Electronics Systems

Federico Casu

Novembre 23, 2023

Digital Circuits Implementation Approaches

Un circuito integrato *special purpose*, noto come ASIC o **A**pplication **S**pecific **I**ntegrated **C**ircuit, è un circuito integrato creato appositamente per risolvere un'elaborazione ben precisa (special purpose, appunto): la specificità della progettazione, focalizzata sulla risoluzione di un unico problema, consente di raggiungere delle prestazioni in termini di velocità di elaborazione e consumo difficilmente ottenibili con l'uso di soluzioni più generiche (general purpose).

Non è tutto oro quello che luccica. Quanto costa produrre un chip ASIC? Prima di tutto dobbiamo introdurre una distinzione. Esistono due tipologie di approcci: *custom* e *semi-custom*.

1. L'approccio *custom* permette di avere il totale controllo sulla progettazione del chip: dalla dimensione dei transistor al posizionamento di quest'ultimi all'interno del chip. Tale controllo (o libertà di progettazione) consente di ottenere i risultati migliori in termini di prestazioni, area e consumo di energia. Ovviamente dobbiamo considerare il rovescio della medaglia: progettare e produrre un chip da zero ha i suoi costi.
2. L'approccio *semi-custom* si basa su design già esistenti, utilizzando una piattaforma predefinita con una struttura logica fissa. Durante la fase di progettazione, si definiscono le connessioni tra le strutture logiche messe a dispozione dalla piattaforma in uso. Il costo di sviluppo, così come il tempo necessario, per un chip semi-custom è **inferiore** rispetto a quello di un chip full custom. Questo perché il processo di ingegnerizzazione di un chip semi-custom è più rapido e meno complesso rispetto al caso full-custom.

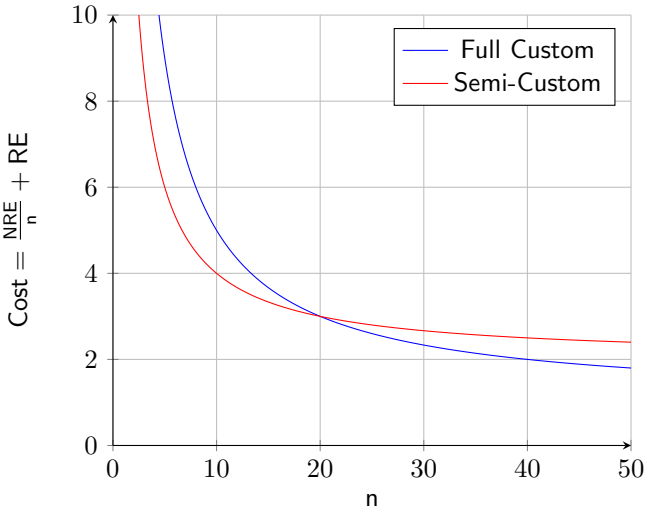


Figure 1: Grafico della curva di costo al variare del numero di chip prodotti.

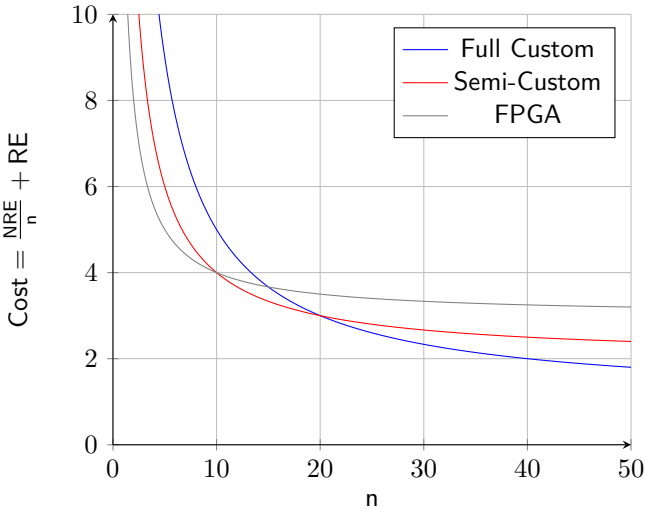
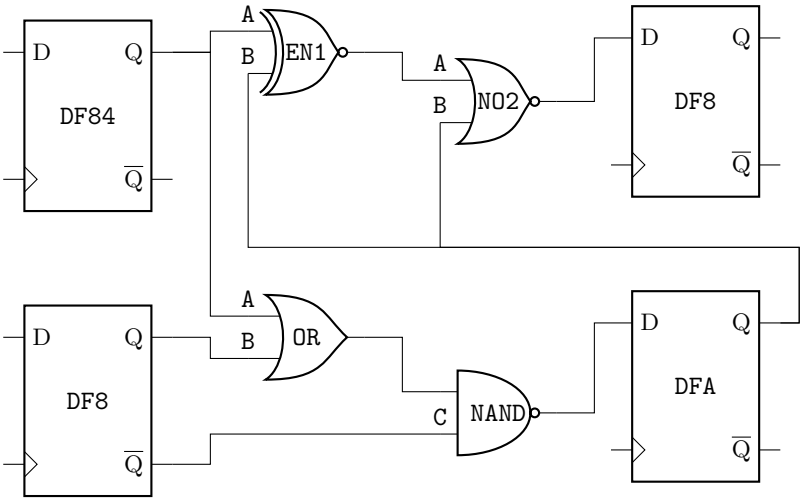


Figure 2: Grafico della curva di costo al variare del numero di chip prodotti: full-custom, semi-custom e FPGA.

Static and Dynamic Power in CMOS Inverters

Timing issues



Clock Skew

Fino ad ora abbiamo considerato che il fronte di salita del clock arrivi a tutti i registri del circuito allo stesso istante t. In realtà questa è solamente un ipotesi:

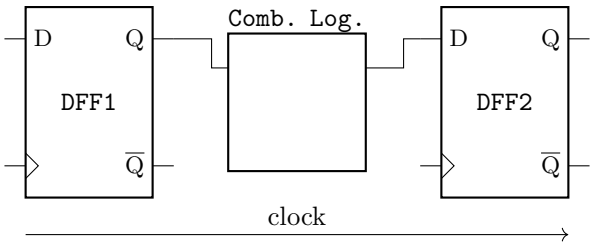
- Supponiamo di prendere come riferimento il segnale (onda quadra) in uscita dal generatore di clock.
- Il fronte di salita del clock visto da un registro i può essere in anticipo/ritardo rispetto al fronte di salita visto dal registro j. Tale scostamento dipende dalla distanza tra il generatore di clock e il registro. Esempio: se il registro i si trova più "vicino" al generatore rispetto al registro j allora il fronte di salita visto dal registro j è in *ritardo* rispetto ciò che ha visto il registro i.

Il fenomeno che abbiamo descritto sopra è detto *clock skew*. Il clock, come qualsiasi altro segnale, impiega del tempo a "percorrere" il mezzo fisico che collega il generatore al registro. Maggiore è la distanza da percorrere, maggiore è il tempo di propagazione.

Quali sono gli effetti del clock skew sui vincoli temporali sul periodo di clock? Esistono due casi:

1. Positive clock skew. Si verifica quando il clock si propaga nella stessa direzione del flusso dei dati.
2. Negative clock skew. Si verifica quando il clock si propaga nella direzione opposta alla direzione del flusso dei dati.

Analizziamo cosa succede nel caso di clock skew *positivo*. Dunque, il circuito in esame è il seguente:



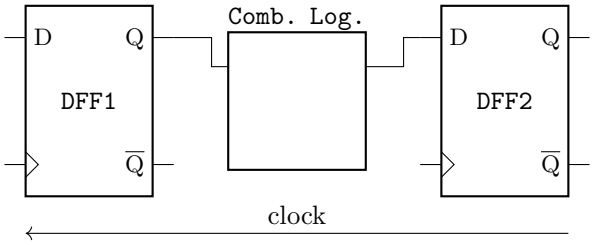
- Il fronte di salita del clock visto dal registro a valle della logica combinatoria è in ritardo rispetto al fronte di salita del clock visto dal registro a monte.
- Ciò significa che la durata del periodo di clock "percepita" dai segnali che si propagano tra i due registri è *maggiore* visto che al periodo di clock T devo sommare il ritardo sperimentato dal fronte di salita del clock visto dal registro a valle.

$$\mathsf{T} + |\delta| \geq \mathsf{t_{cq1}} + \mathsf{t_{plog}} + \mathsf{t_{su2}}$$
$$\mathsf{T} \geq \mathsf{t_{cq1}} + \mathsf{t_{plog}} + \mathsf{t_{su2}} - |\delta|$$

- Un clock skew positivo consente di aumentare la frequenza di clock (diminuire il periodo).
- Per quanto riguarda l'hold time violation, avere clock skew positivo è deleterio. Dato che il fronte di salita visto dal registro a monte è in anticipo rispetto al fronte di salita visto dal registro a valle, la somma dei contamination delay del registro e della logica combinatoria non devono essere minori del hold time del registro a valle a cui si somma il clock skew.

$$\mathsf{t_{hold2}} + |\delta| \leq \mathsf{t_{cd1}} + \mathsf{t_{cdlog}}$$
$$\mathsf{t_{hold2}} \leq \mathsf{t_{cd1}} + \mathsf{t_{cdlog}} - |\delta|$$

Nel caso di clock skew negativo, ciò che cambia nei vincoli temporali è il segno di δ.



$$\mathsf{T} \geq \mathsf{t_{cq1}} + \mathsf{t_{plog}} + \mathsf{t_{su2}} + |\delta|$$
$$\mathsf{t_{hold2}} \leq \mathsf{t_{cd1}} + \mathsf{t_{cdlog}} + |\delta|$$

Sicuramente, il vincolo più difficile da rispettare è l'hold time violation: è più semplice (meno costoso) aumentare il periodo di clock piuttosto che diminuire $\mathsf{t_{hold}}$.

Full Adder

Il nostro viaggio alla scoperta dei sommatori inizia dal più semplice sommatore che si possa implementare: sommatore a 1 bit.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Figure 3: 1-bit full adder truth table.

Possiamo calcolare le due uscite con le seguenti espressioni logiche:

$$\mathsf{S} = \mathsf{A} \oplus \mathsf{B} \oplus \mathsf{C_{in}}$$
$$\mathsf{C_{out}} = \mathsf{A} \cdot \mathsf{B} + \mathsf{A} \cdot \mathsf{C_{in}} + \mathsf{B} \cdot \mathsf{C_{in}}$$

Analizziamo più da vicino la tabella di verità del sommatore:

1. Prendiamo in esame le uscite S e $\mathsf{C_{out}}$. Prendiamo in considerazione solo le uscite associate agli stati diversi da $\{\mathsf{A}, \mathsf{B}, \mathsf{C_{in}}\} = \{0, 0, 0\}$ e $\{\mathsf{A}, \mathsf{B}, \mathsf{C_{in}}\} = \{1, 1, 1\}$. Possiamo notare una proprietà molto utile:

   $$\mathsf{S}(\mathsf{A}, \mathsf{B}, \mathsf{C_{in}}) = \bar{\mathsf{C}}_{\mathsf{out}}(\mathsf{A}, \mathsf{B}, \mathsf{C_{in}})$$

   ovvero l'uscita S può essere ottenuta come la negazione dell'uscita $\mathsf{C_{out}}$ quando almeno un ingresso è diverso da 0 e tutti non sono uguali a 1.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2. Tale proprietà può essere usata per ridurre la complessità (in termini di transistor) del circuito: di fatto possiamo utilizzare la rete combinatoria che produce l'uscita $\mathsf{C_{out}}$ anche per produrre l'uscita S.

   $$\mathsf{S}(\mathsf{A}, \mathsf{B}, \mathsf{C_{in}}) = \bar{\mathsf{C}}_{\mathsf{out}}(\mathsf{A}, \mathsf{B}, \mathsf{C_{in}}) + A \cdot B \cdot C_{in}$$

Cosa possiamo trovare nel datasheet di un 1-bit full adder?

1. La tabella di verità (Figura 3).
2. Le capacità associate a ciascun ingresso. Esempio:

Pin	Cap [pF]
A	0.038
B	0.037
C_{in}	0.032

3. L'occupazione del chip e la *power consumption*.
4. Tutti i ritardi che caratterizzo l'attività della rete combinatoria. In particolare, avremo delle informazioni a riguardo il ritardo di propagazione di ogni percorso, *cioè*:
   - Il ritardo del percorso $A \to S$.
   - Il ritardo del percorso $A \to C_{out}$.
   - Il ritardo del percorso $B \to S$.
   - Il ritardo del percorso $B \to C_{out}$.
   - Il ritardo del percorso $C_{in} \to S$.
   - Il ritardo del percorso $C_{in} \to C_{out}$.

Inoltre, potremo avere delle informazioni a riguardo del contamination delay della rete combinatoria.

Serial Adder

Come possiamo effettuare la somma di numeri rappresentati su $N > 1$ bits? Un possibile approccio è descritto in Figura 4.

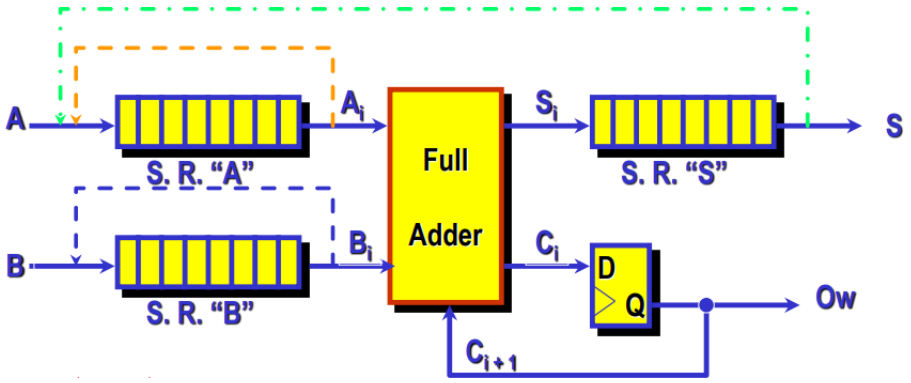


Figure 4: Arch. of N-bits serial full adder.

Quanto tempo si impiega a calcolare la somma di due interi su N bits? Facile: $N \cdot T_{clk}$. Per calcolare il periodo di clock minimo dobbiamo tenere conto dei

vincoli imposti dal *setup time*. Anche in questo caso dobbiamo analizzare tutti i percorsi reg-log-reg e calcolare il periodo di clock minimo in funzione del percorso più "lento".

Facciamo un esempio: supponiamo che il tempo di propagazione da un qualsiasi ingresso all'uscita S sia minore del tempo di propagazione da un qualsiasi ingresso all'uscita C_{out}. In altre parole,

- $T_{A \to S} < T_{A \to C_{out}}$
- $T_{B \to S} < T_{B \to C_{out}}$
- $T_{C_{in} \to S} < T_{C_{in} \to C_{out}}$

Come possiamo vedere dalla Figura 5, abbiamo i seguenti vincoli temporali:

1. $T_{clk} \geq T_{cq,A} + T_{C_{out}} + T_{su,C_{in}}$
2. $T_{clk} \geq T_{cq,B} + T_{C_{out}} + T_{su,C_{in}}$
3. $T_{clk} \geq T_{cq,C_{in}} + T_{C_{out}} + T_{su,C_{in}}$

Possiamo pensare che tutti i registri del circuito sono dello stesso tipo (quindi hanno gli stessi delay e setup time). Da ciò si ricava che il tempo necessario a compiere la somma di due interi su N-bits è $NT_{clk} \geq N(T_{cq} + T_C + T_{su})$.

Parallel Adder: Ripple Carry Adder

Con un architettura parallela, la somma avviene in un unico ciclo di clock!

Supponiamo che $T_{C_{out}} > T_S$. Il percorso "più" critico della rete combinatoria è rappresentanto in verde in Figura 7.

Sotto queste ipotesi il tempo necessario per eseguire la somma di due interi rappresentati su N bits è $N \cdot T_{C_{out}}$.

Parallel Adder: Pipeline solution

Come possiamo risolvere il problema del ritardo della generazione del carry? Introduciamo un stadio di pipeline (Figura 8).

L'architettura presentata in Figura 8 ha un problema:

1. Supponiamo di voler eseguire una somma ad ogni ciclo di clock.
2. Il riporto in input al sommatore 2 è in ritardo di un ciclo di clock rispetto agli input presenti in A_2, B_2, A_3 e B_3. Questo è dovuto al fatto che il registro di pipeline campiona il carry C_1 al clock t_i e l'output associato è valido nel ciclo di clock successivo. *Soluzione*: ritardiamo A_2, B_2, A_3 e B_3 di un ciclo di clock inserendo un registro che campiona gli ingressi in t_i e gli presenta ai sommatori 2 e 3 al ciclo di clock t_{i+1}.

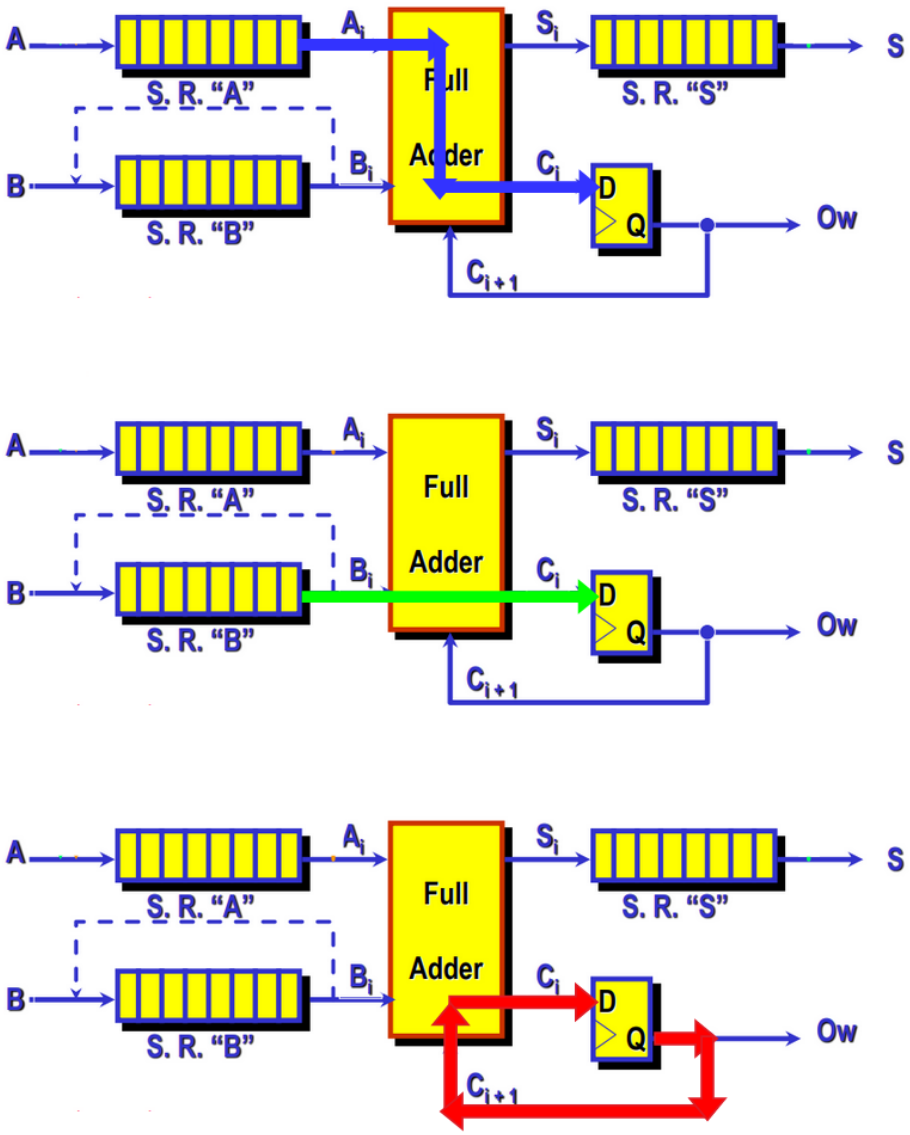


Figure 5: Serial adder: lista dei percorsi critici. Ipotesi: $T_S < T_{C_{out}}$.

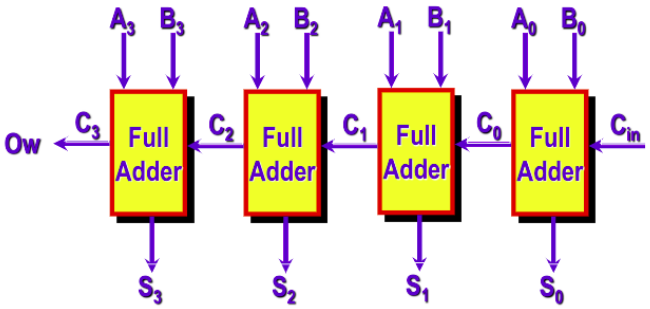


Figure 6: Arch. of 4-bits ripple carry full adder.

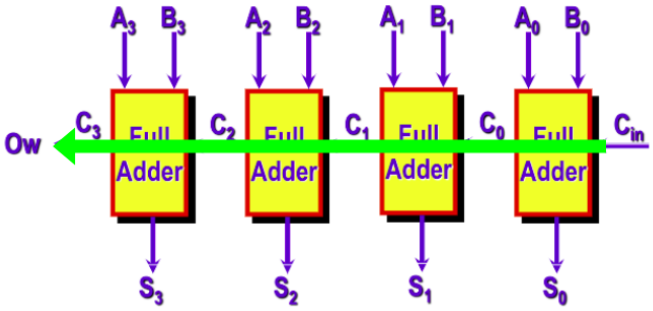


Figure 7: Ripple Carry adder: percorso critico. Ipotesi: $T_S < T_{C_{out}}$.

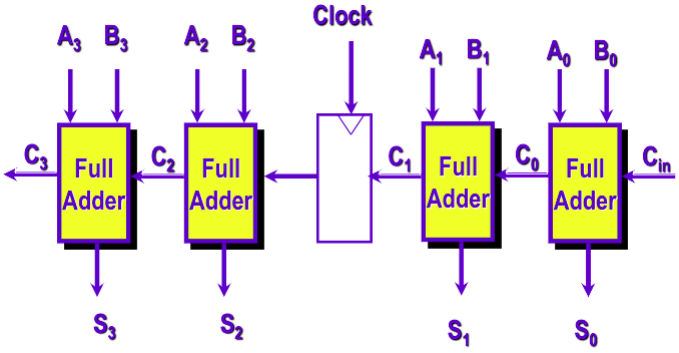


Figure 8: Idea di pipeline applicata ad un ripple carry adder.

3. Ora abbiamo un problema di sincronizzazione sull'output: S_2 e S_3 sono validi al ciclo di clock t_{i+1} ma S_0 e S_1 sono già validi al ciclo di clock t_i. *Soluzione*: inseriamo un registro che ritarda S_0 e S_1 di un ciclo di clock.

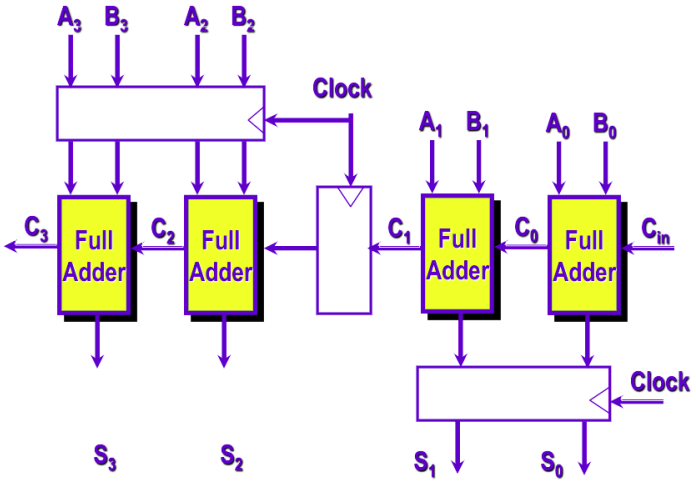


Figure 9: Architettura completa della pipeline applicata ad un ripple carry adder.