

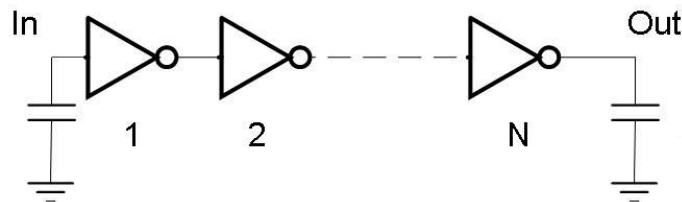
Introduzione alla microelettronica APPUNTI SESTA SETTIMANA

A cura di Luigi Lentini

Ottimizzazione simultanea di funzione e ritardo usando logical effort

Affrontiamo ora un caso molto semplice in cui si può modificare la realizzazione della funzione logica e dimensionare i transistor, con l'obiettivo di ottimizzare il ritardo totale del circuito che stiamo analizzando. Si tratta dunque di decidere, oltre alla dimensione dei transistor utilizzati, quali porte logiche è consigliabile utilizzare al fine di ottenere un guadagno capacitivo fissato a priori.

Analizziamo un semplice circuito, come quello in figura, e facciamo delle considerazioni qualitative.



Fissiamo il guadagno capacitivo $G_{TOT} = \frac{C_{OUT}}{C_{IN}} = 64$ e consideriamo, per semplificare i calcoli, che $p = 1$ e $L = 1$, dove si intende con L il logical effort dell'inverter non normalizzato. Infine, supponiamo che la funzione logica che vogliamo realizzare sia semplicemente $OUT = IN$.

Il caso più semplice è naturalmente quello di utilizzare solo due inverter. Il tempo di propagazione totale sarà dato da

$$T_p = p_1 + L_1 G_1 + p_2 + L_2 G_2 = 2p + L(G_1 + G_2) \quad \text{con il vincolo } G_1 \cdot G_2 = 64$$

$$\text{Banalmente, poiché } L_i G_i = \sqrt[n]{L_1 \cdot L_2 \dots \cdot L_n \cdot G_{TOT}} \Rightarrow G_i = \sqrt{G_{TOT}} = 8$$

Quindi, sostituendo il valore nella formulazione del ritardo, otteniamo $T_p = 18$.

Il caso successivo è quello di utilizzare quattro inverter in cascata. Adattando a tale caso le formule precedenti, otteniamo

$$T_p = 4p + L(G_1 + G_2 + G_3 + G_4) \quad \text{con il vincolo } G_1 \cdot G_2 \cdot G_3 \cdot G_4 = 64$$

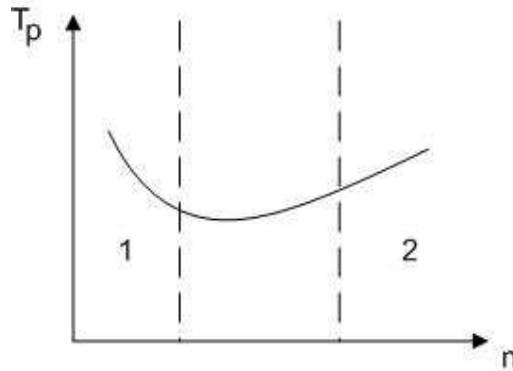
Poiché $G_i = \sqrt[4]{G_{TOT}} = 2.8$, otteniamo un ritardo di propagazione $T_p = 15$. Tale valore è diminuito rispetto al caso precedente, così come il guadagno capacitivo richiesto da ciascun inverter. Questo è naturale a causa della distribuzione uniforme della G_{TOT} .

Se invece utilizziamo un numero ancora maggiore di inverter, ponendo $n = 6$, otteniamo che il guadagno capacitivo sarà $G_i = 2$, ed il ritardo totale invece è $T_p = 18$ (è un puro caso che tale valore sia uguale a quello caso 1).

A questo punto è possibile derivare una formula generale per il ritardo di propagazione di un buffer realizzato con n inverter:

$$T_p = n \cdot p + n \cdot L \cdot \sqrt[n]{G_{TOT}} \quad \text{con } p, L \text{ e } G_{TOT} \text{ assegnati e costanti}$$

L'andamento di questa funzione, al variare di n , è rappresentato in figura.



La zona 1 è dominata dal fattore $n \cdot L \cdot \sqrt[n]{G_{TOT}}$, mentre la zona 2 è dominata da $n \cdot p + n \cdot L$, poiché $\sqrt[n]{G_{TOT}} \rightarrow 1$. Il minimo di questa funzione varia a seconda delle costanti in gioco, ma la zona di minimo per le tecnologie moderne varia tra 2 ed 8. Il guadagno 4 è un buon compromesso empirico tra numero di porte, e quindi area, e prestazioni. Spesso le prestazioni di una tecnologia sono quindi espresse come il ritardo di un inverter con $G=4$, il cosiddetto *fan-out-of-four inverter* (o FO4)

Metodologie di progettazione assistita da calcolatore dei circuiti integrati digitali

Introduzione

Nella seconda parte del corso vediamo come si riesca progettare circuiti integrati che comprendono fino a centinaia di milioni di porte logiche e miliardi di transistor.

Analizzando le componenti della moderna industria elettronica, possiamo dire che essa è caratterizzata da una gerarchia a piramide inversa: la prima parte, più ampia, è costituita dalle aziende che costruiscono il prodotto finale, cioè il sistema elettronico venduto al cliente finale, che può essere un consumatore (p.es. per televisori, computer o telefoni cellulari) o un'altra industria (p.es. per centraline controllo motore o apparecchiature per la commutazione di rete telefonica). Questa parte dell'industria si aggiudica il fatturato maggiore, perché beneficia di un diretto contatto con il cliente finale. A livello intermedio troviamo le aziende che si occupano della progettazione e produzione su semiconduttore, caratterizzata da fatturati medi e non paragonabili a quelli della parte superiore. Al livello più basso della piramide, quindi alla base dell'intera industria, vi sono le aziende (p.es. Cadence, Synopsys, Mentor Graphics, Mathworks) che sviluppano software per la progettazione assistita da calcolatore, anche detta CAD oppure EDA (Electronic Design Automation, per distinguerla dal CAD meccanico). Questa parte, pur essendo di fondamentale importanza per mantenere funzionante il resto, ha un fatturato minore di vari ordini di grandezza rispetto ai livelli superiori, pagando un contatto praticamente assente con il cliente finale.

In realtà, il CAD è necessario per qualunque passo del progetto e realizzazione di dispositivi su semiconduttore, per permetterne la specifica, sintesi logica, layout su silicio e verifica della correttezza.

Il CAD deve soddisfare le richieste di una tecnologia caratterizzata da un numero di transistor sempre maggiore, e frequenze di funzionamento che aumentano di pari passo (Legge di Moore), basandosi sull'*International Technology Roadmap for Semiconductors* (ITRS) per prevedere le richieste e le problematiche future.

E' da notare che la velocità che aumenta esponenzialmente secondo la legge di Moore è solo quella interna dei transistor, quella visibile a livello di sistema varia molto più lentamente, perché dipende dalle resistenze e capacità parassite delle interconnessioni che non diminuiscono, anzi crescono relativamente alle caratteristiche del transistor. Ciò indica che è necessario trovare delle nuove soluzioni architettoniche per migliorare le prestazioni globali, ed anche in questo caso è di supporto lo strumento CAD.

Un esempio può essere quello della pipeline di un processore, per cui attualmente il tempo di esecuzione delle istruzioni è paragonabile al tempo necessario per riportare indietro i dati al register file. E' necessario quindi introdurre stadi di pipeline anche sulle interconnessioni, per mantenere la frequenza di clock richiesta.

E' da notare che il CAD ai livelli più alti di astrazione serve anche per decidere quante e quali parti del progetto vanno realizzate in hardware o in software. Esso deve anche spesso modellare anche il mondo reale in cui il sistema elettronico dovrà operare. Esempi classici sono rappresentati dallo studio della propagazione delle onde elettromagnetiche all'interno di aree urbane, con la difficoltà di progettazione causate da rifrazioni e diffrazioni sui palazzi o alberi, oppure la modellazione dell'orecchio umano utile per realizzare un sistema di ricezione audio di buone prestazioni.

Problematiche

Le problematiche della progettazione di circuiti integrati possono essere classificate in quattro tipologie:

- *Aumento della complessità*, causato sia dal sempre più elevato numero di transistor, sia dalle applicazioni stesse per cui i dispositivi vengono realizzati.
- *Geometrie sempre più piccole*, che rendono problemi originariamente trascurabili sempre più importanti. Per esempio adesso è necessario considerare come fonti di ritardo in un circuito anche le diafonie, o *crosstalk*, tra interconnessioni e le resistenze parassite delle medesime.
- *Eterogeneità dei componenti*, poiché la legge di Moore consente di integrare sullo stesso circuito molte CPU, memorie e blocchi specializzati. Inoltre le applicazioni spesso richiedono questa eterogeneità. Per esempio un telefono cellulare richiede un microcontrollore, uno o più Digital Signal Processor, molte memorie e blocchi specializzati per la gestione del canale radio e la codifica e decodifica di immagini.
- *Cicli di progetto molto rapidi*, per cui si hanno tempi ridottissimi per analizzare il mercato, progettare il circuito o il sistema e farlo uscire sul mercato. Tempi dell'ordine di 6 mesi per il ciclo completo non sono inusuali per l'elettronica diretta al consumatore.

Senza la progettazione CAD, la legge di Moore non potrebbe essere seguita. Infatti la complessità dei problemi elettronici è cresciuta in maniera praticamente esponenziale, mentre le capacità umane sono rimaste essenzialmente le stesse. Ogni qual volta i progressi tecnologici e il divario tra abilità umana e legge di Moore mettono a rischio quest'ultima, avviene una vera e propria "rivoluzione" nel CAD e nei metodi di progetto. Le "rivoluzioni" passate sono, per esempio:

- Negli anni 70 la prima rivoluzione consistette nell'utilizzo di strumenti CAD per il disegno dei layout, precedentemente eseguito con carta e matita (o mylar e taglierina) dal progettista.
- Negli anni 80 vennero creati gli strumenti CAD che permisero la generazione automatica di layout a celle standard, risolvendo i problemi di piazzamento e l'interconnessione.

- Negli anni 90 sono stati introdotti i programmi di sintesi logica, che permettono anche la realizzazione e l'ottimizzazione automatica della funzione logica che il sistema deve realizzare.

La prossima “rivoluzione” potrebbe essere caratterizzata dalla sintesi (cosiddetta “comportamentale”) a partire da una descrizione in un linguaggio di alto livello, per esempio scritta in C, C++ o SystemC (una libreria software che permette di usare il C++ come linguaggio per la specifica dell'hardware, o HDL).

Flusso di progetto di circuiti integrati digitali

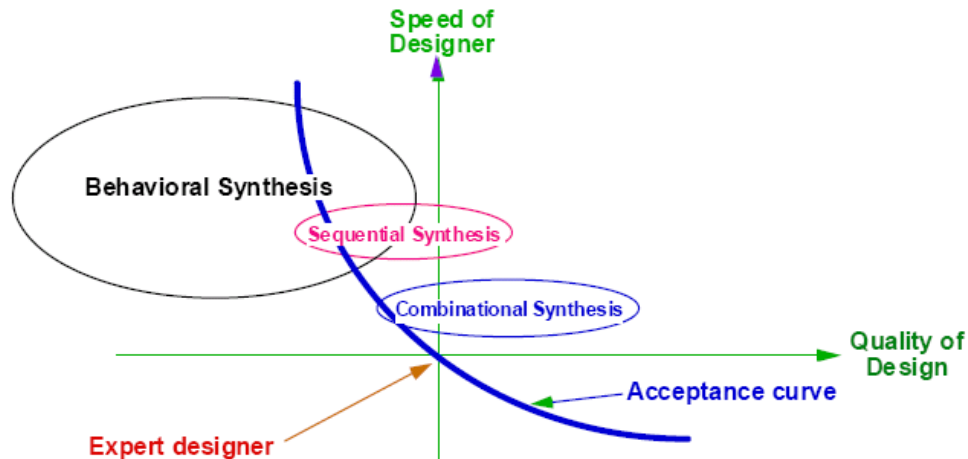
I passi fondamentali sono tre:

- *Specifica*, che definisce le funzionalità necessarie e i vincoli, per esempio di costo, prestazioni e potenza, e che può essere il risultato di un passo di implementazione precedente.
- *Implementazione*, che crea una nuova rappresentazione più dettagliata a partire dalla specifica.
- *Verifica*, che consiste in un controllo della correttezza dell'implementazione rispetto alla specifica (verifica dell'implementazione, implementation verification) o della specifica rispetto alle intenzioni del progettista descritte, per esempio, in un documento (verifica del progetto, design verification).

Le specifiche consistono nella definizione della funzionalità richiesta dall'applicazione, nell'analisi e nel vincolo dei tipici valori di merito di un dispositivo elettronico (area, consumo di potenza, velocità di esecuzione/ritardo), ma anche nell'analisi delle soluzioni proposte dai concorrenti, nonché nel confronto con standard mondiali o nazionali. Bisogna inoltre decidere quali parte del sistema implementare tramite:

- hardware dedicato, perché critiche rispetto a prestazioni o potenza consumata;
- DSP o FPGA, perché meno critiche e riconducibili a operazioni tipiche del digital signal processing, quali moltiplicazioni e somme tra elementi di vettori;
- software, perché richiedono più flessibilità e minore costo di progetto.

La progettazione hardware può partire da vari livelli di astrazione, quali per esempio transistor, porte logiche, trasferimento tra registri (RTL) o comportamentale. Ogni livello permette a una persona di progettare in un giorno un numero di transistor che è approssimativamente superiore di un'ordine di grandezza rispetto al livello inferiore. I livelli di astrazione più elevati quindi hanno una produttività elevatissima, ma la qualità dei risultati è talvolta minore, a causa della minore maturità degli strumenti di progetto. La relazione tra qualità e velocità della progettazione ed il suo effetto sull'accettazione di un livello di progetto è visualizzata in figura.



Livelli di astrazione, e relativi strumenti di progetto, che si trovano a destra della curva che rappresenta l'efficienza di un progettista esperto, sono ben accetti, perché rappresentano un miglioramento. Quelli che si trovano a sinistra (per esempio sintesi comportamentale), devono maturare e spostarsi a destra per essere accettati.

Il meccanismo attualmente più usato per rappresentare la specifica della funzionalità e per verificare sia il progetto sia l'implementazione è costituito dai linguaggi per la descrizione dell'hardware (Hardware Description Languages, HDL). Essi hanno due caratteristiche non presenti nei classici linguaggi di programmazione software: la concorrenza (per specificare e rappresentare il parallelismo) e la temporizzazione (cioè eseguire un'operazione a un certo istante e con un certo ritardo).

La verifica del progetto controlla che un modello HDL, usato come specifica iniziale dal flusso di progetto CAD, sia consistente con le intenzioni del progettista e del cliente finale, espresse tipicamente in documenti testuali. Le tecniche di verifica sono le seguenti:

- Dimostrazione matematica di una determinata funzione/proprietà del sistema. È molto utile, ma è poco diffusa perché richiede conoscenze di logica matematica che solo pochi progettisti possiedono.
- Simulazioni, effettuate applicando degli stimoli agli ingressi ed analizzando le uscite. È la tecnica maggiormente utilizzata perché rappresenta un compromesso tra velocità, costo e qualità dei risultati.
- Emulazioni, utilizzando dei dispositivi hardware che implementano una copia del sistema progettato. È una tecnica molto veloce ma molto costosa rispetto alle simulazioni.
- Prototipo dell'hardware, possibile solo se i vari componenti sono già disponibili in forma di circuiti integrati separati.

La verifica dell'implementazione invece controlla che il risultato sia consistente con le specifiche iniziali. Una progettazione basata su CAD introduce teoricamente pochi errori, ma la verifica è comunque necessaria a causa di errori del software utilizzato o modifiche manuali. Le tecniche di verifica sono le seguenti:

- Dimostrazione matematica di equivalenza tra specifica e implementazione. È molto usata per la verifica di equivalenza logica, per esempio tra RTL e porte logiche, e per la verifica del ritardo massimo di una logica combinatoria.
- Simulazioni, effettuando applicando degli stimoli agli ingressi ed analizzando le uscite, che verifica anche le prestazioni ma richiede tempi molto maggiori.

Un'ulteriore verifica, chiamata collaudo, deve essere poi fatta dopo la fabbricazione del circuito stesso, per via dei possibili errori che possono avvenire nel processo di realizzazione su silicio.

Esistono diverse tipologie di simulatori, in ordine crescente di velocità e decrescente di precisione:

- Simulatori a livello circuitale o elettrico, per esempio Spice;
- Simulatori a eventi, che modellano i cambi di valori dei segnali tra porte logiche o tra gruppi di registri, per esempio NCSim, Modelsim e VCS;
- Simulatori basati sul ciclo di clock, che non modellano i ritardi (spesso sono gli stessi simulatori a eventi che possono essere usati anche in questo modo);
- Simulatori basati su schemi a blocchi funzionali, per esempio Similink, che sono usati solo per specifici ambiti di applicazione.

Per quanto riguarda l'implementazione del circuito integrato, possono essere seguite tre diverse strade:

- Implementazione *automatizzata*, che richiede una specifica RTL, l'ottimizzazione della logica generata, ed infine la generazione automatica del layout sul silicio;
- Implementazione *manuale*, che richiede il progetto a livello di layout degli interi circuiti, non solo delle porte logiche, e quindi è possibile solo per circuiti piccoli, quali le celle base di ALU o le memorie, o per sistemi analogici;
- *Generatore di moduli*, che crea dei layout o dei circuiti logici parametrizzati direttamente, senza usare RTL. Spesso viene usato in abbinamento con l'implementazione manuale delle celle base per ALU, register file o memorie.

Nella seconda parte del corso ci occupiamo della prima strategia, mentre alcuni aspetti della seconda sono stati coperti nella parte iniziale.

Flusso di progetto RTL

Un elemento fondamentale di questo flusso di progetto è la biblioteca (anche detta libreria o *library*) di porte logiche base, quali NAND, NOR, MUX, latch, FF, che rappresenta a livello astratto il layout, in modo da poterlo usare per la sintesi logica. Per esempio la biblioteca rappresenta per ogni porta la funzione logica, il ritardo interno p , la capacità di ingresso C_{IN} , il logical effort L , le dimensioni del layout, la posizione dei contatti di ingresso e uscita, la dissipazione di potenza.

La sintesi iniziale del modello RTL genera una rete logica che lo realizza, ma senza ottimizzazioni. La successiva *sintesi logica* cerca quindi di ridurre l'area complessiva, il ritardo totale e la potenza dissipata.

Dopo l'ottimizzazione otteniamo una rete di porte logiche da disporre sul silicio. Il layout viene realizzato in due fasi:

- il *piazzamento* decide dove mettere le varie celle in modo da non sovrapporne i layout, anch'essi contenuti in forma astratta nella biblioteca di celle, e da minimizzare la lunghezza stimata delle interconnessioni, da cui dipendono le prestazioni.
- *l'interconnessione* crea le linee di metallo e i contatti che connettono tra loro le porte.

Le verifiche di implementazione sono eseguite a ogni passo, controllando *l'equivalenza* della funzione logica con la specifica e verificando che il ritardo massimo sia entro i limiti stabiliti. Se l'implementazione è in forma di layout, strumenti di *estrazione* lo trasformano in una interconnessione di porte logiche.

Dopo la produzione del circuito integrato vero e proprio è necessario *collaudare* che non ci siano stati degli errori (guasti) durante la fabbricazione sul silicio. I difetti fisici possono dipendere da moltissimi fattori (corti circuiti tra interconnessioni, impurità, metalli erosi per errore, interconnessioni o transistor più piccoli o più grandi del previsto). Essi possono causare errori fatali, che compromettono il funzionamento dell'intero circuito, o causare differenze nei parametri come I_{DSAT} , V_T .

Il collaudo viene eseguito da macchine (molto costose) che automaticamente applicano vettori di valori agli ingressi e verificano la correttezza delle uscite come funzione e ritardo.

Per generare in modo automatico i vettori di ingresso che permettono di collaudare il circuito, vengono utilizzati dei modelli logici semplificati dei guasti, perché il CAD non è in grado di utilizzare modelli fisici dettagliati. Attualmente si usa il modello *stuck-at*, che prevede che ogni ingresso guasto di una porta logica possa essere fissato a 0 o ad 1 (*stuck-at-zero* e *stuck-at-one*).

Questo modello è molto lontano dai guasti fisici dei dispositivi MOS, ma continua a essere utilizzato per due ragioni fondamentali:

- È l'unico modello usabile in modo efficiente dagli attuali programmi per la generazione automatica dei vettori di collaudo (Automated Test Pattern Generation, ATPG);
- Si è dimostrato empiricamente che, se il circuito passa questo tipo di collaudo, ha un'altra probabilità di corretto funzionamento durante il suo uso normale.

I programmi di ATPG sono in grado di analizzare e collaudare solo reti combinatorie, non sequenziali. Questo problema viene affrontato permettendo di scrivere e leggere i valori dei flip-flop direttamente dalla macchina di collaudo, riportando il collaudo a livello puramente combinatorio. I registri vengono concatenati in fase di collaudo come un unico registro a scorrimento seriale, usando un multiplexer che ha due posizioni: collaudo seriale e funzionamento normale. Le fasi del collaudo di ogni guasto sono:

1. caricamento seriale all'interno dei registri dei valori di ingresso che farebbero assumere al segnale guasto il valore opposto (*eccitazione*);
2. *propagazione* attraverso la logica combinatoria e memorizzazione nei registri di uscita dei valori risultanti;
3. scaricamento seriale dei risultati e confronto con i valori attesi.

Per i microprocessori, in cui è possibile leggere e scrivere i registri facilmente sotto controllo software, spesso il collaudo avviene eseguendo programmi di collaudo che, se eseguiti correttamente, permettono di considerare funzionante il dispositivo.

La fase di collaudo costa molto, in termini sia di tempo sia di denaro, ma non può essere saltata, perché si rischia di mettere sul mercato componenti non funzionanti, e quindi di perdere clienti.

Sintesi logica

In generale, una funzione logica è una funzione definita come $f(B^n) \rightarrow B$, cioè caratterizzata da n variabili ciascuna con dominio $B = \{0,1\}$, ed un codominio definito su B .

La tavola di verità di una funzione logica rappresenta tutte le possibili combinazioni degli ingressi, con il corrispondente valore dell'uscita. È quindi analogo al grafico di una funzione continua. La funzione è così completamente specificata, ma questa rappresentazione è troppo complessa per essere usata in pratica.

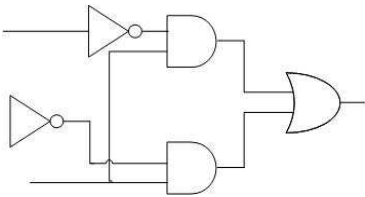
L'obiettivo della sintesi logica è così riassumibile: data la funzione logica da realizzare e la frequenza di funzionamento (quindi T_p max), ed una biblioteca di porte logiche (NAND2, NOR2, MUX, etc...), bisogna trovare la realizzazione di area (e quindi costo) minima che soddisfi il T_p max richiesto.

Sintesi logica a 2 livelli

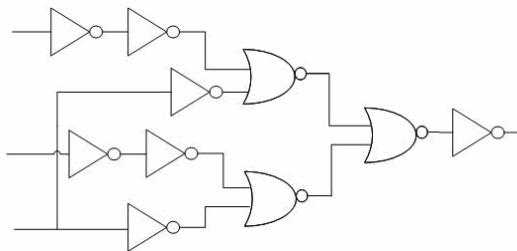
Così enunciato, il problema è troppo complesso, quindi lo si decompone in sottoproblemi più semplici. Una prima restrizione è sull'insieme di porte logiche a disposizione: facilitiamo il tutto usando solo AND, OR e NOT. Inoltre supponiamo di usare due livelli di logica, quindi le porte AND al primo livello e le porte OR al secondo (AND \rightarrow OR). Dualmente, tramite le leggi di De Morgan, possiamo naturalmente scegliere la logica OR \rightarrow AND, con l'inserimento delle corrette negazioni sugli ingressi: $\overline{A \cdot B} = \overline{A} + \overline{B}$ e $\overline{A + B} = \overline{A} \cdot \overline{B}$

La supposizione di avere solo porte AND e OR è realistica, perché possiamo sostituirle, applicando De Morgan, con porte NAND e NOR. Ecco un esempio:

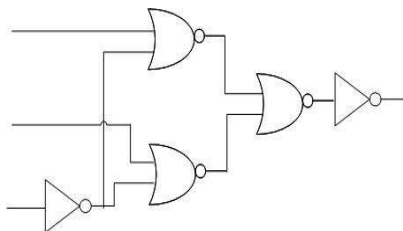
Rete iniziale



Applicazione De Morgan



Semplificazione



In modo duale si può calcolare l'equivalente in logica NAND. Preferibilmente si usa una realizzazione NAND in logica statica (evitando connessioni in serie di MOS nella rete di pull-up);

la realizzazione a NOR è più utilizzata per la logica dinamica (evitando connessioni in serie di MOS nella rete di pull-down).

Supponendo di usare solo due livelli di porte AND-OR, esiste un algoritmo, detto di Quine-McCluskey, che permette di trovare il minimo numero di porte logiche con il minimo numero di ingressi necessarie per realizzare una determinata funzione logica.

Algoritmo di Quine McCluskey

Analizziamo la logica combinatoria nella figura sopra. Abbiamo otto possibili configurazioni degli ingressi:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Ogni combinazione degli ingressi per cui la funzione vale 1 è chiamata mintermine. Un prodotto logico tra variabili (o variabili negate) è un implicante di una funzione se la funzione ha valore 1 per tutte le configurazioni delle variabili che portano ad 1 il prodotto. Nel nostro caso abbiamo due implicanti:

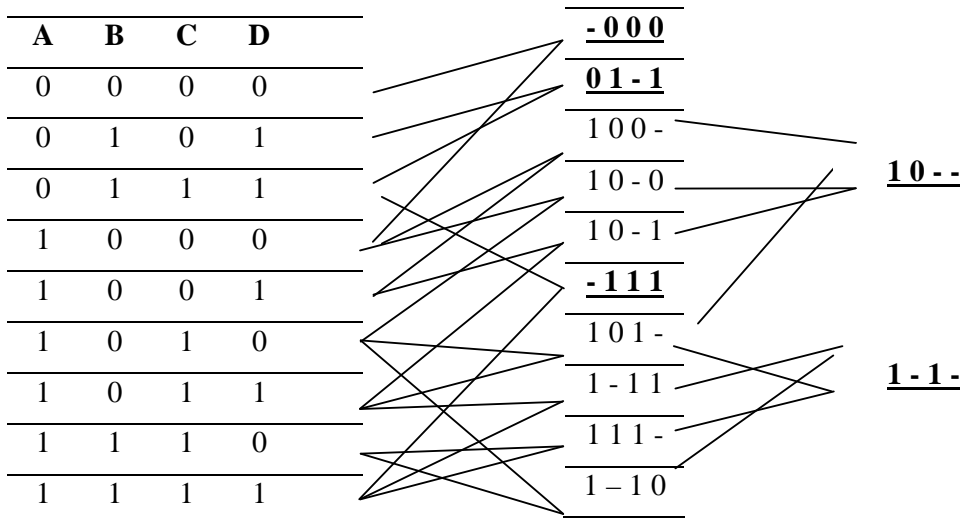
$$A1 = \bar{A} \cdot C \quad \text{e} \quad A2 = \bar{B} \cdot C$$

La funzione rappresentata su due livelli e' quindi una somma di implicanti. L'obiettivo finale è di minimizzare il numero di implicanti con il minimo numero di letterali, che copre tutti gli 1 della tavola di verità.

La regola del consenso applicata ai mintermini permette di trovare tutti gli implicanti. Per esempio $\bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot C = \bar{A} \cdot C$, cioè il letterale B e' ridondante, e quindi puo' essere omesso, semplificando la funzione.

L'algoritmo di Quine-McCluskey procede confrontando ogni mintermine con tutti gli altri, e abbinandoli applicando proprio la regola del consenso. Gli implicanti a cui non è più applicabile questa regola vengono detti implicanti primi (essi coprono un numero massimo di mintermini adiacenti).

Facciamo un esempio (sono indicati solo i mintermini della funzione):



Adesso scriviamo in una tabella, per ogni implicante primo individuato, quali sono i mintermini che esso copre:

	- 0 0 0	0 1 - 1	- 1 1 1	1 0 - -	1 - 1 -
0 0 0 0	X				
0 1 0 1		X			
0 1 1 1		X	X		
1 0 0 0	X			X	
1 0 0 1				X	
1 0 1 0				X	X
1 0 1 1				X	X
1 1 1 0					X
1 1 1 1			X		X

Il vincolo che dobbiamo rispettare per “coprire” tutti gli 1 della funzione è di scegliere abbastanza colonne, quindi implicanti primi, per riuscire a “coprire” tutti i mintermini di partenza.

Il primo passo è cercare, per ogni mintermine, quale implicante primo è essenziale per realizzarlo, cioè qual’è l’unico 1 in una riga. Gli implicanti essenziali sono comunque necessari ai fini della realizzazione minima della funzione.

Nel nostro esempio, andando in ordine, vediamo che il primo mintermine è coperto solo dall’implicante $\bar{0} \bar{0} \bar{0}$ (\overline{BCD}); il secondo è coperto solo dall’implicante $0 \bar{1} \bar{1}$ (\overline{ABD}); il quinto è coperto solo dall’implicante $1 \bar{0} \bar{-}$ (\overline{AB}); l’ottavo è coperto solo dall’implicante $1 - \bar{1} -$ (\overline{AC}). Questi quattro implicanti essenziali *in questo caso semplice* coprono tutti i mintermini della funzione. Resta fuori l’implicante primo $\bar{1} \bar{1} \bar{1}$ quindi viene scartato. La funzione di costo minimo equivalente alla tavola di verità sarà dunque:

$$F = \overline{BCD} + \overline{ABD} + \overline{AB} + AC$$

In generale è però necessario scegliere un insieme minimo di implicanti non essenziali che coprano tutti i mintermini non coperti dagli essenziali.

La procedura di Quine-Mc Cluskey ottiene il minimo assoluto della funzione di costo, ma è molto complessa. Infatti data una generica funzione ad N ingressi, abbiamo 2^N mintermini totali. Inoltre abbiamo un numero di implicanti primi è dell’ordine di $O(3^N)$. Il problema finale di copertura dei mintermini è esponenziale, risultando in una complessità temporale nell’ordine di $O(2 \cdot 3^N)$.

Anche se il caso peggiore non si verifica mai in pratica, l’algoritmo di Quine-Mc Cluskey si applica a funzioni con 15/20 ingressi al massimo.

Per cercare di ridurre la complessità, si usano algoritmi euristici. Questi permettono di raggiungere solo un minimo locale della funzione di costo.

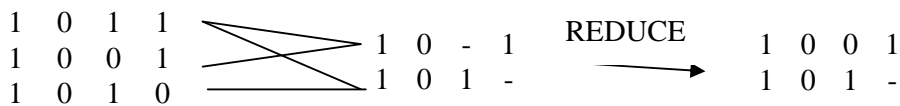
Tali algoritmi appartengono ad alcune grandi famiglie:

- *gradiente o steepest descent*, sceglie la direzione indicata dalla massima riduzione nella funzione di costo senza tornare mai indietro;
- *controllo esaustivo dei vicini*, adatto a funzioni con molti minimi locali poco distanziati, perché qualità della soluzione e tempo di esecuzione dipendono da quanto è grande l’insieme dei vicini analizzato;
- *casuali o randomizzati*, dove spostamenti in salita sono possibili ma penalizzati.

In generale la complessità di questi algoritmi cresce con la qualità del risultato ottenuto.

L' algoritmo attualmente più utilizzato per la sintesi logica a due livelli è l' algoritmo ESPRESSO. Esso conduce ad una soluzione non esatta ma accettabile, poiché molto vicina al minimo assoluto. È basato sulla ripetizione, finché il numero di implicanti primi smette di diminuire, di tre fasi principali:

- EXPAND, dove si usa la regola del consenso per combinare mintermini della funzione per creare implicanti primi, ma non in maniera esaustiva. Diversamente all' algoritmo di Quine-Mc Cluskey non vengono esplorate tutte le possibili combinazioni, ma solo un numero sufficiente per coprire tutti i mintermini;
- IRREDUNDANT, dove vengono eliminati alcuni implicanti primi completamente coperti;
- REDUCE, che rappresenta un tentativo di "uscire" dalla zona di minimo locale. Esso trasforma alcuni implicanti primi in non primi, mantenendo sempre la copertura degli 1 della funzione. Come esempio, supponiamo di avere 3 mintermini di una funzione:



Al termine dell' algoritmo, vengono effettuate, per una sola volta, le fasi di EXPAND e IRREDUNDANT riportare il risultato a una somma di implicanti primi, dopo l' ultima fase di REDUCE.

Realizzazione di funzioni a due livelli

Nel passato, per creare delle funzione logiche ad N ingressi, venivano usate delle ROM, in modo da utilizzare componenti costruiti in numero elevato. La creazione di tutti i mintermini della funzione era affidato al blocco di decodifica della ROM, che consiste in una batteria di porte AND (ne sono quindi necessarie 2^N), ciascuna con N ingressi.

Invece di creare tutti i mintermini della funzione, negli anni 70 si realizzarono porte AND con un numero programmabile di ingressi, in modo da poter realizzare gli implicanti primi della funzione. Vennero dunque create le PLA (Programming Logic Array), che presentavano in uscita dal blocco AND $m \ll 2^N$ segnali. Le PLA erano molto diffuse in quegli anni per due motivazioni:

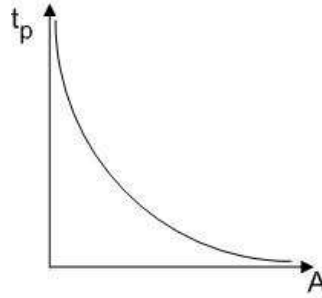
- la tecnologia di allora (basata su NMOS) permetteva di costruirle molto facilmente;
- potevano essere sintetizzate in modo efficiente da algoritmi come ESPRESSO.

Quando il numero di implicanti primi necessari pero' e' comunque troppo elevato, in particolare per le funzioni aritmetiche, è necessario eliminare l' ipotesi di utilizzare solo due livelli di logica.

Ad esempio, per la progettazione di un sommatore è possibile utilizzare una cascata di Full Adder (FA), con una complessità del numero di porte e del ritardo totale di $O(N)$.

Un tale sommatore, progettato con soli due livelli di logica, produce in pratica un sommatore Carry Look Ahead, con complessità del numero di porte nell' ordine di $O(N^2)$ e ritardo invece costante, quindi complessità $O(1)$. Questa complessita' e' eccessiva per un sommatore con 32 ingressi.

Notiamo una proprietà importante di questo esempio, che si puo' empiricamente generalizzare per funzioni logiche generiche: il prodotto area*ritardo è costante per due realizzazioni della stessa funzione.



Sintesi logica multilivello

Passiamo quindi ad occuparci della sintesi logica multilivello, il cui obiettivo è: data una funzione logica ed avendo a disposizione una determinata biblioteca di celle da utilizzare ed un ritardo massimo da rispettare, bisogna trovare l'interconnessione aciclica di porte che realizza la funzione, con il ritardo massimo assegnato.

Per questo problema, diversamente dalla sintesi su due livelli, non esiste algoritmo di minimizzazione esatto usabile in pratica. Si utilizzano dunque tecniche euristiche, basate su:

- identificazione dei fattori comuni,
- divisione booleana e algebrica,
- semplificazione su due livelli,
- eliminazione dei fattori comuni per ridurre ritardo e uscita da minimi locali.

Consideriamo un esempio di identificazione dei fattori comuni. Supponiamo di avere due funzioni con 6 ingressi, rappresentate su 2 livelli come:

$$f_1 = AB + AC + AD \qquad f_2 = \bar{A}B + \bar{A}C + \bar{A}E$$

Trascuriamo gli inverter dovuti alla trasformazione da logica AND \rightarrow OR a NAND/NOR. Il costo della realizzazione delle due funzioni, utilizzando una logica su due livelli, sarà

$$4 \cdot 6 \text{ (transistor per le porte AND)} + 2 \cdot 6 \text{ (transistor per le porte OR)} = 36$$

Sono necessari 36 transistor, che è una buona stima pre-layout dell'area.

Utilizzando una logica multilivello, dunque raccogliendo a fattore comune il termine $K = B + C$, possiamo riscrivere le due funzioni come:

$$f_1 = AK + AD \qquad f_2 = \bar{A}K + \bar{A}E$$

Il costo totale di realizzazione sarà invece

$$4 \text{ (transistor per } K) + 4 \cdot 4 \text{ (transistor per le porte AND)} + 2 \cdot 4 \text{ (transistor per gli OR)} = 28$$

La progettazione multilivello permette di ottenere un risparmio di area di circa un quarto in questo caso.

Divisione booleana e algebrica

La divisione booleana è usata per dividere le funzioni per i fattori individuati ed è definita come segue: date due funzioni F (dividendo) e G (divisore), si cercano le funzioni H (quoziente) ed R (resto) tali che $F = G \cdot H + R$.

In algebra booleana, questo problema ha tantissime soluzioni. Posso scegliere come H qualsiasi funzione che vale 0 per tutte le combinazioni di ingresso in cui F vale 0 ma G vale 1, e posso

scegliere come R qualsiasi funzione che vale 1 per tutte le combinazioni di ingresso in cui GH vale 0 ma F vale 1.

Ad esempio, per la funzione f_1 trattata precedentemente, possiamo trovare diverse soluzioni, come ad esempio

$$f_1 = (A + \overline{BC}) \cdot (B + C) + A \cdot (C + D)$$

$$f_1 = A \cdot (B + C) + A \cdot D$$

Tutte queste diverse soluzioni complicano enormemente il compito di trovare H ed R di costo minimo. Per ottenere una soluzione unica (anche se non è affatto garantito che sia la migliore in assoluto) possiamo ignorare alcuni teoremi tipici dell'algebra booleana, e lavorare applicando solo l'algebra dei polinomi. In particolare, ignoriamo

$$A \cdot \overline{A} = 0$$

$$A + A = A$$

$$A \cdot A = A$$

$$A + \overline{A} = 1$$

Identificazione dei fattori comuni

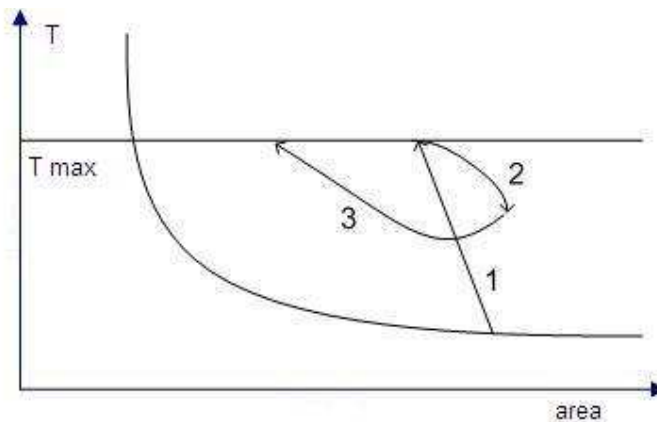
Il principale problema della ricerca dei fattori comuni è per motivi simili, il loro numero elevato, per cui si limita la loro ricerca a quelli che si presentano come combinazione di due letterali, quindi $K = X_1 + X_2$ e $K = X_1 \cdot X_2$.

Si può dimostrare che ripetendo ricorsivamente questa ricerca si identificano tutti i fattori comuni, anche più complessi.

Eliminazione dei fattori comuni

L'identificazione dei fattori comuni riduce notevolmente il numero di transistor stimati. Però questo guadagno in area, come visto prima, viene pagato con un aumento del ritardo totale di propagazione. Questo può essere un problema, perché la sintesi logica è sempre caratterizzata da un T_p max da rispettare.

Si possono quindi usare in modo iterativo le operazioni viste sopra, spostandosi non lungo ma "abbastanza vicino" alla curva che rappresenta il prodotto ideale area*ritardo per la funzione logica specificata:



Ad ogni passo dell'algoritmo, si cerca dunque di "modificare" la situazione (come avveniva nell'algoritmo Espresso con la fase di REDUCE), eliminando dei fattori comuni che riducono di meno la complessità della rete logica e che aumentano di più il ritardo dei cammini critici.

Per esempio, si può partire dalla soluzione ottima a 2 livelli, in basso a destra (elevata area, basso ritardo), che si trova sulla curva ideale o vicino a essa a causa della buona qualità della sintesi logica a 2 livelli. Di qui si possono estrarre molti fattori comuni, per ridurre l'area finché il ritardo rimane accettabile (percorso 1). Si possono quindi eliminare alcuni fattori comuni, per ridurre il ritardo aumentando l'area (percorso 2). A questo punto si possono di nuovo estrarre fattori comuni, forse con risultati migliori della prima volta (percorso 3), e così via.