



Progetto Reti Logiche

Specifiche.....	3
Scopo del progetto.....	3
Tecnologia.....	3
Il NAND.....	3
Software utilizzati.....	3
Specifiche funzionali.....	4
Rappresentazione dei valori.....	4
Dominio e codominio.....	4
Specifiche parametriche.....	5
Impostazione del progetto a livello RT.....	5
Data Flow Graph.....	5
Rete a singolo stadio.....	5
Resource Sharing.....	5
Pipelining.....	6
Risorse.....	6
Progetto delle risorse a livello gate.....	8
Porte logiche elementari.....	8
NAND.....	8
NOT.....	9
AND.....	9
OR.....	10
EXOR.....	11
Componenti e Macro.....	11
Half Adder.....	11
Full Adder.....	12
Potenza.....	13
Moltiplicazione.....	13
Sommatore.....	15
Multiplexer.....	15
Half Subtractor.....	16
Full Subtractor.....	17
Half Divider.....	18
Divisore.....	19
Data Path.....	21
Control Unit.....	23
Specifiche.....	23
Implementazione della macchina di Moore.....	23
Simulazione e analisi del progetto.....	24
Verifica funzionale.....	24
Valutazione di prestazioni e complessità.....	26
Stima di complessità circuitale e prestazioni.....	26



Progetto d'esame di Reti Logiche A.A. 2020/2021

Luca Martinangeli Matricola nr: 293166

Specifiche

Scopo del progetto

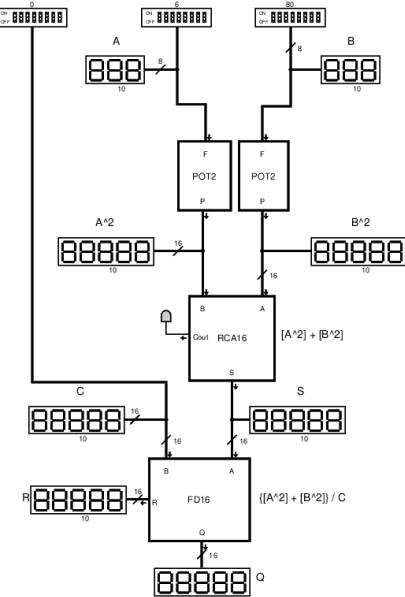
Poniamo come obiettivo del progetto la risoluzione della seguente espressione intera:

$$f = \frac{(A^2 + B^2)}{C}$$



Tecnologia

La logica utilizzata per la costruzione delle porte elementari è la FCMOS (**Fully Complementary MOS**), mentre per la descrizione di alto livello del circuito e del suo funzionamento verrà usata la logica RTL (**Register Transfer Level**).



Il NAND

Tutte le macro ed i moduli della rete sono composti unicamente dalla porta logica elementare NAND (**Not AND**) dimostrando materialmente la completezza funzionale della stessa. Insieme alla porta NOR (**Not OR**) viene definita una porta logica universale proprio per la sua capacità di rappresentare tutte le altre porte logiche. La sua economicità e le sue ottime prestazioni in termini di dissipazione elettrica l'hanno resa una pietra miliare nella rivoluzione tecnologica e nello sviluppo delle memorie utilizzate al giorno d'oggi. È possibile realizzare una rete combinatoria identica usando solo la porta NOR che, seppur logicamente equivalente, avrà prestazioni diverse all'atto dell'implementazione elettronica.



Software utilizzati

L'ambiente di sviluppo è Linux (Pop-Os 20.04), il software usato per la costruzione della rete e le simulazioni è *TkGate v2.1* disponibile nelle repository di Ubuntu. Per la scrittura della relazione è stato utilizzato *LibreOffice* (+ *LATEX*), mentre Draw.io per i disegni.



Specifiche funzionali

La rete realizzata può essere suddivisa in 3 “fasi”. Gli operandi sono numeri naturali ed appartengono tutti ad \mathbb{N} :

1° - Potenza	2° - Somma	3° - Divisione
$A \rightarrow A^2$ $B \rightarrow B^2$	$A^2 + B^2 \rightarrow S$	$\frac{S}{C} = Q, R$

1. Viene eseguito il quadrato di A e B ottenendo A^2, B^2 ;
2. Viene eseguita la loro somma e calcolato S ;
3. Il risultato viene diviso per C trovando Q , e l’eventuale resto R .

Rappresentazione dei valori

Esistono dei limiti di rappresentazione dovuti al **numero di bit** utilizzati per l’input/output che, se non rispettati, possono compromettere la correttezza dell’operazione:

- A e B hanno a disposizione 8 bit, quindi possono assumere $2^8 = 256$ valori;
- C ha a disposizione 16 bit, quindi può assumere $2^{16} = 65536$ valori;
- Anche A^2, B^2, S, Q, R hanno a disposizione 16 bit dunque possono assumere gli stessi valori di C .

Dominio e codominio

Di seguito, analizziamo le funzioni coinvolte in ogni fase ed i rispettivi domini/codomini, formalizzando gli intervalli in cui operano.

1° - Potenza
$Dom(f_{pot}) = \{\forall A, B \in \mathbb{N} \mid A, B \in [0, 255]\}$ $Cod(f_{pot}) = \{\forall A^2, B^2 \in \mathbb{N} \mid A^2, B^2 \in [0, 65535]\}$ $f_{pot} : \mathbb{N} \mapsto \mathbb{N}$
2° - Somma
$Dom(f_{sum}) = \{\forall A^2, B^2 \in \mathbb{N} \mid A^2, B^2 \in [0, 65535]\}$ $f_{sum} : \mathbb{N} \mapsto \mathbb{N}$ $Cod(f_{sum}) = \{\forall S \in \mathbb{N} \mid S \in [0, 65535]\}$
3° - Divisione
$Dom(f_{div}) = \{\forall S, C \in \mathbb{N} \mid S, C \in [0, 65535]\}$ $f_{div} : \mathbb{N} \mapsto \mathbb{N}$ $Cod(f_{div}) = \{\forall Q, R \in \mathbb{N} \mid Q, R \in [0, 65535]\}$



Ne consegue che S non può superare il valore di 65535. Si noti inoltre che qualsiasi divisione per 0 produrrà risultati incoerenti. L’errore in questione verrà affrontato più avanti.



Specifiche parametrica

Da specifiche, occorre analizzare il mutare della complessità dei vari componenti che vengono coinvolti, associandoli all'operazione che implementano. Avendo infatti usato solamente porte NAND è possibile calcolare con una buona accuratezza l'area da loro occupata; inoltre seguendo il **modello di ritardo unitario**, è possibile analizzare la latenza complessiva della rete. Rimane possibile introdurre resource sharing/pipelining facendo uso di registri.

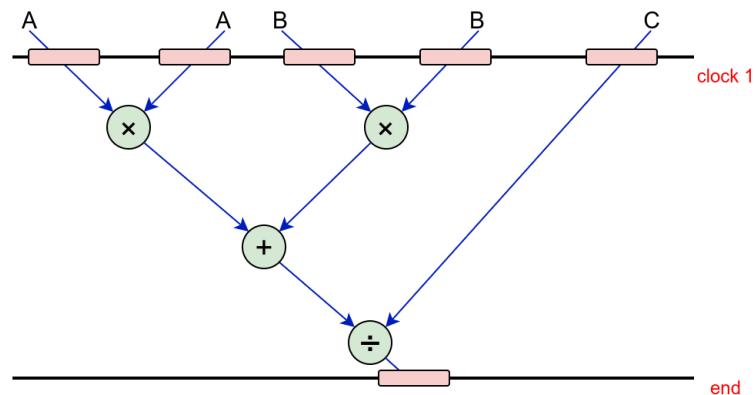
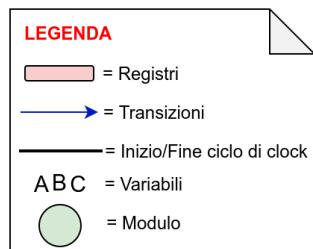
Impostazione del progetto a livello RT

Data Flow Graph

Rete a singolo stadio

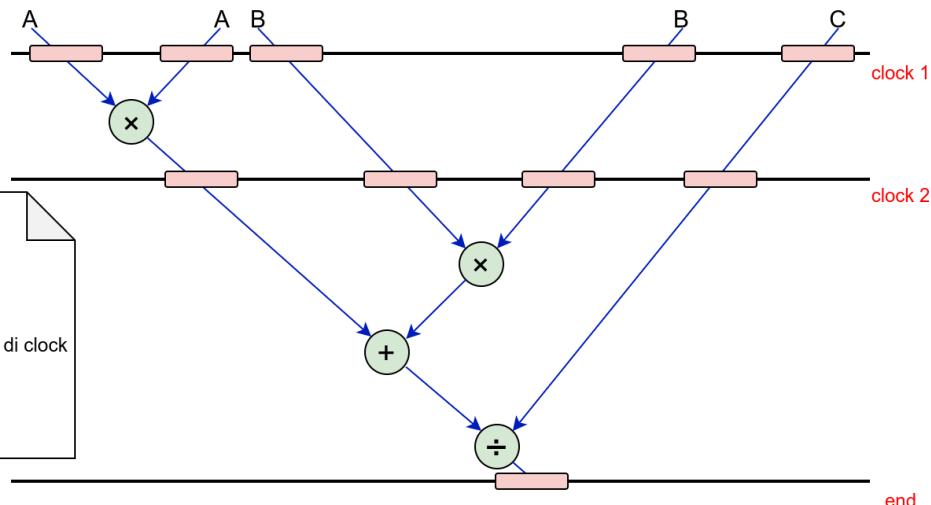
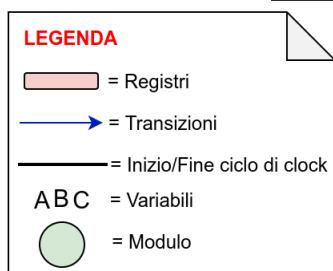


Un'ALU equipaggiata con 2 moltiplicatori, 1 sommatore ed 1 divisore sarà in grado di gestire l'elaborazione in un singolo ciclo di clock. La rete realizzata in questo progetto rientra in questa casistica.



Resource Sharing

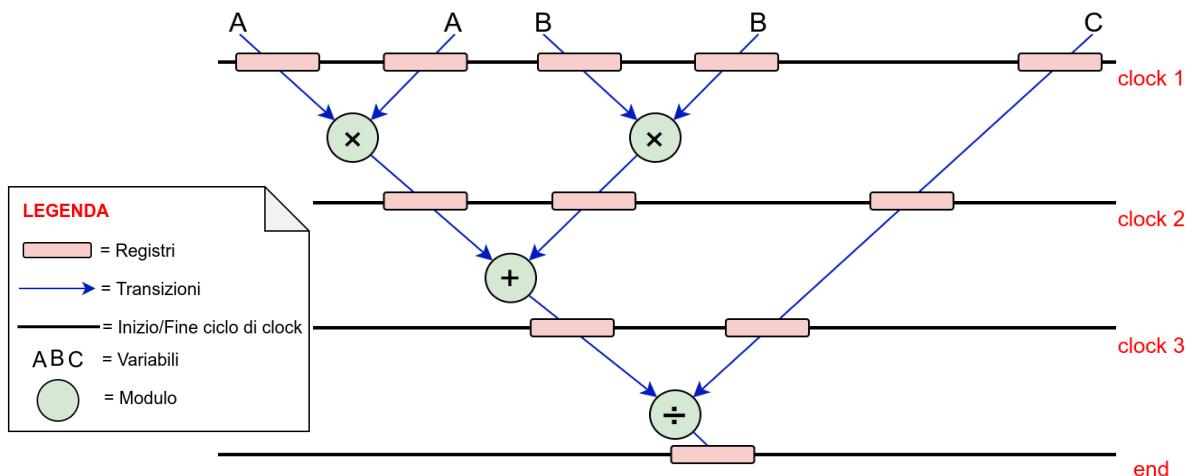
È possibile prevedere l'uso del resource sharing riutilizzando un moltiplicatore. In tal caso, la rete diventa a 2 stadi ed impiegherà 2 cicli di clock per completare l'elaborazione.





Pipelining

Il pipelining può essere introdotto aggiungendo delle batterie di registri sincronizzati dal ciclo di clock. All'inizio di ogni ciclo, tali registri vengono “svuotati” per accogliere i nuovi valori; tale scenario impiega 3 cicli di clock per completare l'elaborazione.



Risorse

Potenza			
Nome Componente	Quantità	Bit in	Bit out
Multiplier	1	16	16
Moltiplicatore			
Nome Componente	Quantità	Bit in	Bit out
Half Adder	8	2	2
Full Adder	48	3	2
AND	64	2	1
Half Adder			
Nome Componente	Quantità	Bit in	Bit out
EXOR	1	2	1
AND	1	2	1
Full Adder			
Nome Componente	Quantità	Bit in	Bit out
HA	2	2	2
OR	1	2	1
Sommatore			
Nome Componente	Quantità	Bit in	Bit out
Ripple Carry Adder	1	32	16



Ripple Carry Adder			
Nome Componente	Quantità	Bit in	Bit out
Half Adder	1	2	2
Full Adder	15	3	2
Divisore			
Nome Componente	Quantità	Bit in	Bit out
Full Divider	1	32	32
Full Divider			
Nome Componente	Quantità	Bit in	Bit out
Half Divider	256	3	4
NOT	16	1	1
Half Divider			
Nome Componente	Quantità	Bit in	Bit out
Full Subtractor	1	3	2
AND	2*	2*	1*
MUX	1	2	1
*gli AND sono necessari dato che TkGate non permette di fare un collegamento diretto input-output			
Full Subtractor			
Nome Componente	Quantità	Bit in	Bit out
Half Subtractor	2	2	1
OR	1	2	1
Half Subtractor			
Nome Componente	Quantità	Bit in	Bit out
EXOR	1	2	1
NOT	2	1	1
NAND	1	2	1
Exor			
Nome Componente	Quantità	Bit in	Bit out
NAND	4	2	1



Progetto delle risorse a livello gate

Nella progettazione dei moduli realizzati verranno analizzate le seguenti metriche:

- **Tempo di propagazione (T_p)**: Data un'uscita, è il tempo di percorrenza più lungo in un segnale per raggiungerla;
- **Tempo di contaminazione (T_p)**: Data un'uscita, è il tempo di percorrenza più breve di un segnale nel raggiungerla;
- **Modello di ritardo unitario**: il ritardo di ogni porta NAND attraversata corrisponde ad 1;
- **Area (A)**: il numero di porte NAND utilizzate per la costruzione di un modulo;

Ove possibile, verrà proposta la tabella di verità associata al componente. Si noti che le porte logiche elementari derivate sono precedute dal prefisso “Nand”, ad indicare la natura delle stesse.

Porte logiche elementari

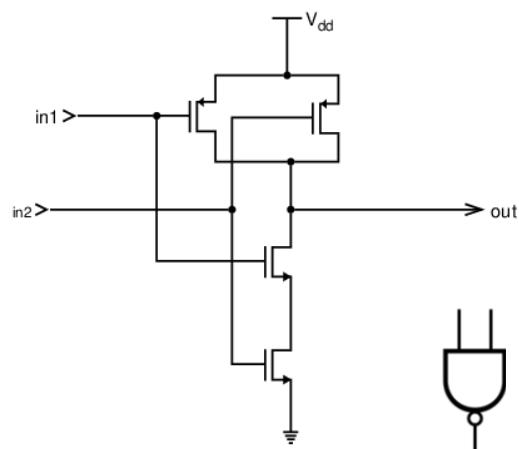
NAND

Il NAND è una porta logica complementare all'AND: $\text{NAND} = (ab)'$.

In logica FCMOS, viene realizzata collegando ogni terminale d'ingresso ad un **pMOS** (MOSFET normalmente aperto) ed un **nMOS** (MOSFET normalmente chiuso), a loro volta collegati all'uscita.

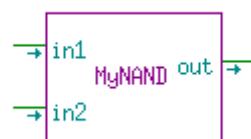
 In una rete come questa possiamo notare la presenza di 2 “sottoreti”: la superiore di pull-up e l’inferiore di pull-down.

Nel NAND, la rete di pull-up è composta da 2 pMOS collegati **in parallelo**, mentre la pull-down da 2 nMOS collegati **in serie**.



La complementarietà dei transistor in questione consente di avere una potenza dissipata estremamente bassa, quindi un'ottima efficienza. Il comportamento delle uscite è descritto nella tabella di verità che segue.

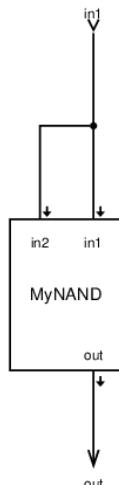
Tabella della verità del NAND			
a	b	ab	$(ab)'$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0



Prestazioni NAND
$T_c = 1$
$T_p = 1$
A = 1



NOT



Il NOT descrive la negazione logica (o “complemento”) e può essere indicato con varie notazioni: $\text{NOT} = \neg a / \bar{a} / (a)'$. In elettronica è chiamato “inverter” ed è ricavabile “cortocircuitando” gli input di un NAND ottenendo quella che in algebra Booleana è definita proprietà di idempotenza: $(a)' \rightarrow (aa)'$.

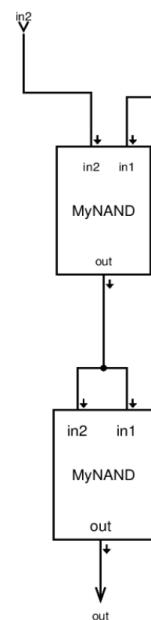


Tabella di verità del NOT		
a	$(a)'$	$(aa)'$
1	0	0
0	1	1



Prestazioni NOT
$T_c = T_{c_{nand}} = 1$
$T_p = T_{p_{nand}} = 1$
$A = A_{nand} = 1$

AND



L'AND corrisponde alla congiunzione logica: $\text{AND} = a \wedge b$, può essere indicata anche con il prodotto aritmetico: $\text{AND} = ab$. Possiamo ricostruire un AND utilizzando 2 NAND, tramite una doppia negazione otteniamo la sua conversione: $ab \rightarrow ((ab)')'$, verificabile anche dalla tabella di verità.



Tabella di verità dell'AND

a	b	ab	$((ab)')'$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0

Prestazioni AND

$T_c = 2 \cdot T_{c_{nand}} = 2 \cdot 1 = 2$
$T_p = 2 \cdot T_{p_{nand}} = 2 \cdot 1 = 2$
$A = 2 \cdot A_{nand} = 2 \cdot 1 = 2$



OR

L'OR corrisponde alla disgiunzione logica: $\text{OR} = a \vee b$, indicato anche come la somma aritmetica $\text{OR} = a + b$. Per convertirlo in logica NAND, basta applicare in successione il secondo teorema di De Morgan e successivamente l'idempotenza:

$$(a + b) \rightarrow (a' \cdot b')' \rightarrow ((aa)'(bb)')'.$$

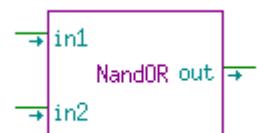
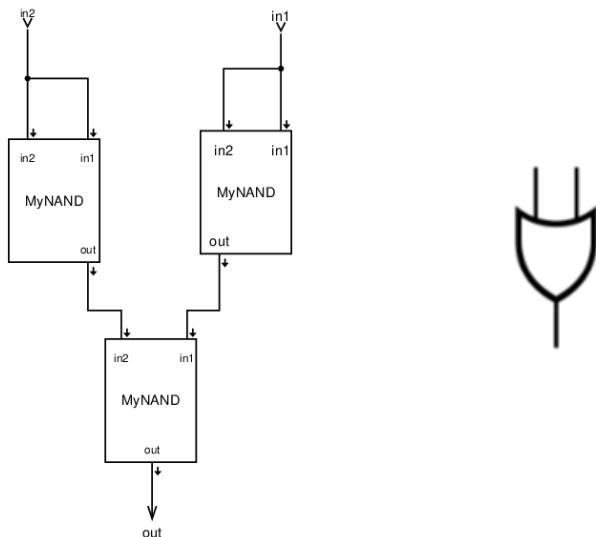


Tabella di verità dell'OR

a	b	$a+b$	$((aa)'(bb)')'$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

Prestazioni OR

$$T_c = 2 \cdot T_{c_{nand}} = 2 \cdot 1 = 2$$

$$T_p = 2 \cdot T_{p_{nand}} = 2 \cdot 1 = 2$$

$$A = 3 \cdot A_{nand} = 3 \cdot 1 = 3$$



EXOR

L'EXOR o "Exclusive OR" è un operatore logico che risulta positivo solamente quando i suoi operandi sono diversi. Ha 2 notazioni: $a \oplus b$ / $a \vee b$. Viene ottenuto in logica NAND, applicando rispettivamente il primo ed il secondo teorema di De Morgan:

$$a \oplus b \rightarrow ab' + a'b \rightarrow ((a' + b)(a + b'))' \rightarrow ((ab')'(a'b)')'$$

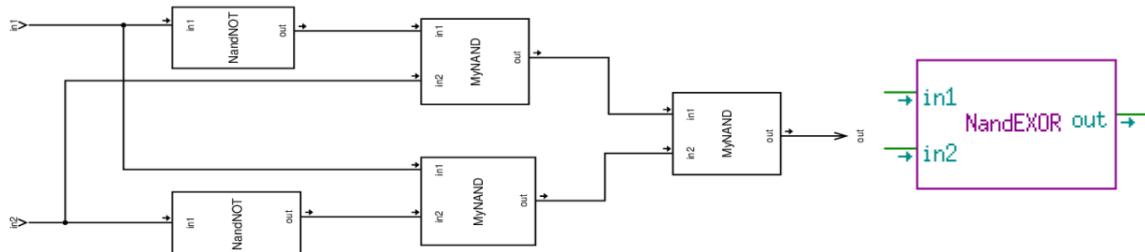


Tabella di verità dell'EXOR			
a	b	$a \oplus b$	$((ab')'(a'b)')'$
1	1	0	0
0	1	1	1
1	0	1	1
0	0	0	0

Prestazioni EXOR	
$T_c = 2 \cdot T_{c_{nand}} = 2 \cdot 1 = 2$	
$T_p = T_{p_{not}} + 2 \cdot T_{p_{nand}} = 1 + (2 \cdot 1) = 3$	
$A = 2 \cdot A_{not} + 3 \cdot A_{nand} = (2 \cdot 1) + (3 \cdot 1) = 5$	

Componenti e Macro

Half Adder

L'**Half Adder (HA)** è un modulo che permette di calcolare la somma S di 2 bit: a e b propagando l'eventuale riporto $Cout$. Implementa 2 funzioni logiche, un Exor per la somma e un AND per il riporto:

$$S = a \oplus b \quad Cout = ab$$

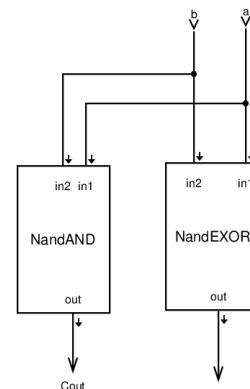
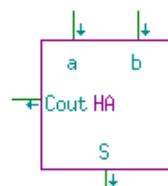


Tabella di verità dell'Half Adder			
a	b	S	Cout
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1



Prestazioni HA	
$T_{c_{HA-S}} = T_{c_{exor}} = 2$	
$T_{c_{HA-Cout}} = T_{c_{and}} = 2$	
$T_{p_{HA-S}} = T_{p_{exor}} = 3$	
$T_{p_{HA-Cout}} = T_{p_{and}} = 2$	
$A_{HA} = A_{and} + A_{exor} = 2 + 5 = 7$	



Full Adder

Il **Full Adder (FA)** permette di sommare 2 bit e di tenere in considerazione il riporto anche in ingresso, propagandolo in uscita se necessario, troviamo 2 HA; il primo si occupa della somma degli addendi ed il secondo li somma ad un'eventuale resto in ingresso. L'OR propaga il resto in caso fosse presente in uno dei due HA. Abbiamo:

$$S = (a \oplus b) \oplus Cin$$

$$Cout = ab + (a \oplus b)Cin$$

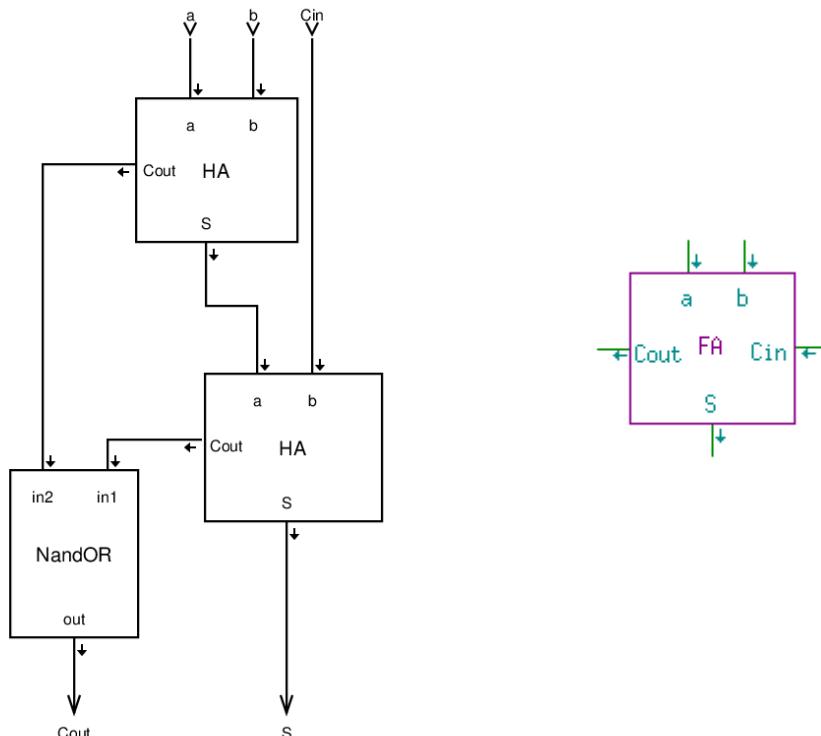


Tabella di verità del Full Adder				
a	b	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	1
1	1	1	0	1

Prestazioni FA

$$Tc_{FA-S} = Tc_{HA-S} = 2$$

$$Tc_{FA-Cout} = Tc_{HA-Cout} + Tc_{or} = 2 + 2 = 4$$

$$Tp_{FA-S} = 2 \cdot Tp_{HA-S} = 2 \cdot 3 = 6$$

$$Tp_{FA-Cout} = Tp_S + Tp_{HA-Cout} = 3 + 2 = 5$$

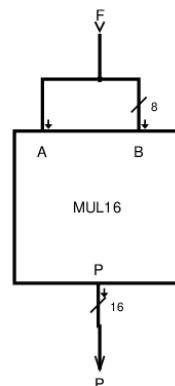
$$A_{FA} = 2 \cdot A_{HA} + A_{or} = 2 \cdot 7 + 3 = 17$$





Potenza

Il modulo potenza è costituito da un moltiplicatore con gli ingressi in comune. In uscita si otterrà il quadrato del valore d'ingresso. Le prestazioni e le metriche saranno quindi identiche a quelle del moltiplicatore, tale macro semplifica la costruzione della rete a livello più alto.

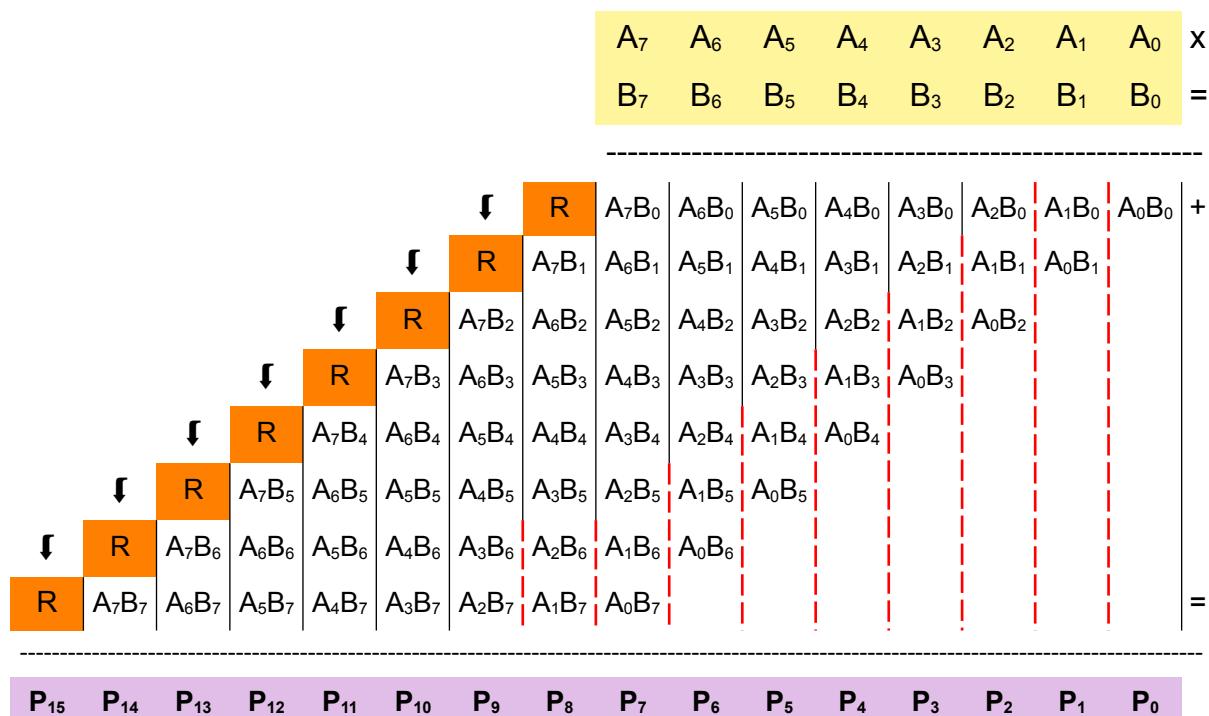


Moltiplicatore

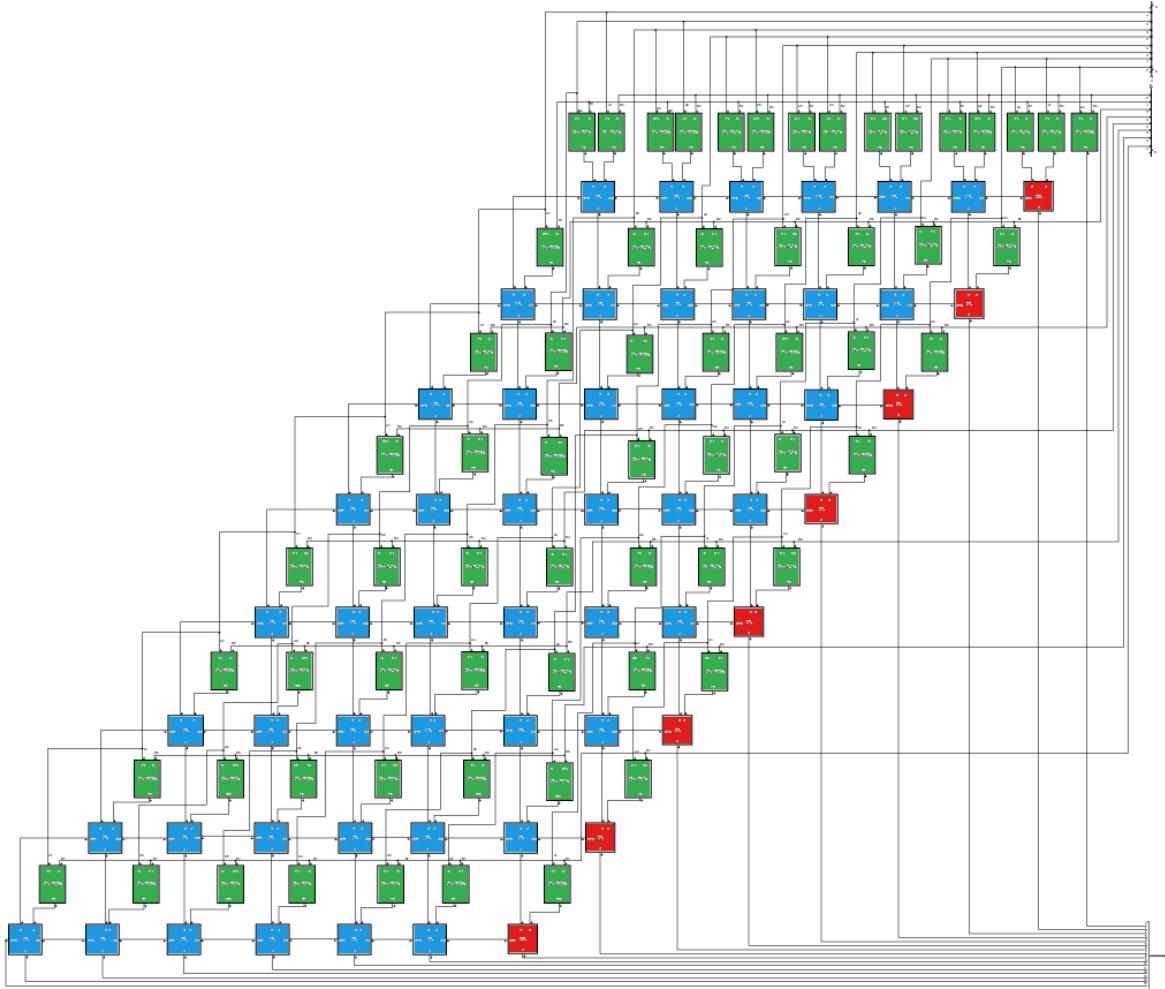
P

Per calcolare il prodotto di 2 numeri ad 8 cifre occorre rispettare il **sistema di numerazione posizionale**: bisogna eseguire una serie di moltiplicazioni e somme in un'ordine ben specifico, scandito dalle colonne sotto riportate. Di seguito viene rappresentato il procedimento considerando i bit dei fattori in ingresso.

Fondamentale sarà l'uso di AND per il prodotto dei singoli bit e di HA/FA per gestire le somme e la propagazione dei riporti. Il primo bit calcolato sarà dunque P_0 , l'ultimo sarà P_{15} , corrispondente al resto residuo delle somme interne al componente.



Le colonne evidenziate in rosso fanno uso di un Half Adder per eseguire l'ultima somma, non avendo riporto in ingresso, abbiamo un leggero risparmio d'area. Sevono 16 bit per rappresentare il risultato.



Nell'immagine sopra è possibile osservare i componenti: sono stati evidenziati in **verde** i NAND, in **rosso** gli half adder ed in **blu scuro** i full adder.

Il primo bit del risultato (P_0) viene calcolato immediatamente, trovando:

$$Tc_{MUL} = 1 \cdot Tc_{and} = 2$$

Per ottenere il settimo bit, dobbiamo attendere la corretta propagazione di tutti i segnali fino all'ultimo FA in basso a sinistra, attraversando tutte le "colonne" e le "righe"; inoltre dalle prestazioni del FA notiamo che il calcolo della somma richiede più tempo del resto, dunque:

$$\begin{aligned} Tp_{MUL} &= Tp_{and} + Tp_{HA-Cout} + 12 \cdot Tp_{FA-Cout} + 8 \cdot Tp_{FA-S} = \\ &= 2 + 2 + (12 \cdot 5) + (8 \cdot 6) = 112 \end{aligned}$$

L'area: $A_{MUL} = 64 \cdot A_{and} + 8 \cdot A_{HA} + 48 \cdot A_{FA} = (64 \cdot 2) + (8 \cdot 7) + (48 \cdot 17) = 1000$

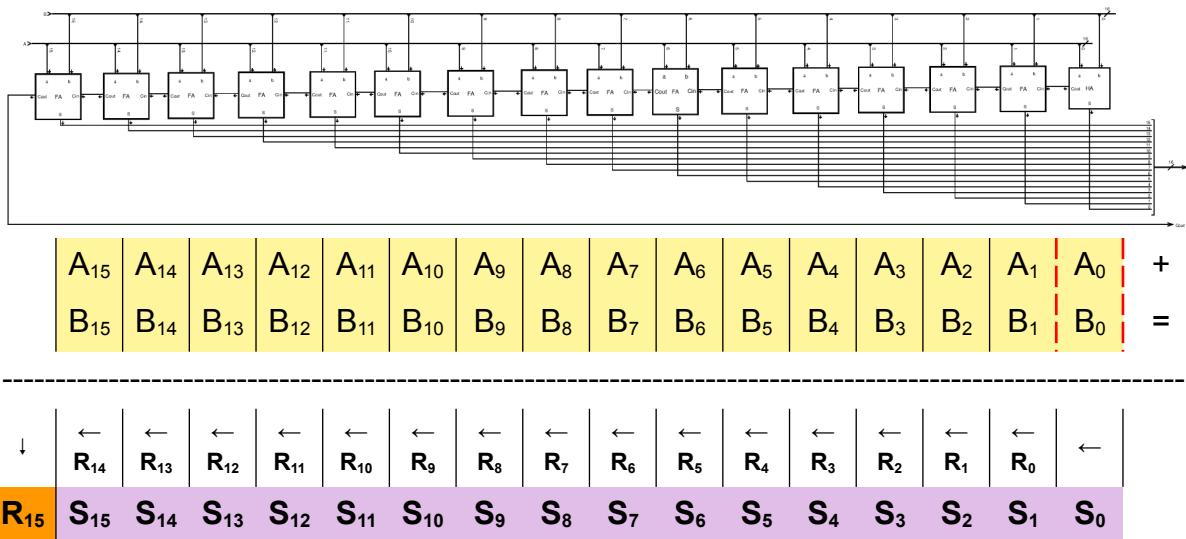
In caso di utilizzo di soli FA, ci sarebbe stato un aumento dell'area di 63 porte, equivalente ad un aumento di circa 6% dell'area.

**Non viene proposta la tabella di verità di questo componente date le sue dimensioni considerevoli.



Sommatore

Il sommatore è composto da un singolo Ripple Carry Adder a 16 bit. Viene ottenuto concatenando 15 full adder ad un half adder, ognuno si occupa della somma di 2 bit e viene segnalato da $Cout$ quando il riporto raggiunge l'ultimo adder.



Il primo bit viene calcolato immediatamente, mentre per $Cout$ occorre attendere la propagazione:

$$Tc_{SUM-S} = Tc_{HA-S} = 2$$

$$Tc_{SUM-Cout} = Tc_{HA-Cout} + 15 \cdot Tc_{FA-Cout} = 2 + (15 \cdot 4) = 62$$

Il tempo di propagazione è molto simile per l'ultimo bit e per il riporto:

$$Tp_{SUM-S} = Tp_{HA-Cout} + 14 \cdot Tp_{FA-Cout} + Tp_{FA-S} = 2 + (14 \cdot 5) + 2 = 74$$

$$Tp_{SUM-Cout} = Tp_{HA-Cout} + 15 \cdot Tp_{FA-Cout} = 2 + (15 \cdot 5) = 77$$

L'area: $A_{SUM} = A_{HA} + 15 \cdot A_{FA} = 7 + (15 \cdot 17) = 262$

Multiplexer

Il multiplexer (**MUX**) restituisce un valore di uscita Y in funzione ad un valore di controllo Cin . Se $Cin = 0$ in uscita troveremo $Y = A$, se $Cin = 1$, troveremo $Y = B$. Tramite De Morgan otteniamo:

$$Y = A(Cin)' + B(Cin) \rightarrow ((A(CinCin)')'(BCin)')'$$

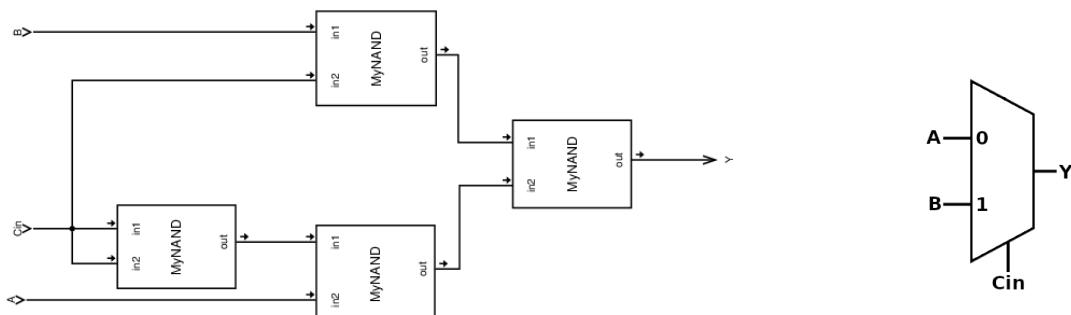




Tabella di verità del MUX

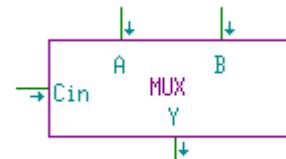
Cin	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Prestazioni MUX

$$Tc_{MUX} = 2 \cdot Tc_{nand} = 2 \cdot 1 = 2$$

$$Tp_{MUX} = 3 \cdot Tp_{nand} = 3 \cdot 1 = 3$$

$$A_{MUX} = 4 \cdot A_{nand} = 4 \cdot 1 = 4$$



Half Subtractor

Come l'Half Adder, l'Half Subtractor(HS) non prevede un prestito in ingresso. Permette di stabilire se 2 bit possono essere sottratti e se sia necessario richiedere un prestito per effettuare la sottrazione, notiamo che l'Exor può descrivere anche una differenza, cambiando il "significato" del resto (che diventa la necessità di prestito), otteniamo:

- Se $A > B$ la differenza avviene senza bisogno del prestito;
- Se $A < B$ viene segnalata la necessità di un prestito;
- Se $A = B$ il risultato è 0;

$$D = A \oplus B \quad Pr = ((A'B)')$$

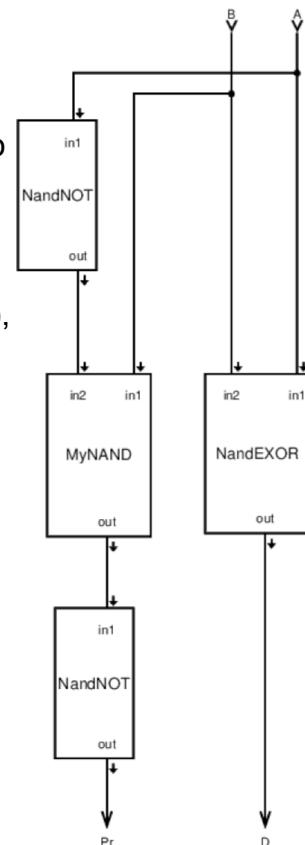
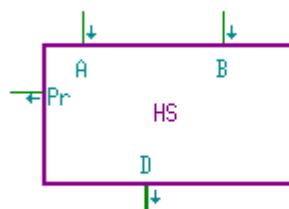


Tabella della Verità dell'HS

A	B	D	Pr
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Prestazioni HS

$$Tc_{HS-D} = Tc_{exor} = 2$$

$$Tc_{HS-Pr} = Tc_{nand} + Tc_{not} = 2$$

$$Tp_{HS-D} = Tp_{exor} = 3$$

$$Tp_{HS-Pr} = 2 \cdot Tp_{not} + Tp_{nand} = (2 \cdot 1) + 1 = 3$$

$$A_{HS} = 2 \cdot A_{not} + A_{exor} + A_{nand} = (2 \cdot 1) + 5 + 1 = 8$$



Full Subtractor

Il Full Sub (FS) come il Full Adder permette di tener conto di un prestito iniziale $PrIn$ in una sottrazione a 2 bit e, se necessario, di richiedere un prestito. Dalla tabella della verità possiamo ricavare:

- Se $A > B$ avviene la sottrazione e nel caso sia presente un $PrIn$, viene propagato;
- Se $A < B$ viene utilizzato il prestito se presente o viene segnalata la sua necessità se assente;
- Se $A = B$ il risultato è 0 e se è presente un prestito iniziale, viene propagato su $PrOut$.

$$D = (A \oplus B) \oplus PrIn \quad \text{PrOut} = Pr_1 + Pr_2$$

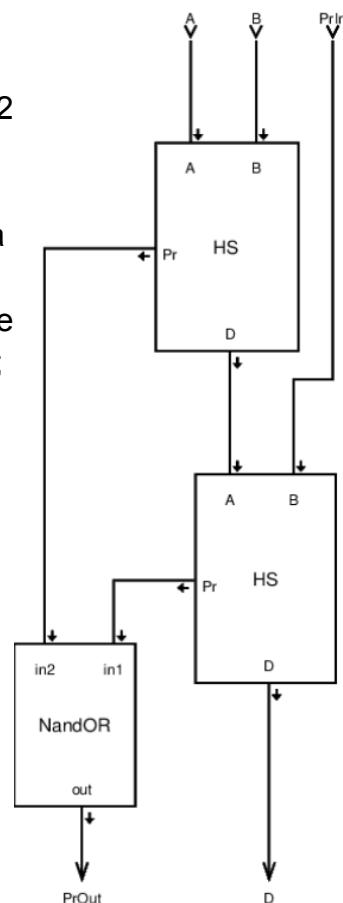
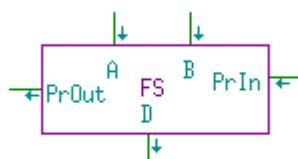


Tabella di verità FS				
A	B	PrIn	D	PrOut
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Prestazioni FS
$Tc_{FS-D} = Tc_{HS-D} = 2$
$Tc_{FS-PrOut} = Tc_{HS-Pr} + Tc_{or} = 2 + 1 = 3$
$Tp_{FS-D} = 2 \cdot Tp_{HS-D} = 3 \cdot 2 = 6$
$Tp_{FS-PrOut} = Tp_{HS-D} + Tp_{HS-Pr} + Tp_{or} = 3 + 3 + 2 = 8$
$A_{FS} = 2 \cdot A_{HS} + A_{or} = (2 \cdot 8) + 3 = 19$



Half Divider

L'Half Div (HD) si occupa della divisione:

- $PrIn$ porta in ingresso al **FS** un'eventuale riporto;
- **A** e **B** vengono portati all'interno del **FS**: se $A > B$ troviamo il risultato su **D** altrimenti viene segnalata la necessità di un prestito su $PrOut$;
- Il risultato del **FS** viene portato agli ingressi del **MUX**, che lo compara al valore iniziale di **A**: $CrIn$ è il bit di controllo del **MUX**;
 - Se $CrIn = 1$ allora $A > B$, dunque troviamo il risultato su **Q**;
 - Se $CrIn = 0$ significa che $A < B$, quindi su **Q** troveremo **A**.

Più avanti vedremo che $CrIn$ risulterà essere $PrOut$ negato.

*l'AND presente su **BOut** e **CrOut** è necessario per esigenze di implementazione con TkGate, che non permette il collegamento diretto tra input e output.

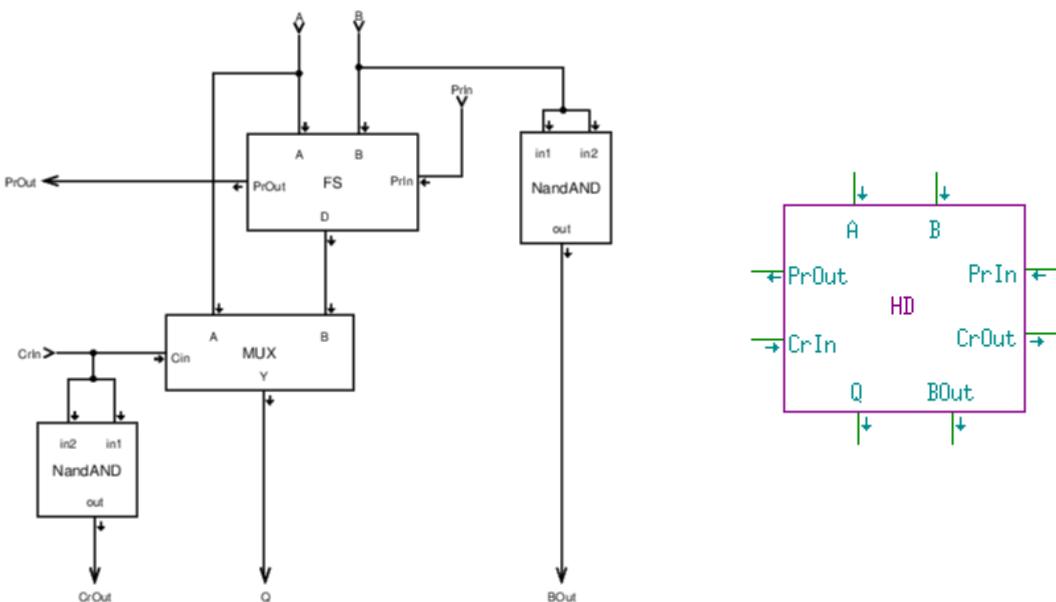


Tabella di verità HD						Prestazioni HD		
A	B	PrIn	CrIn	Q	PrOut			
0	0	0	1	0	0	$Tc_{HD-Q} = Tc_{MUX} = 2$	$Tc_{HD-BOut^*} = Tc_{and} = 2$	$Tc_{HD-PrOut} = Tc_{FS-PrOut} = 3$
0	0	1	0	0	1	$Tc_{HD-CrOut^*} = Tc_{and} = 2$	$Tp_{HD-Q} = Tp_{FS-D} + Tp_{MUX} = 6 + 3 = 9$	$Tp_{HD-BOut^*} = Tp_{and} = 2$
0	1	0	0	0	1	$Tp_{HD-PrOut} = Tp_{FS-PrOut} = 8$	$Tp_{HD-CrOut^*} = Tp_{and} = 2$	$A_{HD} = A_{FS} + A_{MUX} + 2 \cdot A_{and} = 19 + 4 + (2 \cdot 2) = 27$
0	1	1	0	0	1			
1	0	0	1	1	0			
1	0	1	1	0	0			
1	1	0	1	0	0			
1	1	1	0	1	1			

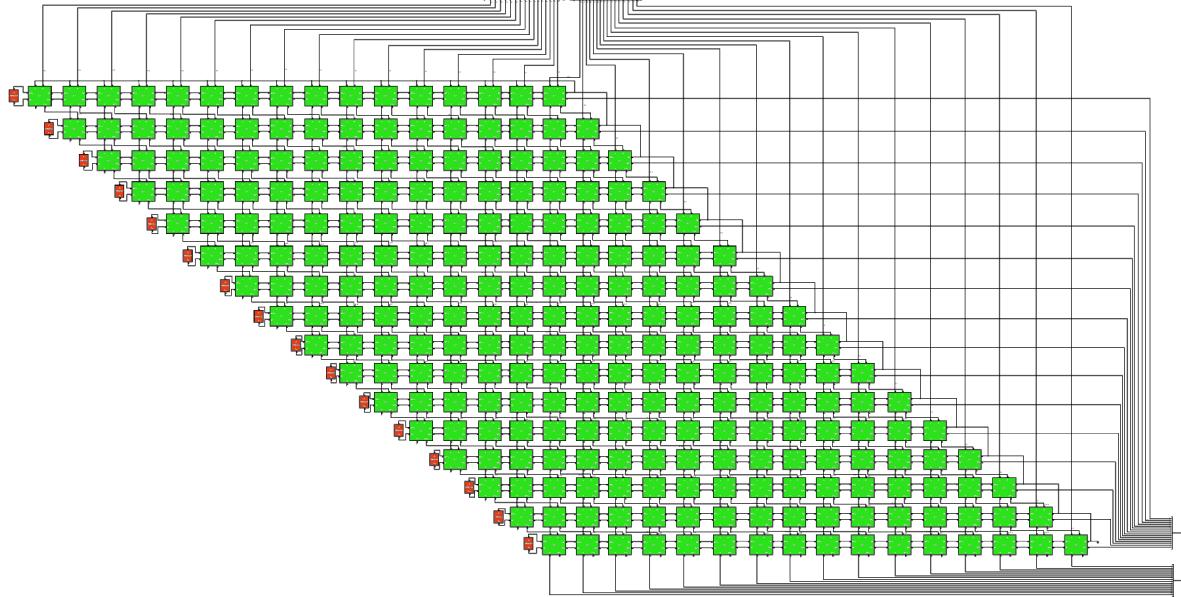


Divisore

Il divisore compie la “**divisione euclidea**” o **divisione con resto**, si vuole controllare quante volte il divisore viene contenuto nel dividendo (dando vita al **quoziente**) e con che resto. L’operazione avviene reiterando ciclicamente una sottrazione e “monitorando” tramite i **MUX** degli **HD** il risultato:

- Il dividendo viene scomposto a partire dal bit più significativo, andando a costituire il primo bit di ogni minuendo della sottrazione;
- La prima sottrazione ha i primi 15 bit inizializzati a 0;
- Il divisore diventa il sottraendo;
- Avvenuta la sottrazione, il *CrOut* del primo **HD** conterrà il valore di **Q₁₅**: comunica se è stato possibile sottrarre i due numeri oppure no;
- I restanti bit contenenti il risultato della differenza (**D₀**, [...], **D₁₅**) “scorrono” in diagonale:
 - Da notare che nella sottrazione successiva, **D₁₅** viene “perduto”, ma non risulta un problema poiché non contiene informazioni importanti ai fini della divisione o del resto
- L’algoritmo procede in questa maniera fino a che non sono popolati tutti i bit del quoziente;
- Il resto corrisponderà al valore dell’ultima differenza.

A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	:
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	=
<hr/>																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	A ₁₅	-
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	= Q ₁₅
<hr/>																
D ₁₅	D ₁₄	D ₁₃	D ₁₂	D ₁₁	D ₁₀	D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	
D ₁₄	D ₁₃	D ₁₂	D ₁₁	D ₁₀	D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	A ₁₄	-
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	= Q ₁₄
<hr/>																
⋮																⋮
D ₁₄	D ₁₃	D ₁₂	D ₁₁	D ₁₀	D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	A ₀	-
B ₁₅	B ₁₄	B ₁₃	B ₁₂	B ₁₁	B ₁₀	B ₉	B ₈	B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	= Q ₀
<hr/>																
R ₁₅	R ₁₄	R ₁₃	R ₁₂	R ₁₁	R ₁₀	R ₉	R ₈	R ₇	R ₆	R ₅	R ₄	R ₃	R ₂	R ₁	R ₀	



Nell'immagine sopra troviamo la rete che compone il divisore. In verde troviamo gli HD ed i collegamenti rispecchiano quanto illustrato nella tabella che descrive l'algoritmo seguito per il calcolo, in arancione troviamo i NOT che invertono i *PrOut*, in alto possiamo notare il dividendo ed il divisore, a destra il quoziente ed il resto (il più basso tra i due). Sia gli input che gli output sono a 16 bit.

Per il tempo di contaminazione si prende in considerazione la prima riga di **HD**:

$$\begin{aligned} Tc_{DIV-Q} &= 16 \cdot Tc_{HD-PrOut} + Tc_{not} (+16 \cdot Tc_{HD-CrOut})^* = \\ &= 8 \cdot 16 + 2 (+16 \cdot 2)^* = 130 (162)^* \end{aligned}$$

Il primo bit del resto dipende dal quoziente del primo **HD**:

$$Tc_{DIV-R} = Tc_{HD-Q} = 2$$

Per il tempo di propagazione, prendiamo in considerazione l'ultimo bit del quoziente:

$$\begin{aligned} Tp_{DIV-Q} &= 16 \cdot [(16 \cdot Tp_{HD-PrOut}) + Tp_{not} (+Tp_{HD-CrOut})^*] + 15 \cdot Tp_{HD-Q} = \\ &= 16 \cdot [(16 \cdot 8) + 1 + (2)^*] + 15 \cdot 9 = 2199 (2231)^* \end{aligned}$$

Anche nel resto prendiamo in considerazione l'ultimo bit:

$$\begin{aligned} Tp_R &= 16 \cdot [(16 \cdot Tp_{HD-PrOut}) + Tp_{not} (+Tp_{HD-CrOut})^*] + 16 \cdot Tp_{HD-Q} = \\ &= 16 \cdot [(16 \cdot 8) + 1 + (2)^*] + 16 \cdot 9 = 2208 (2240)^* \end{aligned}$$

$$A_{DIV} = 256 \cdot A_{HD} + 16 \cdot A_{not} = (256 \cdot 27) + (16 \cdot 1) = 6928$$

* i valori tra parentesi considerano il ritardo introdotto da CrOut che è presente per esigenze di funzionamento con TkGate.

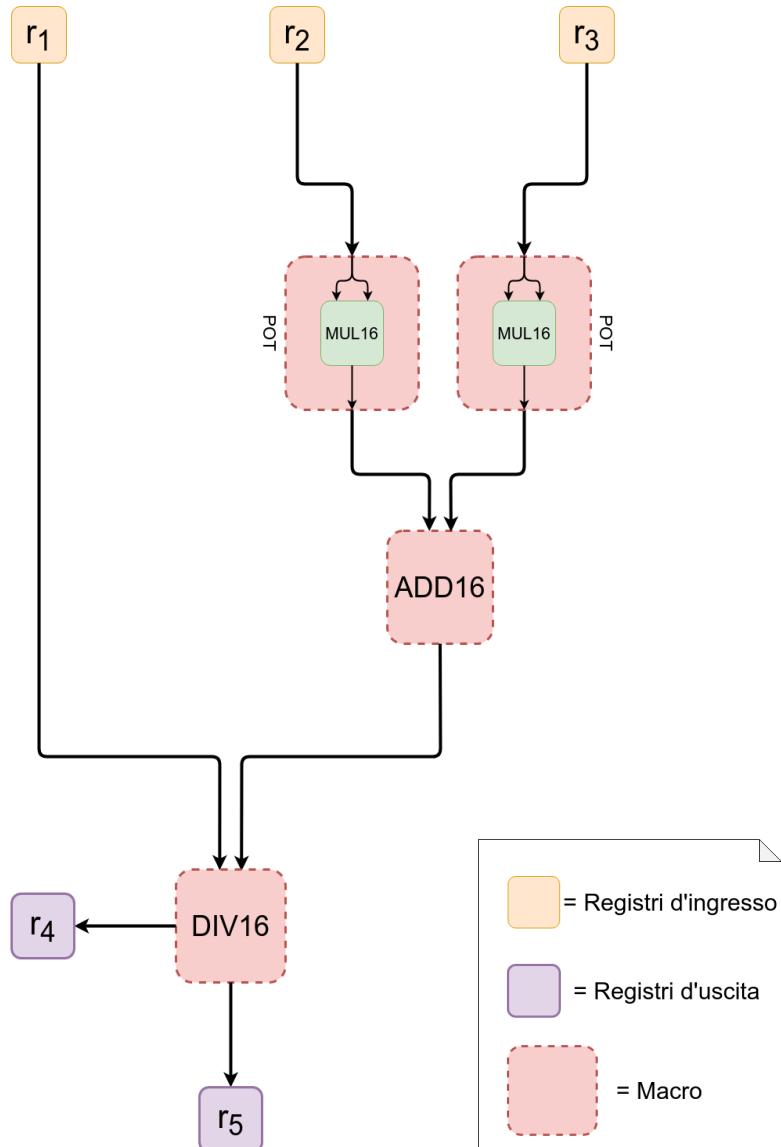
**Non viene proposta la tabella di verità di questo componente date le sue dimensioni considerevoli.



Data Path

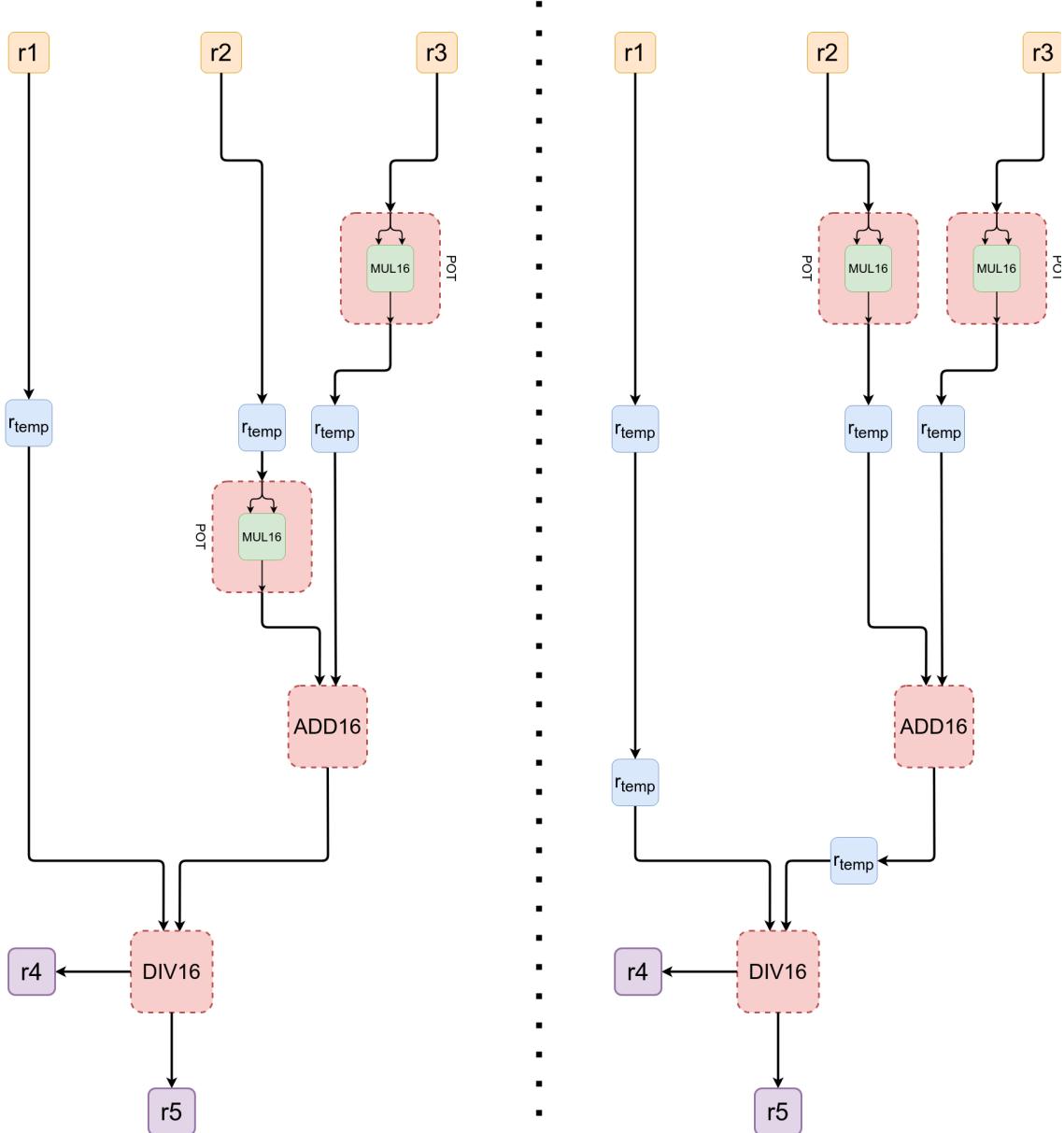
Essendo la rete non regolata da ALU o CU, il ciclo di data path corrisponde ad un ciclo di clock (che avrà una durata corrispondente a $t_{clock} = t_{POT} + t_{SUM} + t_{DIV}$). In questo caso, occorrono 5 registri di appoggio.

$$r_1 = C \quad r_2 = A \quad r_3 = B \quad r_4 = R \quad r_5 = Q$$





Nel caso ci trovassimo di fronte ad una rete multistadio ci troviamo di fronte all'introduzione di ulteriori registri per il campionamento e la propagazione dei risultati al ciclo di clock successivo. In tal caso tocca alla CU controllare debitamente i segnali per evitare problemi di sincronizzazione.



Resource Sharing

Pipelining

	= Registri d'ingresso
	= Registri d'uscita
	= Registri temporanei
	= Macro



Control Unit

Specifiche

Si delinea la specifica di progetto in cui l'automa si trova ad operare:

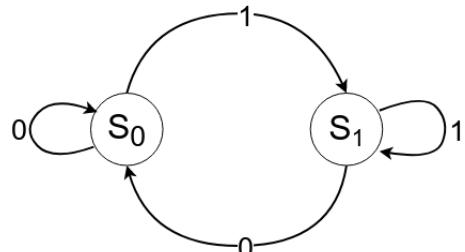
- Non avendo memoria, non si identifica uno stato di reset. Le operazioni precedenti non creano conflitto con le operazioni successive;
- Arbitrariamente possiamo indicare lo stato iniziale il momento in cui gli input sono inizializzati a 0;
- Lo stato finale si identifica quando viene completata la divisione, essendo l'ultima operazione completata.

In sintesi, l'automa esegue ciclicamente una sola operazione: $(A^2 + B^2) \div C$

Di conseguenza indichiamo arbitrariamente come **1** l'immissione di una sequenza di bit diversi da 0 e con **0** l'immissione di soli zeri. Troviamo in **S₀** lo stato di partenza e in **S₁** lo stato in cui l'automa è popolato, quest'ultimo "oscillerà" tra i due, in base agli input che gli vengono forniti.

Tabella degli stati:		
	1	0
S ₀	S ₁	S ₀
S ₁	S ₁	S ₀

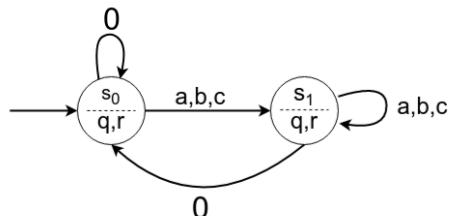
Nella divisione per 0 non troviamo veri problemi di esecuzione, ma la correttezza del calcolo viene meno (in ogni caso, in linea con la regola matematica secondo la quale la divisione per zero è impossibile). Basta dunque implementare un'eccezione nella CU che intercetti **C** in caso fosse zero e farle restituire un errore.



Implementazione della macchina di Moore

Definiamo una sestupla $(S, S_0, \Sigma, \Lambda, T, G)$ dove indichiamo:

- ◆ Un insieme finito di stati $S = \{s_0, s_1\}$;
- ◆ Uno stato iniziale $s_0 \in S$ nel quale tutti gli ingressi sono 0;
- ◆ Un insieme finito $\Sigma = \{a, b, c, 0\}$ che definisce l'alfabeto d'ingresso;
- ◆ Un insieme finito $\Lambda = \{q, r\}$ che definisce l'alfabeto di uscita;
- ◆ Una funzione di transizione $T : S \times \Sigma \rightarrow S$ che associa ad una coppia $<\text{stato, input}>$ lo stato successivo(nell'immagine, le frecce);
- ◆ Una funzione di uscita $G : S \rightarrow \Lambda$ che associa uno stato alla sua uscita, corrispondente alla funzione aritmetica che l'automa risolve(linee tratteggiate nell'immagine);

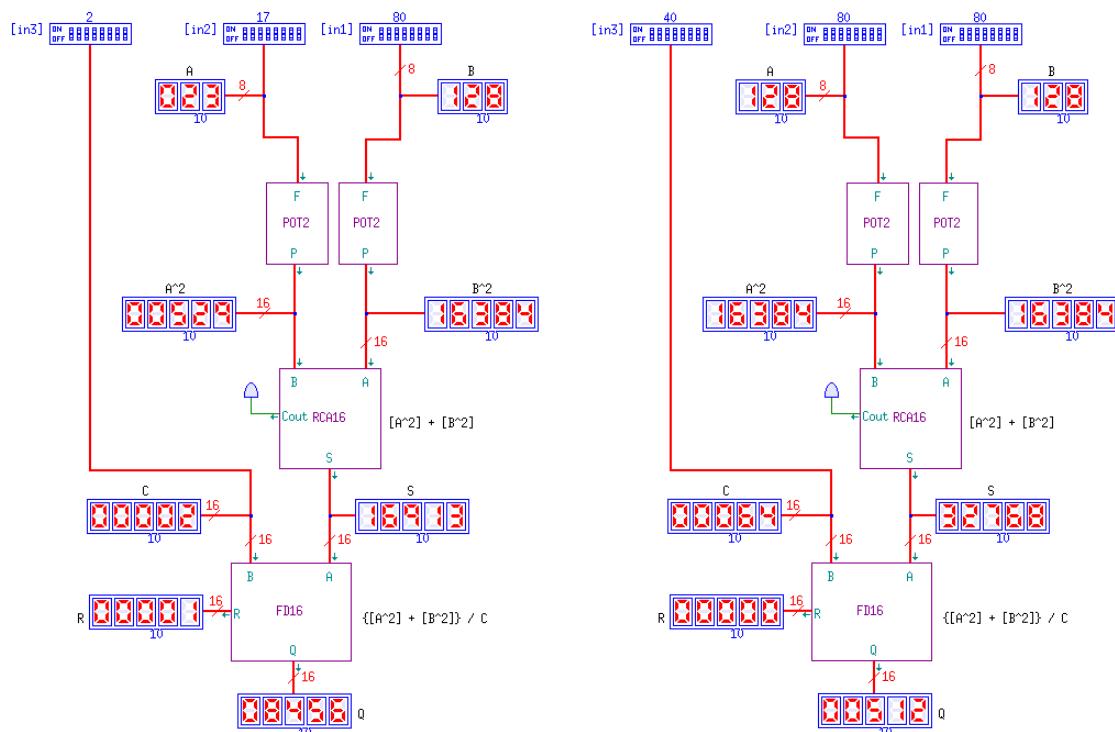




Simulazione e analisi del progetto

Verifica funzionale

Gli input vengono regolati da 3 DIP-Switch, uno a 16 bit collegato direttamente al divisore e 2 ad 8 bit collegati alla macro potenza e successivamente al sommatore. Mimo il valore di A, B, C che deve essere inserito in forma esadecimale, è possibile monitorare il risultato degli switch tramite gli schermi led che rappresentano il valore decimale.



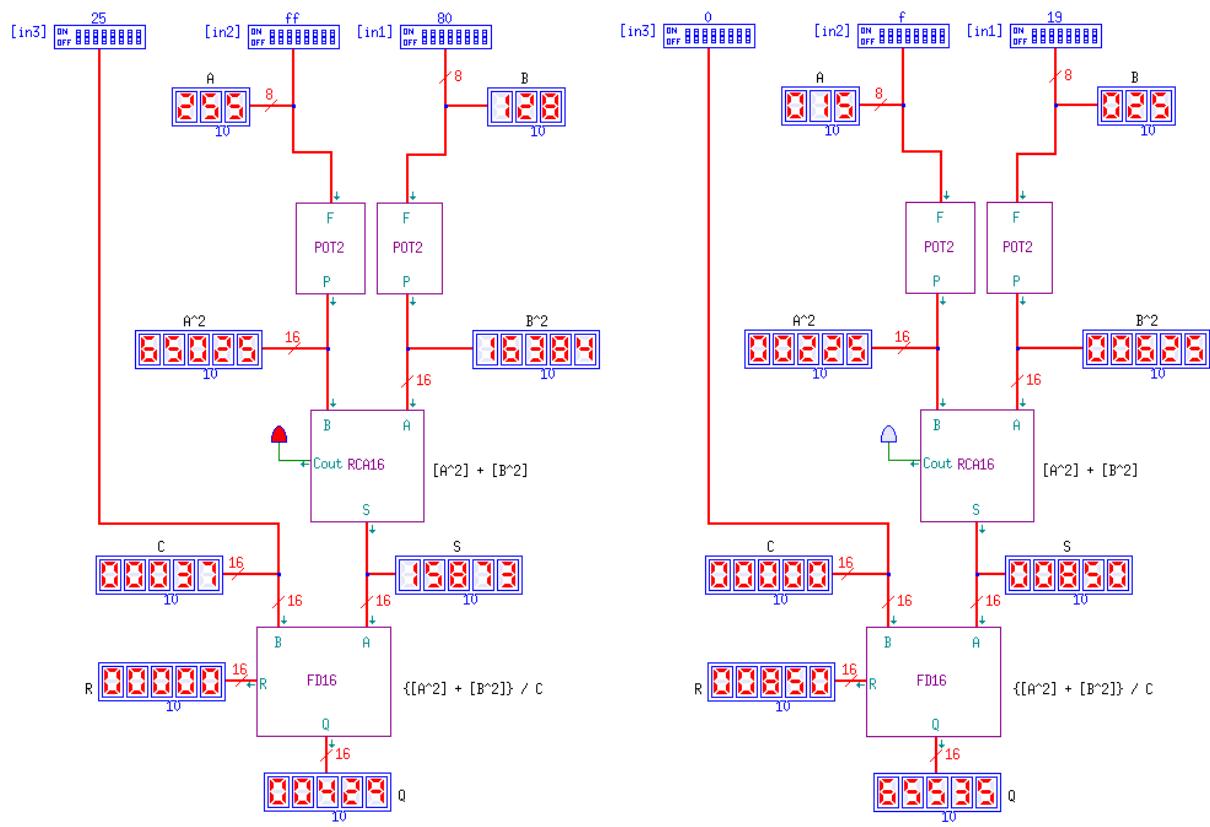
Sopra vediamo un esempio della la rete in funzione, di seguito (per completezza) viene proposto il funzionamento della conversione in esadecimale:

Conversione in esadecimale sx	Conversione in esadecimale dx
$A \rightarrow 23 \div 16 = 1 R : 7 \rightarrow 17$	$A \rightarrow 128 \div 16 = 8 R : 0 \rightarrow 80$
$B \rightarrow 128 \div 16 = 8 R : 0 \rightarrow 80$	$B \rightarrow 128 \div 16 = 8 R : 0 \rightarrow 80$
$C \rightarrow 2 \div 16 = 0 R : 2 \rightarrow 02$	$C \rightarrow 64 \div 16 = 4 R : 0 \rightarrow 40$



Sotto vediamo due errori che possono verificarsi:

- ✗ A sinistra, troviamo l'overflow nel sommatore: tramite il led acceso ci accorgiamo della presenza di errori nel calcolo: abbiamo infatti un risultato parziale in *S* che non corrisponde al vero, infatti dovrebbe essere 81409, ma non vi sono bit sufficienti per la rappresentazione dunque il risultato viene troncato (la divisione procede regolarmente);
- ✗ A destra, troviamo divisione per zero: *C* genera uno squilibrio nei riporti, tutti i bit di *Q* vengono settati ad 1 e in *R* troviamo il valore del dividendo. Ciò è causato perché tutte le differenze interne ai **HD** sono regolate da Exor e non vi è un meccanismo di controllo che permetta di identificare e risolvere questo errore in tempo utile.



Una possibile soluzione alla divisione per 0 potrebbe essere un multiplexer a 16 bit che accenda un led in caso si verificasse l'eventualità $C = 0$, in modo tale da permettere alla CU la notifica/la gestione dell'errore.



Valutazione di prestazioni e complessità

Tipo	Nome	Tc	Tp	A
Porte Elementari	NAND	1	1	1
	NOT	1	1	1
	AND	2	2	2
	OR	2	2	3
	EXOR	2	3	5
Componenti	HA	$S = 2$ $Cout = 2$	$S = 3$ $Cout = 2$	7
	FA	$S = 2$ $Cout = 4$	$S = 6$ $Cout = 5$	17
	MUX	2	3	4
	HS	$D = 2$ $Pr = 2$	$D = 3$ $Pr = 3$	8
	FS	$D = 2$ $PrOut = 3$	$D = 6$ $PrOut = 8$	19
	HD	$Q = 2$ $BOut^* = 2$ $PrOut = 3$ $CrOut^* = 2$	$Q = 9$ $BOut = 2$ $PrOut = 8$ $CrOut = 2$	27
Macro	MUL	$P = 2$	$P = 112$	1000
	RCA	$S = 2$ $Cout = 62$	$S = 74$ $Cout = 77$	262
	DIV	$Q = 130 (162)^*$ $R = 2$	$Q = 2199 (2231)^*$ $R = 2208 (2240)^*$	6928

Stima di complessità circuitale e prestazioni

Analizzando la tabella sopra, possiamo fare alcune considerazioni:

Calcolando il tempo di propagazione complessivo della rete possiamo ricavare la durata minima del periodo di clock. La rete ivi costruita funziona a singolo stadio:

$$t_{clock} = Tp_{MUL} + Tp_{SUM-S} + Tp_{DIV-Q} \rightarrow 112 + 74 + 2199 (+2231)^* = 2385 (2417)^*$$



Inoltre è possibile analizzare la complessità delle macro realizzate, mettendole in relazione al numero di bit ingresso e studiando il numero dei componenti che le compongono.

Nel **moltiplicatore** (in:8 / out: 16, $n = 8$)

- Il tempo di contaminazione risulta indipendente dal numero di bit ingresso, dunque è costante: $Tc_{MUL} = Tc_{and} \rightarrow O(1)$
- Dal percorso del tempo di propagazione vediamo che dipende per la maggior parte dalle performance del FA, dunque :
$$Tp_{MUL} = Tp_{and} + Tp_{HA-Cout} + 12 \cdot Tp_{FA-Cout} + 8 \cdot Tp_{FA-S} \simeq 20 \cdot Tp_{FA} \simeq [(n \cdot 2) + 4] \cdot Tp_{FA} \rightarrow O(n)$$
- Nel calcolo di A_{MUL} dobbiamo tenere presente che l'area di un FA è 8 volte più grande di quella di un AND ed oltre 2 volte quella di un HA, di seguito si tenta una normalizzazione per ottenere un risultato più chiaro:

$$A_{HA} + (5 \cdot A_{and}) = A_{FA} \quad \vee \quad 8 \cdot A_{and} \simeq A_{FA}$$

$$\begin{aligned} A_{MUL} &= 64 \cdot A_{and} + 8 \cdot A_{HA} + 48 \cdot A_{FA} = 24 \cdot A_{and} + 8 \cdot A_{FA} + 48 \cdot A_{FA} = \\ &= 24 \cdot A_{and} + 56 \cdot A_{FA} = 3 \cdot A_{FA} + 56 \cdot A_{FA} = 59 \cdot A_{FA} = \\ &= [(8 \cdot 7) + 3] \cdot A_{FA} \simeq [(n \cdot 7) + 3] \cdot A_{FA} \rightarrow O(n) \end{aligned}$$

Si conclude che l'area del moltiplicatore ha complessità lineare.

Nel **sommatore** (in: 16 / out: 16, $n = 16$)

- Il tempo di contaminazione è costante nella somma e lineare nel resto, possiamo considerarlo costante dato che tra i due il valore più "importante" è sicuramente la somma:

$$\begin{aligned} Tc_{SUM-S} &= Tc_{HA-S} = 2 \rightarrow O(1) \\ Tc_{SUM-Cout} &= Tc_{HA-Cout} + 15 \cdot Tc_{FA-Cout} \simeq \\ &\simeq 16 \cdot Tc_{FA-Cout} = n \cdot Tc_{FA-Cout} \rightarrow O(n) \end{aligned}$$

- I tempi di propagazione della somma e del resto risultano simili:
$$\begin{aligned} Tp_{SUM-S} &= Tp_{HA-Cout} + 14 \cdot Tp_{FA-Cout} + Tp_{FA-S} \simeq 15 \cdot Tp_{FA-Cout} \simeq \\ &\simeq n \cdot Tp_{FA-Cout} \rightarrow O(n) \\ Tp_{SUM-Cout} &= Tp_{HA-Cout} + 15 \cdot Tp_{FA-Cout} \simeq 16 \cdot Tp_{FA-Cout} \simeq \\ &\simeq n \cdot Tp_{FA-Cout} \rightarrow O(n) \end{aligned}$$
- Anche per l'area si registra una complessità lineare:
$$A_{SUM} = A_{HA} + 15 \cdot A_{FA} \simeq 16 \cdot A_{FA} \simeq n \cdot A_{FA} \rightarrow O(n)$$

Nel **divisore** (in:16 / out: 16, $n = 16$)

- Il tempo di contaminazione è lineare per il quoziente e costante per il resto, il quoziente può essere preso come calore più "importante":

$$\begin{aligned} Tc_{DIV-Q} &= 16 \cdot Tc_{HD-PrOut} + Tc_{not} (+16 \cdot Tc_{HD-CrOut})^* \simeq 16 \cdot Tc_{HD-PrOut} \simeq \\ &\simeq n \cdot Tc_{HD-PrOut} \rightarrow O(n) \\ Tc_{DIV-R} &= Tc_{HD-Q} \rightarrow O(1) \end{aligned}$$



- Il tempo di propagazione è simile sia per il quoziente che per il resto, notiamo che differiscono di sole 9 unità ed i componenti attraversati risultano essere gli stessi. Dunque possiamo limitarci a studiare la complessità di uno dei due:

$$\begin{aligned} \text{Tp}_{DIV-Q} &= 16 \cdot [(16 \cdot \text{Tp}_{HD-PrOut}) + \text{Tp}_{not} (+\text{Tp}_{HD-CrOut})^*] + 15 \cdot \text{Tp}_{HD-Q} \simeq \\ &\simeq n^2 \cdot \text{Tp}_{HD-PrOut} + n \cdot \text{Tp}_{not} (+n \cdot \text{Tp}_{HD-CrOut})^* + (n-1) \cdot \text{Tp}_{HD-Q} \simeq \\ &\simeq n^2 \cdot \text{Tp}_{HD-PrOut} \rightarrow O(n^2) \end{aligned}$$

- Il calcolo dell'area risulta quadratico a colpo d'occhio:

$$A_{DIV} = 256 \cdot A_{HD} + 16 \cdot A_{not} \simeq n^2 \cdot A_{HD} + n \cdot A_{not} \rightarrow O(n^2)$$

Completata l'analisi delle singole macro è possibile ricavare la complessità di tutto il circuito:

- Per il tempo di contaminazione:

$$Tc_{tot} = Tc_{MUL} + Tc_{SUM} + Tc_{DIV} = O(1) + O(1) + O(n) = O(n)$$

- Per il tempo di propagazione:

$$Tp_{tot} = Tp_{MUL} + Tp_{SUM} + Tp_{DIV} = O(n) + O(n) + O(n^2) = O(n^2)$$

- Per l'area:

$$A_{tot} = A_{MUL} + A_{SUM} + A_{DIV} = O(n) + O(n) + O(n^2) = O(n^2)$$

I dati ricavati sulla complessità trovano riscontro anche nei numeri registrati durante l'implementazione. Dai dati presenti nella tabella verifichiamo che il divisore è di gran lunga il componente più complesso e più lento nelle operazioni di calcolo.

Nulla di nuovo: nonostante l'implementazione in logica NAND che sicuramente "facilita" l'implementazione elettronica, è da sempre risaputa la difficoltà che hanno i calcolatori ad implementare operazioni inverse come la divisione. Con una certa estensione, possiamo dire che ai giorni nostri la crittografia e parte della sicurezza informatica "sfruttano" a loro vantaggio proprio questa difficoltà dei calcolatori tradizionali. Nei sistemi odierni vengono usati algoritmi più efficienti, ma più complessi da implementare e che potrebbero richiedere hardware più specifico (il metodo di Newton-Raphson o la divisione Goldschmidt).

Degno di nota è il fatto che nei computer quantistici non riscontriamo questo comportamento. Infatti, il cambio radicale di architettura che richiedono i qubit per essere manipolati, agevola la parallelizzazione del calcolo aumentando di molto le prestazioni nella risoluzione di algoritmi di questo genere. Rimane comunque una tecnologia acerba, ma molto promettente.

