

## Appunti VII settimana

### Technology mapping

Il processo di **Technology mapping** ha in ingresso una rete logica realizzata con porte complesse, non necessariamente presenti nelle librerie, indipendenti dalla tecnologia (con stime approssimate di costi e ritardi) e produce una funzione logica (dipendente dalla tecnologia) ottimizzata su una libreria di celle standard. Questa tecnica consiste nella decomposizione della funzione in porte logiche semplici (NAND2, INV) e nel successivo “riconoscimento” di porte logiche più complesse presenti nelle librerie.

Se, per esempio, vogliamo realizzare la funzione, già ottimizzata in modo indipendente dalla tecnologia:

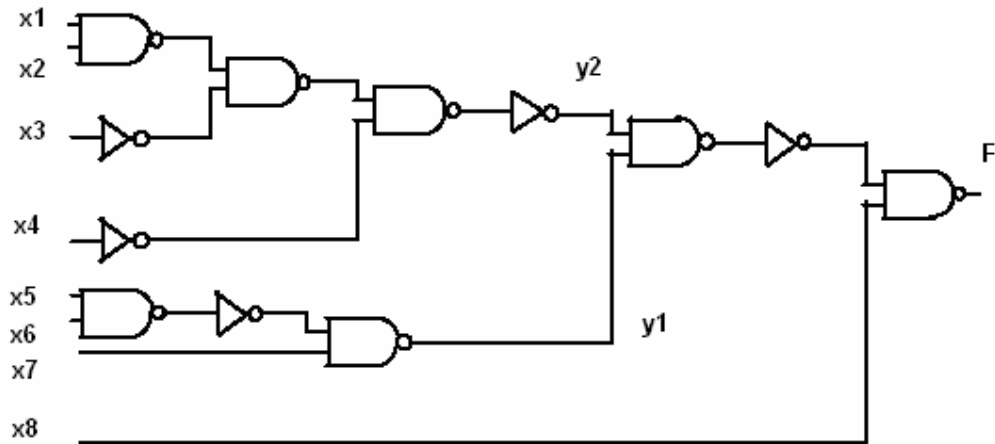
$$F = \overline{x_8 \cdot y_1 \cdot y_2}$$
$$y_1 = \overline{x_5 \cdot x_6 \cdot x_7}$$
$$y_2 = x_3 \cdot \overline{x_4} + x_1 \cdot x_2 \cdot \overline{x_4}$$

dovrei avere in libreria una porta logica per  $y_2$ , che però non esiste. Inoltre non è detto che la decomposizione prescelta sia la migliore, viste le porte logiche effettivamente a disposizione.

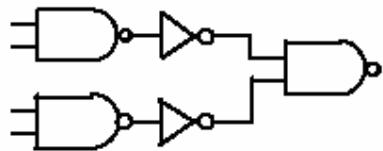
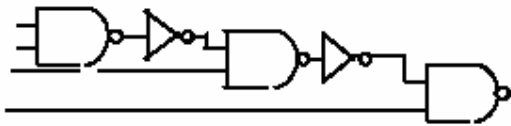
L'algoritmo di technology mapping attualmente più usato:

- spezzare la funzione in soli NAND2 e INV, che sono le porte più semplici che sono necessarie e sufficienti per realizzare qualsiasi funzione logica
- effettuare il riconoscimento delle porte logiche presenti in libreria, cercando la soluzione migliore in termini di area e ritardo.

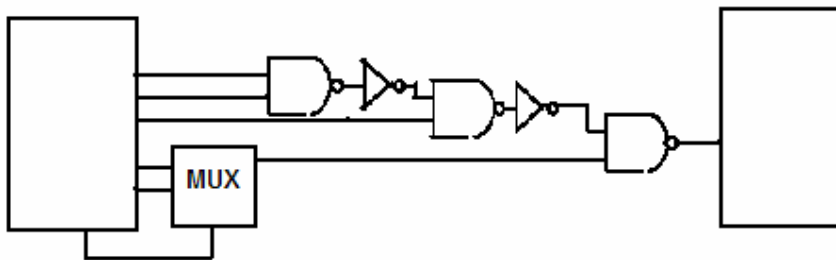
Per capire come ciò avvenga, consideriamo la seguente decomposizione usando NAND2 e INV delle equazioni viste sopra:



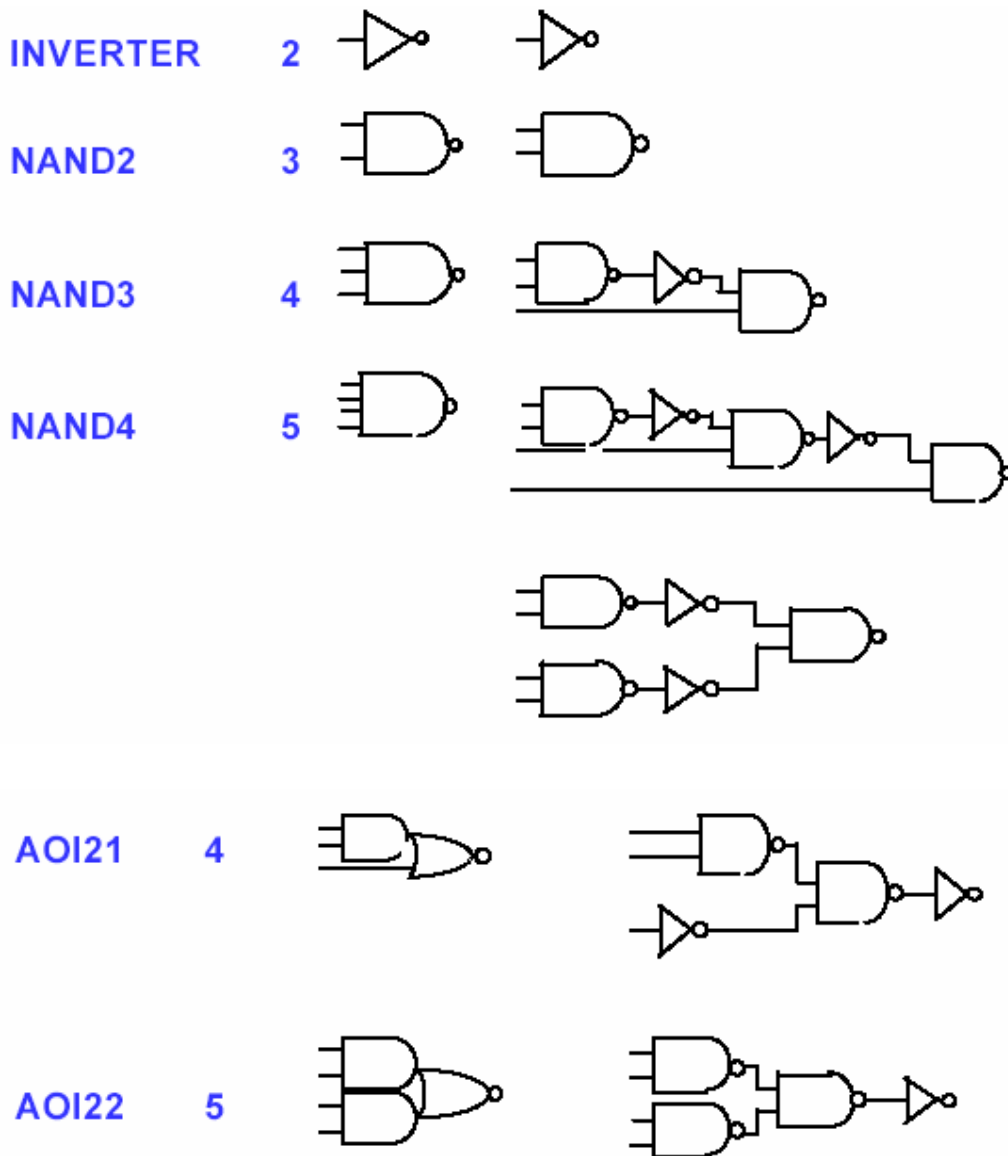
E' da notare come in questa decomposizione, scelte funzionalmente equivalenti possano dare luogo a risultati molto diversi dal punto di vista dei ritardi. Per esempio, consideriamo le due decomposizioni possibili di una funzione logica NAND con 4 ingressi:



Nella prima decomposizione i ritardi non sono bilanciati in quanto il ritardo dell'ingresso in alto e' molto maggiore di quello dell'ingresso in basso. Nella seconda decomposizione invece tutti i ritardi sono simili. La prima decomposizione pero' e' migliore quando i tempi di arrivo dei vari segnali sono diversi tra loro, collegando il segnale che si stabilizza per ultimo a quello con il ritardo rimanente minore, come nell'esempio seguente:



Torniamo all'esempio di technology mapping. Supponiamo che la libreria a disposizione comprenda (accanto il nome del componente è indicato il costo in termini di area):



Calcoliamo inizialmente il costo in termini di area del circuito realizzato con sole porte NAND2 e INV:

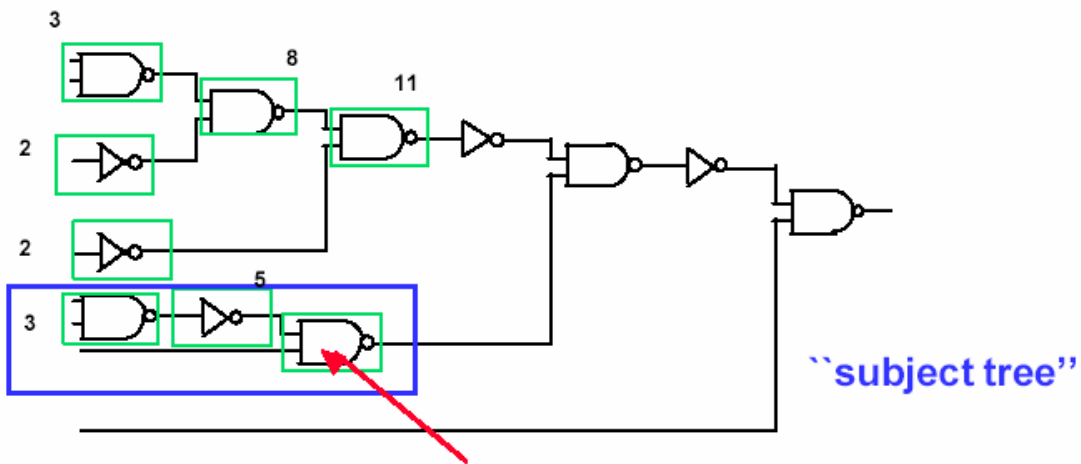
7 NAND2 con costo 3

5 INV con costo 2

Per un costo totale di  $21 + 10 = 31$

Possiamo effettuare ora il “riconoscimento” delle porte logiche più complesse presenti nella libreria: partendo dagli ingressi devo trovare la realizzazione migliore per ogni segnale, considerando le decisioni prese per i segnali precedenti. L’aspetto interessante di questo algoritmo (di programmazione dinamica) e’ che esso non richiede senza mai modificare le decisioni prese nei passi precedenti, se la rete logica decomposta non ha porte che pilotano piu’ di una porta (ovvero se il grafo e’ un albero).

Il primo gruppo di porte logiche, a partire dagli ingressi, che coincide con la rappresentazione di una porta logica presente in libreria è il seguente:



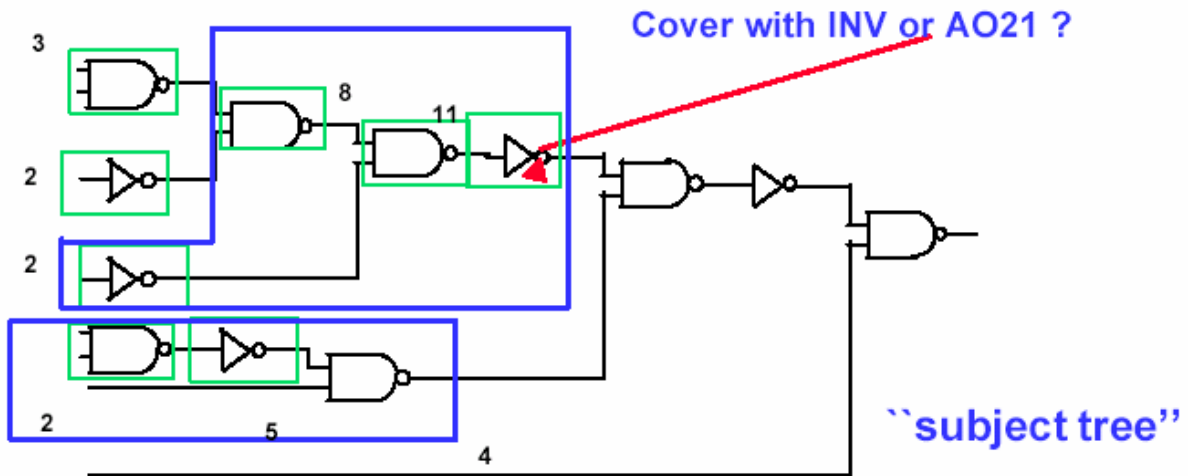
Cover with ND2 or ND3 ?

1 NAND2	3	1 NAND3	= 4
+ subtree	5		

Area cost 8

La realizzazione di costo minore e’ con un NAND3.

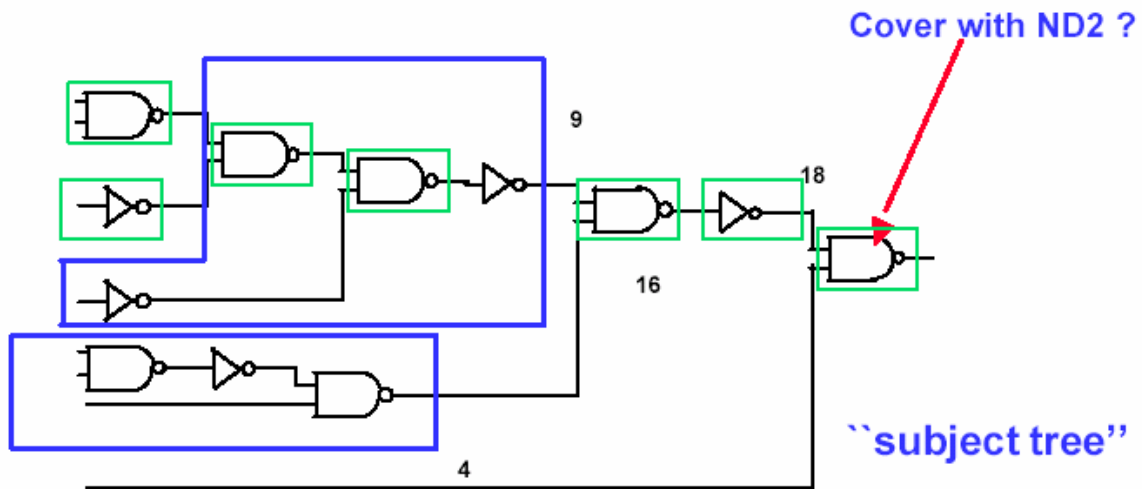
Procedendo a esaminare segnali successivi, sempre in ordine topologico dagli ingressi:



1 Inverter	2	1 AO21	4
+ subtree	11	+ subtree 1	3
		+ subtree 2	2
<b>Area cost 13</b>		<b>Area cost 9</b>	

Quindi la scelta migliore e' AOI21. E' da notare che a ogni passo decido la scelta migliore per ogni segnale, ma non decido ancora se quel segnale verra' realizzato o sara' "nascosto" entro una porta logica. Questa decisione viene presa soltanto quando si arriva all'uscita e si decide come realizzarla.

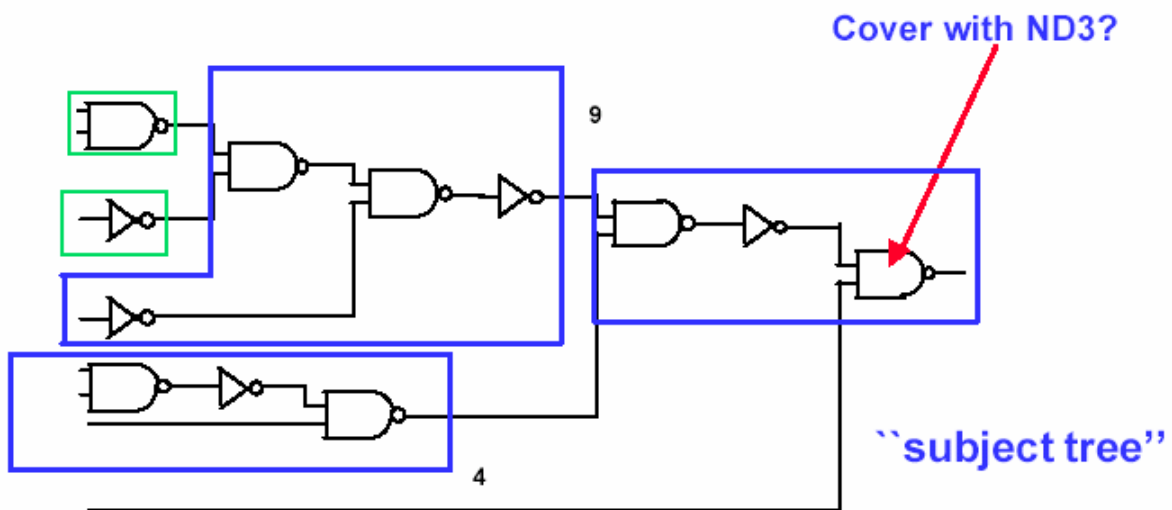
All'uscita della rete, date le scelte precedenti, ottengo come costo se uso un NAND2:



subtree 1	18
subtree 2	0
1 NAND2	<u>3</u>

Area cost 21

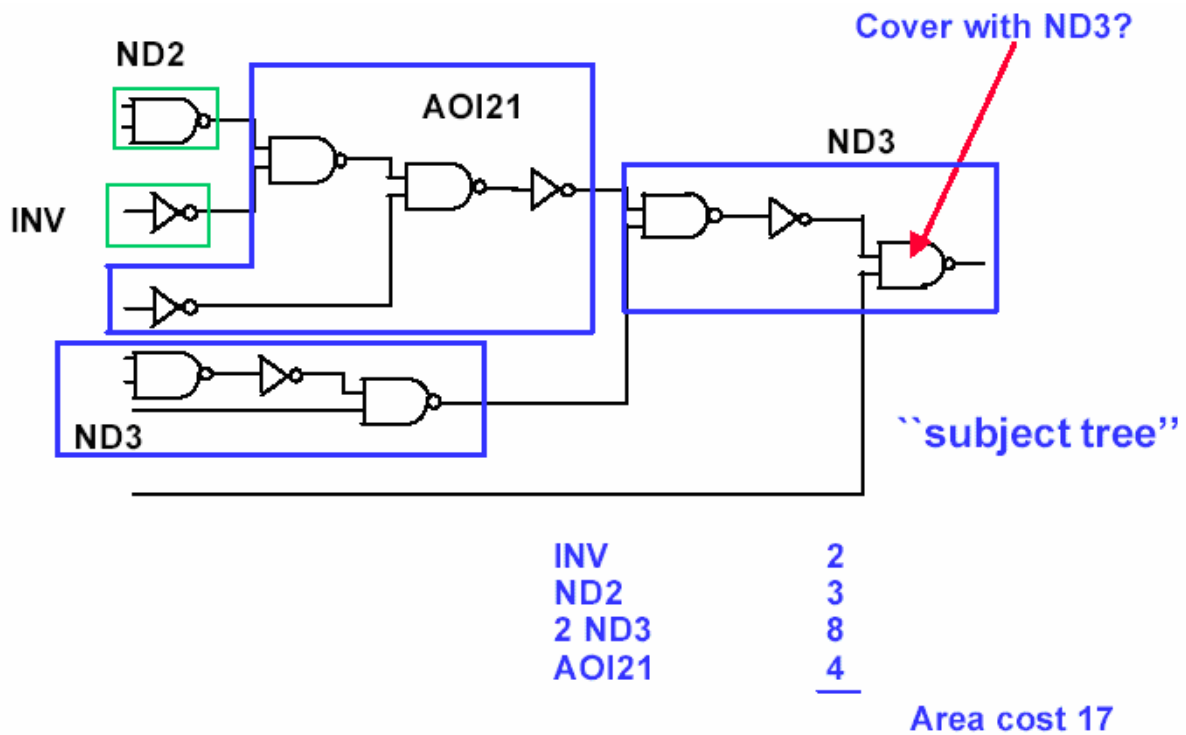
Mentre se uso una porta NAND3:



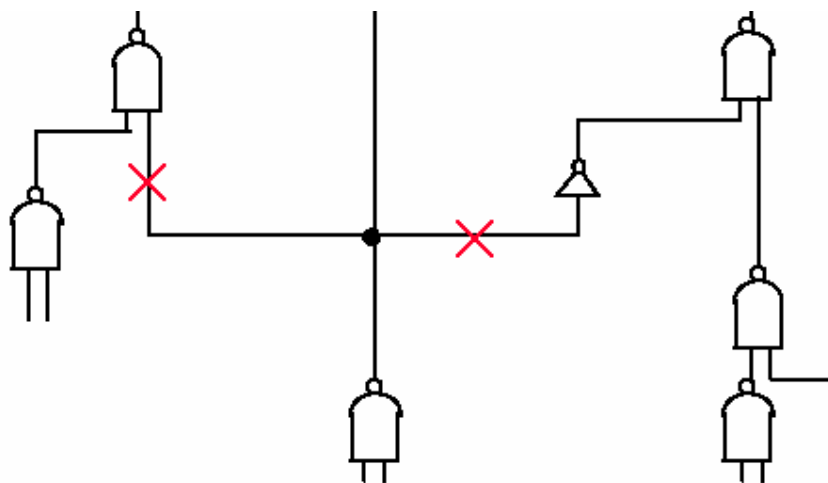
subtree 1	9
subtree 2	4
subtree 3	0
1 NAND3	<u>4</u>

Area cost 17

Che ha costo minore. Quindi la soluzione ottimale è:



Questo algoritmo, come detto prima, ottiene la soluzione ottima solo se la funzione e' rappresentata da un albero senza porte logiche che pilotano piu' di una porta. Nel caso si riscontri tale situazione si puo' effettuare un taglio e analizzare separatamente i due rami; la complessità dell'algoritmo rimane lineare rispetto al numero di porte NAND2 e INV nel circuito decomposto, ma non ottengo più la soluzione ottimale.

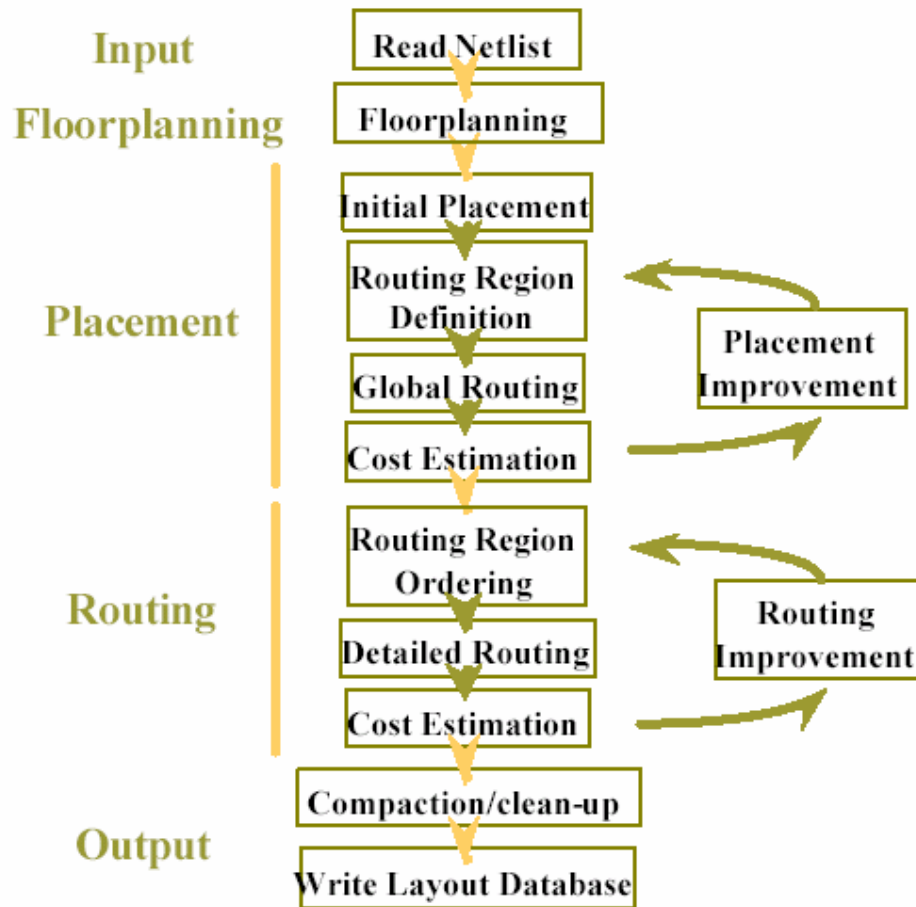


E' da notare che l'algoritmo permette anche, nello stesso modo, di propagare il tempo di arrivo della soluzione prescelta, se il ritardo di ogni porta e' una costante. Quando finisco di analizzare l'intero circuito ho

quindi calcolato costo e ritardo di propagazione totale e posso scegliere la soluzione a costo minimo che rispetti i tempi stabiliti in fase di progetto. In realta' il ritardo dipende anche dalla capacita' pilotata da ogni porta, che non e' ancora nota quando esamino ciascun segnale in ordine topologico. In pratica, si considera un certo numero (tipicamente una decina) di possibili intervalli di capacita' pilotabili e si memorizza per ogni segnale la soluzione migliore per la massima capacita' in quell'intervallo. Quando poi si sceglie la porta che e' pilotata da quel segnale, si puo' usare la soluzione migliore con l'intervallo di capacita' in cui ricade la sua capacita' di ingresso.

## Progetto automatico del layout: piazzamento e interconnessione

Il progetto automatico del layout di un circuito integrato è eseguito in vari passi:



Dopo aver generato la netlist, cioè una descrizione delle celle logiche e delle loro interconnessioni tramite sintesi logica e technology mapping, si esegue il floorplanning. Questo passo (opzionale) permette di scegliere a mano la posizione dei macro blocchi più importanti del circuito complessivo. Successivamente vengono piazzate le porte logiche, usando la stima dei ritardi fornita da passi iterativi di routing globale. Alla fine viene eseguito il routing dettagliato.

E' da notare che la dissipazione di potenza non dipende dal posizionamento, ma l'assegnazione di un gruppo di porte logiche molto attive, e che quindi dissipano molta potenza, in una piccola regione del circuito può causarvi un eccessivo surriscaldamento.

## Piazzamento

Per il piazzamento, gli approcci tradizionali sono:

**Simulated annealing:** lo schema concettuale del SA è relativamente semplice. L'idea base è generare in modo random un punto nello spazio delle soluzioni e valutare la funzione obiettivo ed i vincoli del problema. Se il punto non è compatibile con i vincoli, viene rigettato (oppure penalizzato severamente nella funzione di costo se si possono accettare temporaneamente soluzioni illegali) e si genera un nuovo punto. Se il punto è ammissibile e la funzione obiettivo è minore dell'ottimo sinora raggiunto, allora il punto stesso viene accettato e la soluzione ottima viene aggiornata. Se il punto è ammissibile ma la funzione obiettivo aumenta, allora si utilizza un criterio probabilistico di accettazione, in cui la probabilità tende a zero col decrescere della temperatura (un parametro dell'algoritmo che viene fatto decrescere durante la sua esecuzione) ed è tanto minore tanto più il nuovo punto è peggiore del vecchio. È un algoritmo flessibile, che termina con probabilità 1 nel minimo globale dato un tempo di esecuzione infinito, ma è lento a convergere.

**Recursive partitioning:** l'obiettivo è quello di mettere vicini blocchi che hanno numerose interconnessioni. Procedo dividendo le celle in vari blocchi con delle linee immaginarie che taglino il minor numero di interconnessioni. Il grande vantaggio di questo algoritmo è la possibilità di parallelizzare il lavoro tra i vari blocchi mentre l'inconveniente è che una decisione errata presa nella prima divisione può costare molto alla fine dei vari passi successivi.

**Quadratic placement:** in questo algoritmo ogni interconnessione tra celle logiche è modellata come una molla alla quale è associata una costante elastica tanto maggiore quanto minore è lo slack dell'interconnessione. Si ha in questo caso la possibilità che alcune celle vengano piazzate nello stesso punto, o comunque con parziali sovrapposizioni. Il metodo usato (per esempio nel metodo chiamato Gordian) è di suddividere le celle in due gruppi usando partitioning, e poi ripetere ricorsivamente l'algoritmo su un singolo blocco alla volta (mantenendo fisse le celle dell'altro). Si può anche ripetere questa

procedura di rilassamento più volte. Anche in questo caso il procedimento è parallelizzabile.

## Interconnessione (routing)

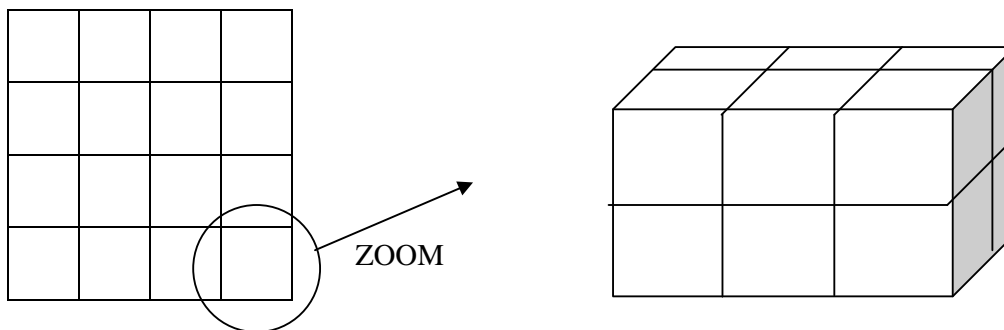
Il **routing** è quella fase di progetto in cui si realizzano i collegamenti necessari tra le celle standard e i macroblocchi per ottenere il layout. Supponendo di avere piazzato le celle logiche, il modello dei ritardi e la frequenza di funzionamento l'obiettivo è quello di trovare le interconnessioni sul silicio che soddisfino i vincoli sulla logica e sulla tempistica. Il routing globale considera come risorse delle regioni del chip che includono l'area di alcune centinaia di celle standard, rappresentate come caselle in una griglia quadrata.

Il piazzamento può essere iterato più volte insieme al routing globale per ottenere una stima sempre più precisa del costo di ogni interconnessione, che inizialmente viene approssimato per esempio come la distanza (euclidea o di Manhattan) tra le coppie di celle interconnesse.

Un algoritmo molto usato per il routing globale è quello di Lee, che analizza una interconnessione alla volta partendo da quella più critica (slack minimo) e si procede numerando le caselle con la loro distanza dalla sorgente. Un modo per considerare le capacità di ciascuna cella nell'algoritmo di Lee usato per il routing globale è di attribuire un peso maggiore alle celle già occupate da interconnessioni in modo che vengano considerate più costose calcolando la distanza dalla sorgente di ogni interconnessione. La figura seguente rappresenta un esempio di applicazione dell'algoritmo di Lee per interconnettere le celle A e B, supponendo che nessun'altra interconnessione sia ancora stata tracciata (il costo per passare attraverso ogni cella è unitario) e che le aree in grigio non possano essere usate (per esempio perché rappresentano blocchi in cui il layout usa tutti i livelli di metallizzazione a disposizione).

4	3	2	3	4	5	6	7	8	9	10	11
3	2	1	2	3	4	5	6	7	8	9	10
2	1	A	1		5	6	7	8			
3	2	1	2		6	7	8	9	10	11	12
4	3	2	3							12	13
5	4	3	4		14				B	13	14
6	5			13	14					14	
7	6	7		11	12	13	14				
8	7	8	9	10	11	12	13	14			
9	8	9	10	11	12	13	14				

Il routing locale puo' essere effettuato usando lo stesso algoritmo, lavorando su una regione alla volta, modellandola con una griglia tridimensionale piu' dettagliata dove la terza dimensione rappresenta i livelli di metallizzazione. Il lato di ogni cella di questa nuova griglia è data dalla distanza minima tra le metallizzazioni più la larghezza minima del metallo, in modo da poter eseguire al massimo una interconnessione per cella.



Visione globale

Visione dettagliata

Bisogna penalizzare l'attraversamento di livelli tramite contatti poiché essi hanno una resistenza maggiore di quella delle connessioni sullo stesso livello e rappresentano punti a rischio più elevato di guasti.

I vantaggi dell'algoritmo di Lee sono di:

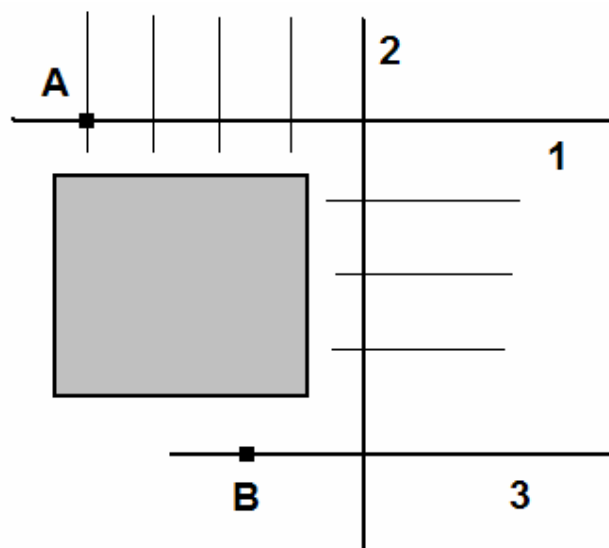
- essere semplice e generale perché basato su distanze geometriche;

- essere adattabile tramite pesi ad analisi di congestione o costi dei contatti;
- essere parallelizzabile, potendo eseguire in parallelo almeno alcune interconnessioni, specie a livello di routing globale;
- garantire una soluzione per ogni interconnessione, se essa esiste.

Il problema però è che le interconnessioni meno critiche possono essere tracciate quando la congestione è già elevata, e quindi può non essere possibile trovare loro un percorso. In questo caso si usa “rip-up and re-route”, cioè si eliminano interconnessioni che bloccano quella che non si riesce a tracciare. Se neanche questo metodo trova una soluzione, allora bisogna tornare alla fase di piazzamento e lasciare più spazio libero nelle regioni bloccate. Un altro svantaggio è che può essere troppo lento.

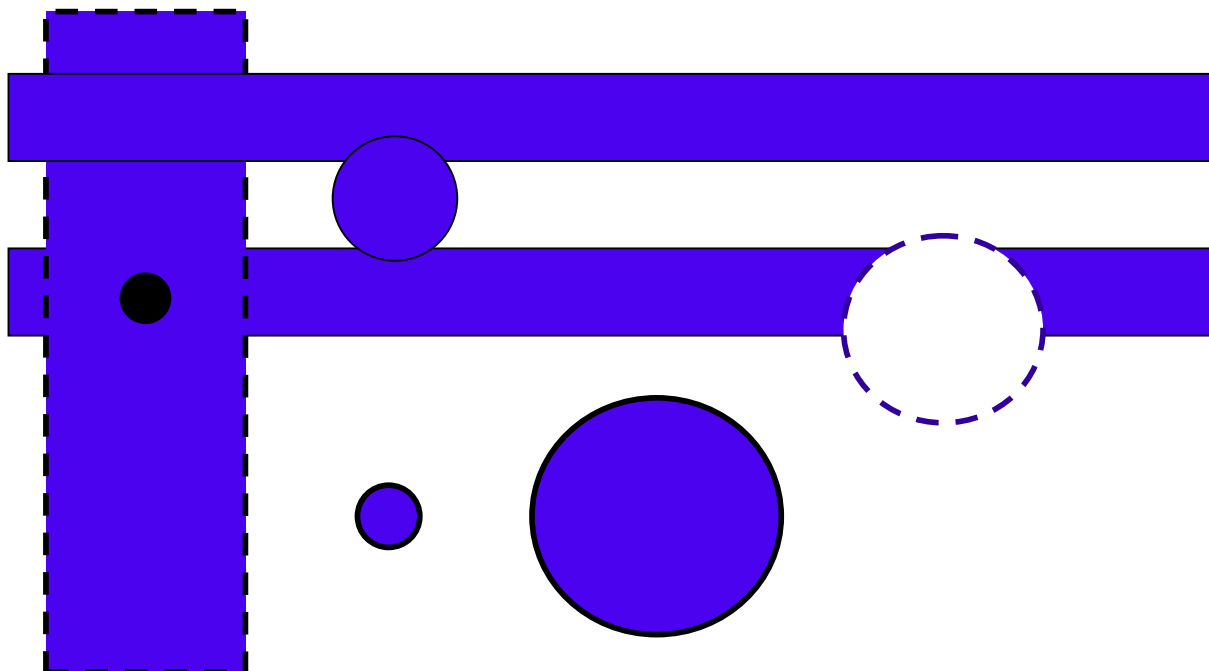
Un'alternativa per ridurre i costi è l'utilizzo dell'algoritmo di Hightower (o la sua variante chiamata A-star), che è semplice e veloce ma non garantisce di trovare una soluzione anche se essa esiste.

Se per esempio dobbiamo interconnettere i punti A e B nella figura seguente, partiamo dal tracciare un segmento di retta orizzontale e uno verticale che passino per il punto A e scegliamo quello che più ci avvicina al nostro obiettivo, in questo caso il segmento orizzontale (1). A questo punto tracciamo tutta una serie di segmenti perpendicolari e valutiamo quello che ancora maggiormente ci avvicina al punto B, cioè (2). Ripetendo lo stesso procedimento su questo segmento troveremo quello che passa proprio per il punto di arrivo (3), ottenendo così in maniera veloce una soluzione al nostro problema.



## Collaudo dei guasti

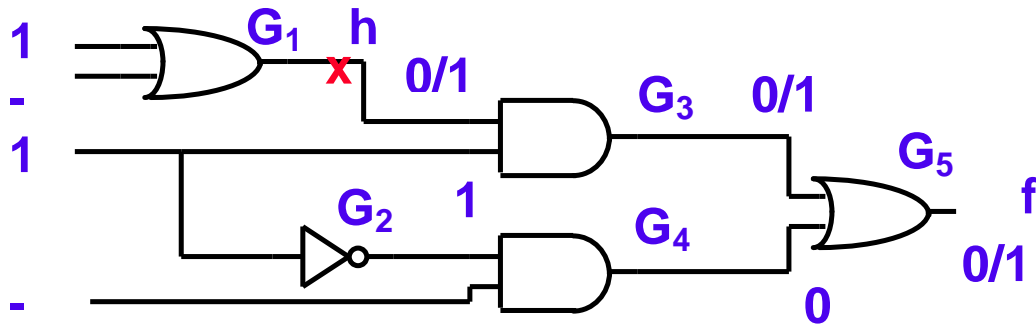
Il collaudo viene eseguito per controllare che non ci siano stati problemi di manifattura che si traducono in difetti logici e/o temporali. Il primo passo è modellare gli difetti fisici con semplici modelli di guasto che il CAD sia in grado di interpretare.



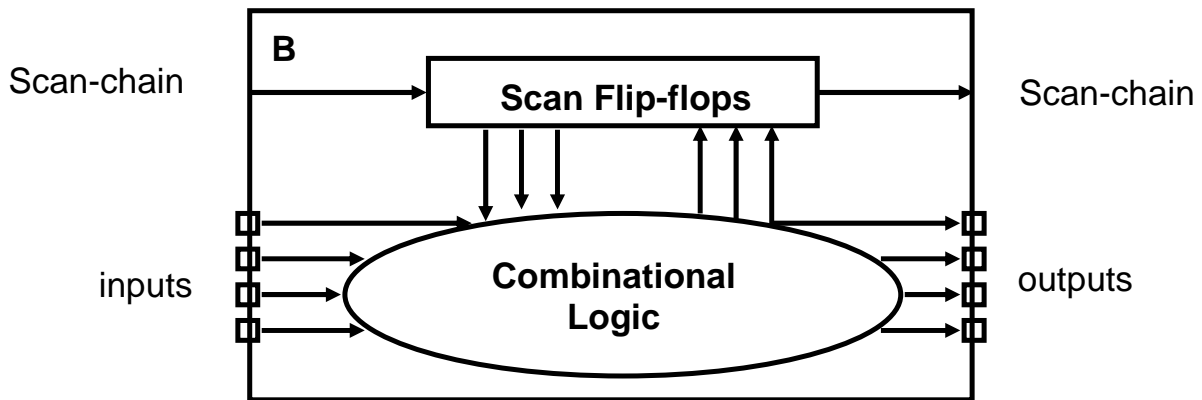
Attualmente i modelli di guasto piu' usati sono:

- difetti logici: stuck-at (fisso gli ingressi delle porte a 0 o a 1)
- difetti temporali: porte più lente

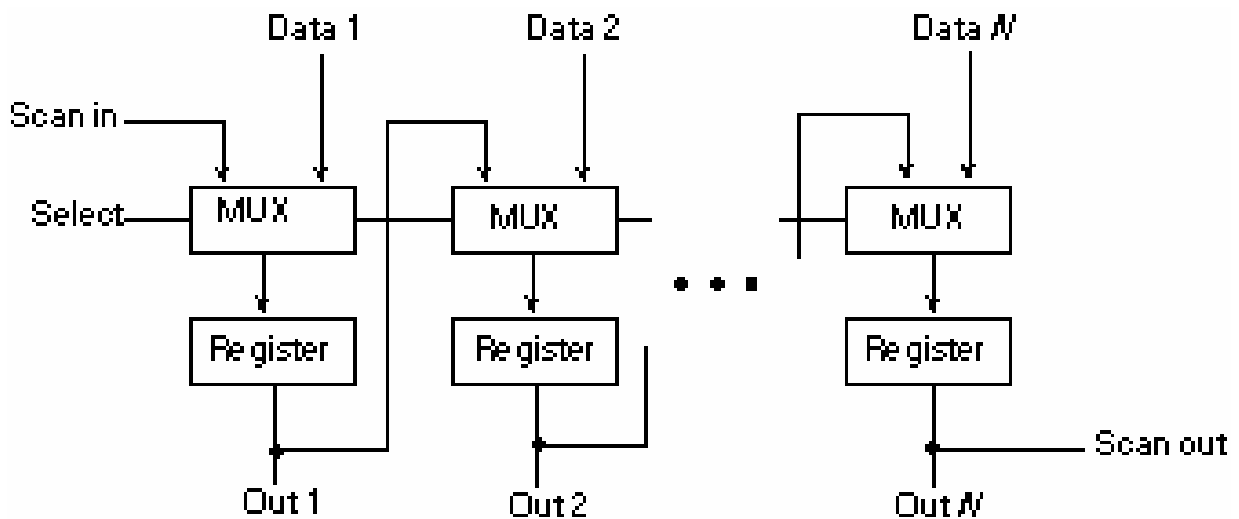
Usando questo modello si possono usare programmi di generazione automatica delle sequenze di collaudo, che per ogni guasto modellato in un circuito logico combinatorio generano un vettore di valori in ingresso che lo *eccita* (cioe' che porta l'ingresso stuck-at al valore opposto nel circuito funzionante) e che lo *propaga* (cioe' che porta un'uscita a un valore diverso nel circuito funzionante e in quello con quel guasto).



Per collaudare i circuiti logici sequenziali, per i quali la generazione delle sequenze di collaudo sarebbe troppo complicata, viene utilizzata la tecnica *scan*.



In pratica a tutti i registri che costituiscono la logica sequenziale vengono preposti dei multiplexer che in modalità normale copiano in uscita il dato in ingresso mentre in modalità *scan* vengono collegati in serie e caricati con i valori di test.



Dopo i cicli di clock richiesti verifico che l'uscita della logica combinatoria sia uguale a quella attesa dal test.