per questo motivo si usa una cache apposita chiamata TLB, *translation look aside buffer*, che contiene parte della page table che, essendo grande, viene salvata nella main memory. Solo la TLB contenente l'ultima locazione di memoria usata dalla page table, è presente nel MMU. Si potrebbe anche salvare tutta la page table sul disco nel caso in cui fosse troppo grande, e quindi, per poterla usare, verrebbe suddivisa tra

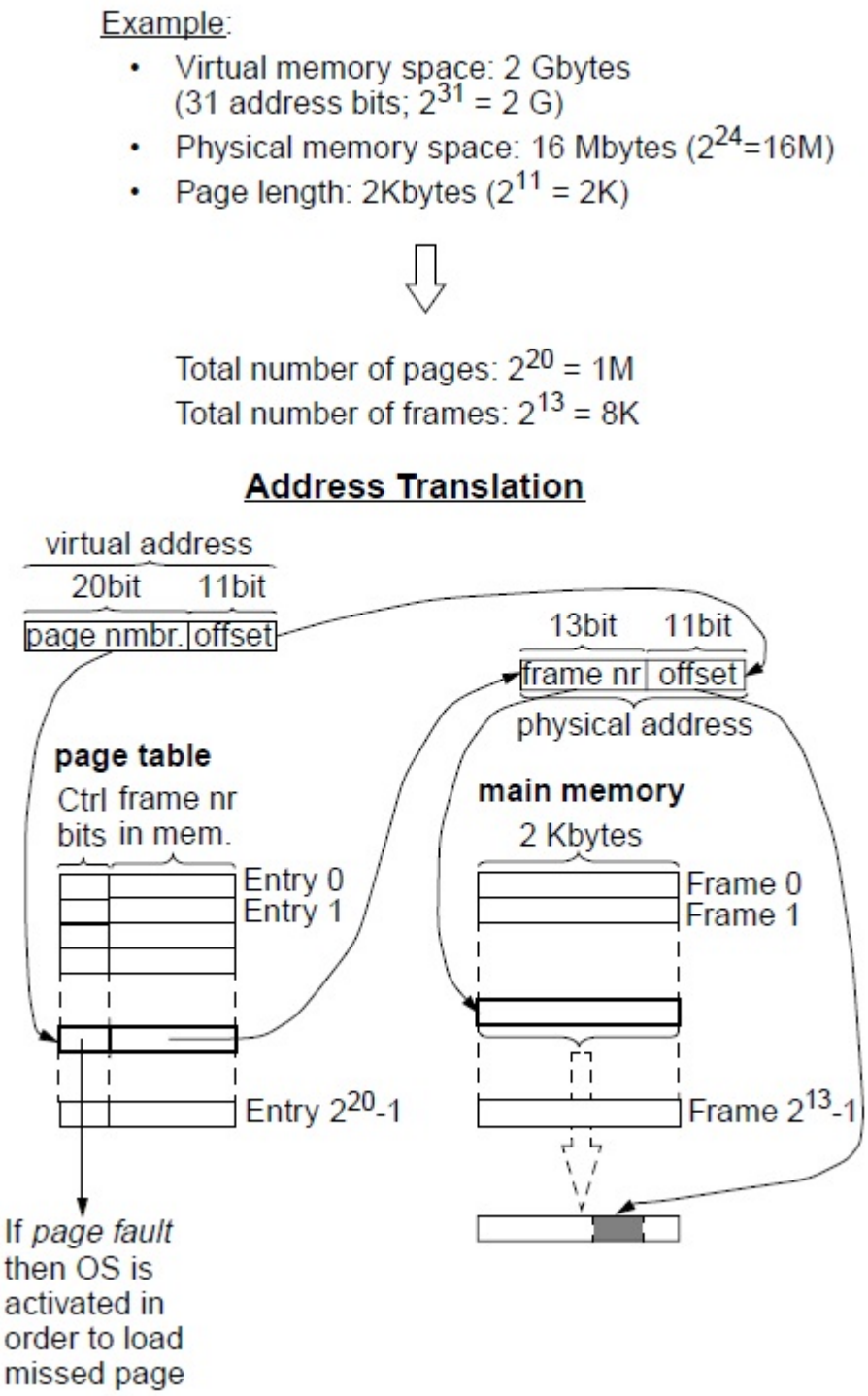

Figura 5.9: Address Translation

Capitolo 6

Microprocessore

Il microprocessore è definito in maniera univoca dall'instruction set. Viene definito un API (*Application Program Interface*), ovvero una sorta di contratto tra chi progetta il μP e chi lo usa, in cui vengono definiti formato dati, istruction set, architettura, ecc...

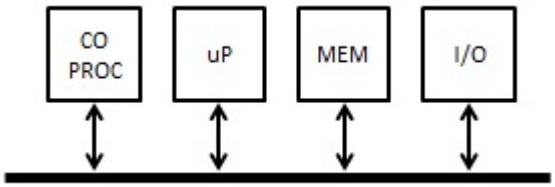

Figura 6.1

Ovviamente, se si prova a modificare l'istruction set accade che il suo volume aumenta e ne aumenta il costo; dunque il μP viene venduto meno e magari alcune delle operazioni aggiuntive non sono usate da tutti gli utenti. Per evitare ciò, si fa uso del Co-processore: questo si occupa di fornire istruzioni addizionali e le esegue in un tempo brevissimo. Il problema è che il μP passa i dati da computare al CO-P e li indirizza verso la memoria o verso i canali di I/O e questo potrebbe causare un *overrate di comunicazione*: significa che potrebbero volerci più colpi di clock per trasferire i dati e pochissimi per computarli. Ovviamente, se il CO-P esegue operazioni molto complesse (cioè in tanti colpi di clock), l'overrate diventa piccolo. Il CO-P si comporta come una periferica ma, a differenza delle periferiche, mentre esegue delle operazioni, il μP può fare altro. Esistono dei CO-P sofisticati in grado di prelevare da soli i dati dalla memoria senza che sia il μP a passarli.

Potrebbe accadere che l'operazione aggiuntiva da aggiungere sia così semplice che aggiungendo il CO-P l'overrate di comunicazione sia eccessivamente

dati sul bus.

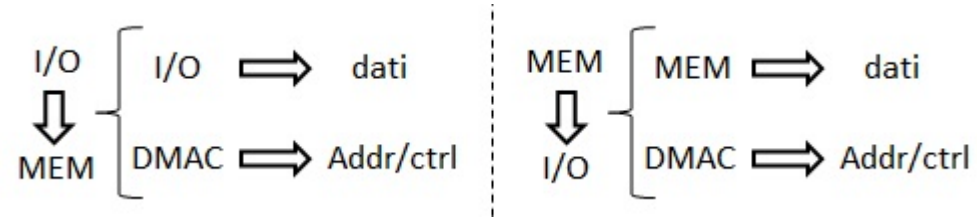

Figura 6.3

Per informare il DMAC che ci sono dei dati, o la periferica modifica un bit in un registro di stato e attraverso il *polling* si legge tale bit, oppure invia un interrupt al μP.

Come fa il DMAC a sapere quale periferica soddisfare, a quale indirizzo in memoria scrivere e quanti dati leggere? E' il μP che programma il DMAC: dopo essere stato programmato, il DMAC attiva il DMAACK, la periferica attiva il DMAREQ quando è pronta a ricevere i dati e al termine il DMAC disattiva il DMAACK.

Per poter comunicare con il μP, DMAC invia degli interrupt per informare che ha terminato oppure il μP dovrebbe stare in polling per sapere quando il DMAC ha terminato.

1. L'apparecchiatura non deve inficiare sul corretto funzionamento degli altri dispositivi e non deve irradiare oltri certi limiti

2. Il ricevitore deve essere in grado di funzionare anche se investito da una radiazione limitata

I disturbi possono essere *irradiati* se trasmessi nell'aria o nel vuoto oppure *condotti* se trasmessi lungo i cavi.

Poichè emissione ed immunità sono due aspetti strettamente correlati, risolvere i problemi legati all'emissione permette di risolvere i problemi legati all'immunità: infatti, il guadagno di un'antenna non cambia in ricezione o in trasmissione.

Se il segnale non è perfettamente periodico è sempre al di sotto di una maschera rappresentata dall'inviluppo di Fourier:

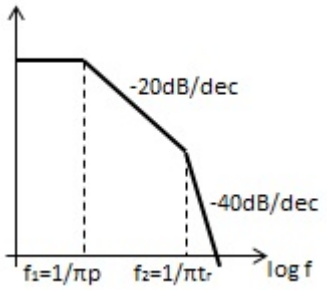

Figura 7.1

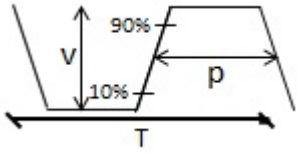

Figura 7.2

Se tale segnale fosse rettangolare e perfettamente periodico avrebbe spettro a righe ma in realtà il segnale è sempre di tipo trapezoidale.

Esistono due modi per irradiare:

- disturbo di modo comune (dipolo elettrico)

- disturbo di modo differenziale (spira)

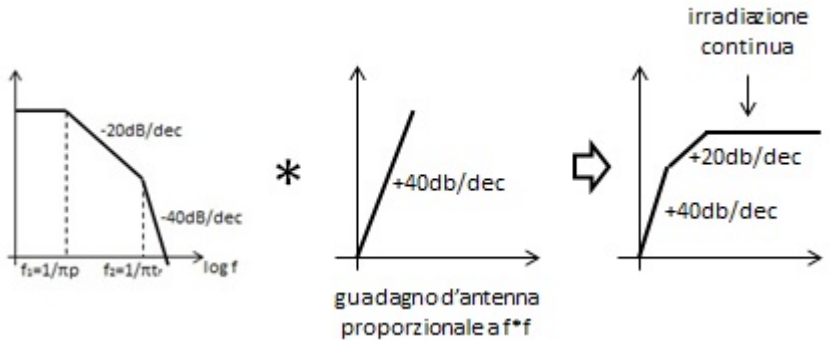

Figura 7.5

In realtà a causa del materiale di cui sono fatti i PCB i circuiti smettono di irradiare dopo i GHz e dissipano solamente. Come è possibile ridurre il guadagno d'antenna? Spostando f_1 ed f_2, dove $f_1 = \frac{1}{\pi P}$ e $f_2 = \frac{1}{\pi t_r}$;

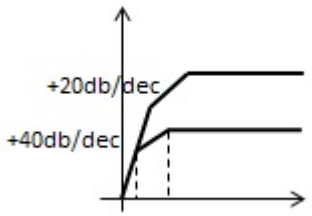

Figura 7.6

f_1 può essere spostato a sinistra evitando che i segnali abbiano commutazioni inutili mentre f_2 può essere spostato aumentando quanto più possbile t_r; quasi tutti i circuiti digitali hanno un controllo che consente di modificare lo slew rate del segnale (ad esempio la USB lavora con segnali quasi sinusoidali).

Per poter ridurre il disturbo differenziale si può anche ridurre l'area della spira introducendo un filo di massa molto vicino a quello di segnale.

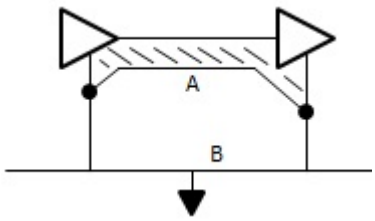

Figura 7.7

1. I_C –> I_X : la corrente del μC diventa corrente di disturbo
2. I_X –> $V_X = Z_{GND}I_X$: la V_X è riducibile con dei condensatori di disaccoppiamento
3. V_X –> $V_{CM} = V_X\alpha$: α= attenuazione e dipende da distanza sorgente-antenna; V_{CM} è riducibile allontanando sorgente del disturbo e antenna
4. V_{CM} –> $I_{CM} = \frac{V_{CM}}{Z_{ANT}}$: Z_{ANT} dipende dalla geometria dell'antenna e dal materiale che la circonda

Quindi, come detto, la corrente di segnale emessa dal microcontrollore diventa corrente di disturbo di modo comune. Come evitare tale problema? Si usano conddensatori di disaccoppiamento.

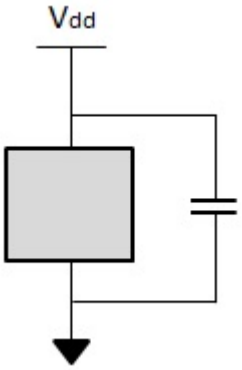

Figura 7.9

Parte della corrente di disturbo va nel piano di massa e parte scorre nel condensatore di disaccoppiamento. Per far entrare la maggior parte della corrente di disturbo, servirebbe un condensatore con impedenza nulla e questo è ovviamente un problema. Il condensatore ha un comportamento capacitivo solo fino ad una certa frequenza, dopo di che il comportamento diventa induttivo; questo accade perchè il condensatore è costituito da un filo.

La soluzione consiste nell'usare delle batterie di condensatori in parallelo con diverse frequenze di risonanza in modo da coprire i disturbi a più frequenze.

Capitolo 8

Generazione degli impulsi di clock

In questo capitolo vedremo come generare segnali di clock e come scegliere l'oscillatore da utilizzare in base al progetto. L'oscillatore più semplice da realizzare è quello a *trigger di Schmittd*:

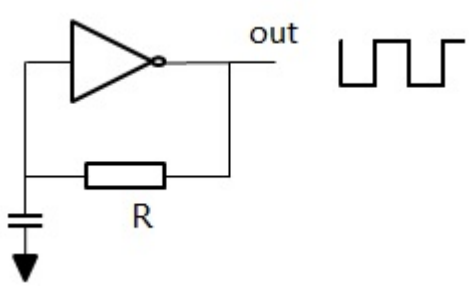

Figura 8.1

Questo oggetto è molto semplice ed economico ma non è un ottimo oscillatore a causa delle grandi tolleranze dei componenti passivi, ovvero circa del 10-15%. Inoltre nel calcolo dell'incertezza bisogna includere la tolleranza sull'ampiezza dell'isteresi del trigger che vale circa il 20%.

Complessivamente l'incertezza è di circa il 30-40%. Questo potrebbe essere un problema oppure no a seconda dell'applicazione! Ovviamente se l'applicazione è di poco importanza, anche il 50% di incertezza potrebbe andar bene, mentre se il dispositivo è di notevole importanza, è fondamentale che sia il più accurato possibile. Inoltre, la soluzione a trigger di Schmittd va bene se l'oscillatore deve essere fatto di componenti discreti; se lo volessimo integrare dovremmo usare un'altra configurazione:

Il generatore di corrente carica la capacità che, una volta caricata, raggiunge un valore di tensione che supera la cosglia e che genera in uscita del comparatore un impulso di tensione che accende il mosfet che scarica la ca-

grafico con picco a f_0 e tale picco dipende dal fattore di merito Q.

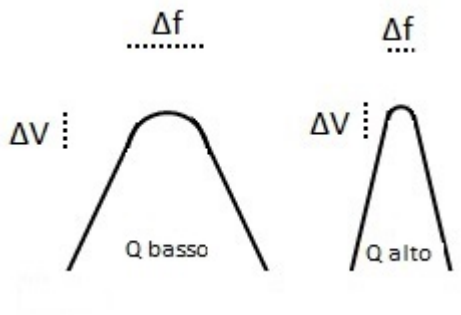

Figura 8.3

Come si nota dalla figura, a parità di variazione di tensione δV, si ha che con un fattore Q più elevato, la corrispondente variazione di frequenza a partire da quella di partenza f_0 è molto minore: avere un Q più elevato significa ridurre sensibilmente la possibilità d'errore. Questo è fondamentale nei sistemi retroazionati. Infatti, inserendo un oscillatore nell'anello di retroazione otterei un circuito che oscilla rispettando le condizioni di Barkausen. Potrebbe accadere che, a causa di tolleranze di fabbricazione e fattori esterni (come il cambiamento di temperatura), il circuito costruito per oscillare a f_0, si sposti da tale frequenza per poter continuare a soddisfare le condizioni di Barkausen. Come già detto, al variare del guadagno δV corrisponde una variazione δf tanto minore quanto maggiore è Q. E' evidente che per avere circuiiti meno sensibili alle condizioni di lavoro, sarebbe opportuno avere un Q elevato e quindi un picco molto stretto nel diagramma di Bode.

Per poter ottenere Q molto grande si usano materiali piezoelettrici, ovvero materiali che generano potenziale elettrico a seguito di deformazioni meccaniche e viceversa, cioè a seguito dell'applicazione di una tensione, il materiale si deforma meccanicamente.

Deformando il parallelepipedo di quarzo si genera una tensione che deve essere rilevata. Il quazo è molto semplice da metallizzare ovvero è molto facile depositare metallo sulle facce per creare degli elettrodi tramite cui leggere/applicare una tensione. Il quarzo è inoltre un materiale che, se applicata una deformazione, oltre a produrre tensione inizia a vibrare e siccome libera poca energia per volta, vibra per molto tempo; le vibrazioni si susseguono ad intervalli di tempo circa uguali. Dal punto di vista meccanico il quarzo è un ottimo risonatore! Per cui posso pensare di usare gli elettrodi per applicare una tensione facendolo vibrare e quindi leggere, durante le vibrazioni, in che modo vibra. La frequenza di risonanza dipende dalle caratteristiche meccaniche del pezzo di quarzo (es. se sottile f aumenta, se spesso f diminui-

che $V_{\text{IN}} = V_{\text{OUT}}$ ed in questo modo si è fissato il punto di lavoro sul tratto di caratteristica a pendenza maggiore. Il circuiito diventa allora il seguente:

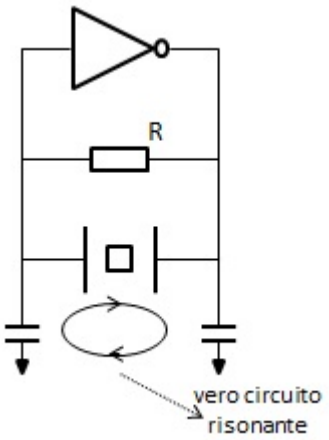

Figura 8.6

L'amplificatore è utile solo a ripristinare l'energia persa per l'oscillazione dal quarzo; al contrario, dopo un po' l'oscillazione si fermerebbe. In uscita dall'amplificatore si ottengono delle onde quadre con i fronti molto smussati a causa del fatto che i transistor usati non sono grossi. Tuttavia il problema è risolvibile connettendo a valle un altro inverter CMOS che squadra l'onda in uscita. I condensatori nel disegno sono le capacità di carico del quarzo e vengono specificate dal costruttore: sono utili per ottenere prestazioni migliori ed hanno una minuscola influenza sulla frequenza di risonanza. In genere, l'uscita di tale circuito ha duty cycle del 50% e, per averne la totale certezza, basta mandare il segnale in ingresso ad un flip flop che però ne dimezza la frequenza.

Tipi di quarzo in commercio:

- LF –> 32768 Hz (poichè se diviso per 2^{15} è uguale a 1 Hz, cioè un periodo al secondo ovvero quello che serve epr gli orologi)
- HF –> 1 MHz : 20 MHz frequenza di risonanza fondamentale

Si potrebbero usare dei tagli particolari che permettono di ottenere quarzi che oscillanoa multipli dell'armonica fondamentale ma questo potrebbe essere problematico poichè bisognerebbe usare filtri che abbattono l'armonica fondamentale e che costano molto. Una soluzione più semplice sarebbe quella di usare dei moltiplicatori di frequenza a PLL:

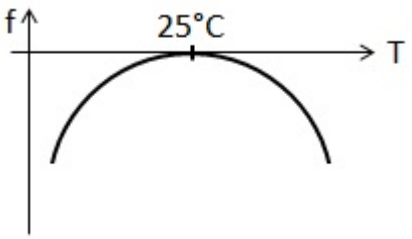

Figura 8.8

la variazione sia di tipo parabolico è ottimo poichè lavorando nel range vicino alla temperatura specificata nel datasheet, la frequenza di risonanza cambia pochissimo.

Altra causa di tolleranza è l'invecchiamento e si misura in [ppm/anno]; la variazione è dovuta allo stress meccanico ed elettrico subito dal quarzo. Ovviamente, minore è la corrente fornita e minore sarà l'invecchiamento.

Come è possibile stabilire quale è l'errore massimo ammissibile a seconda dell'applicazione? Facciamo un esempio: supponiamo che per un orologio da polso un errore di 1 secondo al giorno sia ottimo, dunque:

$$\frac{1sec}{1day} = \frac{1}{86400}sec \approx \frac{1}{100.000} \approx 10ppm$$

ovviamente 10 ppm è una richiesta molto esigente. Infatti i comuni orologi "sbagliano" molto più di 1 secondo al giorno.

E' anche possibile correggere l'errore via hardware tramite un TCXO (Temperature Compensated Cristal Oscillator): oscillatori in quarzo compensati in temperatura con resistenze NTC che variano il loro valore in base alla temperatura. Sono molto costosi poichè la tolleranza è di circa 1 : 5 ppm.