Università degli
Studi di L'Aquila

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

Corso di Elettronica dei sistemi digitali (0.5)
del Prof. Marco Faccio

Mediatore di 6 parole di 4 bit

studente: Mariano Spadaccini
matricola: 140769

Indice

1 Problema posto **1**

2 I sommatori **2**
- 2.1 Somma di bit in binario naturale 2
- 2.2 Circuiti logici per la somma 2
- 2.3 Somma di due parole ad n bit 5
  - 2.3.1 Ripple adder 6
  - 2.3.2 Sommatore parallelo 7
  - 2.3.3 Carry Look Ahead 8
  - 2.3.4 Carry Skip Adder 9
- 2.4 Sommatore temporizzato 12
- 2.5 Sommatore con accumulo 13

3 I registri **15**
- 3.1 I registri PIPO 15
- 3.2 I registri PISO 16
- 3.3 I registri SIPO 16

4 Il progetto **18**
- 4.1 Schema generale 18
- 4.2 Scelta del sommatore 21
  - 4.2.1 Ripple Adder 21
  - 4.2.2 Sommatore parallelo 22
  - 4.2.3 Carry Look Ahead 23
  - 4.2.4 Carry Skip Adder 24
  - 4.2.5 Conclusioni sui sommatori 24
- 4.3 La Rete combinatoria 25
  - 4.3.1 Rete combinatoria 1 26
  - 4.3.2 Rete combinatoria 2 27
- 4.4 La MSF 27
  - 4.4.1 L'ASM della MSF 29

4.4.2 Schema della MSF 30
4.4.3 Sintesi della funzione stato prossimo 32
4.5 Il contatore 37
4.5.1 ASM del contatore 38
4.5.2 Schema del contatore 38
4.5.3 Sintesi della funzione stato prossimo 38
4.6 Visualizzatore 41
4.7 Analisi dell'evoluzione temporale 43

5 La IspLSI 1016 60 LJ della Lattice 50
5.1 Schema interno e suo funzionamento 50
5.2 Programmazione 52
5.3 Download 52

6 Realizzazione del circuito 53
6.1 Test in laboratorio 53

A Listato 54
A.1 Programmazione della PLD I 54
A.2 Programmazione della PLD II 64

Bibliografia 68

Capitolo 1

Problema posto

Progettazione e realizzazione di un mediatore di 6 parole ognuna di 4 bit, con visualizzazione del risultato su display a 7 segmenti.
Si procederà con l'analisi dei vari circuiti sommatori, per poi passare alla descrizione dettagliata del progetto e alla sua sintesi. Infine, si accennerà brevemente all'hardware utilizzato per la realizzazione ed alle metodologie di programmazione.

Capitolo 2

I sommatori

2.1 Somma di bit in binario naturale

Si può riassumere la somma di due bit A e B nella seguente tavola di verità:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 2.1: Tavola di verità della somma di due bit

in cui il bit S rappresenta il bit *somma*, il bit C rappresenta il bit *riporto*[1]. Nell'operazione di somma di due parole, non è però sufficiente sommare semplicemente i due bit del *medesimo posto*, ma è necessario considerare l'eventuale riporto dei *posti precedenti*.
Quindi, in generale è necessario effettuare la somma di tre bit A_x, B_x e C_{x-1}, risultati riporati nella seguente tavola di verità mostrata in Tabella 2.2.

2.2 Circuiti logici per la somma

Dalla tavola di verità mostrata nella Tabella 2.1 è evidente che l'operazione di somma di due bit si riconduce all'operazione logica di *OR esclusivo*[2]:

$$S_x = \overline{A}_x B_x + A_x \overline{B}_x := A \oplus B$$

[1] in inglese *carry*
[2] definito XOR

C_{x-1}	A_x	B_x	S_x	C_x
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	1	1
1	1	0	1	1
1	1	1	0	1

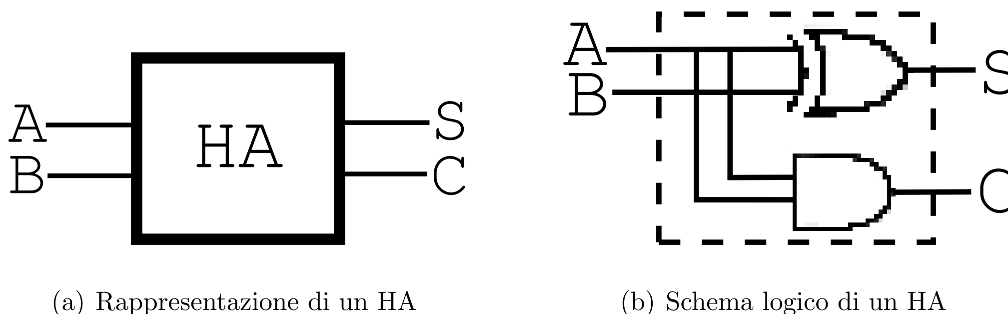
Tabella 2.2: Tavola di verità della somma di tre bit

Dalla medesima Tabella 2.1 è inoltre evidente che il *carry* si riconduce ad un'operazione di AND:

$$C = AB$$

Per sintetizzare le due operazioni, è usuale ricorrere graficamente alle seguenti figure:

Figura 2.1: Rappresentazioni di un *half adder*



(a) Rappresentazione di un HA

(b) Schema logico di un HA

Dopo aver estrapolato dalla Tabella 2.1 l'operazione di somma di due bit, possiamo in maniera analoga sintetizzare dalla Tabella 2.2 l'operazione di somma di tre bit, in cui:

$$\begin{aligned} S_x &= \overline{C}_{x-1}(A_x \oplus B_x) + C_{x-1}(A_x \odot B_x) \\ &\quad \text{in cui } A_x \odot B_x := A_x B_x + \overline{A}_x \overline{B}_x) \\ C_x &= A_x B_x + C_{x_1} A_x + C_{x-1} B_x \end{aligned}$$

Analogamente alla somma di due bit, è possibile sintetizzare le due operazioni ricorrendo alla Figura 2.2, di cui è rappresentata in Figure 2.3 la topologia interna.

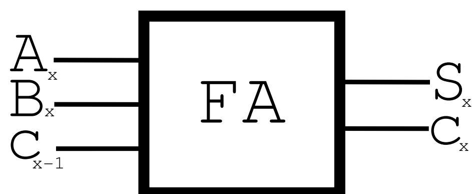


Figura 2.2: Rappresentazione di un *full adder*

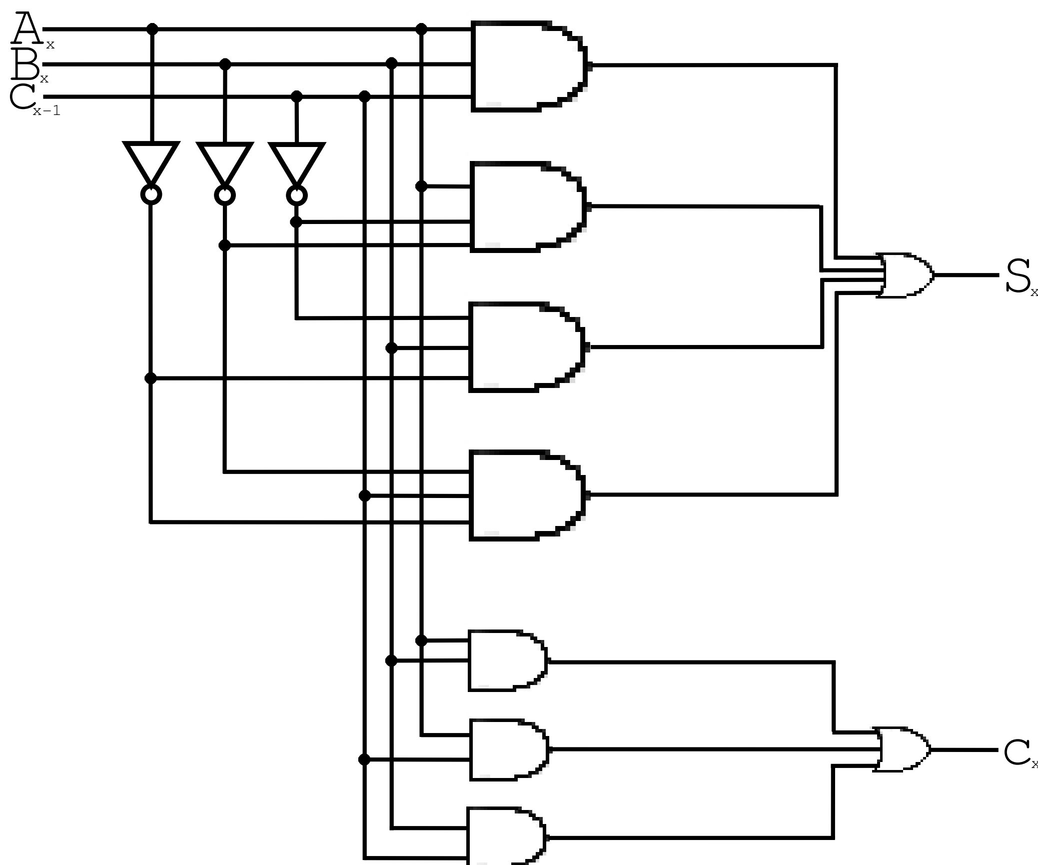


Figura 2.3: Topologia interna di un FA

È importante sottolineare che, per la proprietà associativa, per sintetizzare un FA si possono utilizzare due blocchi HA, come mostrato in Figura 2.4.

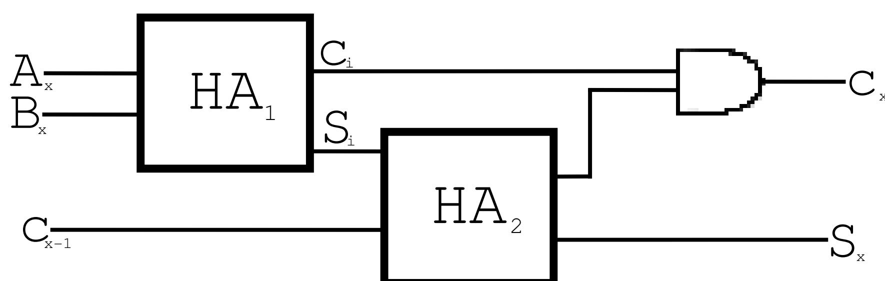


Figura 2.4: Realizzazione di un FA attraverso HA

2.3 Somma di due parole ad n bit

Nelle sezioni 2.1 e 2.2, abbiamo esaminato la somma di 2 o 3 bit; nella somma di due parole, non è possibile procedere in maniera equivalente, poiché, per giungere ad un risultato corretto, è necessario tener traccia degli eventuali carry. La Figura 2.5 illustra in maniera grafica tale concetto, ma

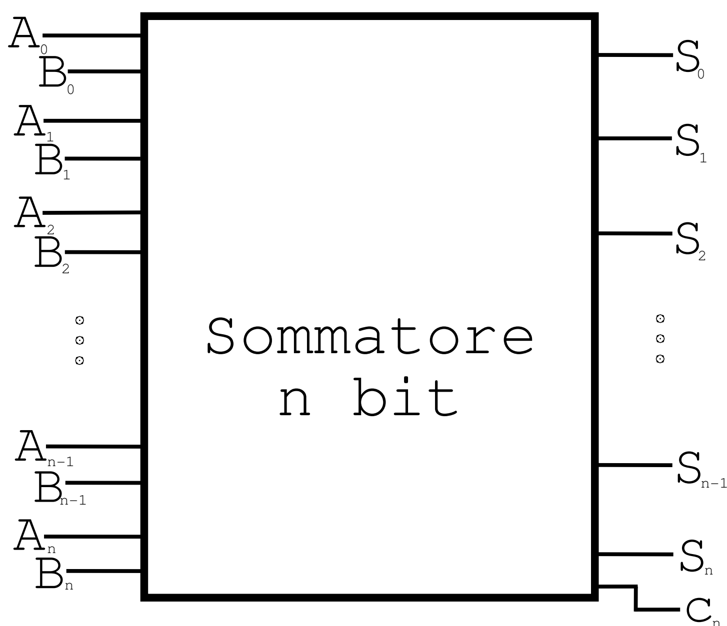


Figura 2.5: Schema generico di un sommatore ad n bit

nasconde nella parola *sommatore* una varietà di algoritmi e di circuiti combinatori che saranno di seguito esaminati, evidenziando per ognuno il tempo di stabilizzazione del risultato e la complessità di realizzazione.

2.3.1 Ripple adder

Il più semplice sommatore è il *ripple adder* (RA) poiché si realizza attraverso una cascata di FA[3], come mostrato in Figura 2.6.

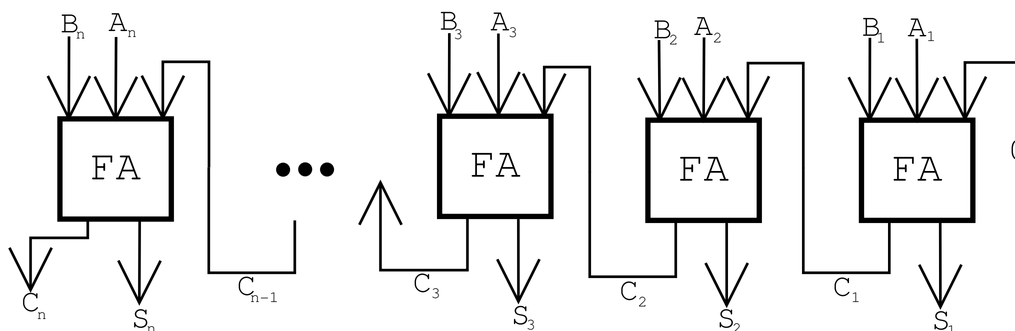


Figura 2.6: Schema di un ripple adder ad 8 bit

Il parametro importante da computare è il tempo richiesto per completare la somma; la determinazione di tale parametro è semplice poiché il tempo di latenza di un FA è $t_{DFA} = 3t_d$[4], quindi il tempo di latenza totale di un sommatore ripple operante su parole ad n bit è $t_{DRA} = 3nt_d$.
In effetti, possiamo estendere l'analisi appena compiuta al tempo di propagazione del carry; possiamo osservare che:

- tutti i FA partono contemporaneamente;
- l'uscita i-esima è stabile (quindi corretta) solo dopo che tutti i precedenti $(i - 1)$ FA presentano un'uscita stabile;
- ipotizzando che il primo FA ha in ingresso $C_{in} = 0$, poiché l'i-esimo FA ha un carry stabile dopo che i precedenti FA hanno un'uscita stabile, il tempo di propagazione del carry è $(i - 1)$, in quanto il primo FA ha già in ingresso un carry stabile.

Possiamo riassumere che il tempo richiesto da un RA per completare la propagazione del carry per parole lunghe n bit è $t_{DRA} = (n - 1)t_{DFA} = 3(n - 1)t_d$.

[3]il primo può semplicemente essere un HA poiché nella somma dei bit meno significativi non è necessario tener conto del carry; se il primo è un FA, si impone che il carry del bit meno significativo sia nullo, cioè $C_0 = 0$

[4]con t_d si è indicato il tempo di ritardo di porta

2.3.2 Sommatore parallelo

Il sommatore parallelo non soffre del ritardo di propagazione del carry poiché tutte le cifre dei due addendi sono sommate contemporaneamente; tale schema è rappresentato in Figura 2.7.

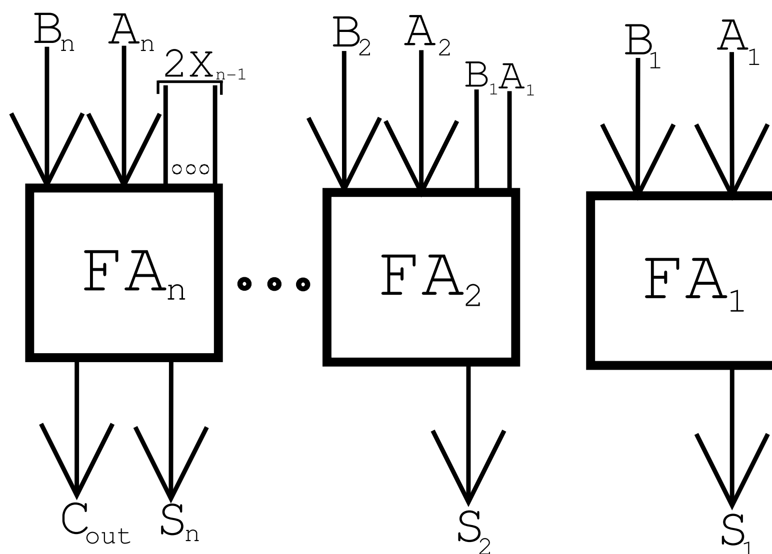


Figura 2.7: Schema generale del sommatore parallelo

È importante notare che il tempo di ritardo è indipendente dal numero di bit, ed è pari a quello di un FA, cioè:

$$t_D = 3t_d$$

In questa configurazione, il singolo FA non deve attendere la stabilizzazione dei precedenti in quanto ogni FA calcola autonomamente il riporto. Affinché ogni FA possa calcolare autonomamente il riporto dei precedenti stadi, un singolo FA dovrà ottenere, oltre alle proprie linee di ingresso[5], anche le linee di ingresso di tutti i FA precedenti[6]; quindi, il numero degli ingressi di ciascun FA cresce al crescere di n, in particolare il FA i-esimo presenta $2i$ ingressi: la rete è complessa al crescere di n, utilizzando porte ad elevato numero di ingressi.

[5] cioè le linee di ingresso dei bit A_x e B_x che hanno lo stesso peso del bit S_x che genererà il FA

[6] cioè che genereranno bit S_x meno significativi

2.3.3 Carry Look Ahead

L'algoritmo CLA prevede una riduzione di complessità rispetto al sommatore parallelo ed un aumento di velocità rispetto al ripple adder; tecnicamente, il carry look ahead prevede una rete opportuna per calcolare direttamente tutti i riporti[7].
Più in dettaglio, esaminiamo la seguente tavola di verità, concentrando l'attenzione sul carry.

C_{i-1}	A_i	B_i	C_i
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

È evidente dalla tabella che il carry si ottiene tramite la seguente formula:

$$C_i = A_iB_i + C_{i-1}(A_i \oplus B_i).$$

Ponendo:

$$G_i = A_iB_i$$

e

$$P_i = A_i \oplus B_i,$$

attraverso una semplice sostituzione, si ottiene:

$$\begin{aligned}
C_1 &= G_1 + P_1C_0 \\
C_2 &= G_2 + P_2C_1 = \\
&= G_2 + P_2(G_1 + P_1C_0) \\
C_3 &= G_3 + P_3C_2 = \\
&= G_3 + P_3(G_2 + P_2G_1 + P_2P_1C_0) = \\
&= G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0
\end{aligned}$$

In generale si ha:

$$\begin{aligned}
C_n &= G_n + P_nG_{n-1} + P_nP_{n-1}G_{n-2} + P_nP_{n-1}P_{n-2}G_{n-3} + \\
&+ \ldots + P_nP_{n-1}\ldots P_2G_1 + P_nP_{n-1}\ldots P_1C_0
\end{aligned}$$

Inoltre, possiamo notare che il CLA è realizzato attraverso una riorganizzazione di blocchi HA; uno schema generale è rappresentato in Figura 2.8.
Dallo schema evidenziato, possiamo determinare il tempo di ritardo:

- il 1° stadio di HA ha un tempo di ritardo pari a $3t_d$;
- il 2° stadio costituito dal carry generator ha un tempo di ritardo pari a $2t_d$;

[7] definita *carry generator*

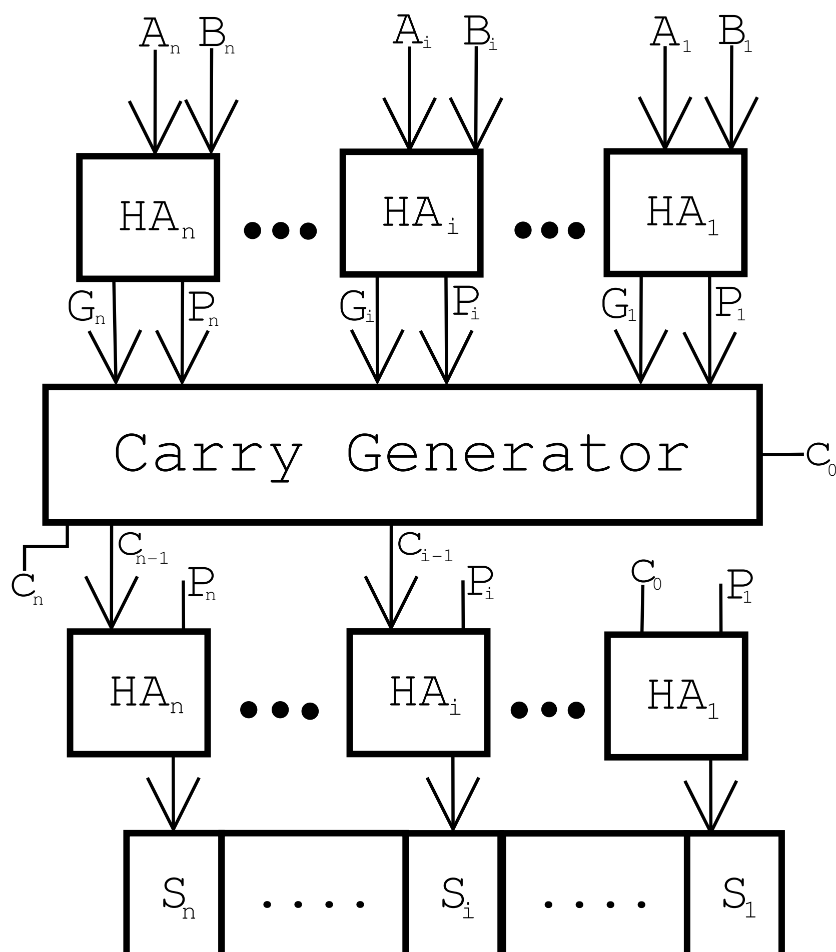


Figura 2.8: Schema generale del CLA che evidenzia il carry generator

- il 3° stadio, analogamente al 1°, ha un tempo di ritardo pari a $3t_d$.

In totale si ha un tempo di ritardo pari a $8t_d$.
Il problema della realizzazione di un CLA è causato dalla complessità della rete carry generator al crescere del numero di bit.

2.3.4 Carry Skip Adder

La configurazione CSA prevede la propagazione del carry come il ripple adder, ma tramite un'opportuna rete di bypass si cerca di stimare, anticipando i tempi di risposta dei FA, se ci sarà riporto negli stadi successivi.
In altre parole, la rete di bypass è una rete anticipattrice di carry in quanto consente di anticipare l'eventuale aggiornamento del carry, senza attendere

la propagazione dentro il blocco ripple.
Il tempo di latenza complessivo è dato dal caso peggiore; risulta essere quando il riporto è generato all'uscita del primo stadio, si propaga nei blocchi di by-pass successivi e si ferma all'uscita dell'ultimo FA: Tecnicamente, l'architettura è formata da un ripple ad n bit, frazionato in m blocchi da k bit ognuno; la caratteristica del CSA è quella di inserire una rete anticipatrice di carry, definita di bypass.

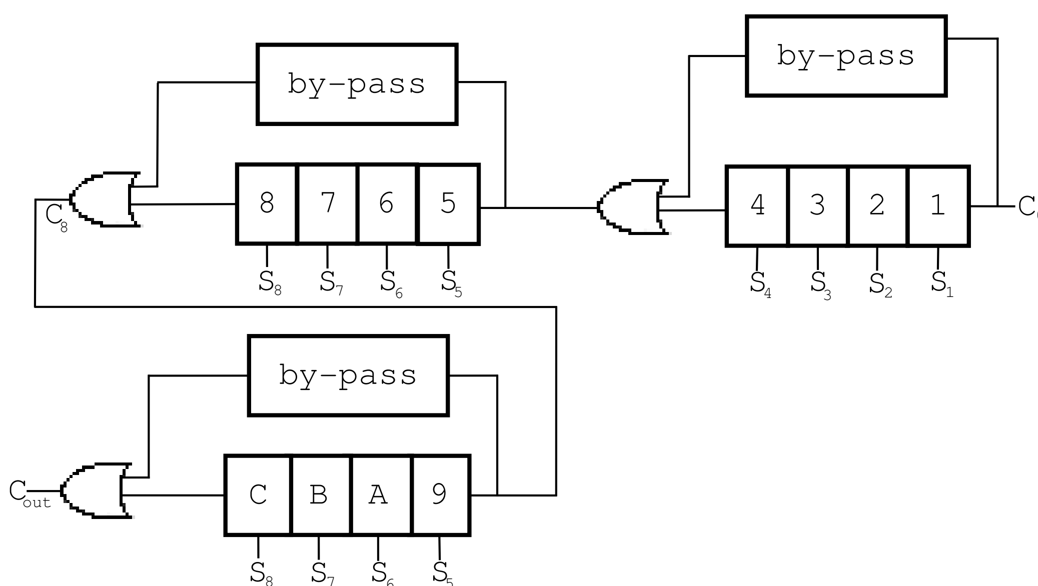


Figura 2.9: Schema di un carry skip adder a 12 bit

Nella Figura 2.9, è rappresentato un esempio di CSA a 12 bit: in questa rappresentazione, un RA a n =12 bit è stato frazionato in $m = 3$ blocchi da $k = 4$ bit ognuno, connessi ciascuno in modalità ripple.
Come già scritto precedentemnte, ogni blocco presenta una rete di bypass che, se ne esistono le condizioni, consentirà di anticipare la propagazione del carry al blocco successivo, senza attendere la propagazione nel blocco stesso (v. Figura 2.10).
Più in dettaglio, esisterà propagazione del carry nel bypass ($C_{outbp} = 1$) se $C_{in} = 1$ e, contemporaneamente, per ogni coppia di bit da sommare, almeno uno di essi è 1; formalmente:

$$C_{out} = C_{in}(A_1 + B_1)(A_2 + B_2)(A_3 + B_3)(A_4 + B_4).$$

Schematicamente, il modulo base è rappresentato in Figura 2.11.
Dallo schema del modulo base si evince che il tempo di latenza del circuito di bypass è $t_{BP} = 3t_d$.

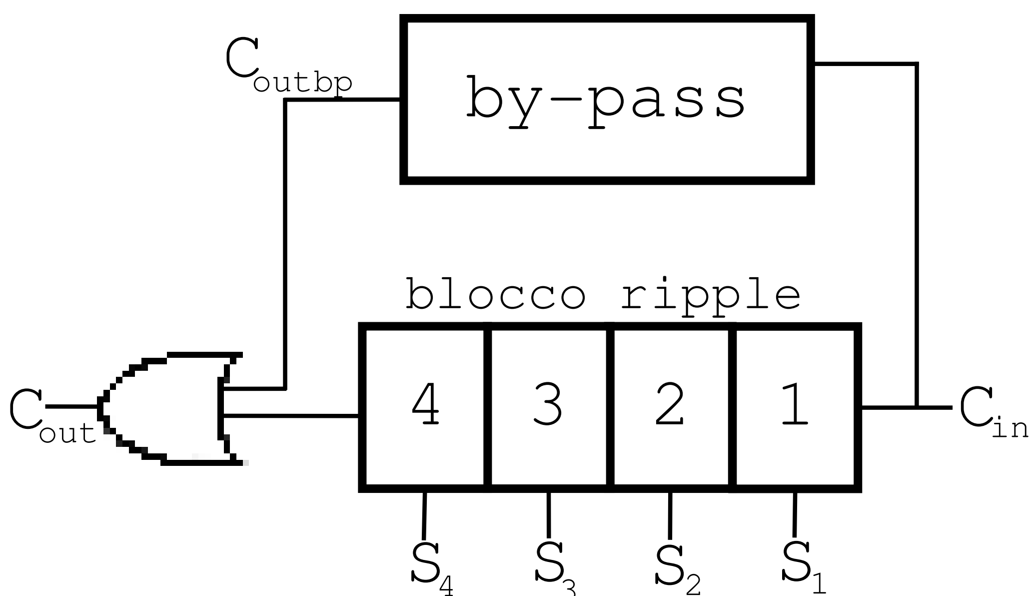


Figura 2.10: Schema di un modulo del carry skip adder

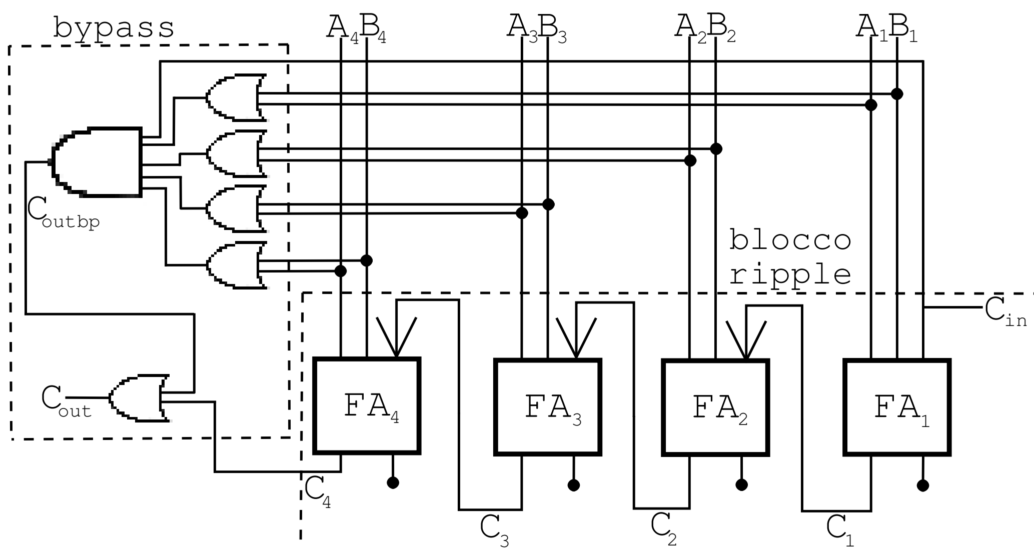


Figura 2.11: Schema di un modulo base del carry skip adder

Nello schema precedente di Figura 2.9, il tempo di latenza complessivo del CSA è dato dal caso peggiore di propagazione. Esso si realizza quando C_0 e C_{out} sono nulli e tutti i C_{in} dei blocchi interni sono pari ad 1, cioè ogni blocco correggerà la stima iniziale di C_{in}; in tale ambito, il caso peggiore per la propagazione del carry si verifica quando il riporto è generato all'uscita del

primo stadio, si propaga nei blocchi di bypass successivi, e si ferma all'uscita del FA_{12}:

$$\Delta T_{dcarry} = 3t_{DFA} + 1t_{DBP} + 3t_{DFA} = 7t_{DFA} = 21t_d$$

In generale, considerando un CSA ad n bit, frazionato in m blocchi da k bit cadauno, si avranno i seguenti tempi di latenza in riferimento:

- alla propagazione del carry:

$$\Delta t_{carry} = 2(k-1)t_{DFA} + (m-2)t_{DBP}$$

- al risultato corretto:

$$\Delta t_{ris} = 2kt_{DFA} + (m-2)t_{DBP}$$

2.4 Sommatore temporizzato

Dopo aver esaminato la topologia interna dei sommatori, possiamo generalizzarlo nelle prossime sezioni come in Figura 2.12.

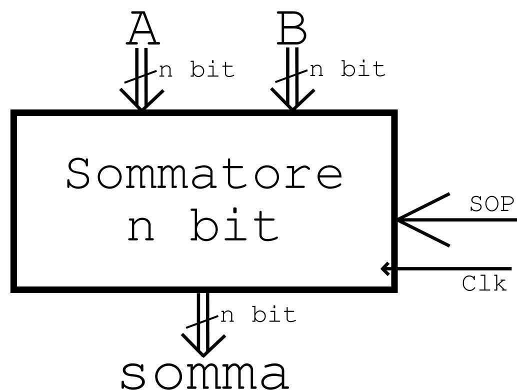


Figura 2.12: Schema generale di un sommatore

Prescindendo dal sommatore scelto, inserendolo in un sistema reale, il sommatore diventa un blocco temporizzato controllato da una MSF (si veda la Figura 2.13).
Ad esempio, una sequenza verosimile potrebbe essere:

- nel primo colpo di clock sono caricati nei registri gli operandi A e B (dipende dal bus esterno);

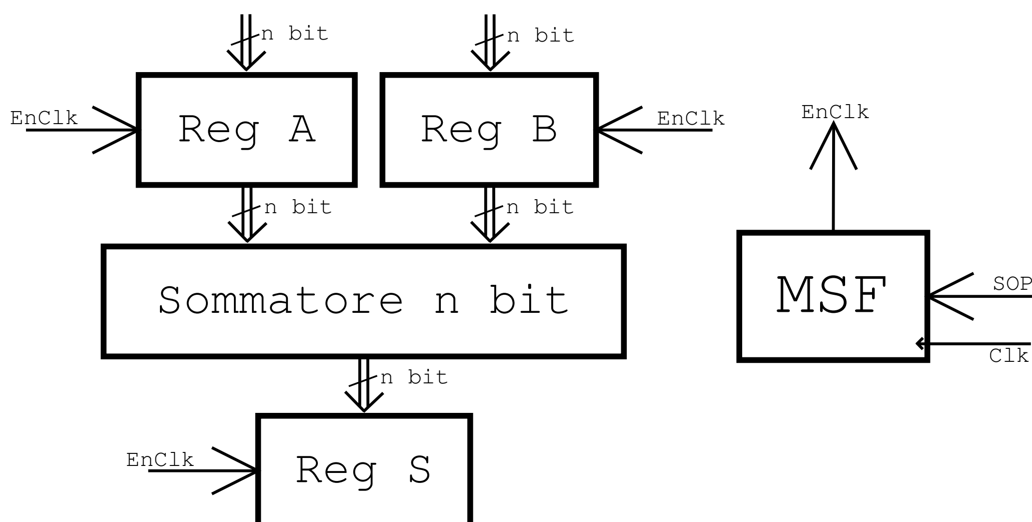


Figura 2.13: Schema generale di un sommatore temporizzato

- è effettuata la somma tramite la rete combinatoria, alla quale assegnamo il ritardo, quantificato tramite le considerazioni effettuate nelle sezioni precedenti;
- al successivo colpo di clock si registra il risultato nel relativo registro.

Dall'esempio appena esaminato, si nota che *il tempo di latenza del sommatore è fondamentale per stabilire la frequenza di clock dell'operazione.*

2.5 Sommatore con accumulo

Un semplice sommatore di più addendi può essere realizzato attraverso un accumulatore, cioè un sommatore con accumulo (si veda la Figura 2.14).
Per il generico accumulatore mostrato, è utile presentare il generico algoritmo di calcolo:

1. Azzerare l'accumulatore (reset);
2. Caricare nel registro *Reg A* l'addendo;
3. Sommare il contenuto del registro *Reg A* e quello dell'accumulatore;
4. Tornare al passo 2..

Ovviamente questo algoritmo non ha termine poiché non prevede l'uscita dal ciclo; si può ovviare modificando l'ultimo passo e compiere il ciclo solo se si

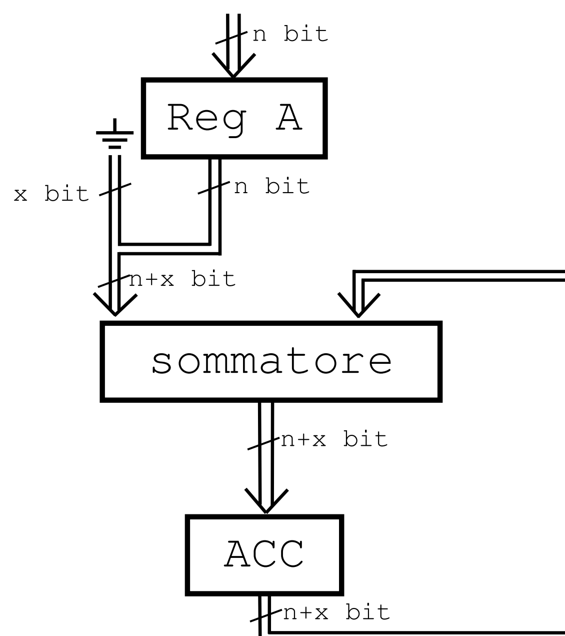


Figura 2.14: Sommatore con accumulo

verifica *una particolare condizione*[8].
Questa condizione può essere, ad esempio, valutata da un comparatore, il quale comunicherà alla MSF l'evento e la MSF procederà conseguentemente.

[8]ad esempio, una possibile condizione potrebbe essere: "Sono stati inseriti tutti gli addendi?"

Capitolo 3

I registri

3.1 I registri PIPO

Un registro PIPO (Parallel In – Parallel Out) a n bit presenta n linee di ingresso e n linee di uscita per permettere sia la lettura sia la scrittura in parallelo.
Essi hanno una struttura interna del tipo rappresentato in Figura 3.1.

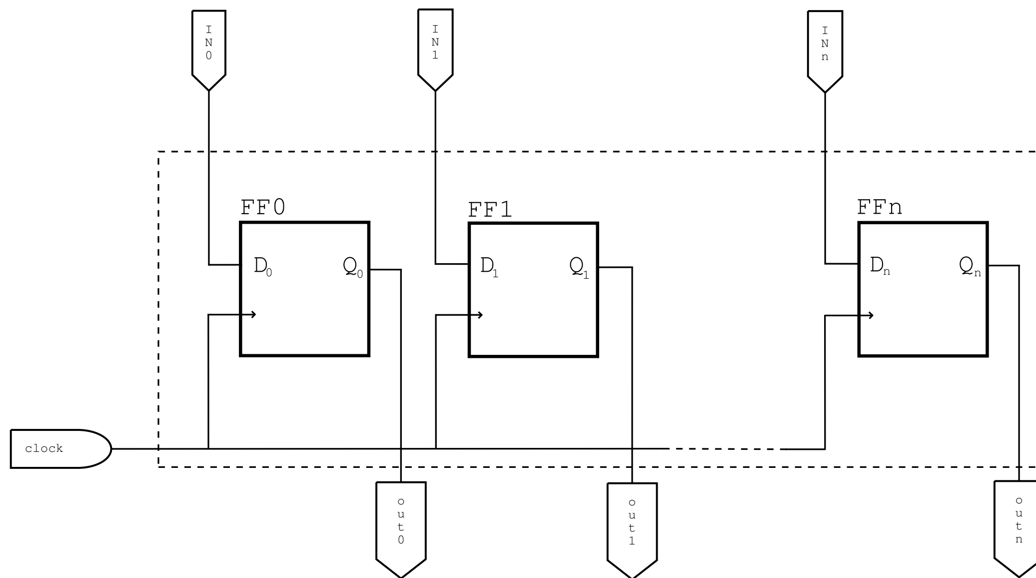


Figura 3.1: Registro PIPO a n bit

Lo schema è molto semplice poiché richiede la presenza di poco hardware: n flip-flop per la memorizzazione dei bit; il segnale di clock abilita la scrittura su ogni singolo flip-flop, sensibile al fronte di salita o a quello di discesa.

Ci possono essere altri due ingressi asincroni: il *Preset* e *Reset*, che comunque non sono necessari per la realizzazione del registro.

3.2 I registri PISO

Un registro PISO (Parallel In – Serial Out) a n bit presenta n linee di ingresso e una linea di uscita per permettere la scrittura in parallelo e l'uscita in serie dei dati.
Essi hanno una struttura interna del tipo rappresentato in Figura 3.2.

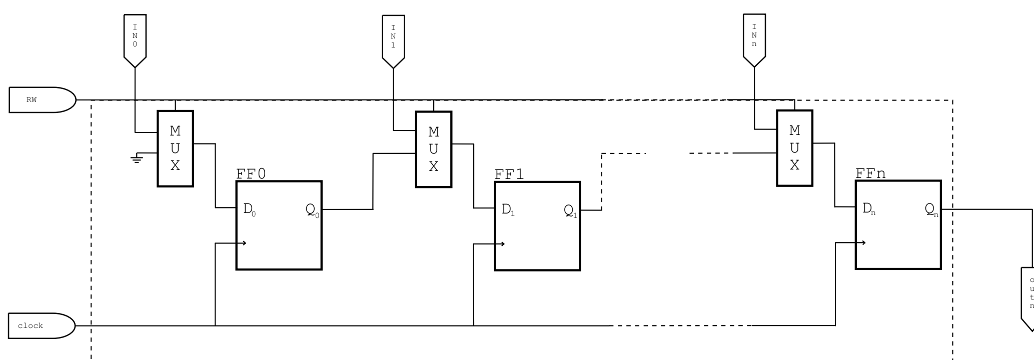


Figura 3.2: Registro PISO a n bit

Oltre alle linee di ingresso-uscita e al clock, è prevista una linea di selezione (RW) che permette di scegliere di volta in volta se effettuare una lettura o una scrittura al colpo di clock successivo. Si noti che nello schema in Figura 3.2, la lettura dal registro è una lettura distruttiva: al termine dell'operazione di lettura di tutti gli n bit del registro, in quest'ultimo sono caricati tutti zero. Per evitare una situazione del genere si può riportare l'uscita in ingresso: in questo modo al termine della lettura il registro si ritrova esattamente nello stato di partenza.

3.3 I registri SIPO

Un registro SIPO (Serial In – Parallel Out) a n bit presenta una linea di ingresso e n linee di uscita che permettono la scrittura in serie e l'uscita in parallelo dei dati. Essi hanno una struttura interna del tipo rappresentato in Figura 3.3.
Anche in questo caso, come per il registro PIPO, lo schema è piuttosto semplice. Da notare che la fase di scrittura richiede n colpi di clock e l'inserimento dei bit dal meno significativo al più significativo. Questo perché i bit di

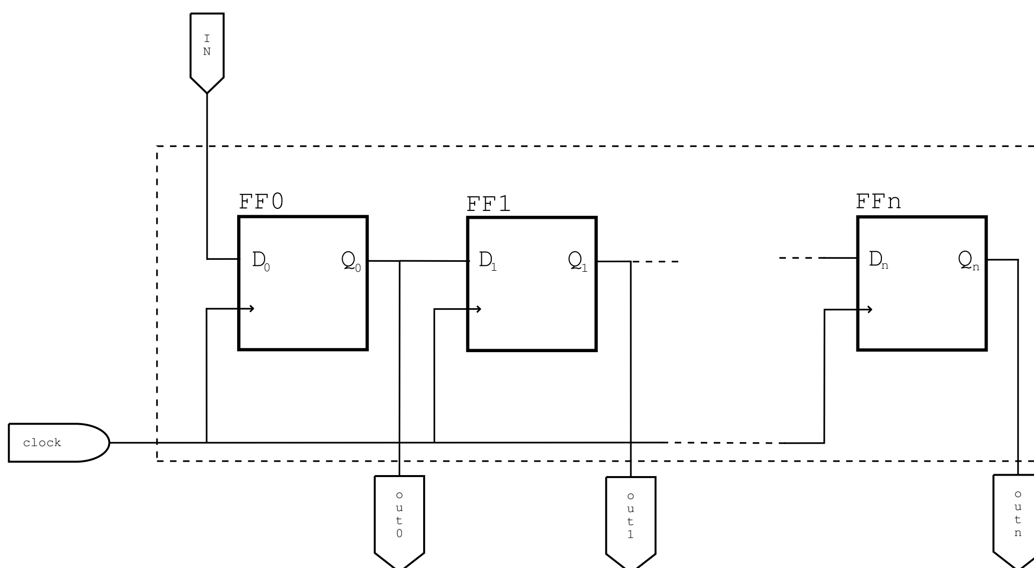


Figura 3.3: Registro SIPO a n bit

ingresso shiftano di un posto verso destra ad ogni colpo di clock in conseguenza del collegamento a cascata dei flip-flop. Altre due linee di ingresso, non previste nello schema, si possono trovare in un registro SIPO: il *Preset* e *Reset*.

Capitolo 4

Il progetto

Nella realizzazione di un mediatore di 6 parole 4 bit, ho effettuato diverse scelte sia per quanto riguarda l'algoritmo utilizzato, sia per quanto riguarda le caratteristiche dell'hardware utilizzato. In particolare:

- nella sezione 4.1 analizzerò le scelte algoritmiche e, quindi, accennerò sia l'hardware sia le linee di controllo utilizzati;
- nelle sezioni seguenti analizzerò nello specifico l'hardware approfondendone la topologia.

4.1 Schema generale

L'algoritmo per compiere una media di n parole si può suddividere in due passi principali:

1. la somma delle n parole;
2. la divisione per n.

Una semplificazione nell'analisi appena proposta è definita dalle specifiche del progetto in quanto è stabilito che il numero delle parole sia fisso, in particolare le specifiche stabiliscono $n = 6$; inoltre, è definito che le parole siano di dimensione fissa di 4 bit.
Per realizzare la somma di 6 parole, si utilizza il sommatore con accumulo illustrato nella Figura 2.14, ripresentandolo nella Figura 4.1 nel caso in esame.

In particolare, si può notare la dimensione dei registri:

- il *Reg. A* di 4 bit, come definito dalle specifiche;

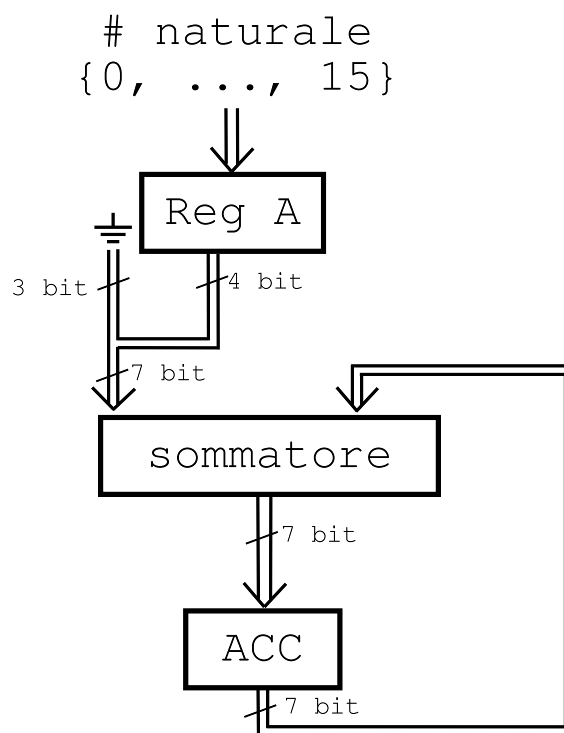


Figura 4.1: Sommatore con accumulo di 7 bit

- il *registro accumulatore* di 7 bit[1].

Il passo successivo per la realizzazione della media, è l'operazione di divisione per 6; si è scelto di compiere la divisione attreverso sottrazioni successive, in particolare il quoto[2] è stato ottenuto contando quante sottrazioni successive si possono realizzare, cioè si è scelto di sommare in C2 di -6 e contare quante volte è possibile effettuare la sottrazione (v. Figura 4.2).
Le differenze più evidenti tra la Figura 4.2 rispetto alla Figura 4.1 sono analizzare di seguito:

1. La dimensione del registro ACC: questa modifica è necessaria poiché sottrarre 6 equivale, ovviamente, a sommare -6; l'introduzione dei numeri relativi comporta l'aggiunta di un bit per il segno, quindi è necessario passare da 7 bit a 8 bit.

[1] nel registro *Reg A* possiamo memorizzare un numero naturale che al massimo sarà 15; nel caso peggiore nel registro ACC sarà memorizzata la somma di 6 numeri ognuno pari a 15, cioè sarà memorizzato $15 \times 6 = 90$; il numero 90 lo possiamo memorizzare in 7 bit, quindi il registro ACC dovrà essere dimensionato di 7 bit

[2] che equivale alla media arrotondata per difetto

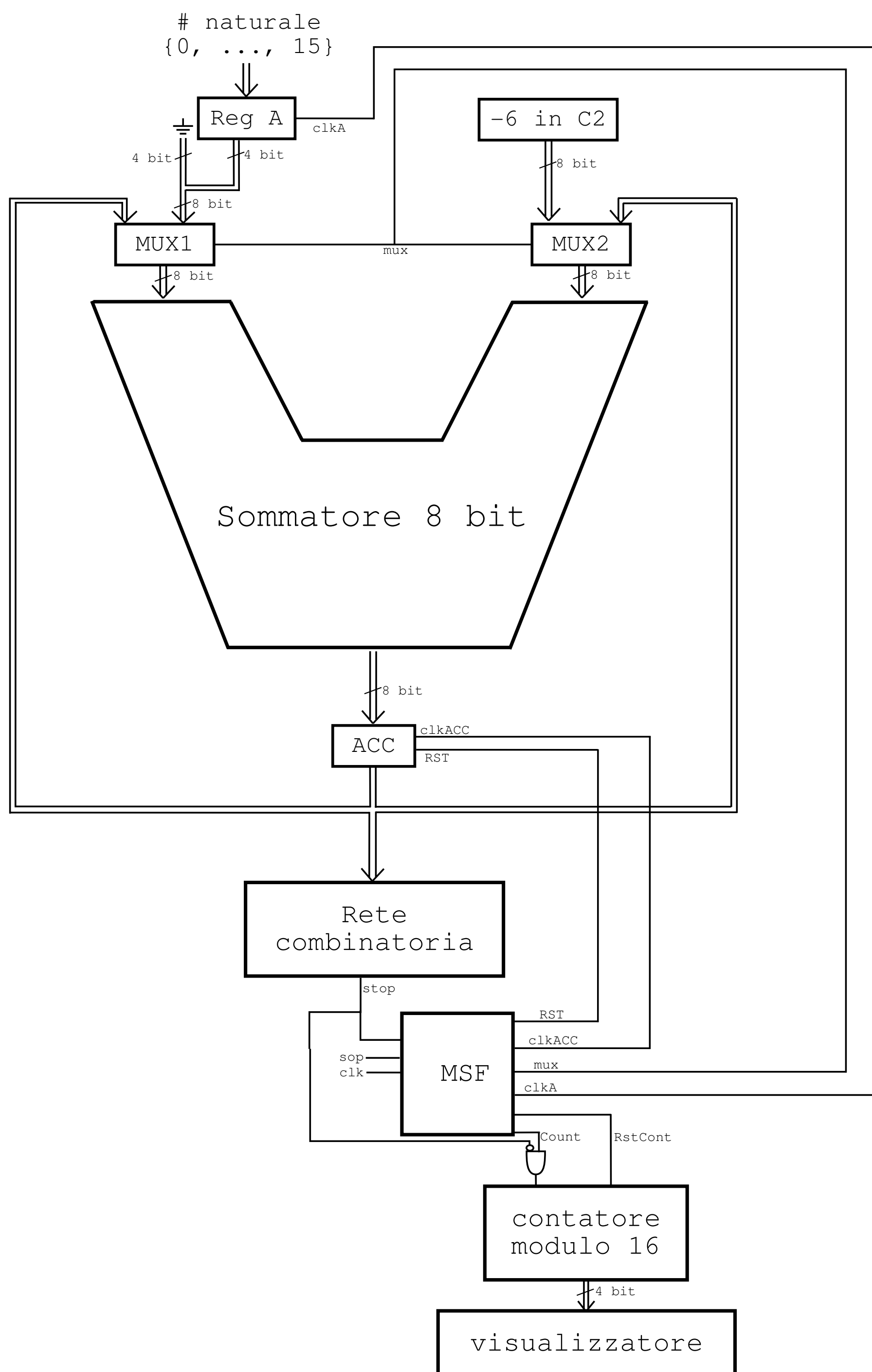


Figura 4.2: Schema generale del mediatore

2. L'introduzione dei *mux*: per utilizzare lo stesso sommatore sia nella somma delle prime 6 parole, sia nelle successive sottrazioni, è necessario selezionare i registri in ingresso, selezione effettuata dal *mux*.

3. L'introduzione della *Rete combinatoria*: questo è un comparatore, la cui funzione è terminare quando le sottrazioni del numero 6 sono terminate: ciò avviene quando nel registro ACC è memorizzato un numero compreso tra 0 e 5.

4. L'introduzione del *contatore*: il contatore è necessaria per contare quante sottrazioni successive è possibile effettuare, quindi rappresenta la media delle parole immesse.

5. L'introduzione[3] della *MSF*: la *macchina a stati finiti* è un'unità di controllo e, in quanto tale, ha il compito di gestire lo stato attuale e determinare l'evoluzione del sistema attraverso la selezione delle appropriate linee di controllo. In particolare, possiamo notare le seguenti linee di controllo:

   - **RST**: abilitata per azzerare il registro ACC;
   - **ClkACC**: abilitata per permettere la scrittura del registro ACC;
   - **mux**: seleziona l'ingresso per il sommatore;
   - **ClkA**: abilita la scrittura nel registro *Reg A*;
   - **RstCont**: resetta il contatore;
   - **Count**: comunica al contatore di avanzare nel conteggio[4].

6. L'introduzione del *visualizzatore*: il visualizzatore è, ovviamente, necessario per visualizzare il risultato, ed è mensionato nelle specifiche.

4.2 Scelta del sommatore

4.2.1 Ripple Adder

In riferimento alla discussione proposta nel paragrafo 2.3.1, proponendo la configurazione di un RA ad 8 bit nella Figura 4.3, è opportuno evidenziare il tempo di ritardo per la propagazione del carry per un RA ad 8 bit:

$$t_{DRA} = 7t_{DFA} = 21t_d.$$

[3] in effetti, essa dovrebbe essere introdotta anche nella Figura 4.1, ma è stata introdotta solo nella Figura 4.2 poiché ho preferito analizzarla solo nel caso implementato

[4] è in *and* con $\overline{stop}$ poiché la Rete combinatoria è l'unica ad accorgersi in tempo reale quando smettere il conteggio

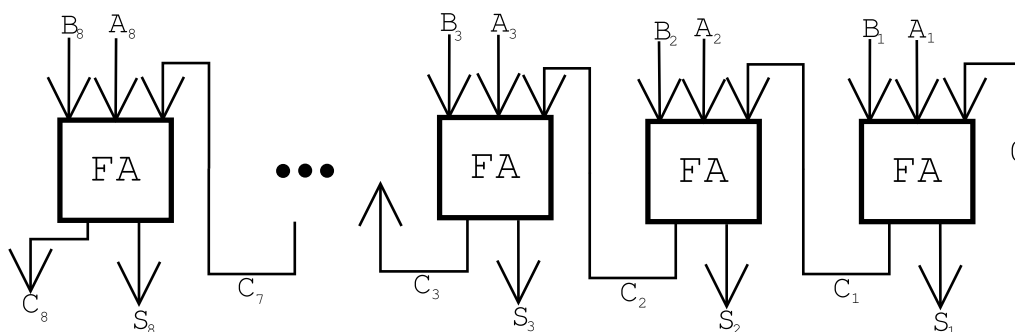


Figura 4.3: Schema di un RA ad 8 bit

4.2.2 Sommatore parallelo

In riferimento alla discussione proposta nel paragrafo 2.3.2, proponendo la configurazione di un sommatore parallelo ad 8 bit nella Figura 4.4, è oppor-

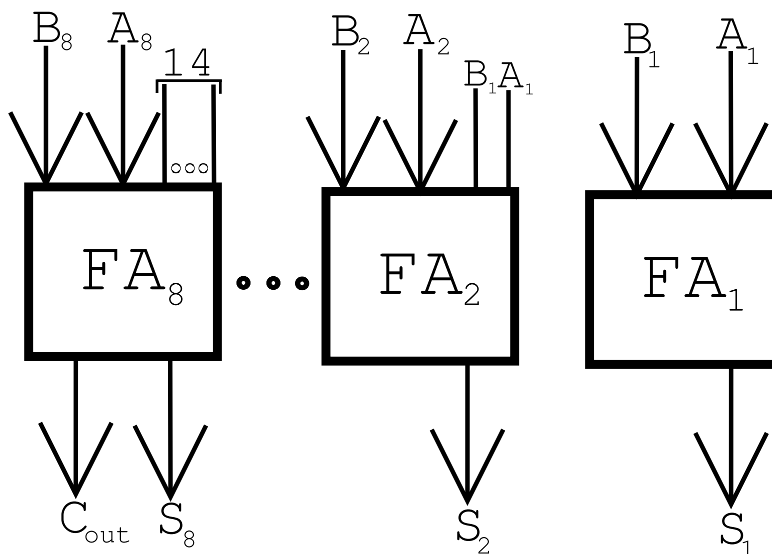


Figura 4.4: Schema di un sommatore parallelo ad 8 bit

tuno evidenziare che:

- il tempo di ritardo è, indipendentemente dal numero di bit, $t_D = 3t_d$;
- il FA i-esimo presenta 2i ingressi, in particolare nella Figura 4.4 è evidenziato che il FA_8 ha 16 ingressi.

4.2.3 Carry Look Ahead

In riferimento alla discussione proposta nel paragrafo 2.3.3, è opportuno ridurre la complessità circuitale del carry generator, sacrificando un po' di velocità a favore di una riduzione di complessità circuitale.
Purtroppo resta il problema della complessità della rete carry generator al crescere del numero di bit. Per tale motivo, è opportuno trasformare la rete CLA mostrata in Figura 2.8 nella rete mostrata in Figura 4.5. Per

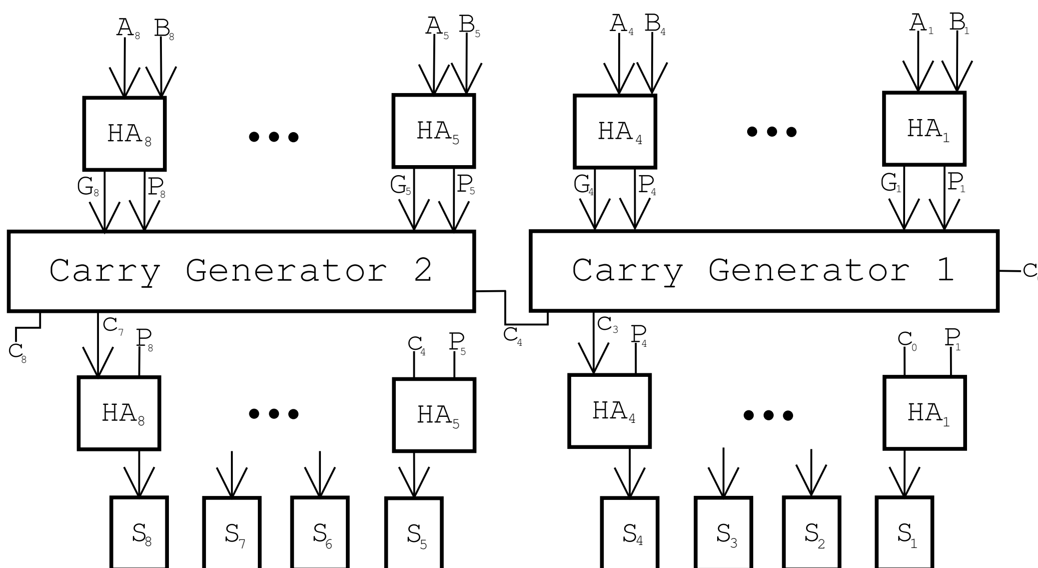


Figura 4.5: Realizzazione del CLA composto da due blocchi in cascata

quest'ultima configurazione, possiamo determinare il tempo di ritardo:

- il 1° stadio di HA (HA_1, ..., HA_4) ha un tempo di ritardo pari a $3t_d$;
- il 2° stadio, costituito dal Carry Generator 1, ha un tempo di ritardo pari a $2t_d$;
- il 3° stadio, costituito dal Carry Generator 2, ha un tempo di ritardo pari a $2t_d$
- il 4° stadio, costituito dagli HA_5, ..., HA_8, analogamente al 1°, ha un tempo di ritardo pari a $3t_d$.

In totale si ha un tempo di ritardo pari a $10t_d$.

4.2.4 Carry Skip Adder

In riferimento alla discussione proposta nel paragrafo 2.3.4, la Figura 4.6 rappresenta una CSA ad 8 bit, frazionato in 2 blocchi da 4 bit cadauno.

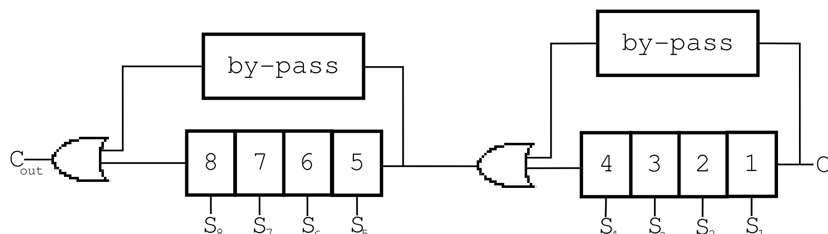


Figura 4.6: Schema di un carry skip adder a 8 bit

Ricordando che:

- il tempo di latenza del circuito di bypass t_{BP} è pari a $3t_d$;
- il tempo di latenza per la propagazione del carry è:
  $\Delta t_{carry} = 2(k-1)t_{DFA} + (m-2)t_{DBP}$;
- il tempo di latenza per la propagazione del risultato corretto è:
  $\Delta t_{ris} = 2kt_{DFA} + (m-2)t_{DBP}$;

con l'ipotesi $t_{DBP} = t_{DFA}$, nel caso specifico in cui sia $m = 2$, $k = 4$, si ottiene:

$$\Delta t_{carry} = 6t_{DFA}.$$

4.2.5 Conclusioni sui sommatori

La scelta di utilizzare un Ripple Adder non è casuale: nella scelta del sommatore i parametri più importanti da analizzare sono il tempo di ritardo e la complessità hardware.
Per quanto riguarda il tempo di ritardo possiamo considerare la Tabella 4.1, valida per sommatori a 8 bit.
Il più veloce è ovviamente il sommatore parallelo che, però, richiede la sintesi diretta di N Full Adder indipendenti; ciò costringe a realizzare l'ultimo FA generando una rete combinatoria con 2x8 = 16 ingressi e il relativo utilizzo di porte AND a 16 ingressi (oltre a una complicazione hardware elevata) che non è permesso dall'harware su cui vogliamo implementare il circuito.
Da scartare anche il Look Up Table, che crea problemi con la realizzazione pratica del circuito, e il sommatore seriale, che rallenterebbe in maniera evidente il mediatore.

Tipo di sommatore	**Tempo di ritardo**
Ripple Adder	$24t_d$
Sommatore Parallelo	$3t_d$
Carry Look Ahead	$10t_d$
Look-up table	tempo di accesso alla memoria
Carry Skip Adder $(m = 2, k = 4)$	$12t_d$
Sommatore Seriale	$n + 1$ colpi di clock

Tabella 4.1: Tempo di latenza di alcuni sommatori

Il Carry Look Ahead ha ottime prestazioni per quanto riguarda la velocità di calcolo, ma la complessità della rete di Carry Generator in un sommatore a 8 bit sconsiglia il suo utilizzo. Possiamo ripetere le stesse considerazioni per il Carry Skip Adder; per tale motivo si è scelto di utilizzare un Ripple Adder.

4.3 La Rete combinatoria

Come scritto precedentemente, la *Rete combinatoria* è un comparatore la cui funzione è segnalare tramite la linea di *stop* quando le sottrazioni del numero 6 sono terminate: ciò avviene quando nel registro *ACC* è memorizzato un numero compreso tra 0 e 5.
Dalla precedente considerazione, si giunge facilmente alla Tabella 4.2, la quale mostra una semplice considerazione nelle colonne centrali:

in base 10	*RC1*	*RC2*	*stop*
5	00000	101	1
4	00000	100	1
3	00000	011	1
2	00000	010	1
1	00000	001	1
0	00000	000	1

Tabella 4.2: Tabella che evidenzia gli unici stati $stop = 1$

Le due colonne centrali mostrano che una parola da 8 bit appartiene nell'intervallo di interi `{0, ..., 5}` quando sono vere le seguenti affermazioni:

1. i 5 bit più significativi sono tutti uguali a 0 (mostrato nella colonna centrale di sinistra della Tabella 4.2);

2. i 3 bit meno significativi sono esaustivamente elencati nella colonna centrale di destra della Tabella 4.2.

Per tale motivo, è possibile semplificare la *Rete combinatoria* in due sottoreti combinatorie, mostrate in Figura 4.7:

1. *RC1*, deputata ad esaminare i primi 5 bit;

2. *RC2*, deputata ad esaminare gli ultimi 3 bit, cioè i tre bit meno significativi.

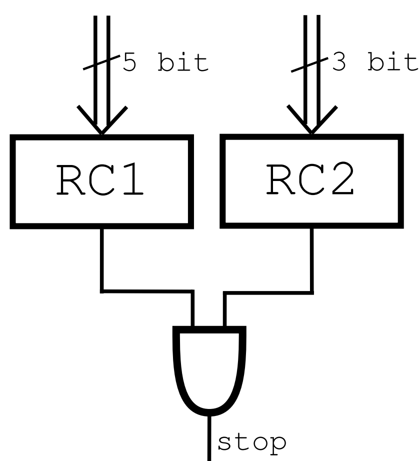


Figura 4.7: Schema generale della rete combinatoria

Nelle due sezioni seguenti, si esamineranno nello specifico la *RC1* e la *RC2*, ricordando che:

- nella rete *RC1* affluiscono i bit $b_7, \ldots, b_3$;
- nella rete *RC2* affluiscono i bit b_2, b_1 e b_0;
- la *Rete combinatoria* è costituita dalla composizione delle precedenti, come mostrato in Figura 4.7.

4.3.1 Rete combinatoria 1

Come precedentemente scritto, la rete *RC1* abilita lo stop quando tutti i bit che affluiscono nella rete sono tutti nulli, quindi semplicemente possiamo graficare la rete *RC1* come in Figura 4.8.

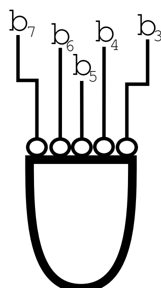


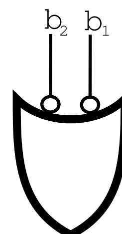
Figura 4.8: Schema rete combinatoria 1

4.3.2 Rete combinatoria 2

Come precedentemente scritto, la rete combinatoria *RC2* abilita lo stop quando i bit che affluiscono sono quelli riassunti nella Tabella 4.2, che possiamo riscrivere come in Tabella 4.3.

Tabella 4.3: Rispettivamente, tabella che evidenzia lo stato stop, e il grafico della sintesi

$b_0 \backslash b_2 b_1$	00	01	11	10
0	1	1	0	1
1	1	1	0	1



Possiamo sintetizzare la Tabella 4.3 con le *mappe di Karnaugh*:

$$stop_2 = \overline{b}_2 + \overline{b}_1$$

4.4 La MSF

L'unità di controllo riceve in ingresso il clock esterno, il segnale di *Start of Operation sop* e il segnale di *Stop of Operation stop*.
Di seguito si analizzerà un esempio che esemplificherà le operazioni e l'evoluzione delle linee di controllo.

WAIT Inizialmente la MSF si trova nello stato WAIT; il registro accumulatore è resettato; se si abilita *sop*, si passa al successivo stato, altrimenti si cicla.

SCRIVI 1^a **parola** Quando è pronta la prima parola nel registro *Reg. A*, la si processa sommando il suo contenuto a quello del registro accumulatore *ACC*.

ATTESA Questo stato non fa nulla e si resta in questo stato finché non si è pronti a processare la prossima parola[5] comunicandolo alla MSF attraverso il segnale *sop*.

SCRIVI 2^a **parola** L'unica differenza rispetto allo stato **Scrivi** 1^a **parola** è nel contenuto del registro accumulatore *ACC*, in questo stato non necessariamente nullo. Come nel caso precedente, dopo aver processato la nuova parola, si va in stato di **ATTESA** e si permane in questo stato finché non si abilita una nuova operazione con il segnale *sop*.
Si possono considerare stati analoghi fino al processamento della 6ª parola.

...

Reset contatore Questo stato si occupa dell'azzeramento del contatore; l'azzeramento del contatore alternativamente può essere effettuato negli stati **SCRIVI n-sima parola** o **Attesa**, ma ho preferito creare uno stato apposito per l'azzeramento.

Divisione Per calcolare la media delle 6 parole, è necessario effettuare la divisione; la divisione è compiuta attraverso sottrazioni successive, in cui ad ogni passo la *Rete combinatoria* comunica se è possibile continuare la sottrazione (*stop=0*) oppure l'operazione di sottrazione è arrivata al termina (*stop=1*); nel caso in cui *stop=0*, si permene in questo stato, altrimenti (*stop=1*) si salta al prossimo.

WAIT In questo stato si permane sino alla prossima operazione, che coincide con l'inserimento della prossima parola, comunicandolo alla MSF attraverso il segnale *sop*; questo stato coincide con il primo stato di una nuova media.

L'analisi appena effettuata è riassunta nel caso generico nella sezione 4.4.1. Dalla precedente analisi, confrontandola con la Figura 4.2, possiamo evincere che nella MSF possiamo trovare sei linee di controllo:

EnClkA È il segnale che abilita il clock per il registro *Reg. A* (PIPO a 4 bit) in cui sono memorizzare le 6 parole.

[5]questo stato è necessario per fornire all'operatore il tempo necessario per selezionare la prossima parola

EnClkACC È il segnale che abilita il clock per il registro accumulatore *ACC* (PIPO a 8 bit) in cui inizialmente è accumulata la somma parziale, successivamente è memorizzato il risultato della sottrazione tra la somma e il numero 6.

EnCount È il segnale che abilita il clock per la rete deputata all'incremento del contatore, quindi al conteggio della media.

Rst È il segnale deputato al reset del registro accumulatore *ACC*.

RstCont È il segnale per il resettaggio del contatore.

mux È il segnale che seleziona l'ingresso del contatore: il sommatore lavorerà con ingressi differenti a seconda che calcoli la somma delle sei parole, oppure che stia nella successiva fase di divisione.

Possiamo evidenziare che dal confronto tra la Figura 4.2 e la Figura 4.11 emergono 3 linee di controllo che sono corrispondenti, ma nella Figura 4.2 sono definite `ClkA`, `ClkACC` e `Count`, nella Figura 4.11 sono definite `EnClkA`, `EnClkACC` e `EnCount`; questa discrepanza è in realtà dovuta ad una differente scala nella visione dell'hardware, poiché, come è correttamente rappresentato dalla Figura 4.11, dalla MSF escono le linee `EnClkA`, `EnClkACC` e `EnCount`; queste ultime, combinate con il clock macchina ed il clock negato, diventano i tre clock `ClkA`, `ClkACC` e `Count`, come mostrato nella Figura 4.9.

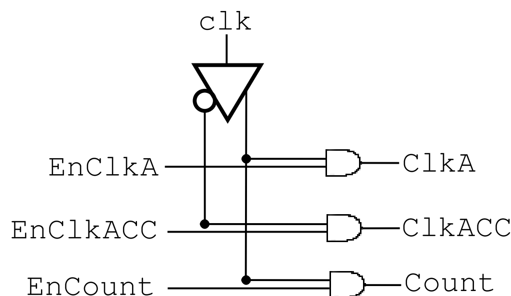


Figura 4.9: Realizzazione delle linee `ClkA`, `ClkACC` e `Count` dalle corrispondenti linee che escono dalla MSF: `EnClkA`, `EnClkACC` e `EnCount`

4.4.1 L'ASM della MSF

L'ASM della MSF è rappresentato nella Figura 4.10, e le linee di controllo degli stati **WAIT**, **ATTESA**, **SCRIVI**, **Reset contatore**, **Divisione** sono illustrate di seguito.

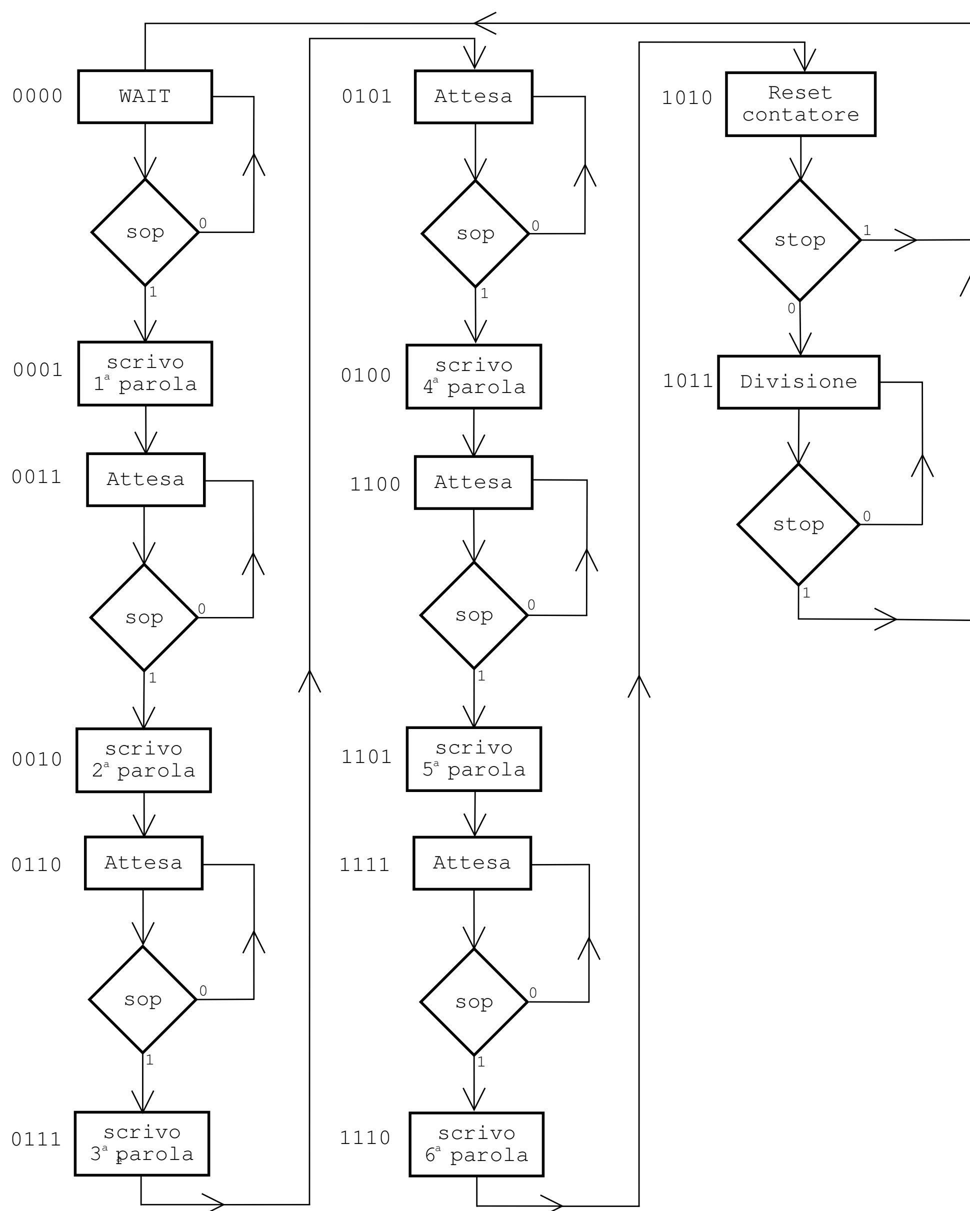


Figura 4.10: Schema ASM della MSF

4.4.2 Schema della MSF

Lo schema della MSF è rappresentato in Figura 4.11, in cui si notano:

- le linee di ingresso *sop* e *stop*;
- le linee di uscita *RstCont*, *EnClkA*, *mux*, *EnClkACC*, *EnCount*, *Rst*.

WAIT
EnClkA = 0
EnClkACC = 0
Rst = 1
EnCount = 0
mux = 0
RstCont = 0

ATTESA
EnClkA = 0
EnClkACC = 0
Rst = 0
EnCount = 0
mux = 0
RstCont = 0

Divisione
EnClkA = 0
EnClkACC = 1
Rst = 0
EnCount = 1
mux = 1
RstCont = 0

SCRIVI
EnClkA = 1
EnClkACC = 1
Rst = 0
EnCount = 0
mux = 0
RstCont = 0

Reset contatore
EnClkA = 0
EnClkACC = 0
Rst = 0
EnCount = 0
mux = 1
RstCont = 1

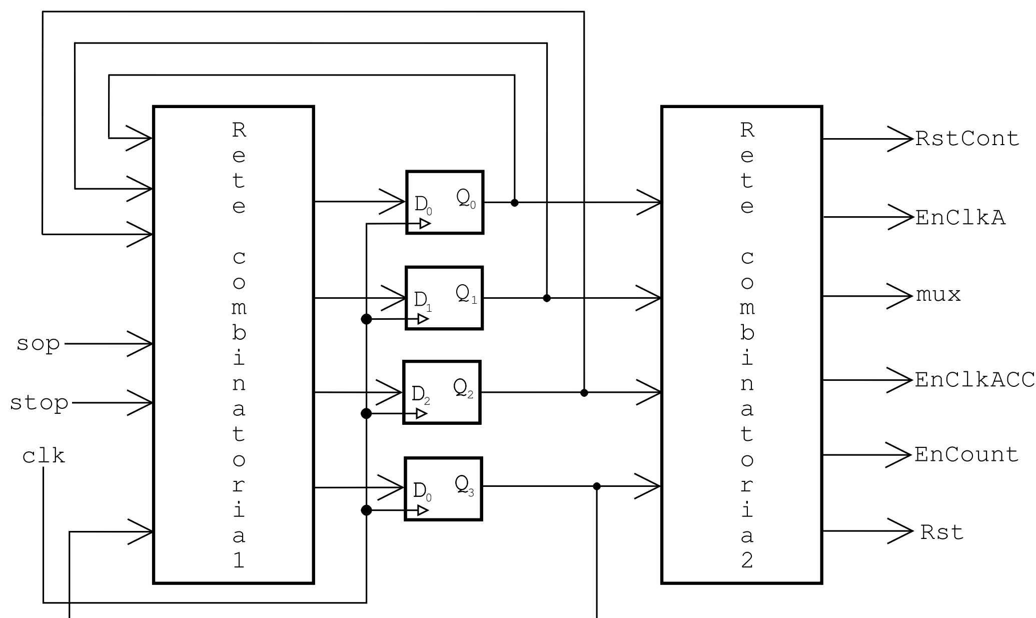


Figura 4.11: Schema della MSF

4.4.3 Sintesi della funzione stato prossimo

Nella Tabella 4.4 sono elencate tutte le linee di controllo e la loro evoluzione. Di seguito, è evidenziata la sintesi delle due reti *Rete combinatoria 1* e *Rete combinatoria 2* dello schema nella Figura 4.11, in cui si ricorda che:

- la *Rete combinatoria 1* è necessaria per l'evoluzione dei flip-flop;
- la *Rete combinatoria 2* è necessaria per l'evoluzione delle linee di controllo.

sop	stop	Q_3	Q_2	Q_1	Q_0	D_3	D_2	D_1	D_0	Reset	EnClkACC	mux	RstCont	EnClkA	EnCount
0	-	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	-	0	0	0	0	0	0	0	1	0	0	0	0	0	0
-	-	0	0	0	1	0	0	1	1	0	1	0	0	1	0
0	-	0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	-	0	0	1	1	0	0	1	0	0	0	0	0	0	0
-	-	0	0	1	0	0	1	1	0	0	1	0	0	1	0
0	-	0	1	1	0	0	1	1	0	0	0	0	0	0	0
1	-	0	1	1	0	0	1	1	1	0	0	0	0	0	0
-	-	0	1	1	1	0	1	0	1	0	1	0	0	1	0
0	-	0	1	0	1	0	1	0	1	0	0	0	0	0	0
1	-	0	1	0	1	0	1	0	0	0	0	0	0	0	0
-	-	0	1	0	0	1	1	0	0	0	1	0	0	1	0
0	-	1	1	0	0	1	1	0	0	0	0	0	0	0	0
1	-	1	1	0	0	1	1	0	1	0	0	0	0	0	0
-	-	1	1	0	1	1	1	1	1	0	1	0	0	1	0
0	-	1	1	1	1	1	1	1	1	0	0	0	0	0	0
1	-	1	1	1	1	1	1	1	0	0	0	0	0	0	0
-	-	1	1	1	0	1	0	1	0	0	1	0	0	1	0
-	0	1	0	1	0	1	0	1	1	0	0	1	1	0	0
-	1	1	0	1	0	0	0	0	0	0	0	1	1	0	0
-	0	1	0	1	1	1	0	1	1	0	1	1	0	0	1
-	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0

Tabella 4.4: Elenco esaustivo dell'evoluzione delle linee di controllo per tutti gli stati. Il simbolo "-" è stato inserito per evidenziare che lo stato della linea di controllo corrispondente non importa ai fini della sintesi e, pertanto, è possibile ignorarlo nella successiva sintesi

Rete combinatoria 2

RstCont $= Q_3\overline{Q}_2Q_1\overline{Q}_0$

Reset $= \overline{Q}_3\overline{Q}_2\overline{Q}_1\overline{Q}_0$

mux $= Q_3\overline{Q}_2Q_1$

EnClkA $= \overline{Q}_3\overline{Q}_2\overline{Q}_1Q_0 + \overline{Q}_3\overline{Q}_2Q_1\overline{Q}_0 + \overline{Q}_3Q_2Q_1Q_0 + \overline{Q}_3Q_2\overline{Q}_1\overline{Q}_0 + Q_3Q_2\overline{Q}_1Q_0 + Q_3Q_2Q_1\overline{Q}_0$

EnClkACC $= EnClkA + Q_3\overline{Q}_2Q_1\overline{Q}_0$

EnCount $= \overline{stop}Q_3\overline{Q}_2Q_1Q_0$

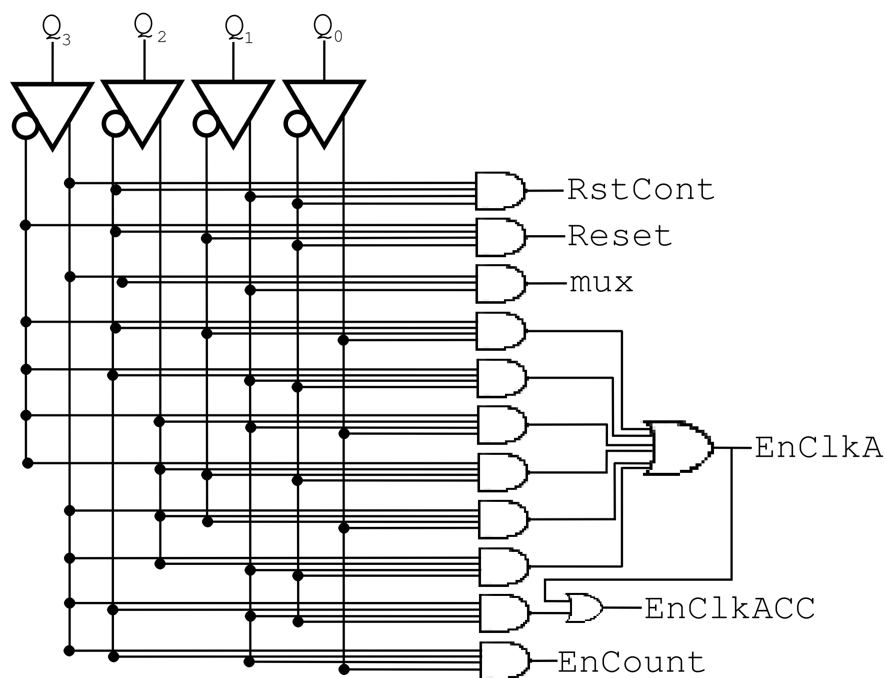


Figura 4.12: Sintesi della *Rete combinatoria 2* della Figura 4.11

Rete combinatoria 1

$nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	*sop*	1	$\overline{sop}$	0
0 1	0	$\overline{sop}$	1	sop
1 1	*sop*	1	$\overline{sop}$	0
1 0	0	0	$\overline{stop}$	$\overline{stop}$

Per ridurre la complessità della Tabella, possiamo suddividerla nelle 4 combinazioni dei due diversi stati delle due linee di controllo `sop` e `stop`:

1. `sop=0`
   `stop=0`

   $nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	0	1	1	0
0 1	0	1	1	0
1 1	0	1	1	0
1 0	0	0	1	1

   $nextD_0 = \overline{Q}_3Q_0 + Q_2Q_0 + Q_3\overline{Q}_2Q_1$

2. `sop=0`
   `stop=1`

   $nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	0	1	1	0
0 1	0	1	1	0
1 1	0	1	1	0
1 0	0	0	0	0

   $nextD_0 = \overline{Q}_3Q_0 + Q_2Q_0$

3. `sop=1`
   `stop=0`

   $nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	1	1	0	0
0 1	0	0	1	1
1 1	1	1	0	0
1 0	0	0	1	1

   $nextD_0 = \overline{Q}_3\overline{Q}_2\overline{Q}_1 + \overline{Q}_3Q_2Q_1 + Q_3Q_2\overline{Q}_1 + Q_3\overline{Q}_2Q_1$

4. `sop=1`
   `stop=1`

$nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	1	1	0	0
0 1	0	0	1	1
1 1	1	1	0	0
1 0	0	0	0	0

$$nextD_0 = \overline{Q}_3\overline{Q}_2\overline{Q}_1 + \overline{Q}_3Q_2Q_1 + Q_3Q_2\overline{Q}_1$$

Possiamo riassumere le 4 combinazioni precedentemente elencate nella Tabella 4.5.

`sop`	`stop`	$nextD_0$
0	0	$\overline{Q}_3Q_0 + Q_2Q_0 + Q_3\overline{Q}_2Q_1$
0	1	$\overline{Q}_3Q_0 + Q_2Q_0$
1	0	$\overline{Q}_3\overline{Q}_2\overline{Q}_1 + \overline{Q}_3Q_2Q_1 + Q_3Q_2\overline{Q}_1 + Q_3\overline{Q}_2Q_1$
1	1	$\overline{Q}_3\overline{Q}_2\overline{Q}_1 + \overline{Q}_3Q_2Q_1 + Q_3Q_2\overline{Q}_1$

Tabella 4.5: $nextD_0$ suddivisi per gli stati delle due linee `sop` e `stop`

Osservando con attenzione la Tabella 4.5, si nota che molte combinazioni si ripetono; tra le tante ripetizioni, è anche possibile verificare che l'ultima riga (`sop=1` e `stop=1`) è ridondante poiché la riga precedente la include; è quindi possibile semplificare la Tabella 4.5 nella Tabella 4.6, procedendo quindi alla sintesi mostrata nella Figura 4.13.

`sop`	`stop`	$nextD_0$
1	-	$\overline{Q}_3\overline{Q}_2\overline{Q}_1 + \overline{Q}_3Q_2Q_1 + Q_3Q_2\overline{Q}_1$
0	-	$\overline{Q}_3Q_0 + Q_2Q_0$
-	0	$Q_3\overline{Q}_2Q_1$

Tabella 4.6: Semplificazione della Tabella 4.5

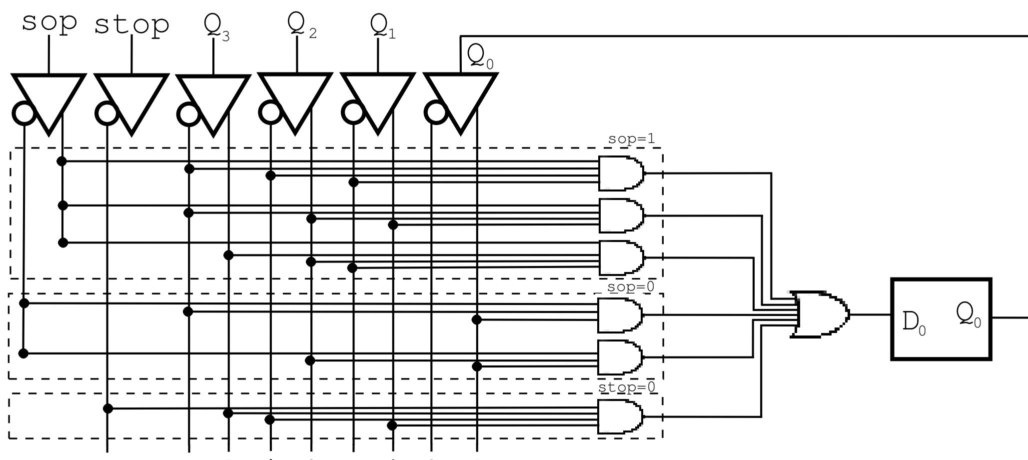


Figura 4.13: Sintesi semplificata della rete combinatoria per $nextD_0$

$nextD_1 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	0	1	1	1
0 1	0	0	0	1
1 1	0	1	1	1
1 0	0	0	$\overline{stop}$	$\overline{stop}$

$$nextD_1 = \overline{Q}_3\overline{Q}_2Q_0 + \overline{Q}_3Q_1\overline{Q}_0 + Q_3Q_2Q_0 + Q_2Q_1\overline{Q}_0 + \overline{stop}Q_3Q_1$$

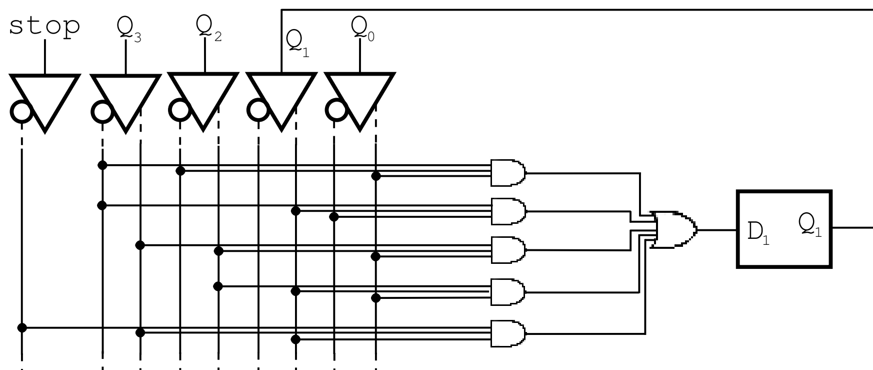


Figura 4.14: Sintesi della rete combinatoria per $nextD_1$

$nextD_2 \longrightarrow$

Q_3Q_2/Q_1Q_0	00	01	11	10
0 0	0	0	0	1
0 1	1	1	1	1
1 1	1	1	1	0
1 0	0	0	0	0

$nextD_2 = Q_2\overline{Q}_1 + Q_2Q_0 + \overline{Q}_3Q_1\overline{Q}_0$

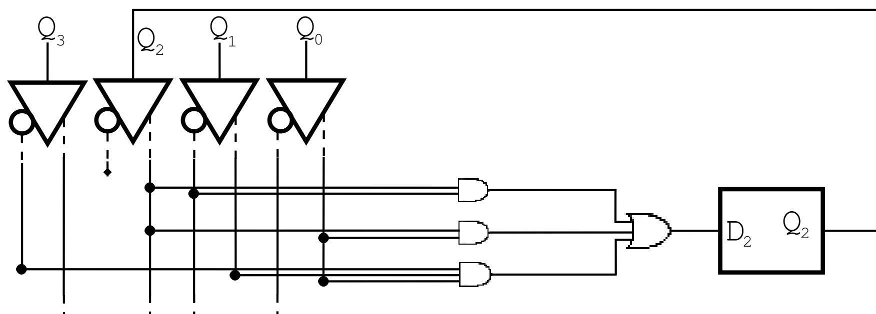


Figura 4.15: Sintesi della rete combinatoria per $nextD_2$

$nextD_3 \longrightarrow$

Q_3Q_2/Q_1Q_0	00	01	11	10
0 0	0	0	0	0
0 1	1	0	0	0
1 1	1	1	1	1
1 0	0	0	$\overline{stop}$	$\overline{stop}$

$nextD_3 = Q_3Q_2 + Q_2\overline{Q}_1\overline{Q}_0 + \overline{stop}Q_3Q_1$

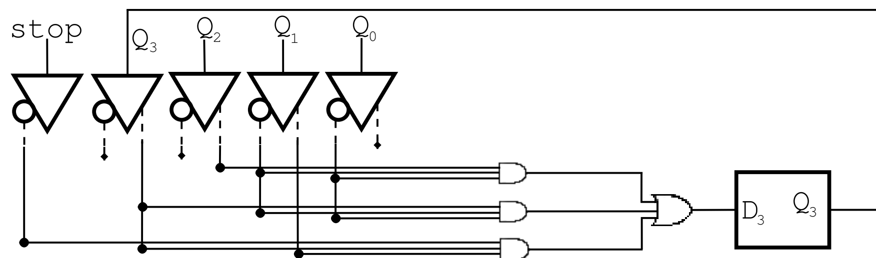


Figura 4.16: Sintesi della rete combinatoria per $nextD_3$

4.5 Il contatore

Il contatore riceve in ingresso il clock costituito dal segnale *add*[6], segnale che comunica al contatore di cambiare stato, cioè aumentare di un'unità.

[6]costituito dall'`and` tra i segnali `Count` e `!stop`

4.5.1 ASM del contatore

Possiamo valutare il semplice ASM del contatore nella Figura 4.17.

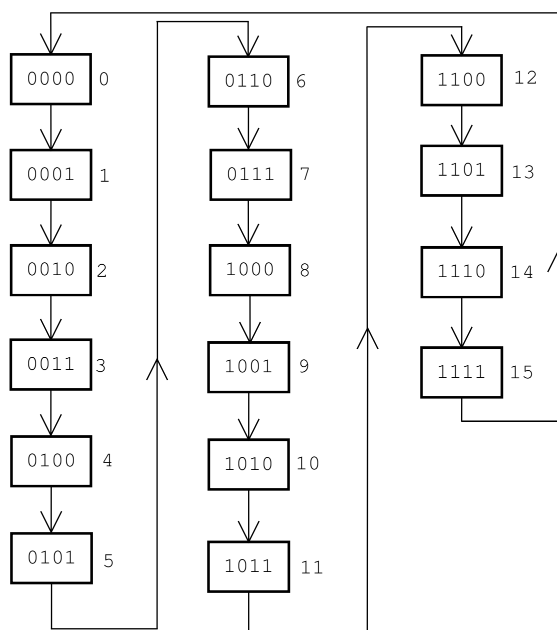


Figura 4.17: Schema ASM del contatore

4.5.2 Schema del contatore

Possiamo valutare il semplice schema del contatore nella Figura 4.18, in cui si ricorda che la *Rete combinatoria* è necessaria per l'evoluzione dei flip-flop.

4.5.3 Sintesi della funzione stato prossimo

Nella pagina seguente, in Tabella 4.7 sono elencati gli stati assunti dal contatore e la sua evoluzione.

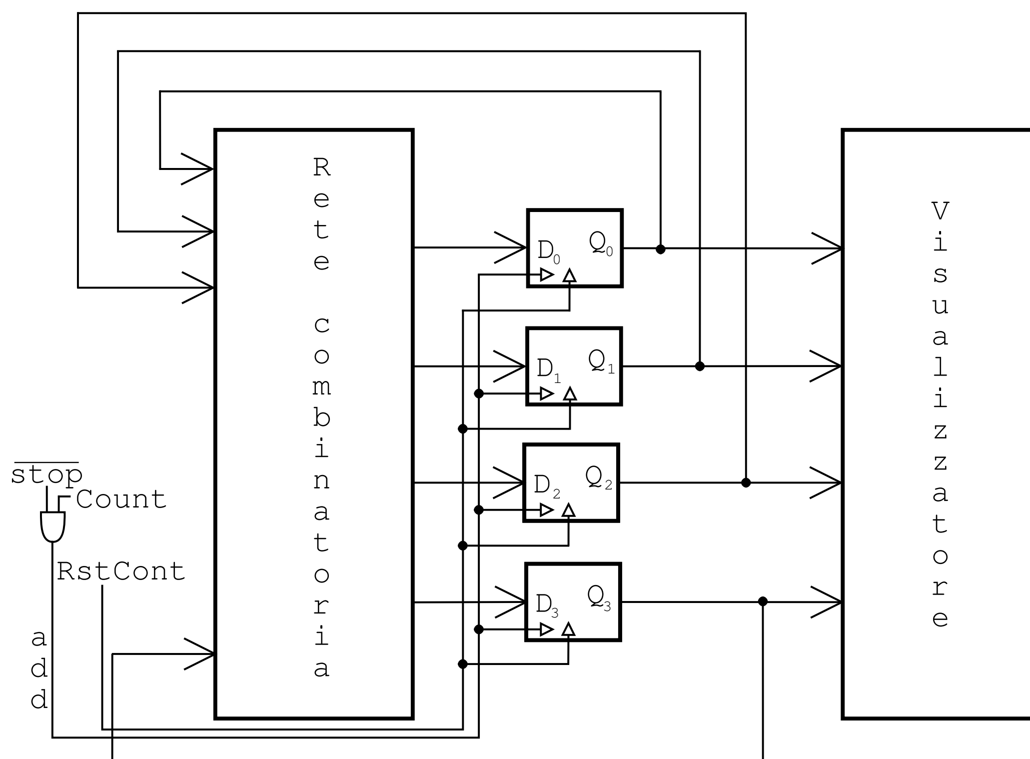


Figura 4.18: Schema del contatore

Rete combinatoria

Di seguito è evidenziata la sintesi della rete *Rete combinatoria* dello schema rappresentato in Figura 4.18.

$nextD_0 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	1	0	0	1
0 1	1	0	0	1
1 1	1	0	0	1
1 0	1	0	0	1

$nextD_0 = \overline{Q}_0$

$nextD_1 \longrightarrow$

$Q_3Q_2 \backslash Q_1Q_0$	00	01	11	10
0 0	0	1	0	1
0 1	0	1	0	1
1 1	0	1	0	1
1 0	0	1	0	1

Q_3	Q_2	Q_1	Q_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Tabella 4.7: Elenco esaustivo dell'evoluzione degli stati del contatore

$$nextD_1 = \overline{Q}_1 Q_0 + Q_1 \overline{Q}_0$$

$nextD_2 \longrightarrow$

Q_3Q_2/Q_1Q_0	00	01	11	10
0 0	0	0	1	0
0 1	1	1	0	1
1 1	1	1	0	1
1 0	0	0	1	0

$$nextD_2 = \overline{Q}_3 \overline{Q}_2 Q_1 Q_0 + Q_2 \overline{Q}_1 + Q_2 \overline{Q}_0 + Q_3 \overline{Q}_2 Q_1 Q_0$$

$nextD_3 \longrightarrow$

Q_3Q_2/Q_1Q_0	00	01	11	10
0 0	0	0	0	0
0 1	0	0	1	0
1 1	1	1	0	1
1 0	1	1	1	1

$$nextD_3 = \overline{Q}_3 Q_2 Q_1 Q_0 + Q_3 \overline{Q}_1 + Q_3 \overline{Q}_0 + Q_3 \overline{Q}_2$$

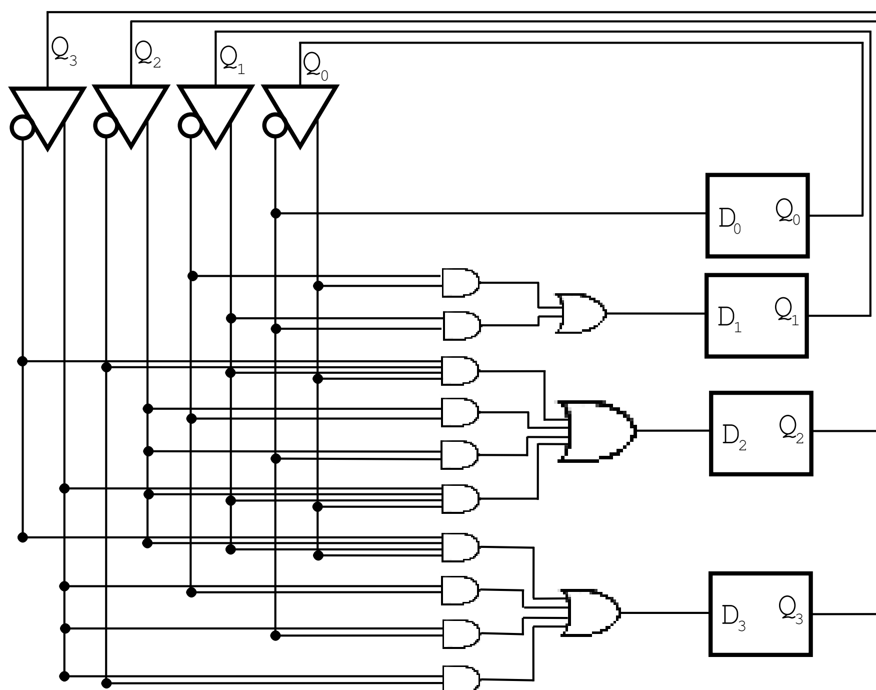


Figura 4.19: Sintesi della *Rete combinatoria* per $nextD_0$, $nextD_1$, $nextD_2$ e $nextD_3$

4.6 Visualizzatore

La progettazione del mediatore può considerarsi conclusa in quanto il dispositivo attualmente visualizza la media delle 6 parole immesse.
Se la progettazione si arrestasse, il dispositivo visualizzerebbe la media delle parole, ma la visualizzerebbe in esadecimale[7]. Per agevolare la lettura, è possibile visualizzare il risultato non in esadecimale, ma in decimale: a questo stato della progettazione, il visualizzatore riceve in ingresso 4 linee poiché il contatore può calcolare un numero intero che appartiene all'intervallo `{0, ..., 15}`, il quale si scrive in binario con al massimo 4 bit[8].
Invece di collegare direttamente queste 4 linee con il visualizzatore, è possibile processare questi 4 valori affinché si possa visualizzare il risultato in decimale.
Per chiarire questo concetto, è utile ricorrere alla Tabella 4.6.

[7]ad esempio, i numeri 10_{10}, 11_{10}, 12_{10}, 13_{10}, 14_{10} e 15_{10} sono visualizzati tramite, rispettivamente, i corrispondenti A_{16}, B_{16}, C_{16}, D_{16}, E_{16} e F_{16}

[8]$15_{10} = 1111_2$

$\mathbf{in_3}$	$\mathbf{in_2}$	$\mathbf{in_1}$	$\mathbf{in_0}$	$\mathbf{o_4}$	$\mathbf{o_3}$	$\mathbf{o_2}$	$\mathbf{o_1}$	$\mathbf{o_0}$
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

Tabella 4.8: Le 4 colonne di sinistra rappresentano la codifica binaria del numero che sarà espressa in decimale tramite le 5 colonne di destra

Dalla precedente tabella emerge che un numero in esadecimale può essere visualizzato direttamente se è inferiore a 1001_2, altrimenti dovrà essere processato; di seguito codificherò la tabella precedente isolando i segnali che saranno inviati al visualizzatore.

$o_0 \longrightarrow$

$in_3in_2 \backslash in_1in_0$	00	01	11	10
0 0	0	1	1	0
0 1	0	1	1	0
1 1	0	1	1	0
1 0	0	1	1	0

$o_0 = in_0$

$o_1 \longrightarrow$

$in_3in_2 \backslash in_1in_0$	00	01	11	10
0 0	0	0	1	1
0 1	0	0	1	1
1 1	1	1	0	0
1 0	0	0	0	0

$o_1 = \overline{in}_3in_1 + in_3in_2\overline{in}_1$

$o_2 \longrightarrow$

$in_3in_2 \backslash in_1in_0$	00	01	11	10
0 0	0	0	0	0
0 1	1	1	1	1
1 1	0	0	1	1
1 0	0	0	0	0

$o_2 = \overline{in}_3 in_2 + in_2 in_1$

$o_3 \longrightarrow$

$in_3in_2 \backslash in_1in_0$	00	01	11	10
0 0	0	0	0	0
0 1	0	0	0	0
1 1	0	0	0	0
1 0	1	1	0	0

$o_3 = in_3 \overline{in}_2 \overline{in}_1$

$o_4 \longrightarrow$

$in_3in_2 \backslash in_1in_0$	00	01	11	10
0 0	0	0	0	0
0 1	0	0	0	0
1 1	1	1	1	1
1 0	0	0	1	1

$o_4 = in_3 in_2 + in_3 in_1$

4.7 Analisi dell'evoluzione temporale

Nella Figura 4.20 sono mostrate tutte le linee di controllo che, insieme al clock e al clock negato, comandano gli elementi del mediatore.
Si può osservare la loro evoluzione quando l'unità di controllo passa da uno stato all'altro.
Un altro modo per comprendere meglio l'evoluzione temporale del circuito può essere la Figura 4.21, che riporta l'evoluzione dei clock generati dalla macchina a stati finiti.
I due grafici mensionati sono frutto di studio e, pertanto, sono stati sviluppati per coprire tutti i casi possibili; nella fase di test è stata effettuata un'analisi temporale delle linee di controllo direttamente sul circuito; in particolare, nella Figura 4.22 sono state inserite parole per un totale di 42_{10}, nella Figura 4.23 sono state inserire parole per un totale di 55_{10}, nella Figura 4.24 sono state inserire parole per un totale di 82_{10}.

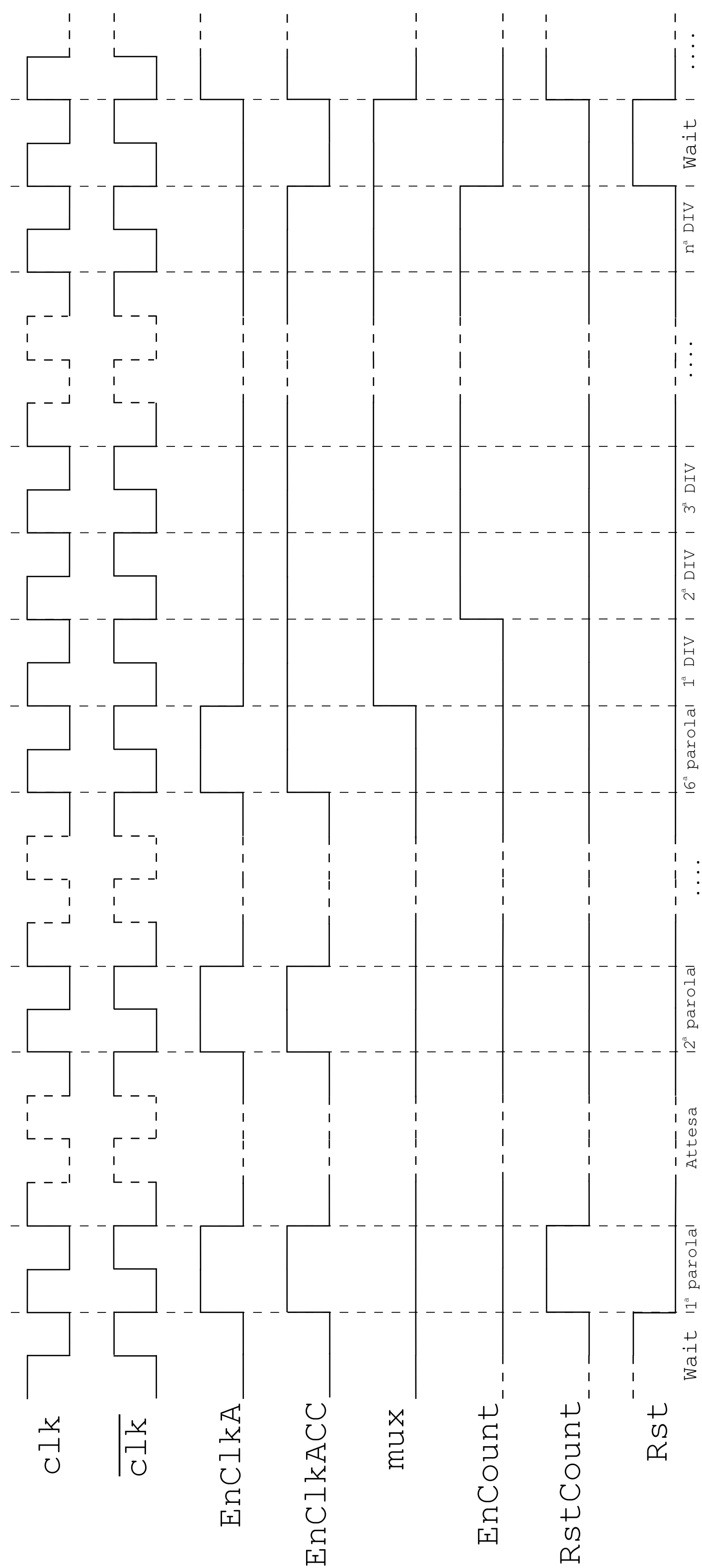


Figura 4.20: Evoluzione temporale delle linee di controllo

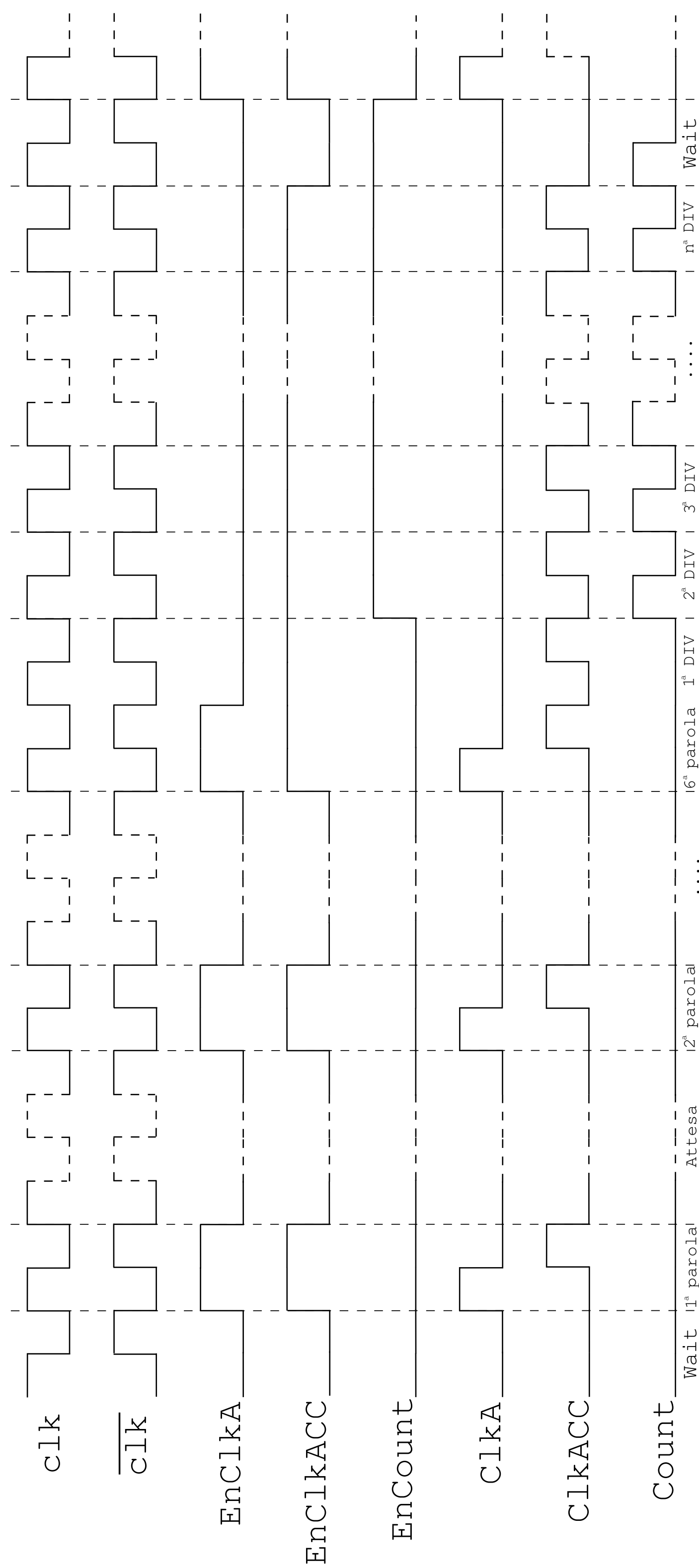


Figura 4.21: Evoluzione temporale delle linee di controllo generati dalla MSF

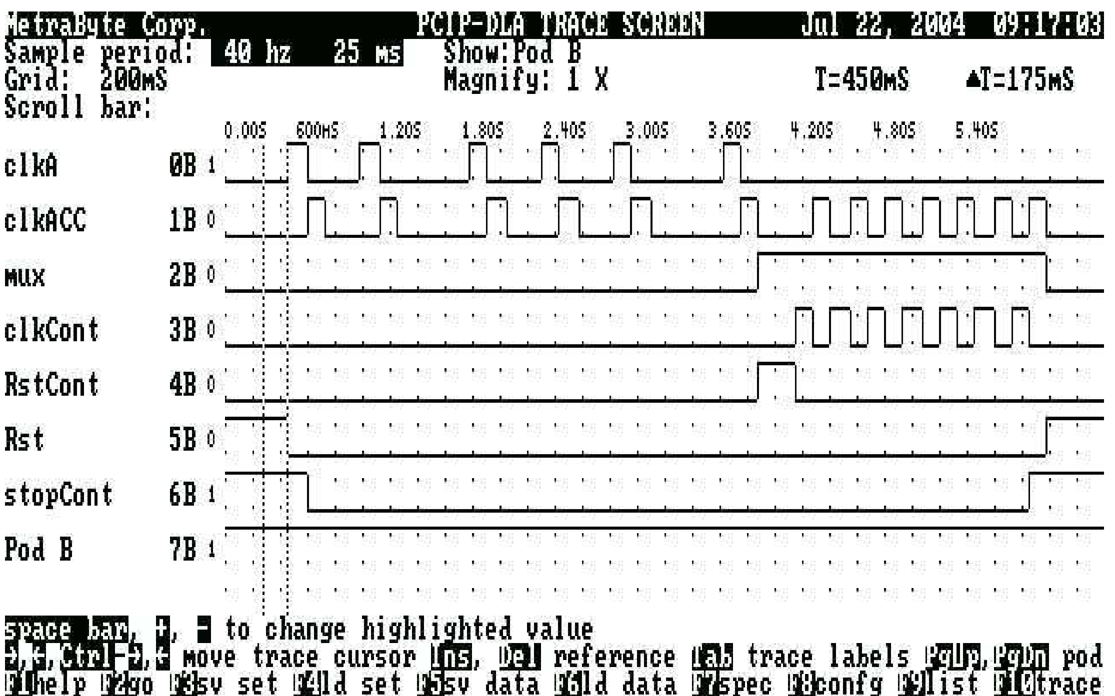


Figura 4.22: Analisi temporale delle linee di controllo generati dalla MSF per parole che sommate risultano pari ai 42_{10}

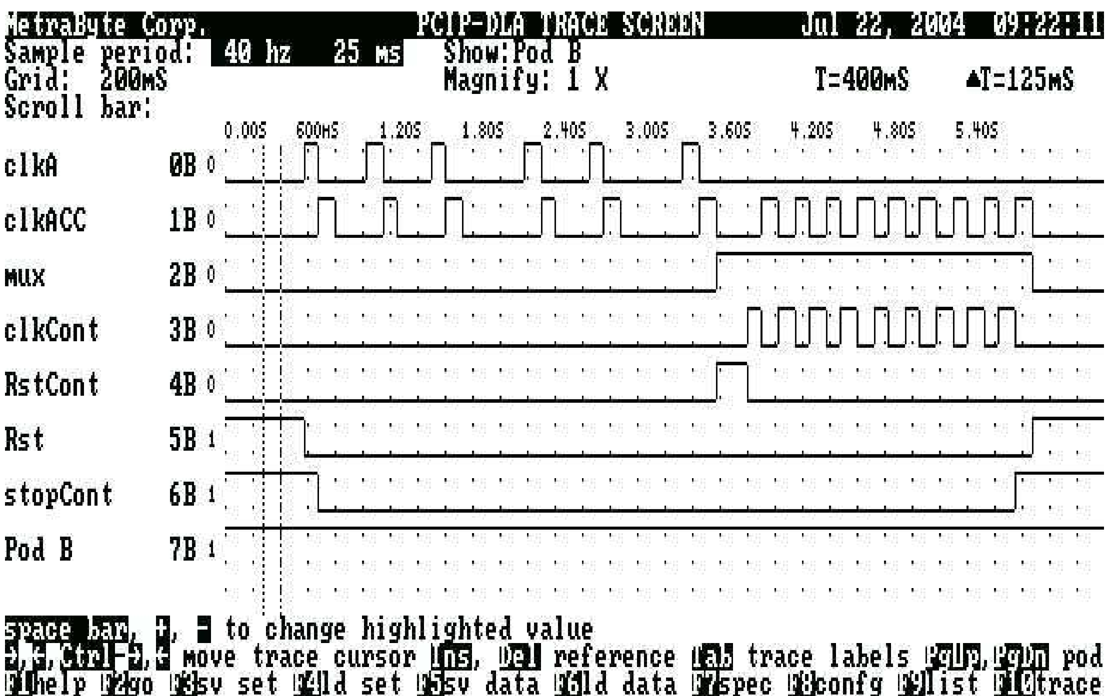


Figura 4.23: Analisi temporale delle linee di controllo generati dalla MSF per parole che sommate risultano pari a 55_{10}

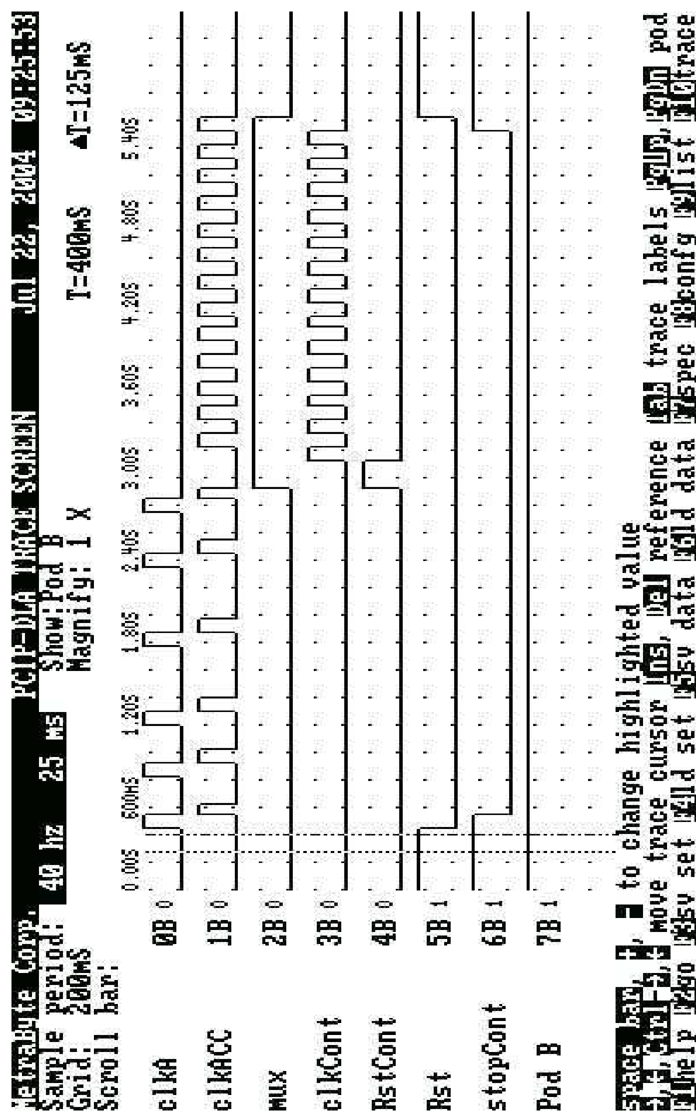


Figura 4.24: Analisi temporale delle linee di controllo generati dalla MSF per parole che sommate risultano pari a 82_{10}

Capitolo 5

La IspLSI 1016 60 LJ della Lattice

5.1 Schema interno e suo funzionamento

La IspLSI 10161 è un dispositivo di logica programmabile ad alta densità contenente 96 registri, 32 pin di input/output, 4 pin per ingressi dedicati, 3 pin di input per clock esterni e un GRP (Global Routing Pool), che garantisce un collegamento completo tra questi elementi.

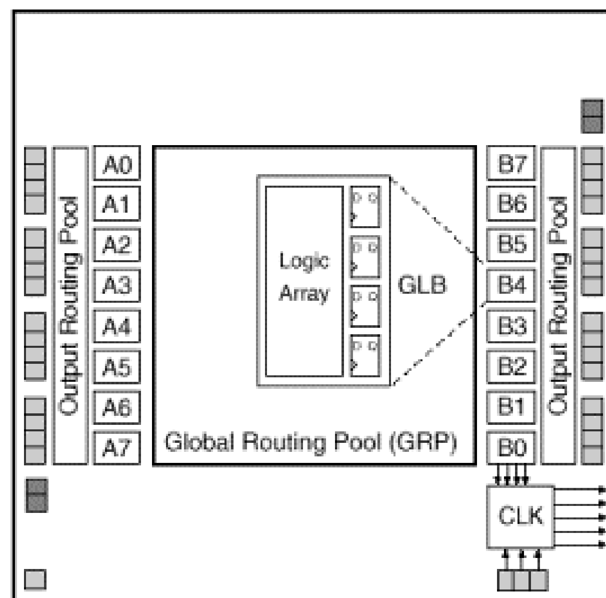


Figura 5.1: Schema interno che evidenzia la struttura del dispositivo a logica programmabile IspLSI1016

L'unità logica di base sull'IspLSI1016 è il GLB (Generic Logic Block). In ogni IspLSI1016 sono presenti 16 GLB. Ogni GLB ha 18 ingressi, un array programmabile di AND/OR/XOR e 4 uscite che possono essere di tipo combinatorio o sequenziale.
Tutti gli ingressi di un GLB provengono dal Global Routine Pool e dai 4 pin per gli ingressi dedicati, mentre tutte le uscite di un GLB sono riportate nel Global Routine Pool così da poter essere riutilizzate come ingresso per un qualunque altro GLB. Il dispositivo presenta inoltre 32 celle di Input/Output, ognuna delle quali è direttamente connessa a un pin di I/O. Ogni cella può essere programmata per contenere un ingresso, un uscita o un pin di I/O bidirezionale.
Otto GLB, 16 celle di I/O, due ingressi specifici e un Output Routine Pool (ORP) sono connessi tra loro per formare un Megablock. Le uscite degli otto GLB sono connesse alle 16 celle di I/O attraverso l'Output Routing Pool.

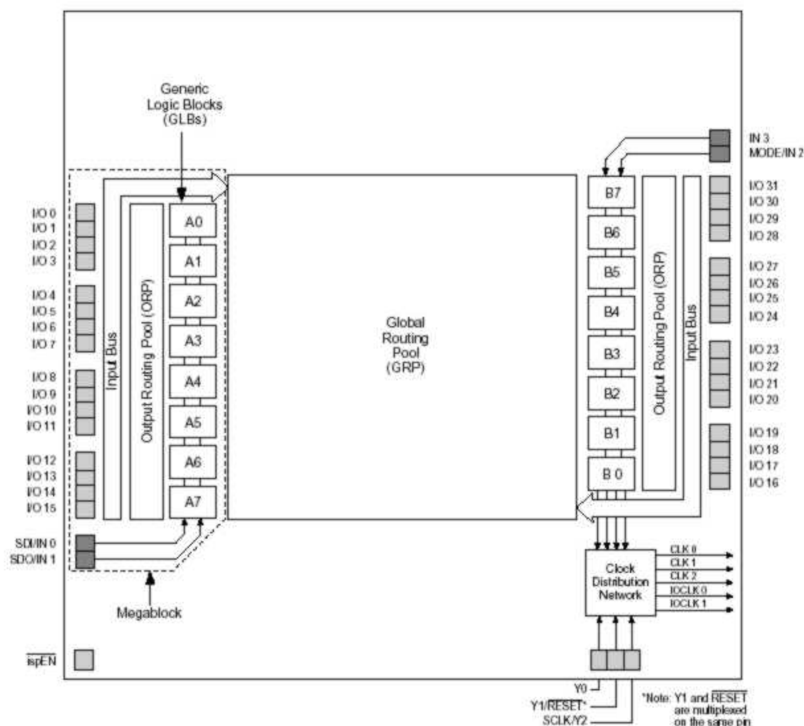


Figura 5.2: Schema interno che evidenzia la struttura del dispositivo a logica programmabile IspLSI1016

La gestione dei clock nell'IspLSI 1016 è lasciata al network per la distribuzione dei clock. Questo ha in ingresso tre pin specifici per clock esterni e le quattro uscite di un GLB apposito (che permette la creazione di clock interni a partire da una combinazione di segnali interni) e genera cinque segnali di clock: clk0, clk1, clk2 che sono usati per la corretta temporizzazione dei GLB e IOclk0, IOclk1 che sono utilizzati per comandare le celle di I/O.

5.2 Programmazione

Per quanto riguarda gli aspetti tecnici della programmazione dei due IspLSI 1016 ci si è attenuti a quanto riportato nel manuale specifico.

5.3 Download

Una volta completata la fase di programmazione al Pc[1], si è passati al download sui due IspLSI 1016 utilizzando una connesione dalla porta LPT del Pc alla basetta per la programmazione.
Ultimata la fase di download si è poi passati alla fase di verifica e al collaudo finale.

[1] nell'Appendice sono presenti per intero i due listati per la programmazione di ciascuno IspLSI 1016 con una descrizione dettagliata di tutte le istruzioni presenti in ogni GLB e in ogni cella di I/O.

Capitolo 6

Realizzazione del circuito

6.1 Test in laboratorio

Una volta eseguito il download dello schema su ciascuna PLD, si sono eseguiti una serie di test per verificare il corretto funzionamento del mediatore, indipendentemente dall'apparente correttezza del risultato.
Vorrei segnalare i due test che ritengo siano i più importanti:

- mostrare le linee di controllo e valutare la corrispondenza tra l'evoluzione effettiva e la sequenza voluta;

- analizzare la frequenza di lavoro: il datasheet della IspLSI1016 assicura un frequenza massima di lavoro pari a 60 Mhz; in laboratorio è possibile arrivare fino a 2 Mhz poiché non ci sono generatori di clock oltre i 2 Mhz.
  Indipendentemente da queste considerazioni, ho lavorato a frequenza di pochi hertz poiché altrimenti non riuscivo a selezionare parole differenti per ingressi consecutivi.

Appendice A

Listato

Di seguito sono elencati i due listati per programmare la 1^a PLD (PLD I) e la 2^a PLD (PLD II) di cui mi sono servito.

A.1 Programmazione della PLD I

```
// Thu Jul 22 11:39:22 2004
// MARIANO2.txt generated using Lattice pDS Version 2.20

LDF 1.00.00 DESIGNLDF;
DESIGN MARIANO2;

PART ispLSI1016-60LJ;

OPTION PULLUP ALL;

DECLARE
END;  //DECLARE

SYM GLB  A1  1  Conta;
sigtype [c0,c1,c2,c3] reg out;

equations

c0=!c0;
c1=c0&!c1 # !c0&c1;
c2=c2&!c1 # !c2&c1&c0 # c2&!c0;
c3=!c3&c2&c1&c0 # c3&!c1 # c3&!c0 # c3&!c2;
```

```
c0.ptclk=clkCont;
c1.ptclk=clkCont;
c2.ptclk=clkCont;
c3.ptclk=clkCont;

c0.RE=RstCont;
c1.RE=RstCont;
c2.RE=RstCont;
c3.RE=RstCont;
end;
END;

SYM GLB  A0  1  clkMSF1;
sigtype [EnclkACC,Cont,RstCont] out;

equations

EnclkACC=EnclkA # m3&!m2&m1&m0;
Cont = !stopcont&m3&!m2&m1&m0;
RstCont=m3&!m2&m1&!m0;

end;
END;

SYM GLB  A2  1  MSF;
sigtype [m0,m1,m2,m3] reg out;

equations

m0=sop& ( !m3 & !m2 & !m1 #
                  !m3 & m2 & m1 #
                  m3 & m2 & !m1) #
      !sop& (!m3 & m0 # m2 & m0) #
      !stopCont & m3&!m2&m1 ;

m1=!m3&!m2&m0 # !m3&m1&!m0 #
      m3&m2&m0 # m2&m1&!m0 # !stopcont&m3&m1;

m2=m2&!m1# m2&m0 # !m3&m1&!m0;
```

```
m3=m3&m2 # m2&!m1&!m0 # !stopCont& m3&m1;

m0.clk=ck;
m1.clk=ck;
m2.clk=ck;
m3.clk=ck;

end;
//#   stopCont &m3&!m2&m1&!m0
END;

SYM GLB  A3  1  clkMSF2;
sigtype [Rst,stopCont,EnclkA]  out;

equations

Rst=!m3&!m2&!m1&!m0;
StopCont = !acc7& !acc6&!acc5&!acc4&!acc3  &(!acc2#!acc1);
EnclkA= !m3&!m2&!m1&m0 # !m3&!m2&m1&!m0 #
          !m3&m2&m1&m0 # !m3&m2&!m1&!m0 #
          m3&m2&!m1&m0 # m3&m2&m1&!m0;

end;
END;

SYM GLB  A7  1  acc0/3;
sigtype [acc0,acc1,acc2,acc3] reg  out;

equations

acc0=som0;
acc1=som1;
acc2=som2;
acc3=som3;

acc0.ptclk= clkACC;
acc1.ptclk=clkACC;
acc2.ptclk=clkACC;
acc3.ptclk=clkACC;

acc0.RE=Rst;
```

```
acc1.RE=Rst;
acc2.RE=Rst;
acc3.RE=Rst;
end;
END;

SYM GLB  A6  1  acc4/7;
sigtype [acc4,acc5,acc6,acc7] reg  out;

equations

acc4=som4;
acc5=som5;
acc6=som6;
acc7=som7;

acc4.ptclk= clkACC;
acc5.ptclk=clkACC;
acc6.ptclk=clkACC;
acc7.ptclk=clkACC;

acc4.RE=Rst;
acc5.RE=Rst;
acc6.RE=Rst;
acc7.RE=Rst;
end;
END;

SYM GLB  A5  1  som0-2;
sigtype [som0, r0, som2, r2]  out;

equations

som0=mux0&!mux20 # !mux0&mux20;

r0=mux0&mux20;

som2=mux2&mux22&r1 # mux2&!mux22&!r1 #
           !mux2&mux22&!r1 # !mux2&!mux22&r1;

r2=mux2 & mux22 # mux2&r1 # r1&mux22;
```

```
end;
END;

SYM GLB  B7  1  mux4/7;
sigtype [mux4,mux5,mux6,mux7]  out;

equations

mux4= 0 & !mux # acc4&mux;
mux5= 0 & !mux # acc5&mux;
mux6= 0 & !mux # acc6&mux;
mux7= 0 & !mux # acc7&mux;

end;
END;

SYM GLB  B6  1  mux0/3;
sigtype [mux0,mux1,mux2,mux3]  out;

equations

mux0=a0 & !mux # acc0&mux;
mux1=a1 & !mux # acc1&mux;
mux2=a2 & !mux # acc2&mux;
mux3=a3 & !mux # acc3&mux;

end;
END;

SYM GLB  B5  1  mux24/7;
sigtype [mux24,mux25,mux26,mux27] out;

equations

mux24= 1 & mux # acc4&!mux;
mux25= 1 & mux # acc5&!mux;
mux26= 1 & mux # acc6&!mux;
mux27= 1 & mux # acc7&!mux;

end;
```

```
END;

SYM GLB  B4  1  mux20/4;
sigtype [mux20,mux21,mux22,mux23]  out;

equations

mux20=0 & mux # acc0&!mux;
mux21=1 & mux # acc1&!mux;
mux22=0 & mux # acc2&!mux;
mux23=1 & mux # acc3&!mux;

end;
END;

SYM GLB  B3  1  som5-7;
sigtype [som5, r5, som7, r7] out;

equations

som5=mux5&mux25&r4 # mux5&!mux25&!r4 #
           !mux5&mux25&!r4 # !mux5&!mux25&r4;

r5=mux5 & mux25 # mux5&r4 # r4&mux25 ;

som7=mux7&mux27&r6 # mux7&!mux27&!r6 #
           !mux7&mux27&!r6 # !mux7&!mux27&r6;

r7=mux7 & mux27 # mux7&r6 # r6&mux27 ;

end;
END;

SYM GLB  B2  1  som1-3;
sigtype [som1, r1, som3, r3] out;

equations

som1=mux1&mux21&r0 # mux1&!mux21&!r0 #
           !mux1&mux21&!r0 # !mux1&!mux21&r0;
```

```
r1=mux1&mux21 # mux1&r0 # r0&mux21;

som3=mux3&mux23&r2 # mux3&!mux23&!r2 #
           !mux3&mux23&!r2 # !mux3&!mux23&r2;

r3=mux3&mux23 # mux3&r2 # r2&mux23;

end;
END;

SYM GLB  A4  1  som4-6;
sigtype [som4, r4, som6, r6] out;

equations

som4=mux4&mux24&r3 # mux4&!mux24&!r3 #
           !mux4&mux24&!r3 # !mux4&!mux24&r3;

r4=mux4 & mux24 # mux4&r3 # r3&mux24 ;

som6=mux6&mux26&r5 # mux6&!mux26&!r5 #
           !mux6&mux26&!r5 # !mux6&!mux26&r5;

r6=mux6 & mux26 # mux6&r5 # r5&mux26 ;

end;
END;

SYM GLB  B1  1  clks;
sigtype [clkCont, mux, clkACC, clkA]  out;

equations

mux=m3&!m2&m1;
clkACC = EnclkACC&!clock;
clkA=EnclkA &clock;
clkCont=Cont & !StopCont&clock;

end;
END;
```

```
SYM GLB  B0  1  RegA;
sigtype [a0,a1,a2,a3] reg  out;

equations

a0=in0
a1=in1;
a2=in2;
a3=in3;

a0.ptclk= clkA;
a1.ptclk=clkA;
a2.ptclk=clkA;
a3.ptclk=clkA;

end;
END;

SYM IOC  Y0  1  CK;
XPIN clk XCK  LOCK11;

IB11 (CK,XCK);
END;

SYM IOC  IO31  1  c0;
XPIN IO Xc0 LOCK 28;

OB11(Xc0,c0);
END;

SYM IOC  IO30  1  c1;
XPIN IO Xc1 LOCK 29;

OB11(Xc1,c1);
END;

SYM IOC  IO29  1  c2;
XPIN IO Xc2 LOCK 30;

OB11(Xc2,c2);
END;
```

```
SYM IOC  IO28  1  c3;
XPIN IO Xc3 LOCK 31;

OB11(Xc3,c3);
END;

SYM IOC  IO0  1  Clk;
XPIN IO XClock LOCK 20;

IB11(Clock,XClock);
END;

SYM IOC  IO1  1  sop;
XPIN IO Xsop LOCK 10;

IB11(sop,Xsop);
END;

SYM IOC  IO2  1  in0;
XPIN IO Xin0 LOCK 41;

IB11(in0,Xin0);
END;

SYM IOC  IO3  1  in1;
XPIN IO Xin1 LOCK 42;

IB11(in1,Xin1);
END;

SYM IOC  IO4  1  in2;
XPIN IO Xin2 LOCK 43;

IB11(in2,Xin2);
END;

SYM IOC  IO5  1  in3;
XPIN IO Xin3 LOCK 44;

IB11(in3,Xin3);
```

```
END;

SYM IOC  IO23  1  clkA;
XPIN IO XclkA LOCK 3;

OB11(XclkA,clkA);
END;

SYM IOC  IO22  1  clkACC;
XPIN IO XclkACC LOCK 4;

OB11(XclkACC,clkACC);
END;

SYM IOC  IO21  1  mux;
XPIN IO Xmux LOCK 5;

OB11(Xmux,mux);
END;

SYM IOC  IO20  1  clkCont;
XPIN IO XclkCont LOCK 6;

OB11(XclkCont,clkCont);
END;

SYM IOC  IO10  1  c0;
XPIN IO Xm0 LOCK 37;

OB11(Xm0,m0);
END;

SYM IOC  IO11  1  c1;
XPIN IO Xm1 LOCK 38;

OB11(Xm1,m1);
END;

SYM IOC  IO12  1  c2;
XPIN IO Xm2 LOCK 39;
```

```
OB11(Xm2,m2);
END;

SYM IOC  IO13  1  c3;
XPIN IO Xm3 LOCK 40;

OB11(Xm3,m3);
END;

SYM IOC  IO19  1  RstCont;
XPIN IO XRstCont LOCK 7;

OB11(XRstCont,RstCont);
END;

SYM IOC  IO18  1  Rst;
XPIN IO XRst LOCK 8;

OB11(XRst,Rst);
END;

SYM IOC  IO17  1  stopCont;
XPIN IO XstopCont LOCK 9;

OB11(XstopCont,stopCont);
END;
END;  //LDF DESIGNLDF
```

A.2 Programmazione della PLD II

```
// Wed Jul 21 12:48:42 2004
// MARIANOV.txt generated using Lattice pDS Version 2.20

LDF 1.00.00 DESIGNLDF;
DESIGN MARIANOV;

PART ispLSI1016-60LJ;

OPTION PULLUP ALL;
```

```
DECLARE
END;  //DECLARE

SYM GLB  A7  1  vis0/3;
sigtype [v0,v1,v2,v3] out;

equations

v3 =  in3&!in2&!in1;

v2 =  !in3&in2 #in2&in1;

v1 =  !in3&in1 # in3&in2&!in1;

v0 =  in0;

end;
END;

SYM GLB  A6  1  vis4;
sigtype [v4] out;

equations

v4=in3&in2#in3&in1;

end;
END;

SYM IOC  Y0  1  CK;
XPIN clk XCK  LOCK11;

IB11 (CK,XCK);
END;

SYM IOC  IO31  1  v0;
XPIN IO Xv0 LOCK 7;

OB11(Xv0,v0);
END;
```

```
SYM IOC  IO29  1  v2;
XPIN IO Xv2 LOCK 5;

OB11(Xv2,v2);
END;

SYM IOC  IO28  1  v3;
XPIN IO Xv3 LOCK 4;

OB11(Xv3,v3);
END;

SYM IOC  IO2  1  in0;
XPIN IO Xin0 LOCK 21;

IB11(in0,Xin0);
END;

SYM IOC  IO3  1  in1;
XPIN IO Xin1 LOCK 20;

IB11(in1,Xin1);
END;

SYM IOC  IO4  1  in2;
XPIN IO Xin2 LOCK 19;

IB11(in2,Xin2);
END;

SYM IOC  IO5  1  in3;
XPIN IO Xin3 LOCK 18;

IB11(in3,Xin3);
END;

SYM IOC  IO30  1  v1;
XPIN IO Xv1 LOCK 6;

OB11(Xv1,v1);
END;
```

```
SYM IOC  IO27  1  v4;
XPIN IO Xv4 LOCK 3;

OB11(Xv4,v4);
END;
END;  //LDF DESIGNLDF
```

Bibliografia

[1] Daniels, *Digital Design from Zero to One* - 616 pagg.

[2] Herbert Taub, Donald Schilling, *Elettronica Integrata Digitale* - 721 pagg.

[3] Datasheet IspLSI 1016 60 LJ

[4] Dispense del docente

[5] Appunti delle lezioni