Prestazioni dinamiche dei circuiti digitali

Michele Favalli
Universita' di Ferrara

Introduzione

- **Prestazioni dinamiche di un circuito digitale**
- **Parametri da cui dipendono le prestazioni**
- **Analisi**
- **Sintesi**

Sommario

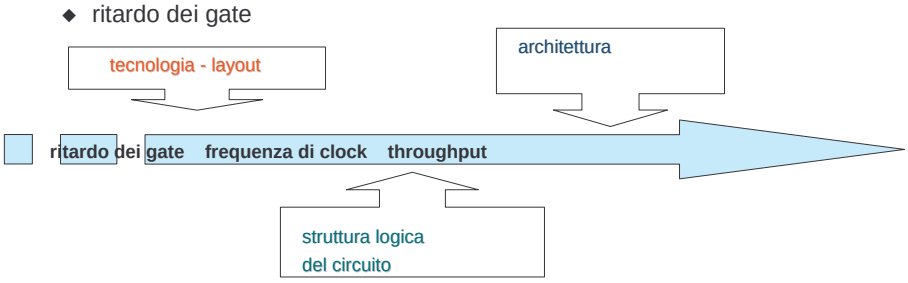
- **Prestazioni dinamiche di un circuito digitale**
  - dalle prestazioni di un sistema a quelle di un singolo gate
  - metriche per la valutazione delle prestazioni: latenza e throughput
- **Legame fra le prestazioni a livello di sistema e quelle dei singoli componenti**
  - valutazione della frequenza di clock di una rete sincrona
  - ritardo massimo di una rete combinatoria, concetto di cammino critico
- **Analisi delle prestazioni**
  - algoritmo per la valutazione del ritardo massimo (static timing analysis)
- **Metodologie per il miglioramento delle prestazioni**
  - livello tecnologico, elettrico e di layout
  - livello gate
  - livello architetturale (parallelismo, pipelining)

Prestazioni: specifiche

- **Le prestazioni dinamiche di un circuito digitale possono essere specificate in diversi modi, dipendentemente dal tipo di applicazione considerata**
  - throughput (banda): *Digital Signal Processing, CPU per applicazioni di calcolo*
  - latenza (ritardo nell'eseguire un compito): *microcontrollori, sistemi real-time*
- **Questo tipo di specifiche sono del tutto indipendenti dalla modalità di funzionamento del circuito**
- **Le specifiche indirizzano le scelte architetturali del progettista**

Prestazioni dinamiche di un sistema digitale

- **Throughput:***quantità di informazione elaborata nell'unità di tempo*
- **In un sistema sincrono il throughput dipende da**
  - architettura
  - frequenza di clock
- **La frequenza di clock dipende da**
  - struttura del circuito
  - ritardo dei gate



Esempi

- **CPU per applicazioni di calcolo**
- **MIPS (Million of Operations per Second)**

$$\text{MIPS} = \frac{\text{number of instructions}}{\text{CPU time} \times 10^6}$$

- **E' noto che tale misura può dare luogo ad errori nelle valutazioni delle prestazioni di una CPU in quanto non tiene in conto la potenza delle singole istruzioni**
- **Un esempio piu' corretto viene dai Digital Signal Processor (DSP)**
- **Il numero di campioni un segnale elaborati nell'unità di tempo è una tipica specifica di throughput**

Latenza

- Si supponga di disporre di un applicazione di tipo reattivo che riceve un qualche stimolo e deve fornire entro un certo tempo il risultato di una qualche elaborazione (sistemi real-time)
- In questo caso, è evidente che la latenza è la principale cifra di merito del sistema considerato
- In alcuni tipi di sistemi, latenza e throughput sono spesso specificati contemporaneamente nell'ambito dello stesso progetto

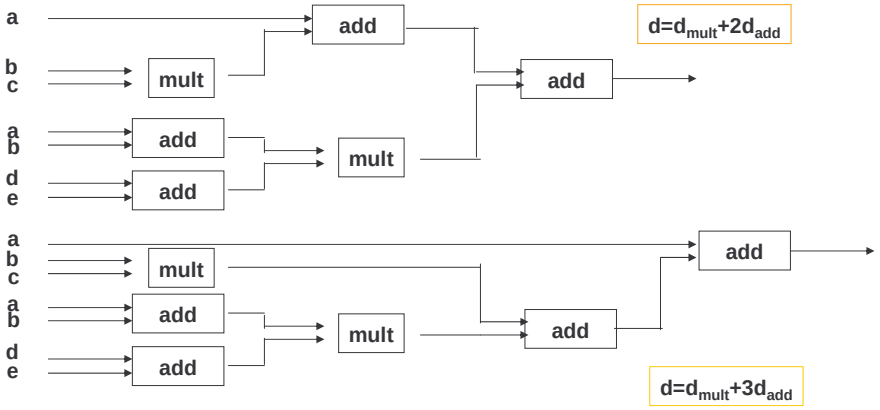
Prestazioni a livello di architettura

- L'architettura di un sistema, ovvero il modo in cui sono interconnessi fra loro i blocchi funzionali (addizionatori, moltiplicatori etc.) contribuisce alla definizione di latenza e throughput del sistema
- La possibilità di elaborare dati in maniera parallela da luogo ad elevate prestazioni, anche se chiaramente per ottenerle è necessario utilizzare un maggior numero di componenti rispetto a una soluzione che utilizza il minor numero possibile di risorse
- L'architettura deve poi consentire di sfruttare al meglio le prestazioni dei singoli blocchi, sia per quello che riguarda la frequenza di clock a cui essi operano, sia per l'efficienza con cui questi sono utilizzati

Latenza (esempio)

Supponiamo di voler calcolare l'espressione out=a+b*c+(a+b)*(d+e)

(dove a, b, …. sono interi senza segno) utilizzando una rete combinatoria di sommatori e moltiplicatori: sono possibili diverse alternative con diversi valori di latenza



Throughput

- **Nel caso di sistemi sincroni, la quantità di informazione elaborata nell'unità di tempo (throughput) può essere dato come**

$$t = f_{ck} \cdot n_{ck}$$

- **dove f_{ck} è la frequenza di clock e n_{ck} è il numero di "dati" elaborati in un periodo di clock**
- **il tipo di "dati" dipende dall'applicazione considerata**
  - in una CPU si considerano le istruzioni
  - in un DSP che elabora il flusso di dati proveniente da un convertitore A/D, i "dati" corrispondono ai campioni elaborati
- **come si vedrà in seguito, è molto difficile aumentare contemporaneamente f_{ck} e n_{ck}**

Throughput - esempio

- Nel caso di CPU il throughput viene tipicamente espresso come numero di istruzioni per unità di tempo. Ad esempio MIPS:

$$MIPS = \frac{n_{istr.}}{10^6 \cdot CPU_{time}} = \frac{f_{ck}}{10^6 \cdot CPI}$$

(CPI = numero medio di cicli di clock per istruzione)

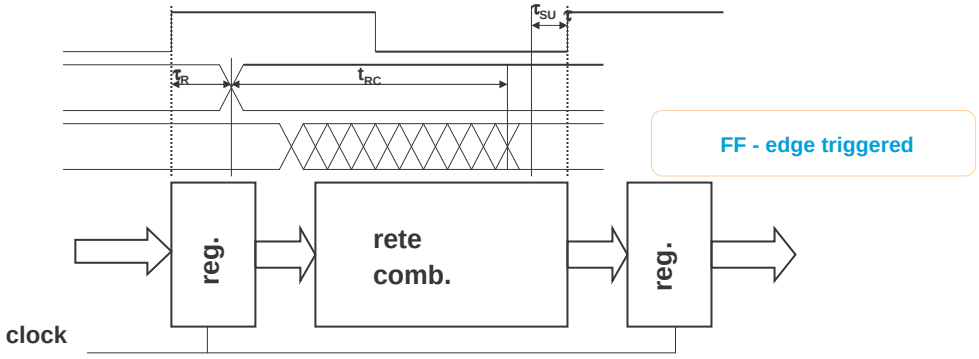
- Chiaramente $n_{ck}=1/CPI$ da cui si ottiene la definizione di throughput vista in precedenza

$$t = f_{ck} \cdot n_{ck}$$

- Questa misura non è particolarmente significativa a causa del fatto che non tiene conto del potere semantico di una istruzione
- Si noti anche che i MIPS dipendono dall'applicazione considerata
- Architetture RISC con poche istruzioni semplici hanno MIPS molto elevati rispetto ad architetture CISC che però hanno istruzioni più potenti

Frequenza di clock

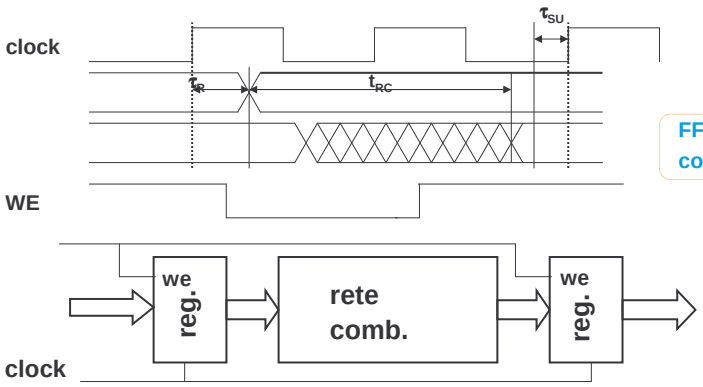
- In un sistema sincrono il campionamento dei segnali deve avvenire quando questi si sono stabilizzati



$$T > t_R + t_{RC} + t_{SU}$$

Multicycling

- La frequenza di clock può essere dettata da considerazioni di sistema e può essere maggiore di quella minima
- Il problema è risolubile tramite l'utilizzo di FF con segnali di write enable WE

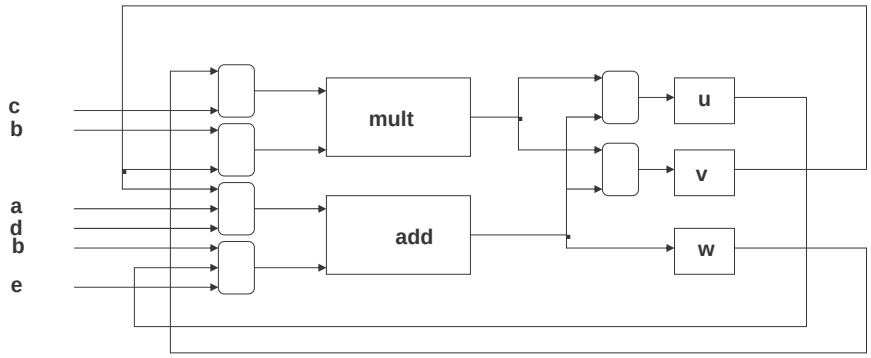


FF - edge triggered
con WE

$$kT > t_R + t_{RC} + t_{SU}$$

Esempio - I

L'espressione vista in precedenza può essere valutata ricorrendo a un solo moltiplicatore e a un solo addizionatore



Questo richiede l'utilizzo di registri per la memorizzazione dei risultati intermedi e di multiplexer per condividere le risorse

Esempio - II

- **Il calcolo del risultato richiede l'esecuzione delle seguenti operazioni**

  u=b*c

  v=a+b

  u=a+u

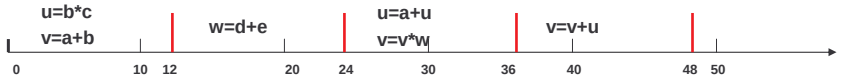
  w=d+e

  v=v*w

  v=v+u /* risultato */
- **L'ordine di queste operazioni è in parte arbitrario e verrà discusso nel seguito del corso**
- **Si osserva comunque che alcune di tali operazioni possono essere eseguite in parallelo dall'architettura illustrata**
- **Dal punto di vista delle prestazioni rimangono aperti alcuni problemi**
  - come scegliere il periodo di clock
  - quando eseguire le varie operazioni

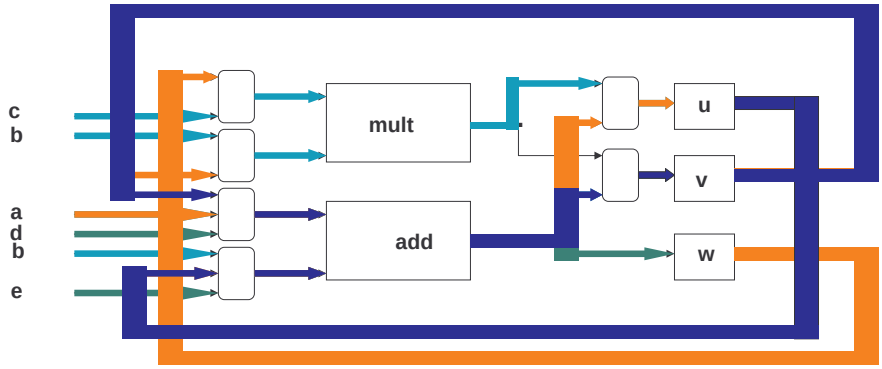
Esempio - III

- **Si facciano le seguenti ipotesi sui ritardi:**
  - $d_{add}=1/3d_{mult}=3.6ns$ e che sia $d_{mpx}=0.3ns$
- **Il ritardo massimo della rete combinatoria risulta pari a 11.7ns**
- **Si considerino due possibili periodi di clock**
  - T=12ns
  - T=5ns
- **Nel primo caso il periodo di clock è maggiore del ritardo massimo della rete combinatoria e quindi le operazioni possono essere programmate nel seguente modo**



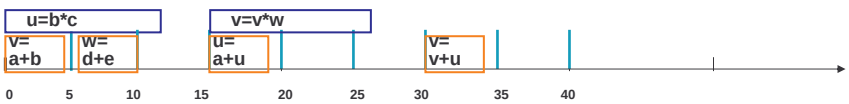
- **La latenza risulta chiaramente d=4*T=48ns**
- **Un risultato viene calcolato in 4 periodi di clock, quindi il numero di dati elaborati per periodo di clock è pari a 0.25 la frequenza di clock è f=83.3MHz**
- **$t=20.8*10^6$ (dati * s^{-1})**

Esempio (operazioni)



Esempio - IV

- Si consideri un periodo di clock pari a T=5ns
- Il periodo di clock è minore del ritardo massimo della rete combinatoria e quindi bisogna sfruttare registri con WE
- In particolare il ritardo di una somma è d=4.2ns e quella di un prodotto è d=11.7ns
- Quindi per una somma è necessario un periodo di clock e per un prodotto ne servono 3, chiaramente si usano FF con WE
- Le operazioni possono essere programmate nel seguente modo



- La latenza risulta chiaramente d=7*T=35ns
- Un risultato viene calcolato in 7 periodi di clock, quindi il numero di dati elaborati per periodo di clock è pari a 0.14 la frequenza di clock è f=200MHz
- $t=28*10^6$ (dati * s^{-1})

Limiti sul ritardo minimo

- **Esistono anche limiti sul ritardo minimo per evitare violazioni del tempo di hold dei flip-flop**

$$t_{RCmin} + t_R > t_H$$

- **Sono particolarmente importanti in circuiti veloci con pochi livelli di logica (ad esempio shift-register)**
- **Nelle architetture attuali, la frequenza di clock è determinata per una frazione consistente (10-30%) anche dallo skew dei segnali di clock**

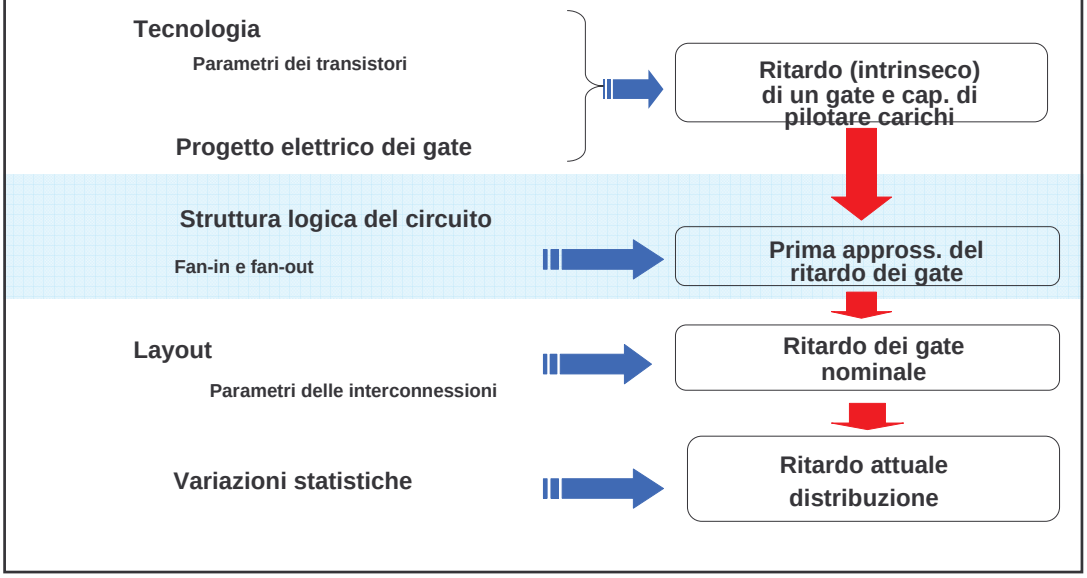
Ritardo di reti combinatorie: cammino critico

- **Per t_{RC} si intende il massimo ritardo della rete combinatoria (valutato supponendo che gli ingressi cambino tutti contemporaneamente)**
- **Tale ritardo, dipende da:**
  - tecnologia
  - implementazione fisica del circuito (layout e dimensionamento dei transistor)
  - struttura logica della rete
  - condizioni di funzionamento (T, V_{DD})
  - per una data tecnologia, implementazione e struttura logica:
    - coppia di vettori di ingresso
    - uscita
- **Problemi al livello logico**
  - stimare t_{RC}
  - verificare il circuito
  - modificare la struttura logica del circuito per ridurre t_{RC}

Conclusioni

- *Il legame fra architettura e prestazioni verrà discusso nella parte di sintesi ad alto livello*
- *Di seguito si considereranno i problemi relativi all'analisi delle prestazioni delle reti combinatorie che costituiscono l'ingrediente fondamentale per il calcolo delle frequenze di clock*

Ritardo di una rete combinatoria



Valutazione del cammino critico

- **Struttura della rete logica**
- **Modello del ritardo di propagazione dei gate**
  - singolo parametro (d)
  - ritardo in discesa e in salita
  - modello *timing arcs*
  - modello *pattern dependent*
- **Tempo di arrivo di una transizione**
  - distribuzione
  - Early arrival time (EAT) e Latest stabilization time (LST)
- **Algoritmo per il calcolo di t_{RC}**
  - impossibilità di usare un metodo esaustivo: 2^{2n} vettori di ingresso
  - necessità di un metodo algoritmico

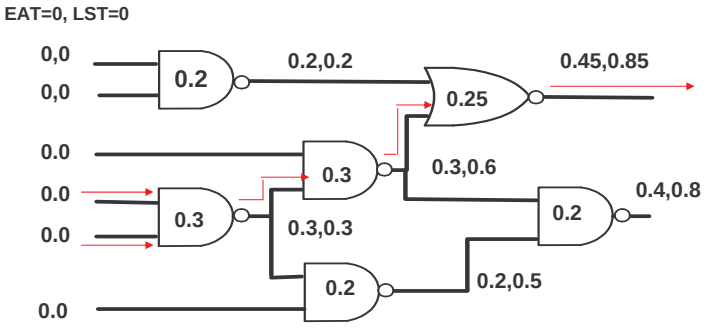
Algoritmo: static timing analysis

```
forall gates
  evaluate prop_delay[gate];
  evaluate level[gate];
forall inputs
  eat[input]=0;
  lst[input]=0;
forall gates
  push(gate,stack[l]);
for (l=1; l<=lmax; l++)
  while (stack[l]!=empty)
    gate=pop(stack[l]);
    eat[gate]=prop_delay[gate]+min(eat of fan_in);
    lst[gate]=prop_delay[gate]+max(lst of fan_in);
trcmin=min(eat of gates);
trc=max(lst of gates);
```

Esempio

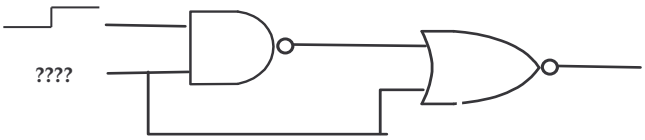
$$d = d_0 + d_1 \cdot fo$$

gate	d_0	d_1
NAND	0.1*ns*	0.1*ns*
NOR	0.1*ns*	0.15*ns*
...	...	...



Problema dei false path

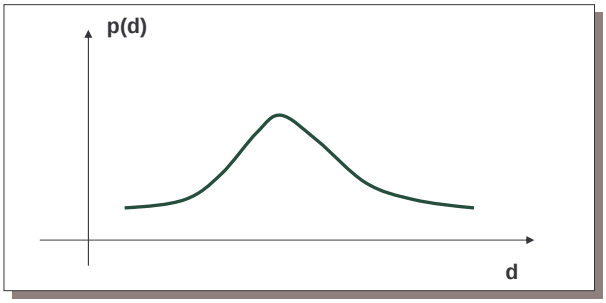
- Nei circuiti possono esistere cammini non sensitizzabili funzionalmente



- L'algoritmo STA in tale caso può sovrastimare t_{RC}
- Esistono degli algoritmi in grado di tenere conto del problema

Aspetti statistici

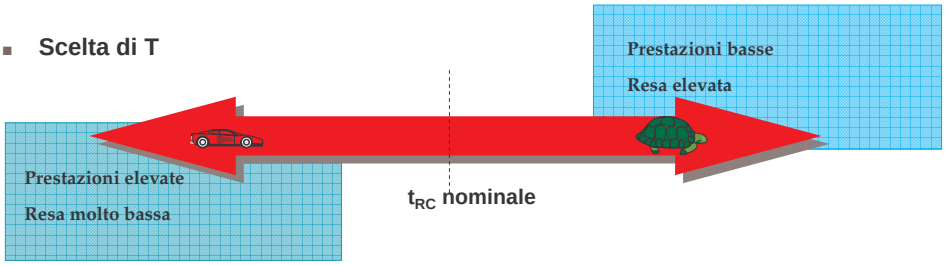
- I parametri elettrici di dispositivi e interconnessioni sono caratterizzati da una dispersione statistica che tende ad aumentare nelle tecnologie correnti
- Il ritardo dei gate ha una distribuzione statistica



- Anche t_{RC} ha una distribuzione statistica

Aspetti statistici

- Scelta di T



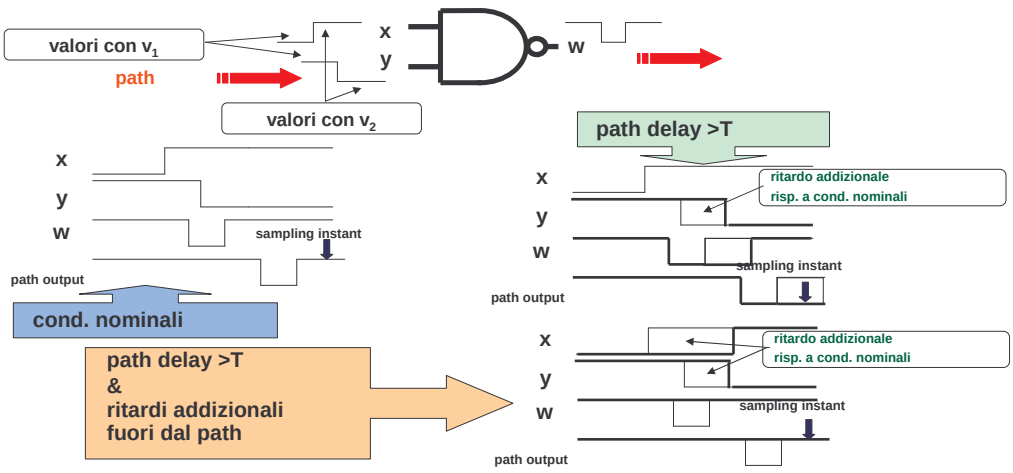
- T può essere scelto in modo ottimale se è nota la distribuzione di t_{RC}
- Bisogna generare degli stimoli per determinare se ciascun circuito funziona correttamente dal punto di vista delle temporizzazioni
- Sulla base dell'esito di tale verifica di può aumentare T

Generazione di sequenze di stimoli per la verifica del timing - I

- **Si tratta di verificare che il circuito funzioni correttamente nel caso in cui una transizione sia propagata lungo i possibili critici del circuito**
  - la dispersione dei parametri circuitali, infatti, non rende possibile determinare a priori quale sarà il cammino più lungo in un circuito
  - per questo motivo si sceglie di verificare un insieme di possibili cammini (chiaramente selezionati fra quelli che presentano i ritardi maggiori in condizioni nominali)
- **Devono essere determinati due vettori $<t_1,t_2>$ tali da propagare un opportuna transizione attraverso il cammino da verificare**
- **L'applicazione della transizione di ingresso e il campionamento dell'uscita a cui tale transizione si propaga avvengono in accordo al periodo di clock T**

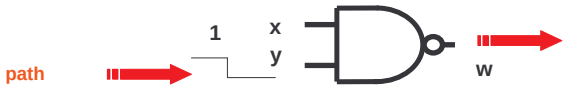
Generazione di sequenze di stimoli per la verifica del timing - II

- **Per permettere la propagazione della transizione, il cammino deve essere opportunamente sensibilizzato**
- **Bisogna evitare che lungo il cammino si propaghino delle alee che potrebbero mascherare gli effetti di un guasto**

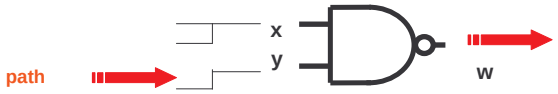


Generazione di sequenze di stimoli per la verifica del timing - III

- **Si vuole collaudare il cammino in modo che si campioni un valore errato se il suo ritardo è > T, in maniera indipendente da ogni altro ritardo del circuito**
- **Condizioni sui valori degli ingressi fuori dal path**
  - transizione di ingresso in discesa



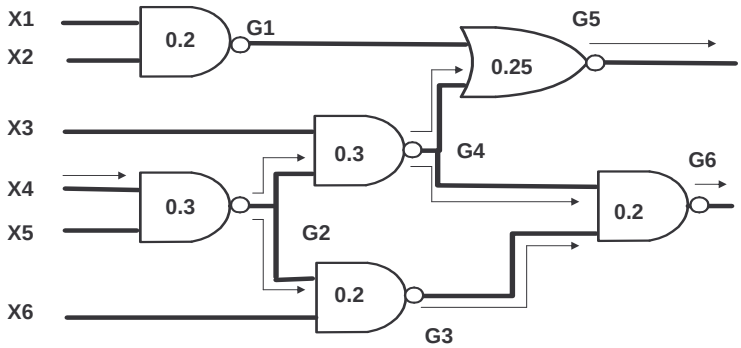
  - transizione di ingresso in salita



  - come mai in questo caso y può avere due diversi valori con v_1 ?

Esempio

- **Non si sa a priori quale è il cammino critico**
- **In un circuito di grandi dimensioni c'è un numero esponenziale di path**
- **Si considerano quelli col maggiore ritardo nominale**

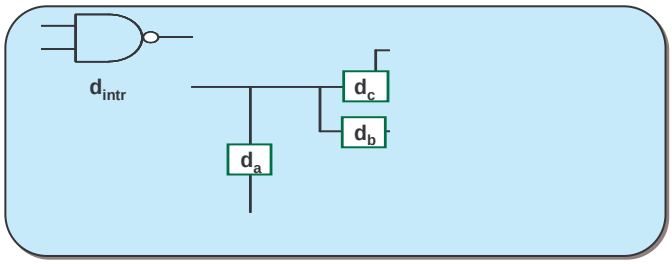


Esempio

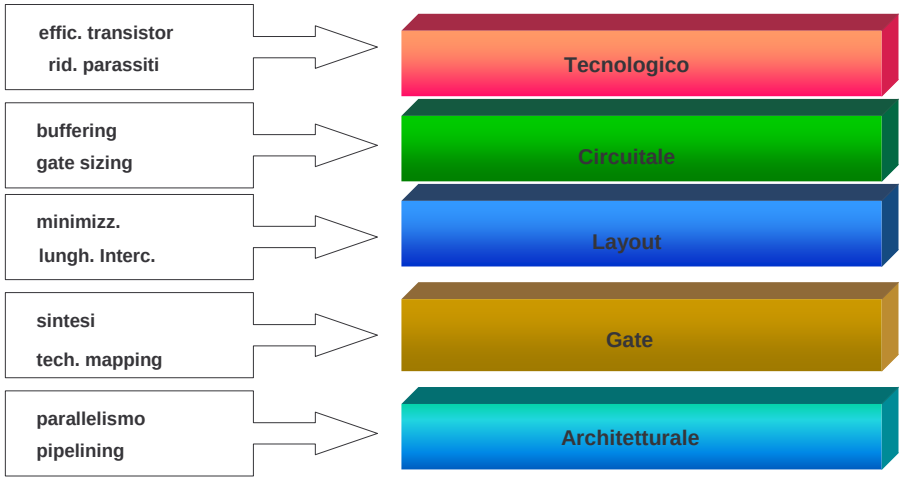
PATH	t_{nom}	$<t_1,t_2>$
X4,G2,G4,G5	0.85	11101- 11111-
X4,G2,G4,G6	0.8	--1010 --1110
X5,G2,G4,G6	0.8	--0110 --1110
X4,G2,G3,G6	0.7	--0011 --0111
X5,G2,G3,G6	0.7	--0101 --0001
X5,G2,G4,G5	0.85	11011- 11111-

Verifica delle prestazioni dopo il layout

- **Nelle tecnologie attuali una frazione consistente del ritardo è dovuto alle interconnessioni**
- **Le metodologie illustrate fno ad ora servono per**
  - una prima stima delle prestazioni
  - supportare algoritmi per il miglioramento delle prestazioni
- **Il ritardo deve essere stimato dopo la realizzazione del layout**
- **Le interconnessioni non possono essere modellate come una capacità di carico**



Miglioramento delle prestazioni dinamiche



Miglioramento delle prestazioni a livello gate

- **I metodi che riducono l'area non danno luogo a circuiti ottimizzati dal punto di vista delle prestazioni a causa dell'utilizzo del fan-out**
- **L'ottimizzazione dell'area ha comunque degli effeti positivi sul ritardo in quanto un layout compatto riduce la lunghezza delle interconnessioni**
- **Metodi di riduzione di t_{RC} in reti combinatorie:**
  - riduzione della profondità logica del circuito
  - riduzione del fan-out
  - semplificazione della logica sul critical path
- **Ciascuno di questi metodi presenta dei problemi che ne limitano in parte l'efficacia**

Riduzione della profondità logica

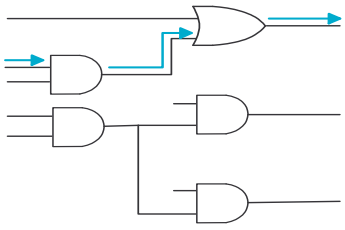
- **Stile di progetto basato su espressioni di tipo SP o PS**
  - ben si adatta a stili di progettazione basati su PLA
  - la dipendenza di t_{gate} dal fan-in invalida la riduzione del ritardo
  - molte funzioni non possono essere realizzate in maniera efficiente con espressioni SP
- **Nella sintesi multilivello si può minimizzare localmente la profondità logica se questo consente di non aumentare il ritardo**

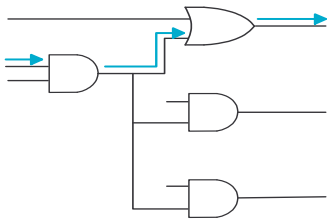
$$(a + b) \cdot c + d = a \cdot c + b \cdot c + d$$

Riduzione del fan-out

- **Buffering**
- **Limitare il fan-out (es. fan-out < 3,4)**

Circuito con il fan-out sfruttato a pieno

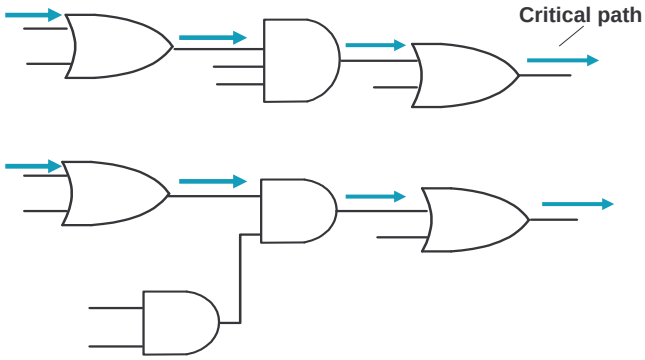




Circuito con il fan-out limitato

Semplificazione della logica sul cammino critico

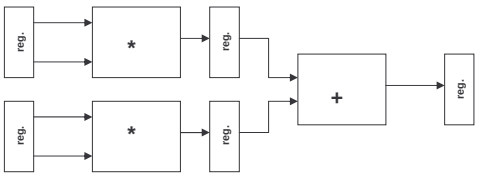
- I blocchi logici sul critical path possono essere semplificati spostando fuori dal path alcune operazioni



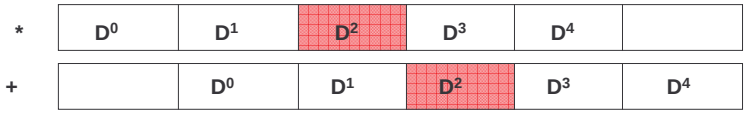
- Bisogna fare attenzione a non creare un nuovo critical path

Pipelining

- L'idea base dei sistemi di tipo pipelined è data dal cercare di utilizzare ciascuna risorsa nel modo più efficiente possibile
- In un sistema pipelined, una volta che un unità ha calcolato il suo risultato, tale unità può passare ad elaborare il dato successivo
- Questo richiede l'inserimento di registri che garantiscono il corretto sequenziamento dei dati



- Operazioni pipelined



Pipelining - periodo di clock

- Il critical path è dato dal maggiore fra quello dell'adder e del moltiplicatore

  $T>\max\{t_{RCmult}, t_{RCadd}\}+t_R+t_{SU}$

- Questo implica che i ritardo di ciascuno stadio della pipeline deve essere bilanciato (cosa che non accede nell'esempio precedente)
- Si supponga di voler realizzare una versione pipelined a *n* stadi di un blocco combinatorio con ritardo t_{RC}
- Nell'ipotesi di ritardi bilanciati, risulta

  $T>\max\{t_{RCi}\}+t_R+t_{SU}= t_{RC}/n+t_R+t_{SU}$

Pipelining - latenza e throughput

- La latenza di un operazione (il tempo che intercorre fra l'inizio dell'elaborazione di un dato e la produzione in uscita del risultato) risulta pari a

  $nT= t_{RC}+n(t_R+t_{SU})$

- Esistono strategie di clock che eliminano il problema dei tempi dei FF
- Dal punto di vista del throughput si osserva che il numero di dati elaborati per periodo di clock (dati di cui inizia l'elaborazione) è pari a 1 come nel caso combinatorio
- Quindi il miglioramento del throughput dipende dall'aumento della frequenza di clock:
- Caso combinatorio: $t=(t_{RC}+t_R+t_{SU})*1$
- Caso pipelined: $t= (t_{RC}/n+t_R+t_{SU})*1$
- Trascurando i tempi dei FF, l'aumento di throughput è pari al numero di stadi inseriti della pipeline

Pipelining: problemi

- **Abbiamo utilizzato l'ipotesi che le risorse siano fra loro indipendenti, se così non è la pipeline può avere degli stalli (CPU)**
- **Minore è il ritardo della parte combinatoria, maggiore è l'effetto dei tempi di risposta e di setup dei FF**
- **Non ha senso aumentare *n* indefinitamente**
- **La tendenza è quella di avere pochi 2-4 livelli di logica per stadio**