Prestazioni dinamiche dei circuiti digitali

M. Favalli

DE: Department of Engineering Ferrara

Introduzione

- Le prestazioni sono attualmente il parametro di progetto piú importante nei circuiti digitali
- Le prestazioni devono essere adeguate alle specifiche di progetto che possono chiedere di
  - calcolare un risultato entro un certo tempo
  - produrre risultati con un certo rateo
  - svolgere queste operazioni con un certo consumo di potenza e energia
- Qui si considereranno gli aspetti relativi alla velocitá computazionale lasciando ai corsi di elettronica quelli energetici

Sommario

- **Prestazioni dinamiche di un circuito digitale**
  - parametri da cui dipendono le prestazioni
  - 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 massima di una rete sincrona a partire dal suo ritardo massimo
- **Analisi delle prestazioni**
  - algoritmo per la valutazione del ritardo massimo (statico) e del cammino critico
- **Metodologie per il miglioramento di prestazioni**
  - livello tecnologico, elettrico e di layout
  - livello gate
  - livello architetturale (parallelismo, pipelining)

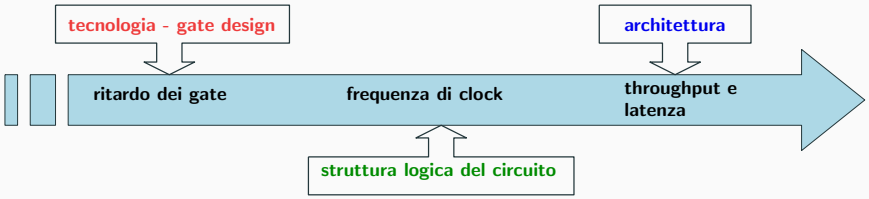
Banda e latenza dei sistemi digitali

Prestazioni: tipi di specifica

- 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, processori grafici, audio e video
  - **latenza (ritardo nell'eseguire un compito)**: microcontrollori, sistemi real-time
- Questo tipo di specifiche sono del tutto indipendenti dalla modalitá di funzionamento del circuito
- Le specifiche indirizzano le scelte architetturali del progettista

Throughput o banda

- In prima approssimazione si puó definire il throughput come la *quantitá di informazione elaborata nell'unitá di tempo*
- In un sistema sincrono, ignorando per il momento il layout, si puó dire che la banda dipende da
  - architettura
  - frequenza di clock
- La frequenza di clock dipende da
  - struttura del circuito
  - ritardo dei gate



Esempio di misure di banda

- Esempio storico che riguarda CPU per applicazioni di calcolo
- MIPS (Million of Operations per Second)

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

- Misura dipendente da algoritmo e dati
- É noto che tale misura puó dare luogo ad errori nella valutazione delle prestazioni di una CPU in quanto non tiene conto della potenza delle singole istruzioni
- Attualmente si considerano decine o centinaia di GFLOPS (miliardi di operazioni in floating point per secondo) per macchine multicore che comunque contengono piú CPU (4-16) sullo stesso chip

Banda dei sistemi sincroni

- Nel caso di sistemi sincroni la quantitá di informazione elaborata nell'unitá di tempo (b) puó essere data come

$$b = f_{ck} n_{ck}$$

- Dove f_{ck} é la frequenza di clock e n_{ck} é il numero di informazioni/dati elaborati in un periodo di clock
- Il tipo di informazioni dipende dall'applicazione considerata
  - in una CPU si considerano le istruzioni
  - in un DSP che elabora i dati provenienti da un convertitore A/D, si considerano i campioni elaborati
- Come si vedrá, é molto difficile aumentare contemporaneamente f_{ck} e n_{ck}

Legame fra MIPS frequenza di clock e architettura

- Il numero medio di periodi di clock per istruzione (CPI) permette di descrivere i MIPS in funzione della frequenza di clock
- Sia n_{istr} il numero di istruzioni eseguite, n_{ck} il numero di cicli di clock e $T_{ck} = 1/f_{ck}$ il periodo di clock
  - $CPU\ time = n_{ck} \times T_{ck}$
  - $n_{ck} = CPI \times n_{istr}$
- Si ha quindi

$$MIPS = \frac{n_{istr}}{CPU\ time \times 10^6} = \frac{n_{istr}}{n_{ck} \times T_{ck} \times 10^6} = \frac{n_{istr}}{n_{istr} \times CPI \times T_{ck} \times 10^6} = \frac{1}{CPI \times T_{ck} \times 10^6} = \frac{f_{ck}}{CPI \times 10^6}$$

Considerazioni

- Una CPU che implementava un instruction set molto semplice aveva una frequenza di clock elevata (logica semplice) e un CPI basso (RISC)
- Una CPU che implementava istruzioni piú complesse (CISC) aveva una frequenza di clock inferiore e un CPI piú elevato
- Questo poteva peró essere compensato dalla maggior potenza delle singole istruzioni
- Nel caso di Digital Signal Processors le misure di banda caratterizzano meglio le prestazioni dei circuiti
- Il numero di campioni di segnale elaborati nell'unitá di tempo é una tipica specifica di throughput

Latenza

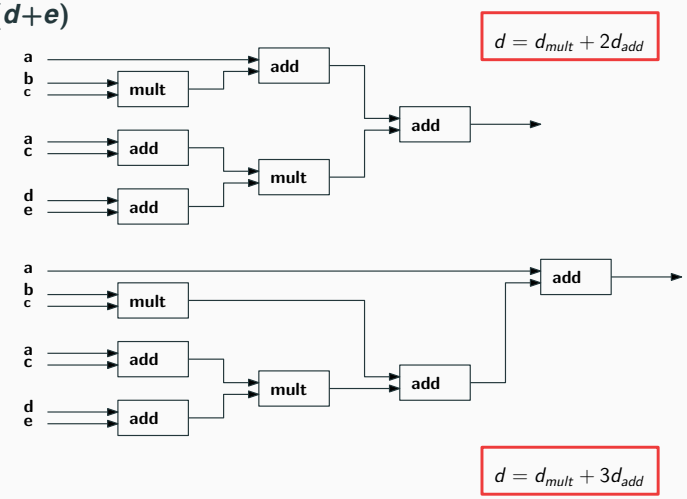
- Si considerino applicazioni di tipo reattivo che ricevono un qualche stimolo e devono fornire entro un certo tempo il risultato di un elaborazione (sistemi real-time)
- In questo caso, é evidente che la **latenza** é la principale cifra di merito del sistema considerato
- In alcuni tipi di sistemi, latenza e throughput sono spesso specificati contemporaneamente nell'ambito dello stesso progetto

Latenza nel calcolo di un espressione

- Espressione

  $out = a+bc+(a+b)(d+e)$

  (dove $a, b, \ldots$ sono interi senza segno)
- Utilizzo di una rete combinatoria composta di sommatori e moltiplicatori
- Architetture alternative con diversi valori di latenza



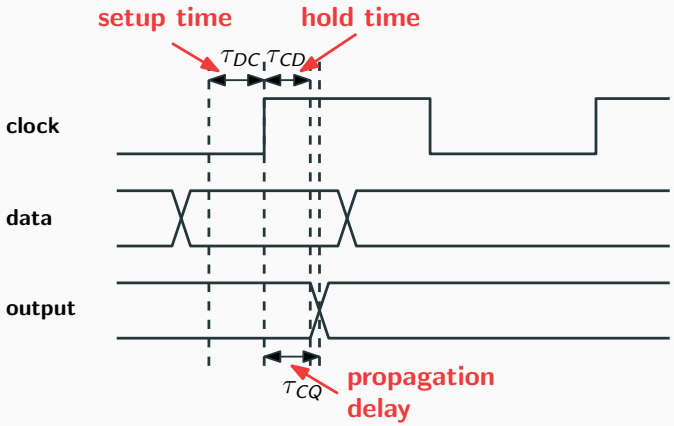
Dimensionamento della frequenza di clock

Frequenza di clock di rete sincrone

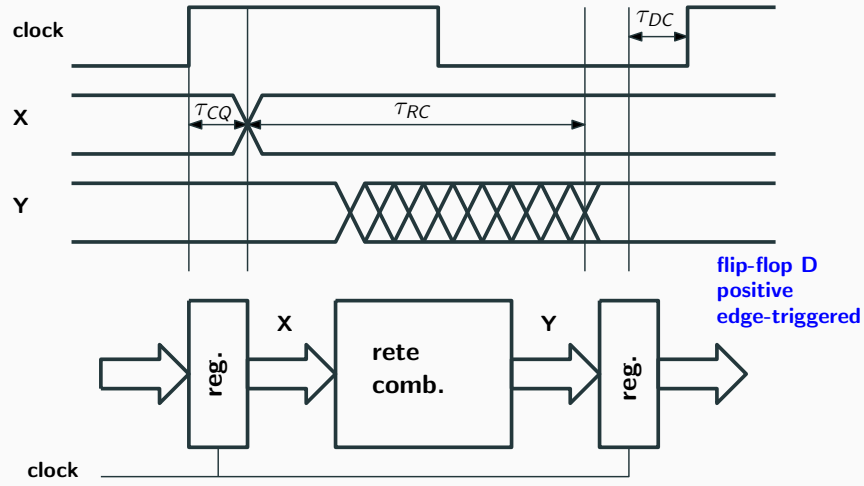
- Nelle reti di tipo sincrono gli ingressi delle reti combinatorie sono fornite da registri (flip-flop) e le uscite di tali reti vanno poi in ingresso a registri
- I registri sono controllati dallo stesso clock con periodo T_{ck} e frequenza $f_{ck} = 1/T_{ck}$
- Le reti combinatorie sono caratterizzate dal loro ritardo massimo
- I flip-flop sono caratterizzati dal loro tempo di risposta, dal tempo di setup e da quello di hold
- **In un sistema sincrono il campionamento dei segnali in ingresso ai flip-flop deve avvenire quando questi si sono stabilizzati**

Parametri da cui dipende la frequenza di clock in condizioni ideali

- Tempo di risposta dei flip-flop τ_{CQ}
- Ritardo massimo della rete combinatoria t_{RC}
- Tempo di setup dei flip-flop τ_{DC}



Vincolo sul periodo di clock minimo

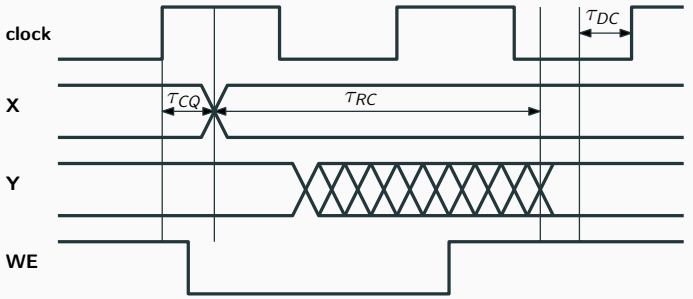


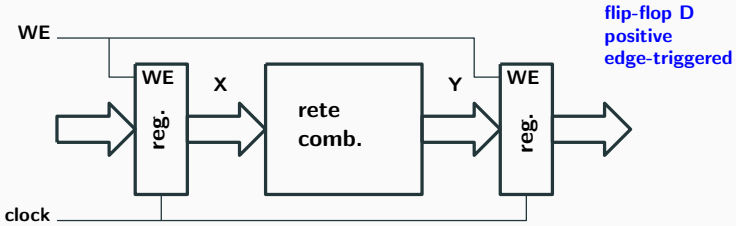
$$T_{ck} > \tau_{CQ} + \tau_{RC} + \tau_{DC}$$

Periodo di clock

- Il vincolo precedente (timing budget) se soddisfatto garantisce il corretto sequenziamento delle operazioni
- Cosa succede se la frequenza di clock é dettata da considerazioni di sistema e le reti combinatorie non soddisfano il vincolo precedente?
- Si utilizza il **multicycling** per cui la rete combinatoria dispone di piú periodi di clock per completare le sue operazioni
- Questa tecnica richiede l'utilizzo di flip-flop con il comando di write-enable

Multicycling





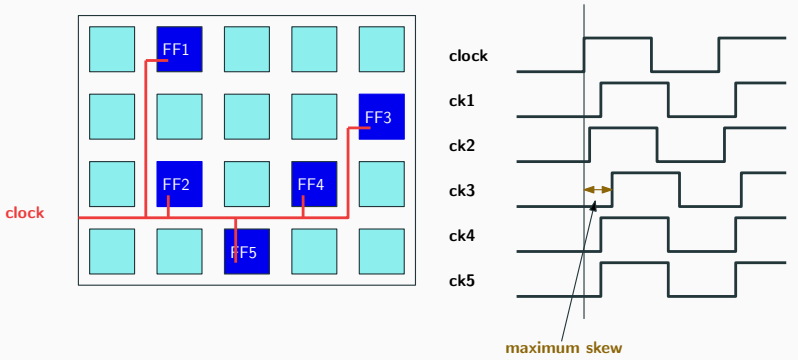
$$\mathbf{kT_{ck}} > \tau_{\mathbf{CQ}} + \tau_{\mathbf{RC}} + \tau_{\mathbf{DC}}$$

Caso ideale e caso reale

- I vincoli precedenti sul periodo di clock sono riferiti a un caso ideale dal punto di vista
  - della rete di distribuzione del clock: si suppone che la transizione del segnale di clock avvenga nello stesso istante in ingresso a ogni flip-flop
  - dei valori dei parametri di ritardo della rete combinatoria e dei flip-flop che si assume abbiano un valore nominale

Il problema della rete di distribuzione del clock

- Il segnale di clock percorre cammini di lunghezza diversa (e quindi con reti parassiti diverse) per raggiungere i diversi flip-flop
- Questo introduce sfasamenti di fra i tempi di arrivo dei fronti di campionamento



Come tenere conto dello skew sul segnale di clock

- La disequazione che determina il periodo di clock minimo diventa

$$T_{ck} > \tau_{cq} + \tau_{RC} + \tau_{dc} + \sigma_{max}$$

- Dove σ_{max} é lo skew massimo fra i diversi ingressi di clock dei flip-flop
- Se non si tiene conto del problema, a frequenze alte, si puó avere un 30% di budget utilizzato per lo skew del segnale di clock
- Il problema puó essere ridotto utilizzando: 1) metallizzazioni dedicate alle connessioni del segnale di clock; 2) bilanciando la rete di distribuzione (albero); 3) utilizzando buffer e altri circuiti per avere temporizzazioni simili

Limiti sul ritardo minimo

- Per evitare violazioni del tempo di hold, esistono vincoli sul ritardo minimo

$$t_{RCmin} + \tau_{CQ} > \tau_{CD} + \sigma_{max}$$

  - dove t_{RCmin} é il ritardo minimo della rete combinatoria
- Sono particolarmente importanti in circuiti con pochi livelli di logica come, ad esempio, gli shift-register
- Vengono analizzati dalla STA e possono essere soddisfatti aggiungendo linee di ritardo o utilizzando particolari tipi di elementi di memoria e strategi di clocking

Lo spazio di progetto dei sistemi digitali: costo e prestazioni

Compromesso fra costo e prestazioni

- Seppure meno importante delle prestazioni, il costo rimane comunque una specifica rilevante
- Ad esempio, la valutazione di espressioni in ASIC per DSP di solito non avviene tramite reti combinatorie
- L'utilizzo di una rete combinatoria richiede infatti un numero di risorse ridondante in quanto diverse operazioni potrebbero essere svolte dallo stesso componente
- Per abilitare la condivisione di risorse sequenziando correttamente le operazioni si utilizzano:
  - registri con write-enable per memorizzare risultati temporanei e sequenziare le operazioni
  - multiplexer per permettere allo stesso componente di operare su diversi operandi

Compromesso fra costo e prestazioni

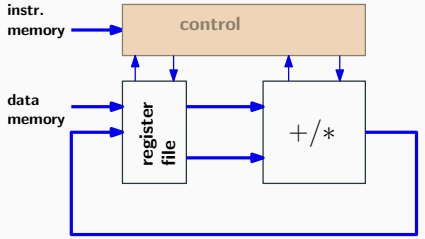
- Si considera di nuovo l'espressione

$$out = a + bc + (a + b)(d + e)$$

- Nella sua realizzazione combinatoria ogni operazione é mappata su un blocco funzionale
- In alternativa, il massimo della condivisione di risorse corrisponde a una semplice CPU (architettura di Von Neumann) che esegue un operazione per volta con una ALU
- Abbiamo quindi due estremi nello spazio di progetto
  - rete combinatoria che massimizza il parallelismo
  - semplice CPU che svolge le operazioni in maniera sequenziale
- Inizialmente analizzeremo queste due soluzioni che **non sono peró le uniche nello spazio di progetto**

Calcolo del periodo di clock minimo e della latenza

- Caso **parallelo** (combinatorio, mettendo un registro in ingresso e uno in uscita), il periodo di clock deve essere $T_p > \tau_{CQ} + d_{mult} + 2d_{add} + \tau_{DC}$ e la latenza $D_p = T_p$
- Caso **sequenziale** (CPU) per un possibile ordine delle istruzioni




```
u=b*c
v=a+b
u=a+u
w=d+e
v=v*w
v=v+u
```

- Quindi, $T_s > \tau_{CQ} + d_{st} + d_{mult} + \tau_{DC}$ e $D_s = 6T_s$
- Si noti che $d_{mult} > d_{add}$, che d_{st} tiene conto dei ritardi attraverso i multiplexer del register file e che non si é considerato il controllo

Elementi per calcolare costo e ritardo

- Si confrontano le due architetture con ipotesi molto semplici su costo e ritardi
  - si trascurano quelli relativi a flip-flop e multiplexer
  - se n é la dimensione delle parole (a, b, ...), il ritardo massimo di un semplice adder é proporzionale a n volte quello di un full-adder e quello di un moltiplicatore a $2n$ volte
  - nelle tecnologie attualmente utilizzate per circuiti full-custom, il ritardo di un full-adder (in un sommatore) va da 20ps a 50ps
  - il costo di un adder é proporzionale a n e quello di un moltiplicatore a n^2
- Chiaramente, si deve notare che la prima architettura é specializzata e la seconda é invece in grado di calcolare altre espressioni

Calcolo approssimato di latenza e costo in funzione della word width

- Quindi, si puó assumere che il periodo di clock minimo per l'architettura parallela sia $T_p = k(2n + 2n) = 4kn$ e per quella sequenziale $T_s = 2kn$
  - k é una costante che tiene conto di diverse cose e che un po irrealisticamente assumiamo essere la stessa
- Quindi per la latenza, $D_p = 4kn$ e $D_s = 12kn$
- Per il costo (area), abbiamo approssimativamente $A_p = j(2n^2 + 4n)$ e $A_s = j(n^2)$, dove si assume che l'adder sfrutti logica presente nel moltiplicatore e che per j valgano le stesse considerazioni fatte per k
- La soluzione parallela é circa 3 volte piú veloce e al crescere di n 2 volte piú costosa (per l'espressione considerata)

Un alternativa concorrente alle soluzioni completamente parallele e sequenziali

- Analizzando il codice si osserva che l'espressione puó essere eseguita dispondendo di un solo moltiplicatore e un solo addizionatore
- Diversamente dalla semplice CPU supporremo che le due operazioni possano essere svolte concorrentemente nello stesso ciclo di clock (non siamo piú in un caso sequenziale)
- Dal codice si osserva anche che l'ordine sequenziale delle istruzioni é soltanto parziale
  - **ogni operazione deve essere eseguita prima di quella che ne utilizza il risultato**

Ordinamento delle operazioni

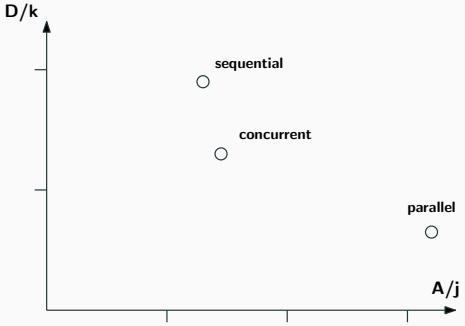
Possibile ordinamento delle operazioni che soddisfa il vincolo di precedenza e sfrutta la concorrenza di somme e prodotti per ridurre il numero di cicli di clock

cycle	mult	add
1	u=b*c	v=a+b
2		w=d+e
3	v=v*w	u=a+u
4		v=v+u

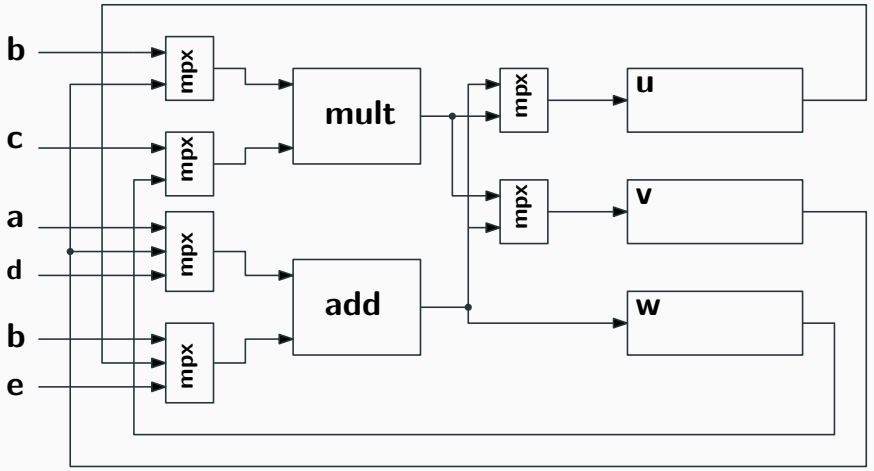
Ritardo, latenza e costo per l'implementazione concorrente

- Il vincolo su T_{ck} rimane simile a quello della CPU (i multiplexer qui sono piú semplici) $T_h > \tau_{CQ} + d_{mult} + t_{st} + \tau_{DC} = 2kn$
- La latenza risulta $D_h = 4T_h = 8kn$
- Il costo é piú o meno equivalente a quello della parte di data-path della CPU ($A_h = n^2 + n$)

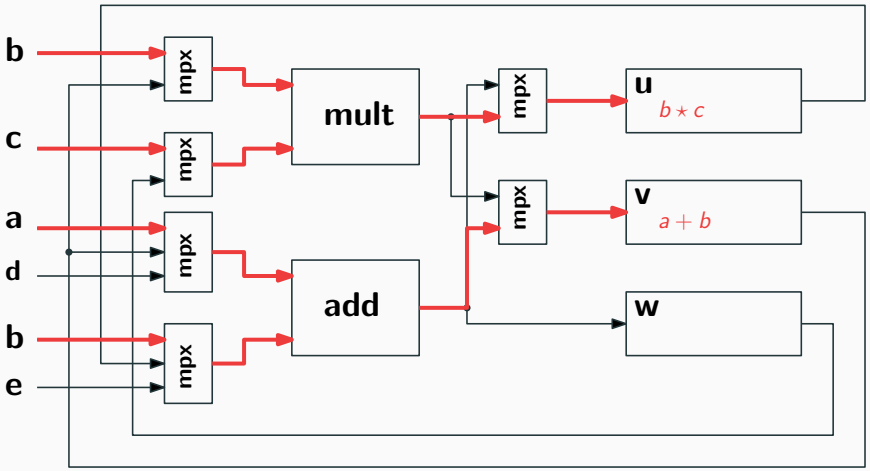
Punti nello spazio di progetto ($n = 8$)



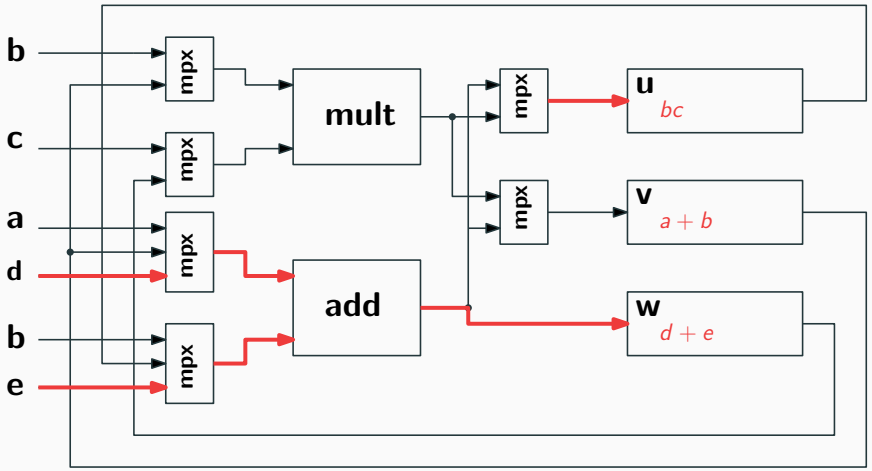
Schema circuitale e operazioni



Schema circuitale e operazioni

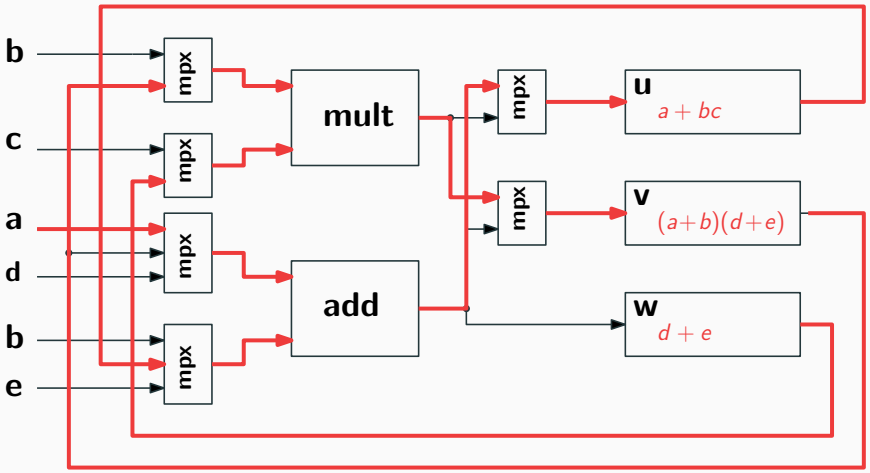


Schema circuitale e operazioni



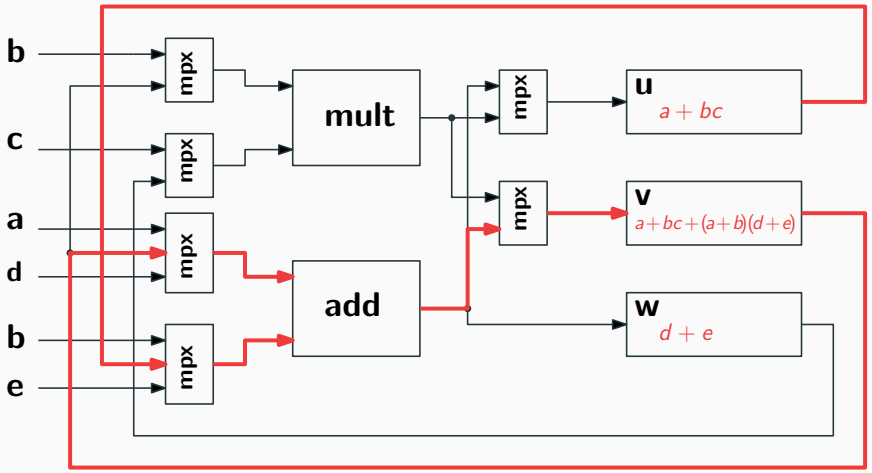
Lo spazio di progetto dei sistemi digitali: costo e prestazioni 28

Schema circuitale e operazioni



Lo spazio di progetto dei sistemi digitali: costo e prestazioni 28

Schema circuitale e operazioni



Lo spazio di progetto dei sistemi digitali: costo e prestazioni 28

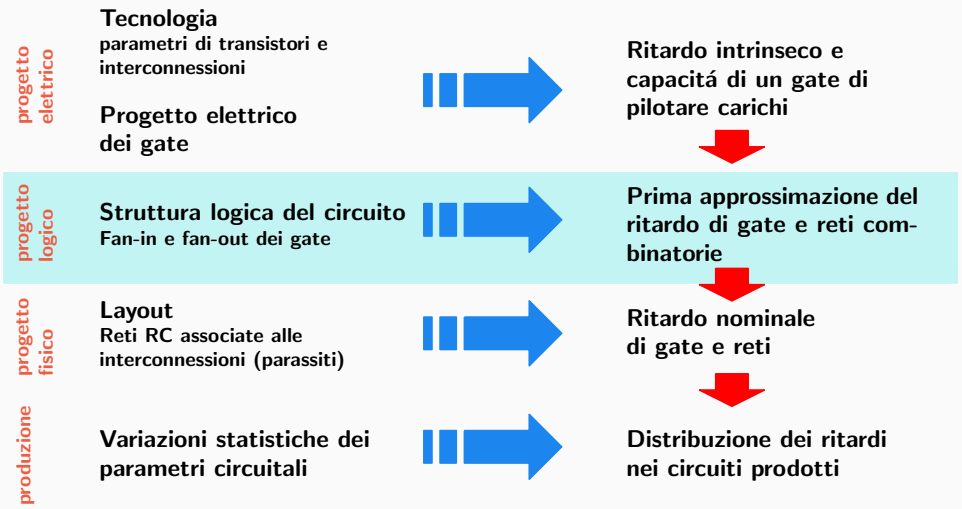
Banda

- In tutte le architteture considerate, la banda t é semplicemente uguale all'inverso della latenza:
  $t_p = 1/4kn \quad t_s = 1/12kn \quad t_h = 1/8kn$
- Allo stesso risultato si puó pervenire utilizzando l'espressione $b = f_{ck} n_{ck}$
- n_{ck} é il numero di dati elaborati per periodo di clock, quindi $n_{ck,p} = 1$, $n_{ck,s} = 1/6$ e $n_{ck,h} = 1/4$
- Ricordando che $T_p = 4kn$, $T_s = 2kn$ e $T_h = 2kn$, si ottiene lo stesso risultato mostrato nella tabella

Ritardo massimo delle reti combinatorie

- Si é visto il ruolo del ritardo massimo t_{RC} delle reti combinatorie nel calcolo del periodo minimo di clock
- Tale ritardo dipende da
  - tecnologia (caratteristiche di transistori e interconnessioni)
  - implementazione fisica del circuito (layout e dimensionamento dei transistori)
  - struttura logica della rete
  - condizioni di funzionamento (T e V_{DD})
  - per una data tecnologia, implementazione e struttura logica
    - coppia di vettori di ingresso
    - uscita
- Problemi al livello logico
  - stimare t_{RC}
  - determinare gli stimoli da utilizzare per verificare tale stima
  - modificare il circuito per ridurre t_{RC}

Parametri che determinano il ritardo di una rete combinatoria



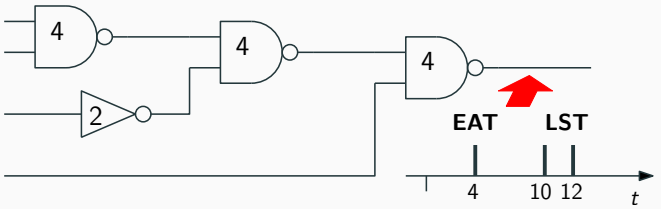
Static Timing Analysis

Calcolo del ritardo massimo

- La simulazione logica non é fattibile anche per reti di piccolissime dimensioni
  - un adder a 16 bit ha 33 ingressi e contiene 80 gate
  - richiede la simulazione di 2^{33} coppie di vettori di ingresso
- É stato quindi sviluppato un algoritmo alternativo detto di **Static Timing Analysis** (STA)
  - é un algoritmo approssimato
  - non ha un costo esponenziale (lineare nel numero di interconnessioni)

Static Timing Analysis

- Struttura della rete logica
- Ritardo di propagazione dei gate
  - singolo parametro (***d***)
  - ritardo in salita e discesa
  - modello *timing arcs*
  - modello *pattern dependent*
- Tempo di arrivo di una transizione
  - distribuzione per ogni segnale
  - **Early Arrival Time (EAT) e Latest Stabilization Time (LST)**



Static Timing Analysis

Interessano solo l'estremo superiore e quello inferiore dei tempi di arrivo che vengono stimati in maniera indipendente dagli ingressi (staticamente)

1. La STA inserisce gli ingressi in una lista
2. L'algoritmo procede fino a quando tale lista non é vuota
   - 2.1 estrae un elemento (gate o ingresso) dalla lista
   - 2.2 se l'elemento é un ingresso mette il suo EAT e LST a 0 e lo marca come assegnato
   - 2.3 se é un gate e tutto il suo fan-in é assegnato
     - calcola EAT come il minimo EAT del fan-in piú il ritardo del gate
     - calcola LST come il massimo LST del fan-in piú il ritardo del gate
     - marca il gate come assegnato
     - carica il suo fan-out nella lista (se non presente)
   - 2.4 uscita dal ciclo
     - il minimo EAT é uguale a $t_{RC,min}$
     - il massimo LST é uguale a $t_{RC,max}$ (che stima t_{RC})

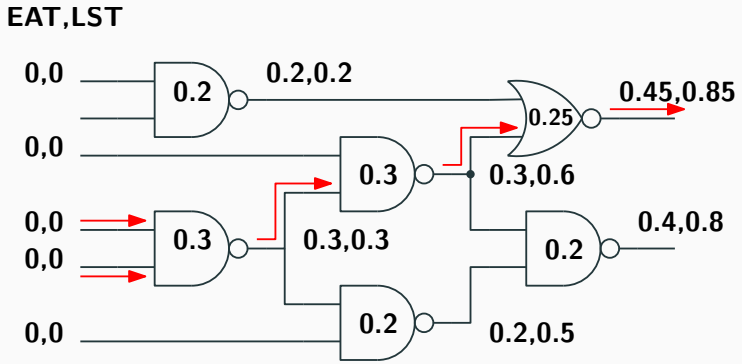
Static Timing Analysis: pseudocode

```
eval_list=empty;
forall gate
  evaluate prop_delay[gate];
  assigned[gate]=0;
forall input
  insert input to eval_list;
while eval_list!=empty
  extract gate from eval_list;
  if (gate==input)
    eat[gate]=0; lst[gate]=0;
    assigned[gate]=1;
  else if (assigned[fan_in]==1 forall fan_in of gate)
      eat[gate]=prop_delay[gate]+min(eat of its fan_in);
      lst[gate]=prop_delay[gate]+max(lst of its fan_in);
  forall fan_out of gate
    if fan_out is not in eval_list
      insert fan_out in eval_list;
  assigned[gate]=1;
trcmin=min(eat of gates); trcmax=max(lst of gates);
```

Esempio

Propagation delay $d = d_0 + d_1 fo$

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

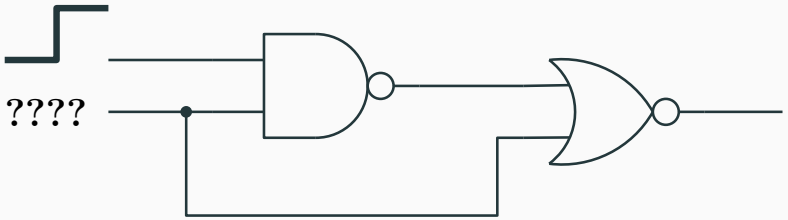


Critical path

- Una volta calcolati gli EAT e gli LST di tutti i segnali, si individua il critical path del circuito (potrebbero essercene piú di uno)
- Una transizione che si propaga lungo il critical path arriva in uscita con un ritardo pari a t_{RC}
- Per individuare tale cammino(i), si parte dall'uscita con il LST massimo e e si procede verso i PI selezionando l'ingresso di ciascuna porta logica con il LST massimo

Problema dei false path

- Nei circuiti possono esistere cammini statici non sensibilizzabili funzionalmente



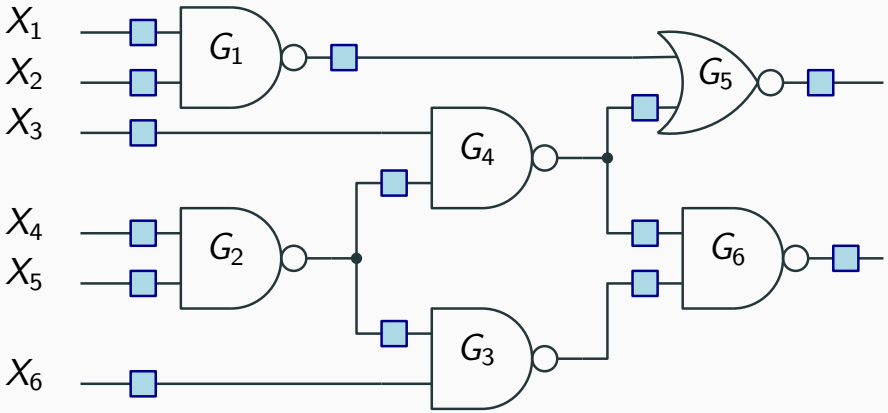
- La STA in tali casi puó sovrastimare il ritardo massimo
- Esistono algoritmi in grado di tenerne conto

Verifica delle prestazioni dopo il layout

- Nelle tecnologie attuali una frazione consistente del ritardo é dovuto alle interconnessioni che non possono essere modellate come una capacitá di carico
- Il ritardo di un gate dipende da quale ingresso commuta
- Le metodologie illustrate fino 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 con un modello piú accurato del ritardo di propagazione
- Questo ruolo é svolto dal modello di ritardo pin-to-pin o timing arc in cui il ritardo associato all'interconnessione e quelli del gate sono annotati sul fan-in del gate stesso

Esempio di circuito annotato con ritardi di tipo pin-to-pin

Ciascun quadratino corrisponde a un diverso valore di ritardo



Descrizione VHDL di un NAND con ritardo di tipo timing arc

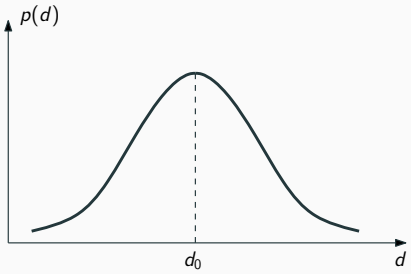
- Per semplicitá non differenziamo fra ritardo in salita e in discesa
- Si noti che questo modello é ibrido dal punto di vista del ritardo inerziale/trasporto

```
entity nand2 is
 generic(d1,d2: time);
 port(in1,in2: in std_logic;
      output: out std_logic);
end entity nand2;

architecture behav of nand2 is
signal tmp1,tmp2: std_logic;
begin
  tmp1<=in1 after d1;
  tmp2<=in2 after d2;
  output<=tmp1 nand tmp2;
end architecture;
```


Aspetti statistici dei ritardi

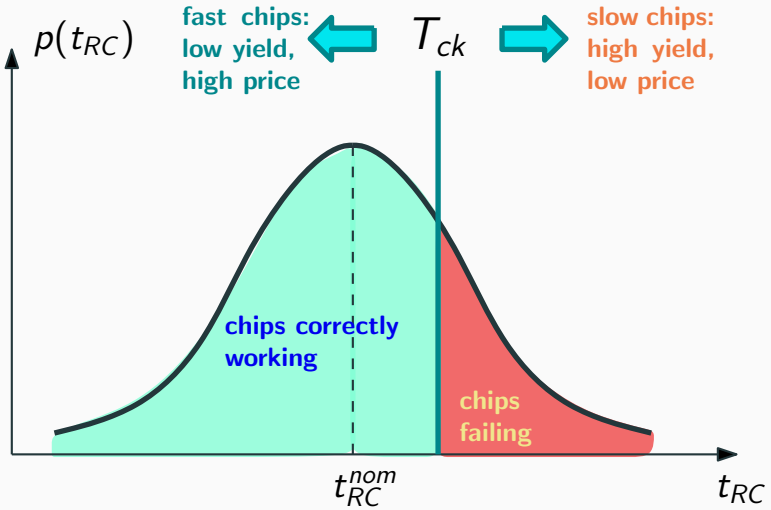
- I parametri elettrici di dispositivi e interconnessioni sono caratterizzati da una dispersione statistica che tende ad aumentare con la diminuzione delle dimensioni dei dispositivi
- Il ritardo dei gate cambia da gate a gate nello stesso chip e da integrato a integrato per lo stesso chip
- Il ritardo ha una distribuzione di probabilitá approssimabile con una normale
- Di conseguenza, anche t_{RC} ha una distribuzione statistica



- Copie diverse dello stesso schema circuitale possono avere critical path diversi

Aspetti statistici nel dimensionamento del periodo di clock

- T_{ck} puó essere scelto in modo ottimale se é nota la distribuzione di t_{RC}
- La scelta determina resa e prezzo di vendita del chip



Verifica delle temporizzazioni e adattamento della frequenza di clock

- Bisogna generare degli stimoli per determinare se ciascun circuito prodotto funziona correttamente dal punto di vista delle temporizzazioni
- Sulla base dell'esito di tale verifica, si puó aumentare T_{ck} arrivando ad avere piú versioni dello stesso integrato che lavorano con frequenze di clock diverse
- Questa operazione nelle CPU recenti viene fatta durante il normale funzionamento del circuito adattandosi a temperatura, alimentazione e anche invecchiamento

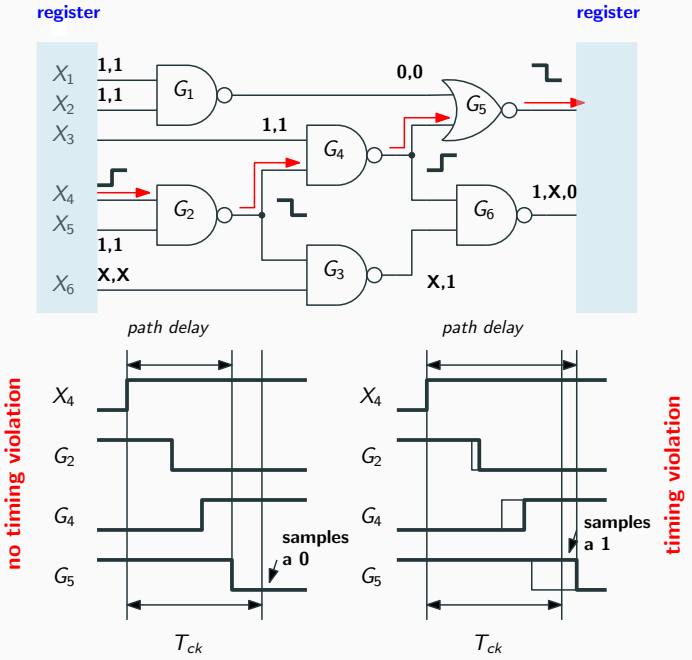
Generazione di sequenze di stimoli per la verifica del timing - I

- Non é possibile applicare tutte le possibili coppie di vettori di ingresso
- In alternativa, si potrebbero verificare i cammini critici del circuito in condizioni nominali
- La dispersione dei parametri circuitali rende non affidabile questo approccio perché non si puó determinare a priori il cammino piú lungo
- Non é possibile verificare tutti i cammini che possono essere in numero esponenziale rispetto ai gate
- Si sceglie di verificare un **insieme di possibili cammini selezionando quelli che presentano i ritardi maggiori in condizioni nominali**

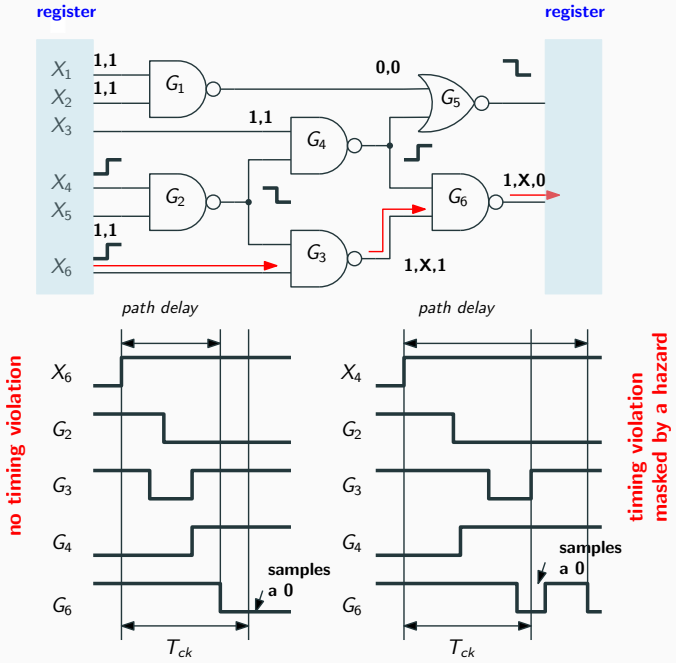
Generazione di sequenze di stimoli per la verifica del timing - II

- Per ogni cammino e tipo di transizione (rise, fall) devono essere determinati due vettori $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ tali da propagare la transizione attraverso tale cammino
- Se la transizione in ingresso viene applicata al tempo $\boldsymbol{t = 0}$, il campionamento dell'uscita a cui tale transizione si propaga avviene a T_{ck}
- L'osservazione del valore campionato determina se c'é stato un errore o meno
- La transizione dovrebbe propagarsi senza hazard per evitare che tale fenomeno transitorio mascheri un ritardo del cammino che da luogo a una violazione dei vincoli sul timing e che in condizioni diverse potrebbe dare luogo a un errore

Esempio



Esempio dei problemi dati dalla presenza di hazard



Esempio di test set

- I test che collaudano un cammino senza hazard sono chiamati robusti
- Nel caso di reti realistiche, il calcolo di questi test é un processo complesso e computazionalmente costoso
- Qui abbiamo un esempio di test per i cammini della rete con ritardo $\geq$ **0.7**

PATH	t_{nom}	$\langle u, v \rangle$
X_4, G_2, G_4, G_5	0.85	11101X 11111X
X_4, G_2, G_4, G_6	0.8	XX1010 XX1110
X_5, G_2, G_4, G_6	0.8	XX0110 XX1110
X_4, G_2, G_3, G_6	0.7	XX0011 XX0111
X_5, G_2, G_3, G_6	0.7	XX0101 XX0111
$X_5, G_2.G_4, G_5$	0.85	11011X 11111X

Miglioramentio delle prestazioni dinamiche

Miglioramento delle prestazioni dinamiche

miglioramento dei transistori e riduzione delle capacitá parassite	$\Rightarrow$	**livello tecnologico**
buffering e gate sizing	$\Rightarrow$	**livello circuitale**
minimizzazione della lunghezza delle interconnessioni	$\Rightarrow$	**livello di layout**
sintesi e technology mapping orientati alla riduzione dei ritardi	$\Rightarrow$	**livello gate**
parallelismo e pipelining	$\Rightarrow$	**livello architetturale**

Miglioramento delle prestazioni al 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 effetti positivi sul ritardo in quanto un layout compatto riduce la lunghezza delle interconnessioni
- Metodi di riduzione di t_{RC} in reti combinatorie:
  - riduzione della profonditá 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 profonditá logica - I

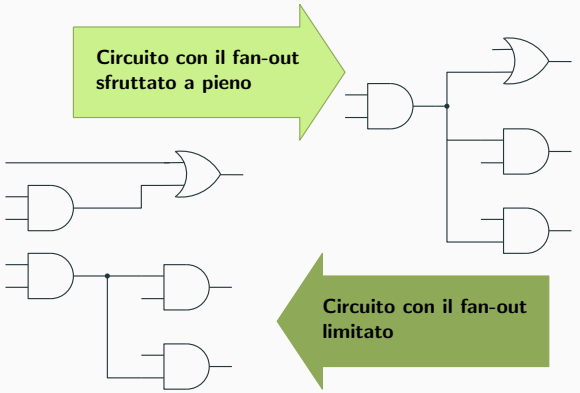
- Esiste chiaramente una relazione fra la profonditá logica e il ritardo
  - la condizione ottimale é in teoria data dall'utilizzo di espressioni SP o PS che si usavano con le PLA
  - alcune funzioni hanno un costo proibitivo se implementate in tale modo
  - attualmente si utilizza logica multilivello basata su gate con un fan-in minore o uguale a 4 per considerazioni tecnologiche
  - le tecnologie correnti hanno meno problemi a realizzare gate con fan-in elevato (FinFET)
- Invece di avere tutta la rete realizzata a 2 livelli, si puó applicare questa trasformazione localmente in una rete multilivello

Riduzione della profonditá logica - II

- Esempio di riduzione della profonditá logica (prima del technology mapping)
- Sia y un segnale in una rete multilivello descritta come insieme di equazioni logiche
  - $y = a(b + a(c + b'(d' + e)))$: costo 7 letterali, profonditá logica 6
  - $y = ab + ac + ad' + ae$: costo 8 letterali, profonditá logica 2

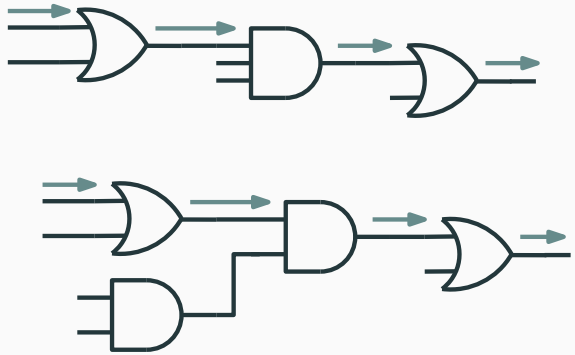
Riduzione del fan-out

- Il ritardo dei gate é in buona parte proporzionale al fan-out
- In presenza di fan-out elevati si puó utilizzare il buffering o limitare il fan-out (ad esempio ≤ 4)
- Questo richiede la duplicazione di parte della logica e quindi puó aumentare il fan-out dei gate a monte



Semplificazione del cammino critico

- I blocchi logici lungo il critical path possono essere semplificati spostando fuori da esso alcune computazioni
- In questo modo i gate sul critical path avranno un ritardo minore
- Bisogna stare attenti a non creare altri cammini critici
- Estendendo questo approccio si arriva al path balancing



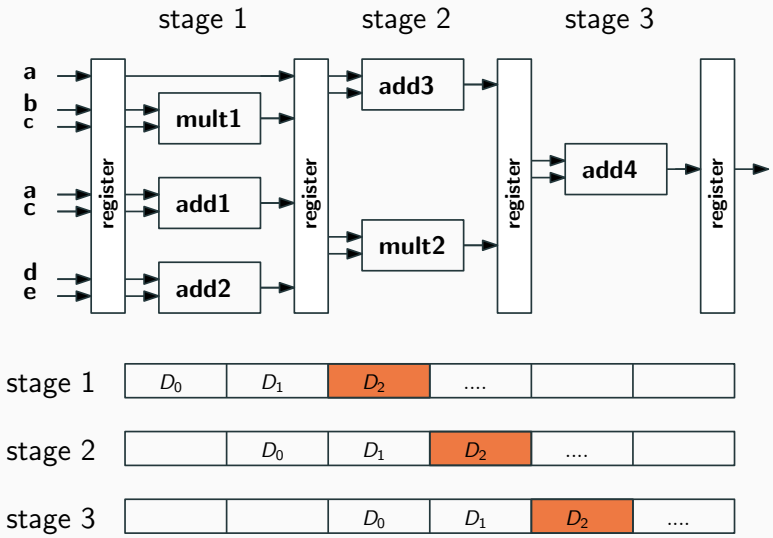
Considerazioni sul miglioramento delle prestazioni a livello gate

- I circuiti ottimizzati dal punto di vista del ritardo tendono ad avere i cammini bilanciati e quindi una distribuzione dei ritardi piú stretta di quella dei circuiti ottimizzati rispetto all'area
- Nel caso di circuiti aritmetici si possono utilizzare considerazioni algoritmiche nel miglioramento delle prestazioni (es. ridurre the catene di propagazione dei riporti)
- Nel caso di reti sequenziali in alcuni casi si puó spostare della logica attorno ai flip-flop
- Queste tecniche di ottimizzazione producono in genere risultati inferiori rispetto a quelle al livello architetturale descritte nel seguito

Pipelining

- Il pipelining é un tecnica al livello architetturale che serve ad aumentare la banda
- Torniamo alla rete combinatoria che calcolava l'espressione e notiamo che le risorse funzionali non lavorano per un tempo pari a t_{RC}, ma solo per una parte di questo
- Si potrebbe pensare di inserire nuovi dati prima che il risultato dei precedenti sia stato calcolato, ma il diverso ritardo dei cammini nella rete mischierebbe i due insiemi di dati
- Sono quindi possibili 3 alternative che garantiscono il corretto sequenziamento dei dati
  - equalizzazione dei ritardi lungo i diversi cammini (wave pipelining): tecnologicamente quasi infattibile
  - fare in modo che ogni componente segnali ai successivi quando ha finito il calcolo: pipeline asincrona
  - **dividere la rete in stadi utilizzando registri controllati da un segnale di clock**: pipeline sincrona

Esempio



Frequenza di clock

- Il valore di t_{RC} nell'esempio considerato é dato dal massimo ritardo del moltiplicatore
- Si ha quindi $T_{ck} > t_{mult} + \tau_{DC} + \tau_{CQ}$
- Si noti che il massimo ritardo combinatorio dei singoli stadi é diverso, ma nel caso sincrono interessa solo quello massimo
- **Conviene avere il miglior bilanciamento possibile fra i ritardi dei diversi stadi**
- Se in una pipeline a n stadi ottenuta da una rete combinatoria con ritardo massimo t_{comb}, la condizione di bilanciamento é stata rispettata, si ha:

$$T_{ck} > t_{RC} + \tau_{DC} + \tau_{CQ} = t_{comb}/n + \tau_{DC} + \tau_{CQ}$$

- Quindi trascurando i parametri dei FF, la frequenza di clock aumenta di circa n volte

Latenza e banda

- La latenza D, ovvero il tempo che intercorre fra l'inizio dell'elaborazione di un dato e il calcolo del risultato é data da

$$D = nT_{ck} = t_{comb} + n(\tau_{DC} + \tau_{CQ})$$

  - trascurando i tempi dei FF é simile a quella combinatoria
  - per n grande, questa ipotesi non é peró piú valida
  - esistono strategie di clock che usano latch che risolvono in parte tale problema
- Dal punto di vista del throughput, si osserva che il numero di dati (n_{ck}) elaborati per periodo di clock é sempre pari a 1
- Quindi, il miglioramento nel throughput rispetto al caso combinatorio é sempre dovuto all'aumento della frequenza di clock

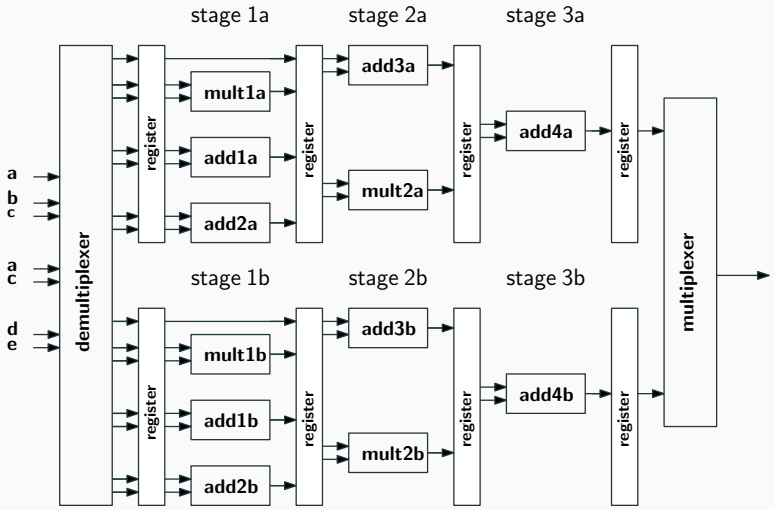
Problemi nel pipelining

- Abbiamo utilizzato l'ipotesi che le operazioni svolte da ciascuno stadio siano indipendenti da quelle degli stadi precedenti, altrimenti si potrebbero avere degli stalli
- Si nota che non si puó aumentare n indefinitamente sia per i parametri dei FF che per lo skew nei segnali di clock
- C'é stata una generazione di dispositivi con pipeline con pochissimi livelli di logica (2-4), attualmente si utilizzano piú livelli di logica e si preferisce puntare sul parallelismo

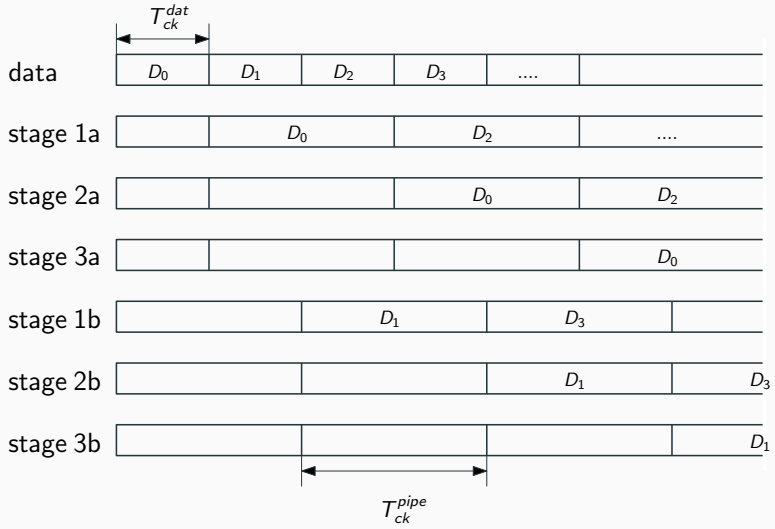
Parallelismo

- L'utilizzo del parallelismo viene qui solo accennato tramite un semplice esempio
- Si consideri la pipeline vista nell'esempio precedente e si supponga che i dati arrivino con un rateo pari a due volte la frequenza massima della pipeline calcolata in precedenza
- Ci sono due possibili alternative
  - si dividono i moltiplicatori con l'inserimento di un registro addizionale in modo da portare t_{RC} a $d_{mult}/2$: pipelining srutturale
  - si duplica la pipeline mantenendo la stessa frequenza di clock: parallelismo
- La soluzione parallela richiede chiaramente un opportuno schema di clock per gestire i due diversi domini (dati e pipeline) che funzionano a frequenze di clock diverse
- Si puó supporre che si usi un solo clock (quello dei dati) e che la pipeline utilizzi multicycling

Esempio di parallelismo



Esempio di parallelismo



Prestazioni dell'architettura parallela

- Si noti che questo parallelismo é a un livello piú alto di quelli implicitamente presenti in altre architetture
- In questa architettura
  - il clock e la latenza sono invariati rispetto a quelli di una singola pipeline
  - la banda, invece, si raddoppia come si puó vedere in due diversi modi
    - $b = f_{ck}^{dat} n_{ck}^{dat} = f_{ck}^{dat} \times 1$
    - $b = f_{ck}^{pipe} n_{ck}^{pipe} = f_{ck}^{pipe} \times 2$
- Nel caso il data-path sia quello di una CPU, questa soluzione viene definita come superscalare

Conclusioni

- L'analisi delle prestazioni é pienamente automatizzata e inserita nel flusso EDA per circuiti digitali
- Lo stesso vale sia per il livello logico sia per quello che riguarda i problemi relativi a condivisione delle risorse e scheduling delle operazioni che per quelli relativi al livello gate
- Ai livelli piú alti, alcune scelte riguardanti pipelining e parallelismo sono meno supportate dalla EDA