UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO
Corso di Laurea in Informatica Applicata

Prodotto di differenze di valori a 4 bit

Progetto d'esame di Reti Logiche

A.A 2022/23

Balducci Milena Matricola 321791

1 Specifica

1.1 Scopo del progetto

L'obiettivo è quello di realizzare una rete combinatoria in tecnologia CMOS dotata di quattro ingressi A e B a 4 bit, rappresentanti quattro valori in codifica binaria naturale, e di una uscita Z a 8 bit che rappresenta il prodotto delle differenze dei quattro valori in ingresso.

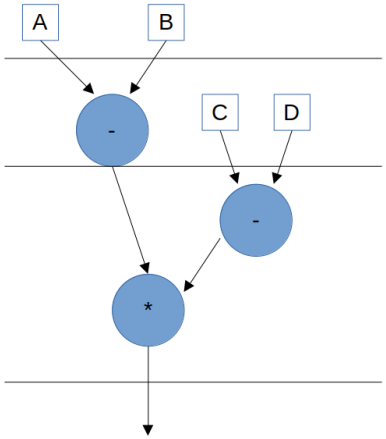
1.2 Specifica funzionale

$f : \mathbb{N}^4 \longrightarrow \mathbb{N}, \;\; f(a,b,c,d) = (a-b) \cdot (c-d).$

2 Impostazione progetto a livello RT

2.1 Data Flow Graph

$f = (a-b)*(a-b)$



Dal momento che l'operazione di sottrazione compare due volte all'interno del data flow graph, è possibile utilizzare il resource sharing, usando una unica macro aritmetica per entrambe le differenze. Per fare questo sono necessari due cicli di clock: durante il primo viene svolta la differenza tra i primi due valori, e durante il secondo viene svolta la seconda differenza e il prodotto tra i due risultati. Per svolgere la differenza tra due numeri si è scelto di utilizzare la notifica in complemento a due, che permette di svolgere le sottrazioni attraverso la somma. Il secondo valore andrà dunque convertito nel rispettivo complemento prima di essere passato ad un addizionatore.
Per mantenere i valori degli ingressi e delle uscite durante i due cicli di clock, sono necessari dei registri per gli ingressi C e D e per l'uscita di (A-B).

2.2 Risorse

- 3 Registri parallelo/parallelo a 4 bit
- 2 Multiplexer a 2 ingressi a 4 bit
- 1 Demultiplexer a 2 uscite a 4 bit
- 1 Complementatore a 4 bit
- 1 Addizionatore a 4 bit
- 1 Moltiplicatore a ingressi a 4 bit e uscita a 8 bit

3 Progetto delle risorse a livello gate

Per progettare i componenti necessari alla realizzazione della rete combinatoria illustrata nella specifica, è necessario partire dalle porte logiche. Queste vengono realizzate partendo dai due componenti base della tecnologia C-MOS: i transistor **nMOS** e **pMOS**. Questi due transistor sono formati da tre terminali: **Source (S)**, **Drain (D)** e **Gate (G)**.

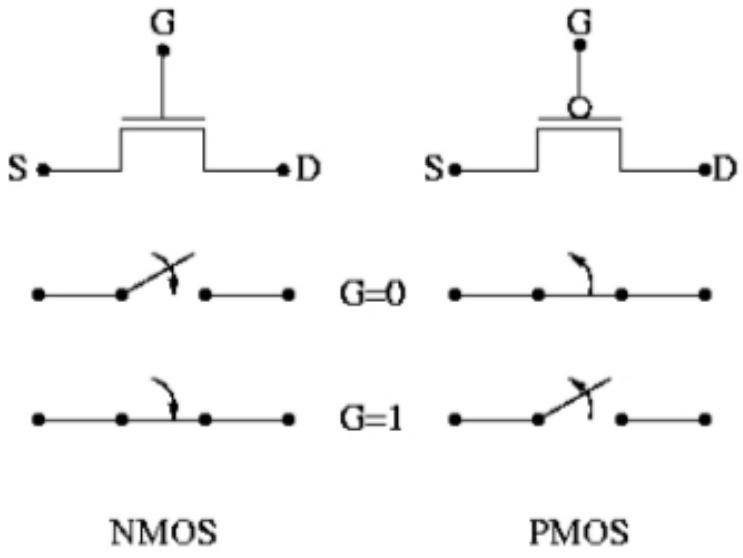


Figura 1: Schema dei transistor nMOS (sinistra) e pMOS (destra)

- Il transistor **nMOS** crea un collegamento tra S e D solo se il Gate assume valore 1 (alto). Quando viene collegato a massa, si forma la rete di **"pull-down"**.
- Il transistor **pMOS** collega S e D solo se il valore del Gate è 0 (basso). Quando collegato alla Vdd, forma quella che si chiama rete di **"pull-up"**.

3.1 Porte logiche semplici

Le porte logiche elementari sono quelle funzionalmente complete, ovvero che permettono di costruire tutte le altre porte logiche. Queste sono le porte **NOT**, **NAND** e **NOR**.

3.1.1 Porta NOT

La porta NOT (o inverter), corrispondente all'omonimo operatore logico, prende il segnale in ingresso e lo restituisce negato. Viene realizzata collegando pull-up e pull-down. Di seguito sono illustrate la tabella della verità e l'implementazione della porta NOT.

X	X’
0	1
1	0

Tabella 1: Tabella della verità inverter

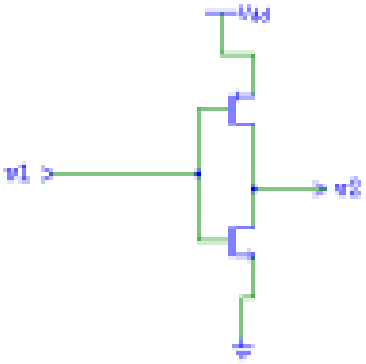


Figura 2: Inverter

3.1.2 Porta NAND

La porta NAND è una porta logica invertente che restituisce 1 solo se almeno uno dei due ingressi vale 0, e corrisponde alla negazione della congiunzione logica dei due ingressi. Viene realizzata collegando in parallelo due transistor pMOS e in serie due transistor nMOS.

A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

Tabella 2: Tabella della verità NAND

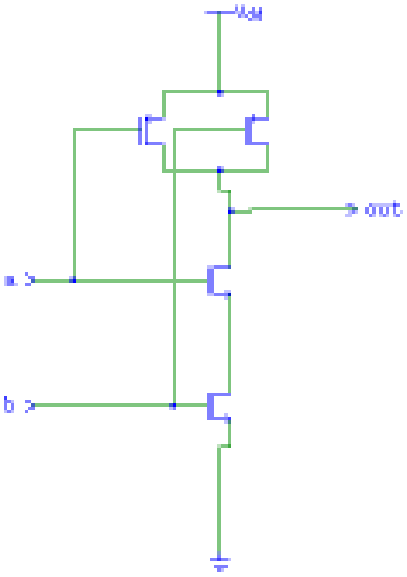


Figura 3: Porta NAND

3.1.3 Porta NOR

La porta NOR è una porta logica invertente che restituisce 1 solo se entrambi gli ingressi valgono 0, e corrisponde alla negazione della disgiunzione logica dei due ingressi. Viene realizzata collegando in parallelo due transistor nMOS e in serie due transistor pMOS.

A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

Tabella 3: Tabella della verità NOR

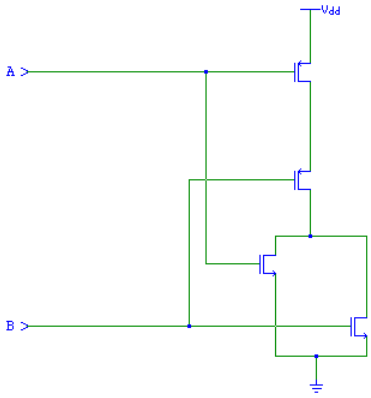


Figura 4: Porta NOR

3.2 Porte logiche composte

Le porte logiche composte utilizzate in questo progetto sono le porte AND, OR e EXOR. Ciascuna di queste è realizzata combinando porte NOT, NAND e NOR.

3.2.1 Porta AND

La porta AND (congiunzione logica) viene implementata tramite una porta NAND, la cui uscita viene invertita tramite una porta NOT.
(AB)=((AB)')'

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

Tabella 4: Tabella della verità AND

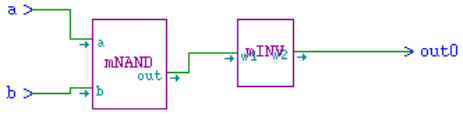


Figura 5: Porta AND

3.2.2 Porta OR

E' possibile realizzare una porta OR, che corrisponde all'omonimo operatore logico, semplicemente invertendo l'uscita di una porta NOR.
(A+B)=((A+B)')'

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

Tabella 5: Tabella della verità OR

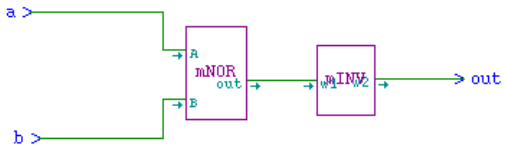


Figura 6: Porta OR

3.2.3 Porta EXOR

L'operatore logico EXOR (EXclusive OR) corrisponde all'equazione logica $ab' + a'b$. Una porta logica di questo tipo è dunque realizzabile tramite porte NOT, AND e OR. Tuttavia, utilizzando le leggi di DeMorgan è possibile realizzare questa porta logica esclusivamente tramite porte NAND e NOT. Infatti:
((A'B)'(AB')')'=A'B+AB'

A	B	A'B + AB'
0	0	0
0	1	1
1	0	1
1	1	0

Tabella 6: Tabella della verità EXOR

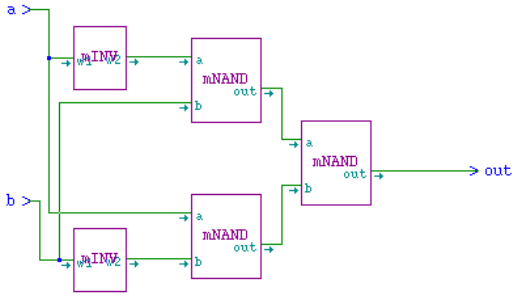


Figura 7: Porta EXOR

3.3 Elementi di memoria

Gli elementi di memoria sono circuiti sequenziali che possono avere solo due stati (0 e 1) in grado di passare da uno all'altro sulla base di uno o più segnali di ingresso, e di mantenere lo stato corrente per un periodo di tempo indefinito. Sono dunque elementi in gradi di memorizzare un singolo bit.

3.3.1 Latch SR

Il Latch SR (Set-Reset) è un elemento di memoria asincrono dotato di due segnali di ingresso ***s*** e ***r*** e di due porte NOR, ciascuna avente un ingresso collegato a *s* o *r*, e l'altro collegato all'uscita dell'altra porta NOR. Il Latch SR ammette un cambiamento di stato sulla sola base dei segnali di ingresso:

- Con *s*=1, *r*=0 si passa allo stato di **SET**: *y*'=0, *y*=1;
- Con *s*=0, *r*=1 si passa allo stato di **RESET**: *y*'=1, *y*=0;
- Con *s*=0, *r*=0 lo stato rimane invariato (**HOLD**).

La configurazione *s*=1, *r*=1 non è ammessa.

y'	s	r	f (y)	g (y')
0	0	0	1	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Tabella 7: Tabella della verità Latch SR

Il problema legato a questa rete sequenziale, che è asincrona, è la possibilità che si presentino situazioni transitorie che potrebbero modificare le configurazioni degli ingressi in maniera non desiderata o non prevedibile.

3.3.2 Flip-flop SR

Nel Flip-flop SR, i due segnali *s* e *r* sono generati da due porte AND, alle quali sono collegati due segnali di ingresso ***S*** ed ***R*** e un segnale di sincronismo denominato **Clock**, di natura generalmente periodica. In questo modo, ci si assicura che l'aggiornamento degli stati avvenga solo in momenti ben definiti.

- con Clock=1, *S*=1, *R*=0 si ha lo stato di **SET**;
- con Clock=1, *S*=0, *R*=1 si ha lo stato di **RESET**;
- con Clock=1, *S*=0, *R*=0 si ha lo stato di **HOLD**;
- con Clock=0, il flip-flop rimane in stato di HOLD indipendentemente dalla configurazione degli ingressi.

Clk	S	R	s	r	
0	0	0	0	0	HOLD
0	0	1	0	0	HOLD
0	1	0	0	0	HOLD
0	1	1	0	0	HOLD
1	0	0	0	0	HOLD
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	1	1	N.A.

Tabella 8: Tabella della verità Flip-flop SR

3.3.3 Flip-flop D-level sensitive

Il funzionamento di un Flip-flop D-level sensitive è analogo a quello del FFSR, ma ai due segnali *S* e *R* viene sostituito un unico segnale *D*: *S* corrisponde a *D*, *R* a *D'*. Questo permette di escludere la configurazione non ammessa *s*=1, *r*=1, e di avere durante il periodo di salita del clock due sole configurazioni possibili, ovvero quelle di SET e di RESET. Il FFDls rimane sensibile alle variazioni degli ingressi per tutta la durata di attività del segnale di clock.

Clk	D	S	R	s	r	
0	0	0	1	0	0	HOLD
0	1	1	0	0	0	HOLD
1	0	0	1	0	1	RESET
1	1	1	0	1	0	SET

Tabella 9: Tabella della verità FF D-level sensitive

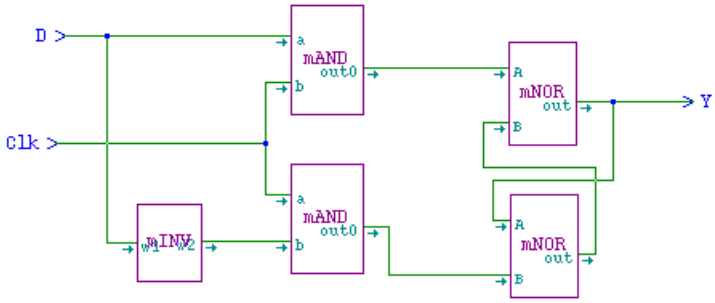


Figura 8: Flip-flop D-level sensitive

3.3.4 Flip-flop D-edge triggered

Il Flip-flop D-edge triggered è un elemento di memoria nel quale l'aggiornamento dello stato è consentito solo sul fronte di salita del clock. E' costruito tramite due FFDls, denominati **master** e **slave**. Il *master* prende in ingresso il segnale di clock negato e il bit da memorizzare, *slave* mentre lo slave riceve l'uscita del *master* e il segnale di clock.

- Quando Clock=0, il *master* è abilitato a cambiare stato mentre lo *slave* mantiene le uscite stabili;

– Quando Clock=1, lo *slave* rileva le variazioni di stato del *master* e le propaga, mentre il *master* è insensibile alle variazioni degli ingressi.

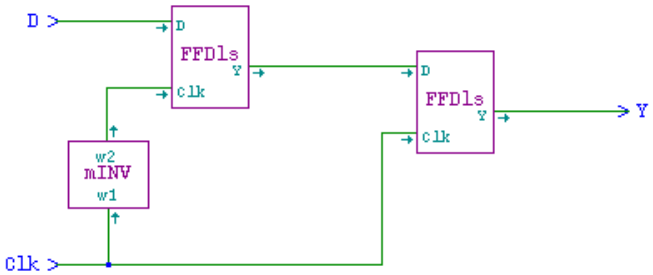


Figura 9: Flip-flop D edge-triggered

3.4 Registri

Un registro è un elemento di memoria in grado di memorizzare vettori di n bit. Ne esistono di differenti tipi, quelli utilizzati in questo progetto sono registri parallelo/parallelo a 4 bit. Questo tipo di registri è dotato di quattro FFD-edge triggered. Tutti i bit vengono ricevuti in ingresso contemporaneamente, collegando l'ingresso dell'i-esimo flip-flop all'i-esimo bit del vettore in ingresso, e le uscite vengono tutte lette contemporaneamente.

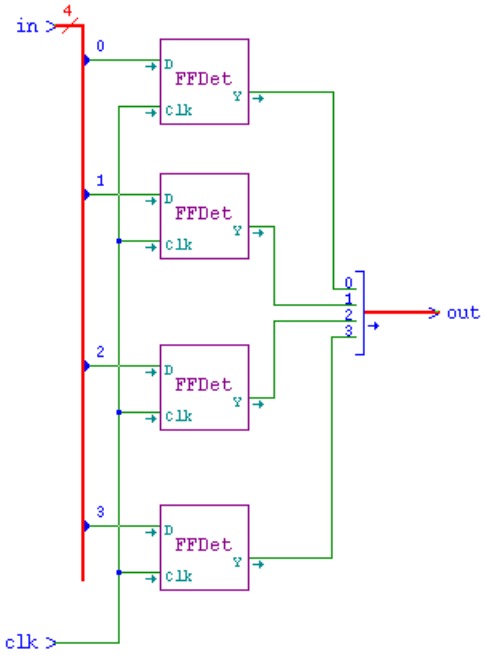


Figura 10: Registro parallelo/parallelo a 4 bit

3.5 Multiplexer

Il Multiplexer (MUX) è un componente combinatorio dotato di x ingressi a n bit e di una uscita a n bit, in grado di scegliere il valore che assumerà l'uscita tra quelli in ingresso sulla base di uno o più segnali di controllo. Il più semplice multiplexer ha 2 ingressi a 1 bit e una uscita a un bit. Funzione, tabella della verità e implementazione sono illustrati di seguito:

$f(a,b,c) = ac' + bc$

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tabella 10: Tabella della verità MUX

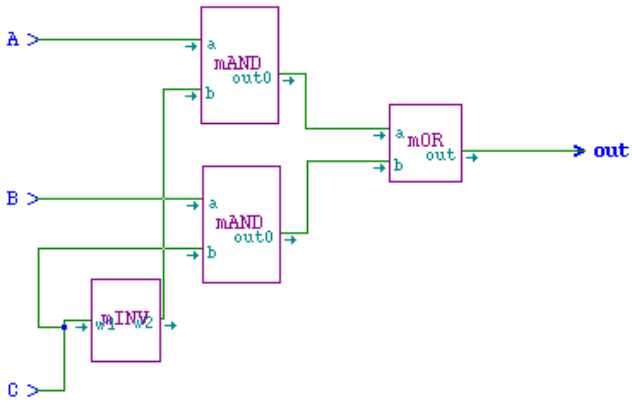


Figura 11: Multiplexer a 1 bit

E' possibile realizzare MUX in grado di scegliere tra due valori a n bit combinando n MUX a 1 bit, ognuno dei quali seleziona l'i-esimo bit del primo o del secondo ingresso a seconda del segnale di controllo. In questo progetto è stato fatto uso di MUX con due ingressi a 4 bit e una uscita a 4 bit.

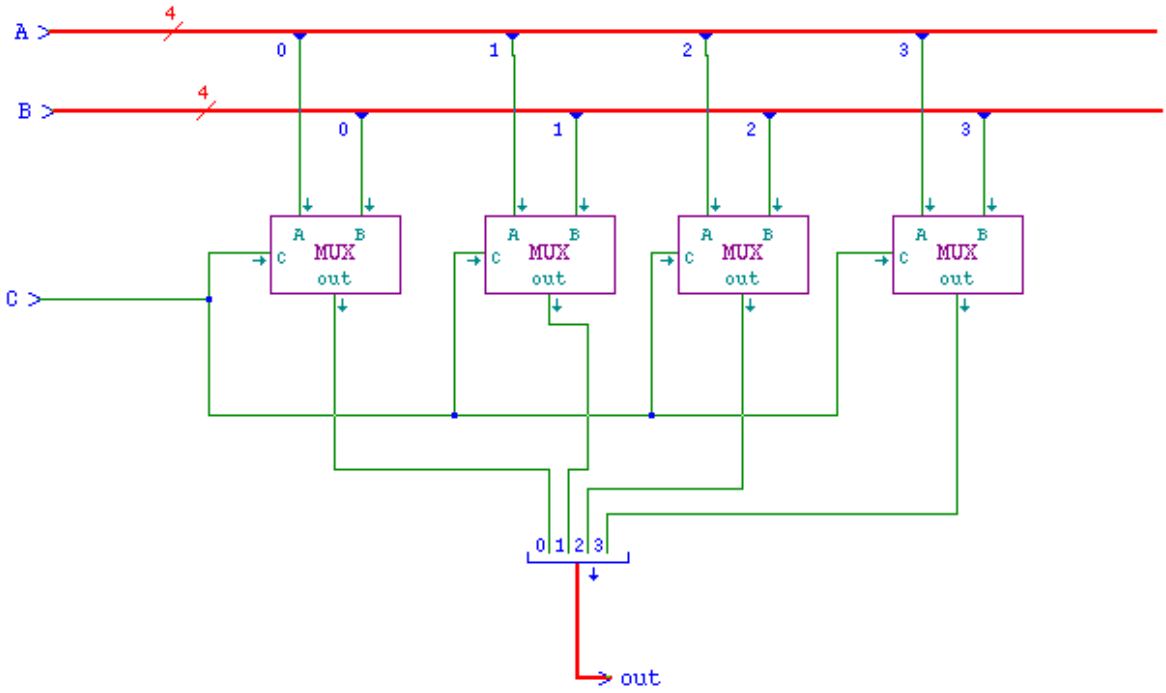


Figura 12: Multiplexer a 4 bit

3.6 Demultiplexer

Un Demultiplexer (DEMUX) è un componente che funziona in maniera opposta a un MUX: è dotato di un ingresso a n bit e di x uscite a n bit; una di queste uscite assume il valore dell'ingresso, mentre le altre valgono zero. La selezione è operata sulla base di uno o più segnali di controllo. Il più semplice DEMUX, illustrato di seguito, è dotato di un ingresso a 1 bit e due uscite a 1 bit.
$f_1 = ac'$
$f_2 = ac$

A	C	F1	F2
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

Tabella 11: Tabella della verità DEMUX

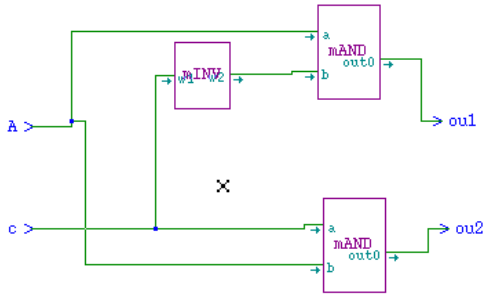


Figura 13: Demultiplexer a 1 bit

Come per i multiplexer, è possibile implementare un DEMUX con due uscite a n bit combinando n DEMUX a 1 bit, ciascuno dei quali prende in ingresso l'i-esimo bit del vettore in ingresso. Il DEMUX raffigurato di seguito è dotato di un ingresso e due uscite a 4 bit.

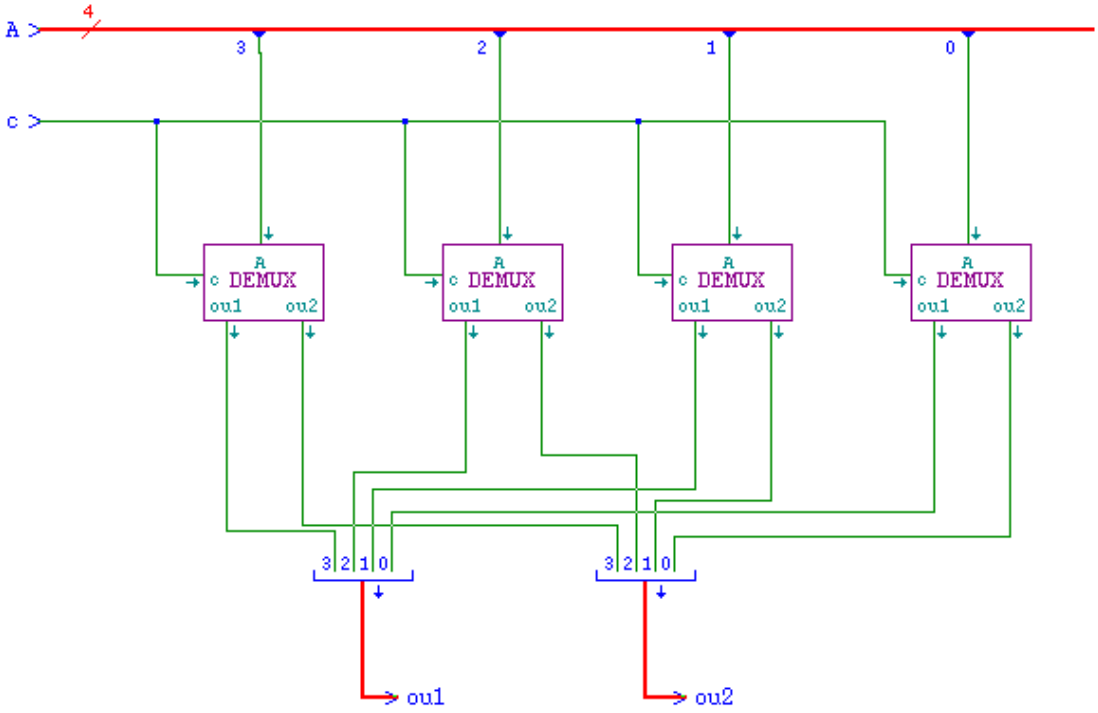


Figura 14: Demultiplexer a 4 bit

3.7 Macro aritmetiche

3.7.1 Half adder

Un Half Adder (HA) è una rete combinatoria in grado di calcolare la somma di due bit. E' dotato di due ingressi a un bit (gli addendi) e di due uscite S e Cout che rappresentano Il risultato della somma e il riporto in uscita, rispettivamente. Le equazioni corrispondenti alle due uscite sono:

$S = A \oplus Cin$
$Cout = ACin$

A	Cin	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabella 12: Tabella della verità HA

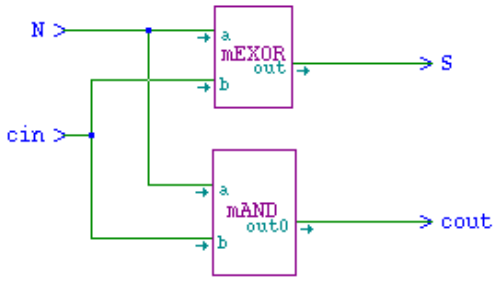


Figura 15: Half Adder

3.7.2 Full adder

Un Full Adder (FA) è una rete combinatoria che calcola la somma di tre bit. E' dotato di tre ingressi A, B e Cin che rappresentano il primo addendo, il seccondo addendo e il riporto in ingresso, e di due uscite S e Cout che rappresentano il risultato della somma e il riporto in uscita, rispettivamente. Le equazioni delle due uscite sono:
$S = Cin \oplus (A \oplus B)$
$cout = AB + Cin(A + B)$
E' possibile implementare questo componente utilizzando due HA in cascata e una porta NOT. Il primo HA calcola la somma dei due addendi, e al risultato viene sommato il riporto in ingresso tramite il secondo HA. Il riporto in uscita del FA corrisponde alla disgiunzione logica dei riporti in uscita dei due HA.

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	0	1
1	1	0	0	1
1	1	1	1	1

Tabella 13: Tabella della verità FA

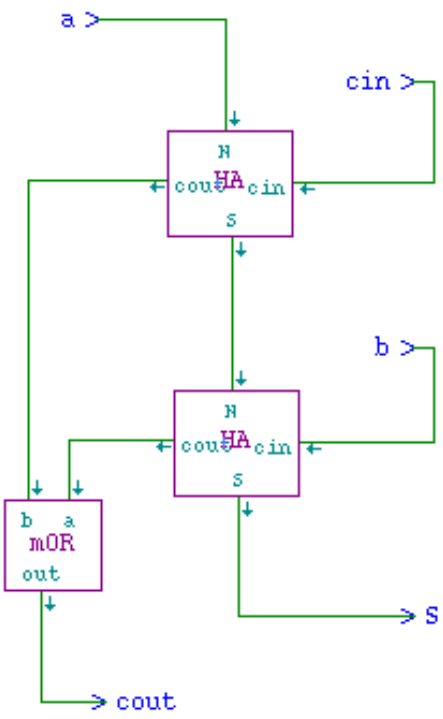


Figura 16: Full Adder

3.7.3 Complementatore

Per convertire un numero intero nel rispettivo complemento a 2, è sufficiente complementare i singoli bit che lo compongono, e poi sommare 1 al vettore di bit così ottenuto. Per realizzare un circuito in grado di svolgere questa operazione per valori a 4 bit si utilizzando dunque degli inverter, le cui uscite vanno in ingresso a degli HA in cascata. Il riporto in ingresso del primo HA viene collegato alla differenza di potenziale.

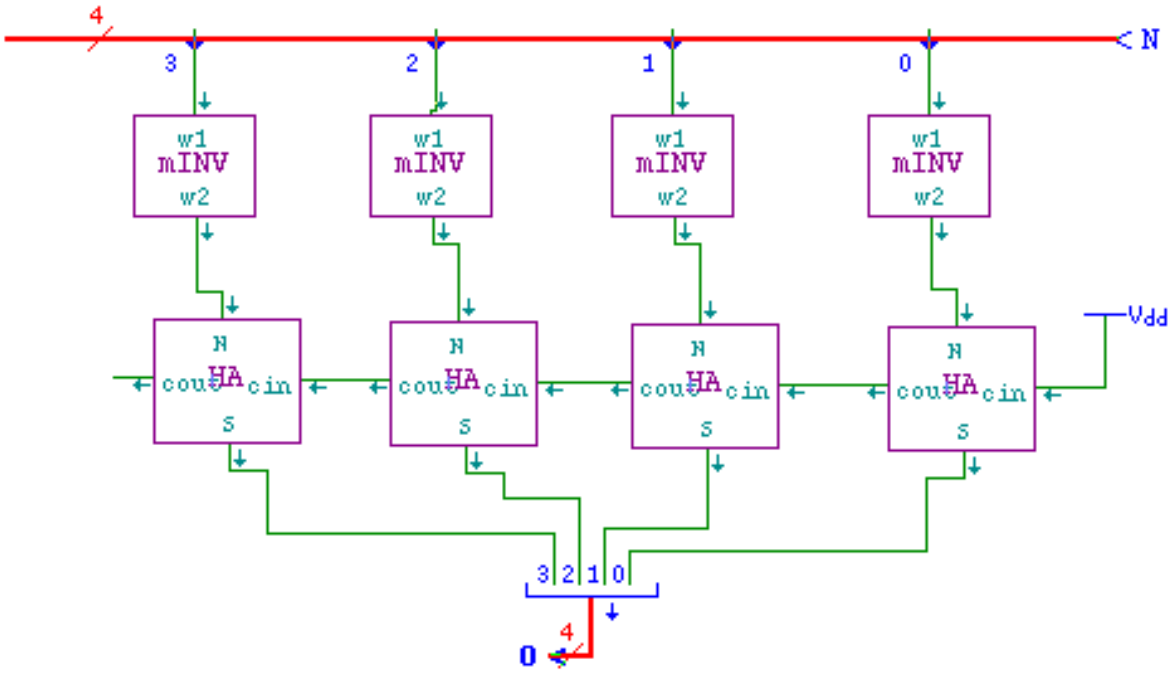


Figura 17: Complementatore a 4 bit

3.7.4 RCA a 4bit

Un Ripple Carry Adder (RCA) è una macro funzionale per l'addizione di valori a *n*bit. Il suo funzionamento si basa sullo schema classico dell'addizione in colonna:

	c3	**c2**	**c1**	**c0**	
	x3	x2	x1	x0	+
	y3	y2	y1	y0	=
c4	s3	s2	s1	s0	

Tabella 14: Schema addizione in colonna

Questo schema viene implementato tramite FA, collegando il riporto in ingresso di ogni FA al riporto in uscita del FA precedente, con l'eccezione del primo FA, il cui riporto in ingresso è collegato alla massa.

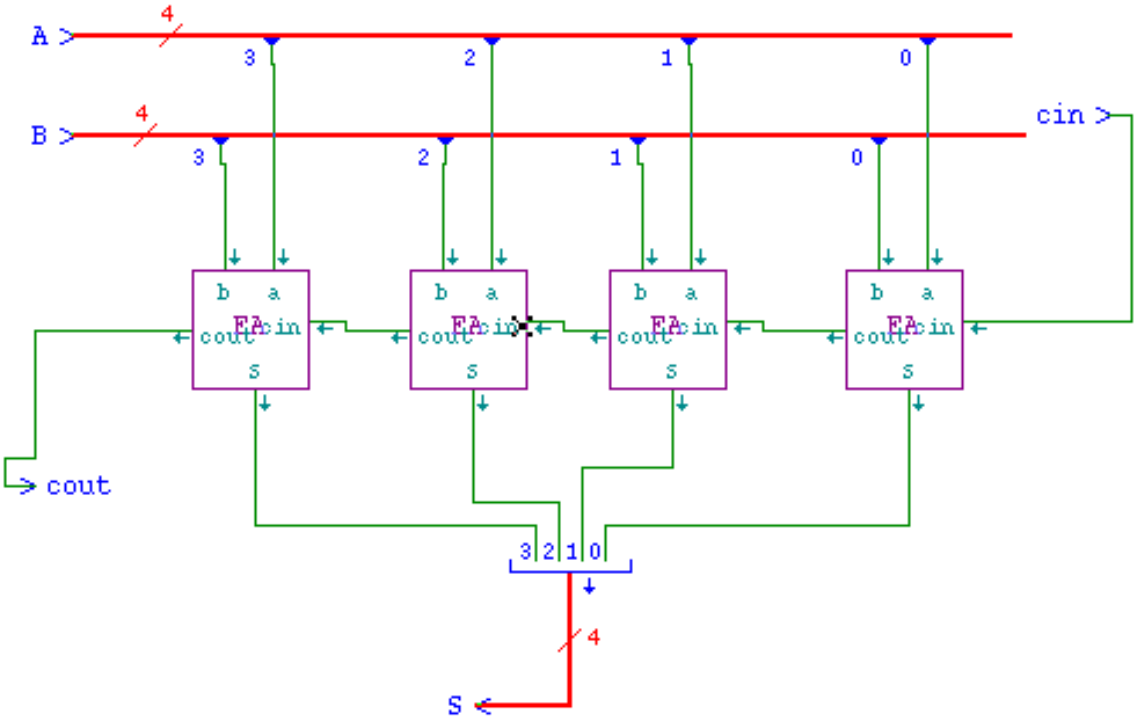


Figura 18: Ripple-Carry Adder a 4 bit

3.7.5 Moltiplicatore a 4 bit

Un moltiplicatore è, come suggerisce il nome, un componente combinatorio in grado di effettuare il prodotto di valori a *n* bit. In questo progetto è stato utilizzato un moltiplicatore a 4 bit. Per costruire questo componente, è necessario innanzitutto ricavare la funzione logica del prodotto tra due bit. Osservando la tabella della verità sottostante, risulta evidente che questa corrisponde al prodotto logico di due bit, e dunque la funzione logica della moltiplicazione è **S=AB**.

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

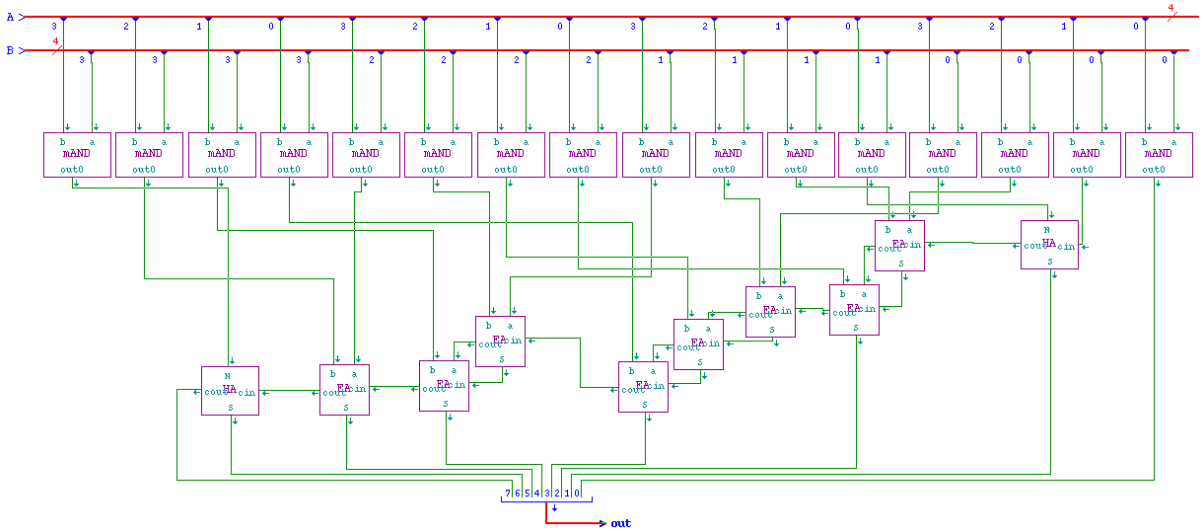
Tabella 15: Tabella della verità moltiplicazione 1 bit

Una volta determinata la funzione del prodotto matematico tra due bit, per realizzare un moltiplicatore è sufficiente seguire il classico schema della moltiplicazione in colonna.

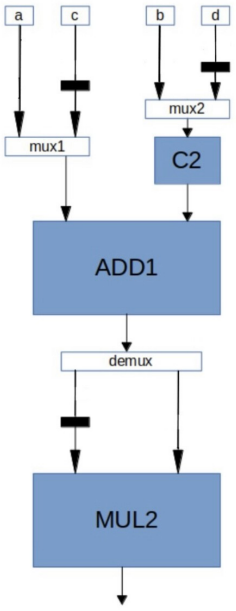
				x3	x2	x1	x0	*
				y3	y2	y1	y0	=
		0	0	0	y0x3	y0x2	y0x1	y0x0	+
		0	0	y1x3	y1x2	y1x1	y1x0	0	+
		0	y2x3	y2x2	y2x1	y2x0	0	0	+
		y3x3	y3x2	y3x1	y3x0	0	0	0	+
s7	s6	s5	s4	s3	s2	s1	s0		

Tabella 16: Schema moltiplicazione in colonna

Seguendo questo schema, è possibile implementare il moltiplicatore utilizzando delle porte AND per svolgere il prodotto dei singoli bit, e poi FA e HA per sommare tra loro i prodotti parziali che si trovano lungo la stessa colonna, prestando attenzione a propagare correttamente il riporto in uscita alla colonna successiva.



4 Data Path



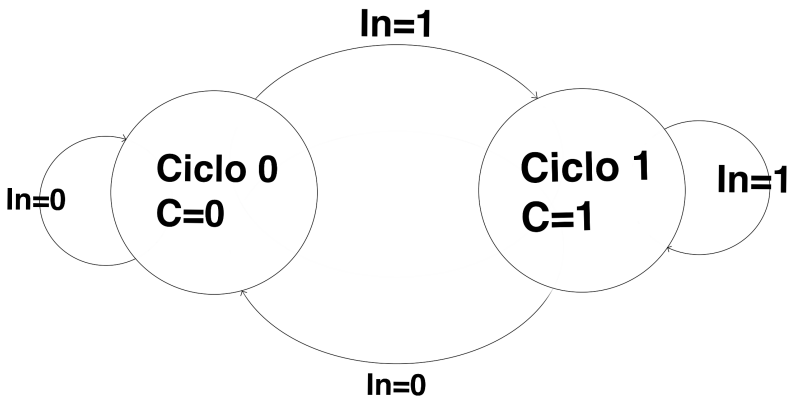
5 Control Unit

5.1 Specifica

Il circuito che si sta progettando necessita di due multiplexer e un demultiplexer, tutti con un singolo segnale di controllo; e di due cicli di clock. Durante il primo ciclo di clock, tutti e tre i componenti devono selezionare l'ingresso/uscita di sinistra (0), mentre nel secondo ciclo di clock viene selezionato l'ingresso/uscita di destra (1). Dal momento che per tutti e tre i componenti controllati dalla CU il segnale assume lo stesso valore all'interno del medesimo ciclo di clock, è possibile utilizzare un unico segnale di controllo (denominato C).

Ciclo di Clock	MUX1	MUX2	DEMUX	C
0	0	0	0	0
1	1	1	1	1

E' possibile implementare questa specifica tramite una macchina a stati finiti che può assumere due stati (Ciclo 0 e Ciclo 1) e produce un unico segnale in uscita. La funzione di uscita dipende unicamente dallo stato presente, mentre la funzione di stato futuro dipende dallo stato presente e dagli ingressi.



5.2 Implementazione della macchina di Moore

Dal momento che la funzione di uscita dipende unicamente dallo stato presente, è possibile implementare la control unit come una macchina di Moore. I due stati della macchina di Moore possono essere codificati con un singolo bit S, e l'ingresso è a sua volta rappresentato da un segnale ad un singolo bit.

In	S	C	Sn
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	0

Le funzioni di uscita e stato futuro risultano dunque essere:
Sn = InS'
C = In'S + InS = S

Lo stato presente viene conservato tramite un FFD-edge triggered, che viene aggiornato solo sul fronte di salita del clock (che in questo caso è rappresentato da un interrutore manovrato manualmente).

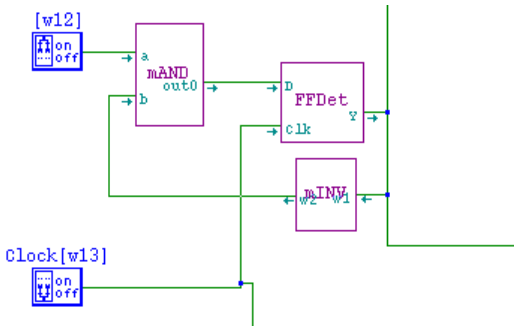


Figura 19: Control Unit

6 Simulazione e analisi del progetto

6.1 Verifica funzionale

In questa sezione sono riportate delle immagini illustrative di alcune simulazioni effettuate per testare la funzionalità del circuito. In tutto sono riportate tre simulazioni, per ciascuna vengono illustrati entrambi i cicli di clock.

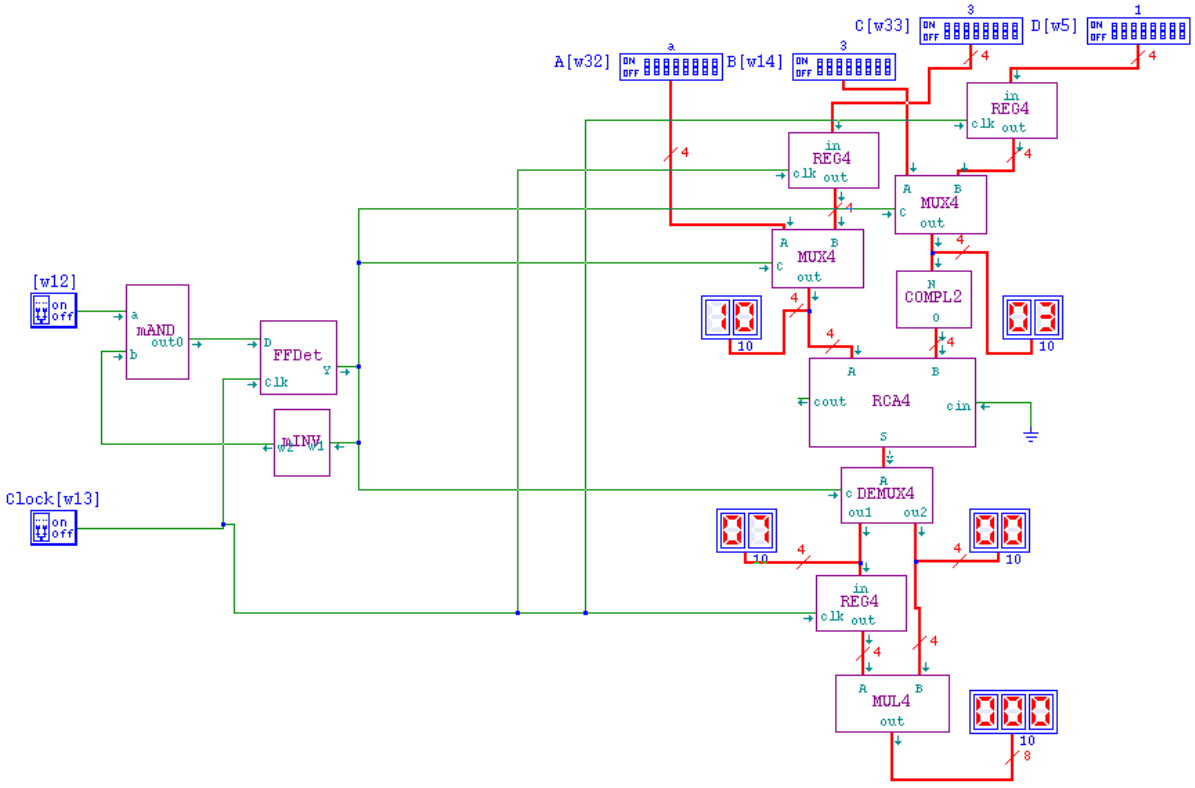


Figura 20: Simulazione 1: ciclo di clock 0

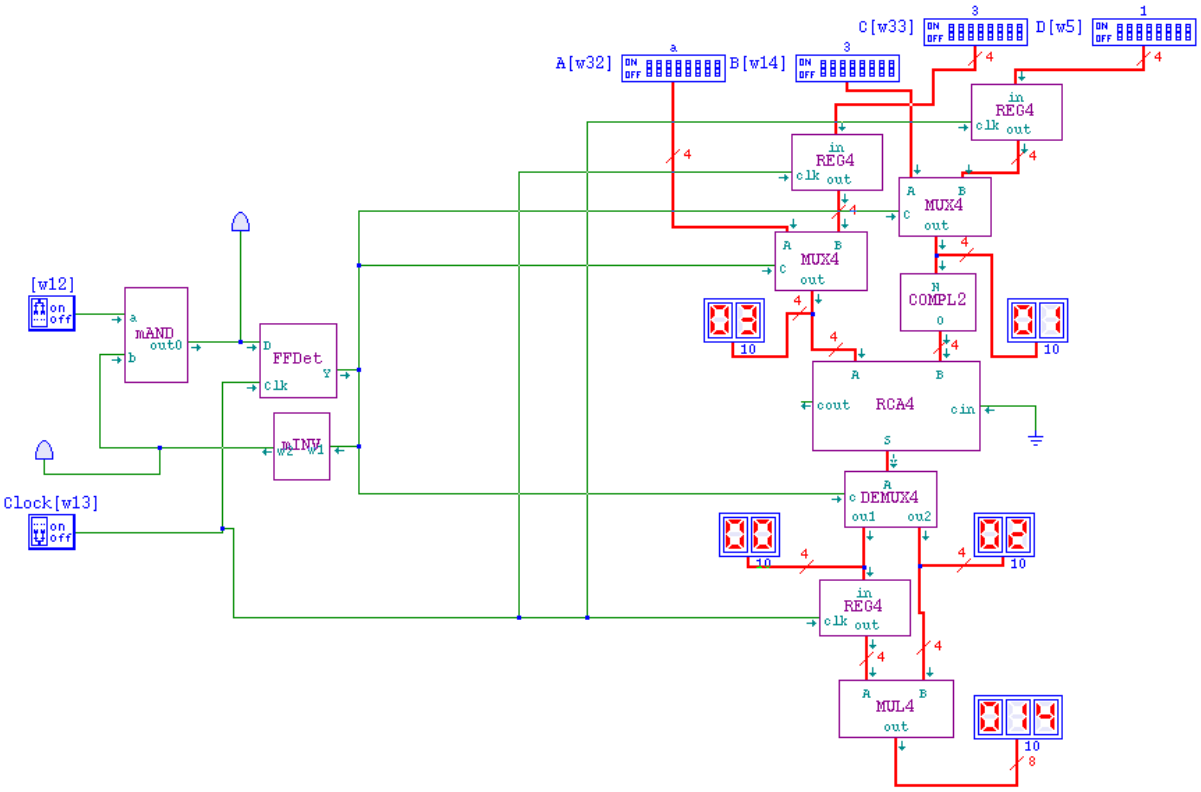


Figura 21: Simulazione 1: ciclo di clock 1

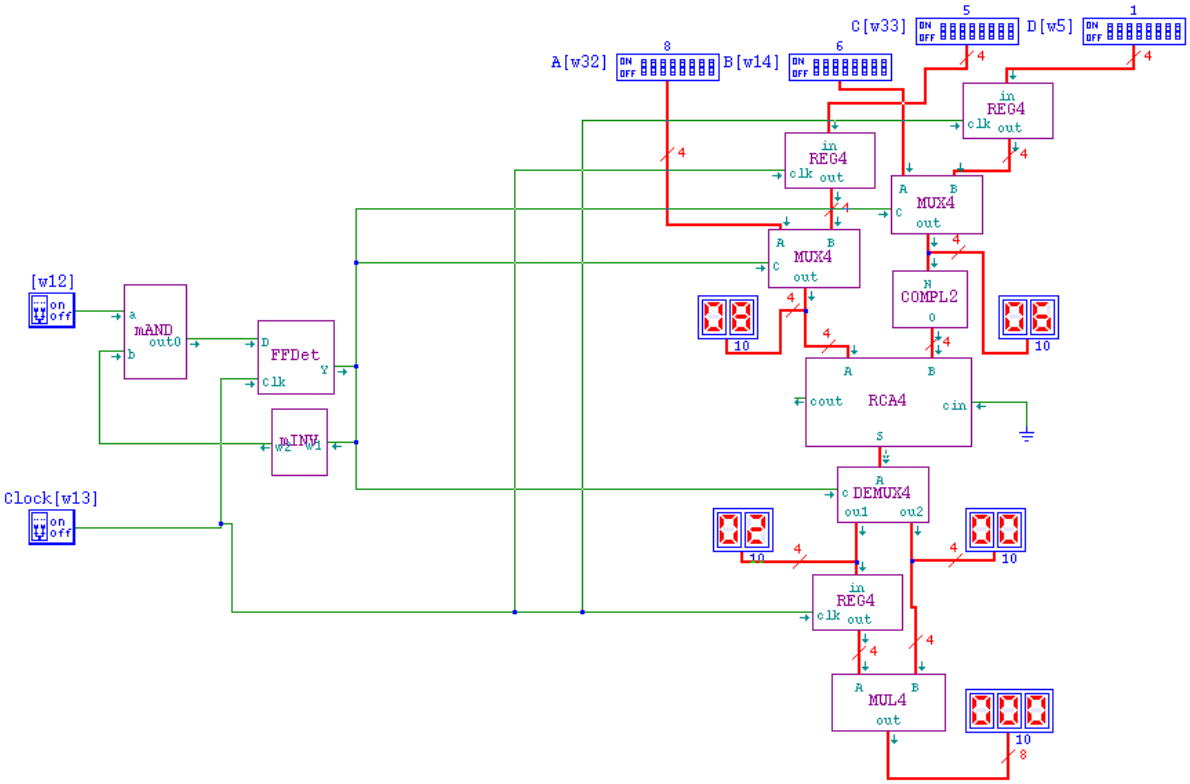


Figura 22: Simulazione 2: ciclo di clock 0

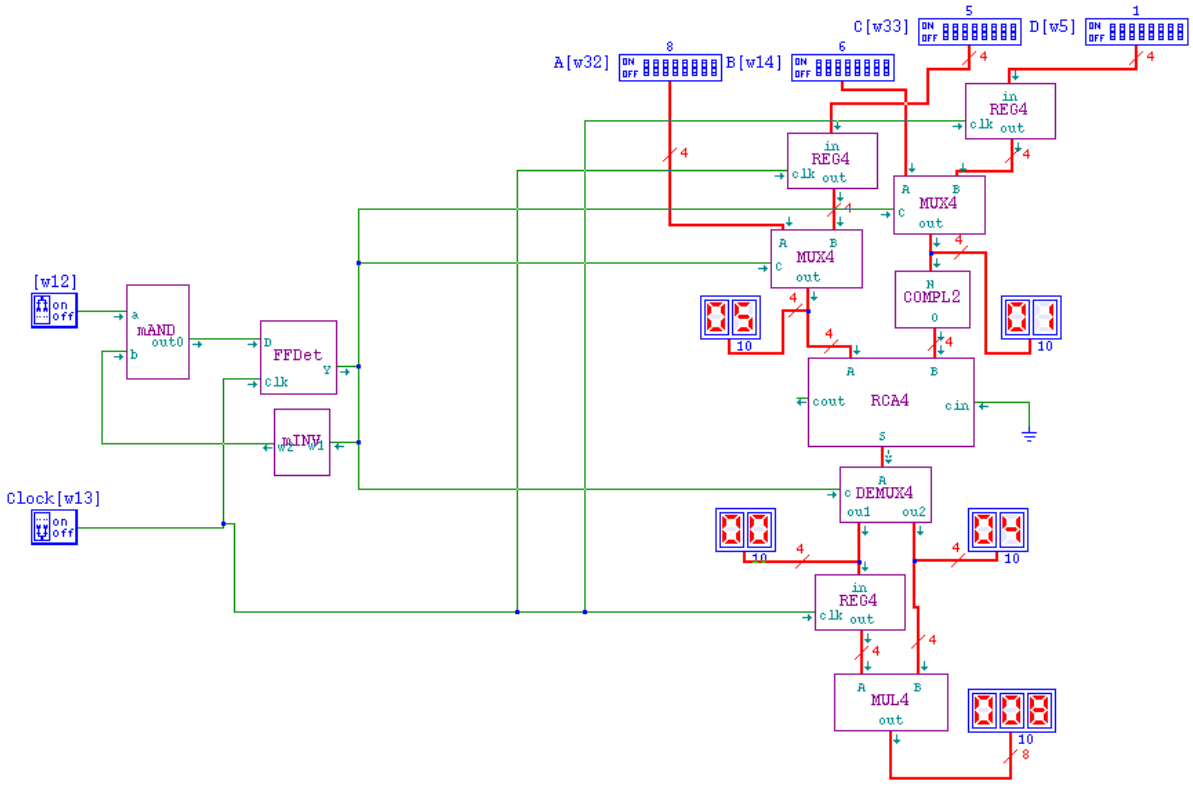


Figura 23: Simulazione 2: ciclo di clock 1

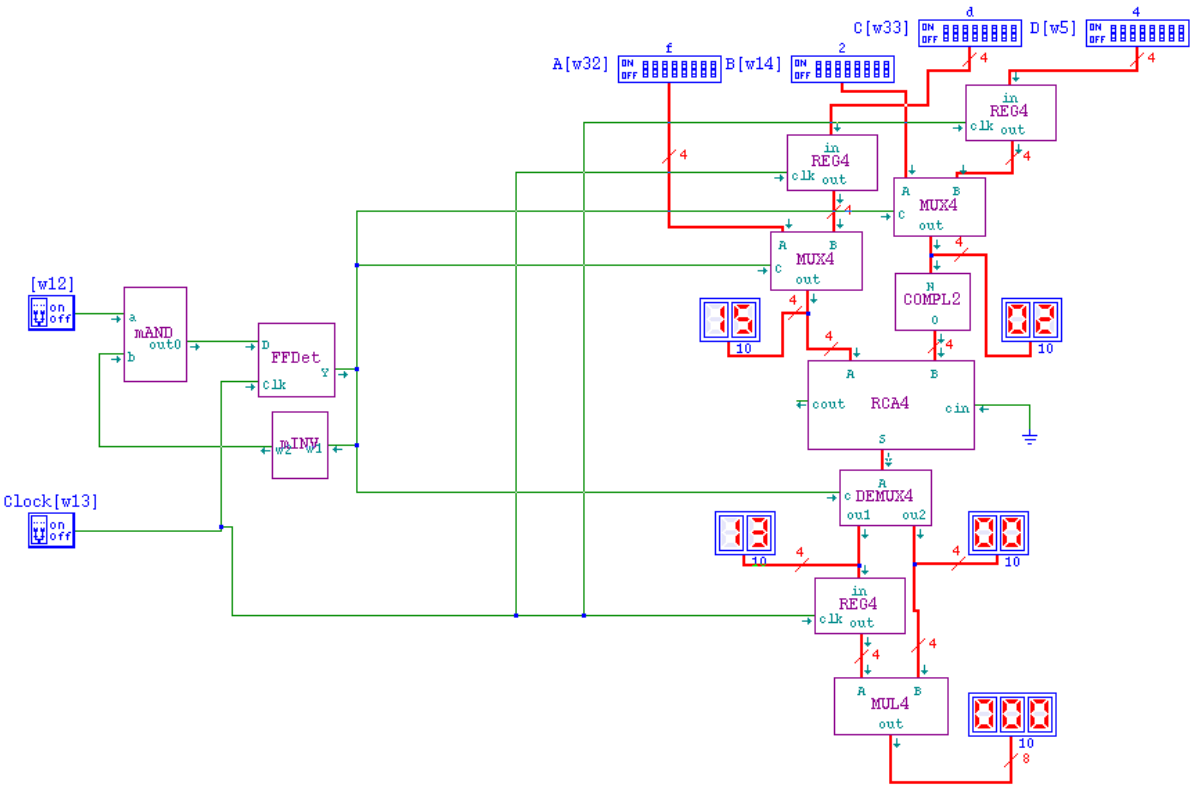


Figura 24: Simulazione 3: ciclo di clock 0

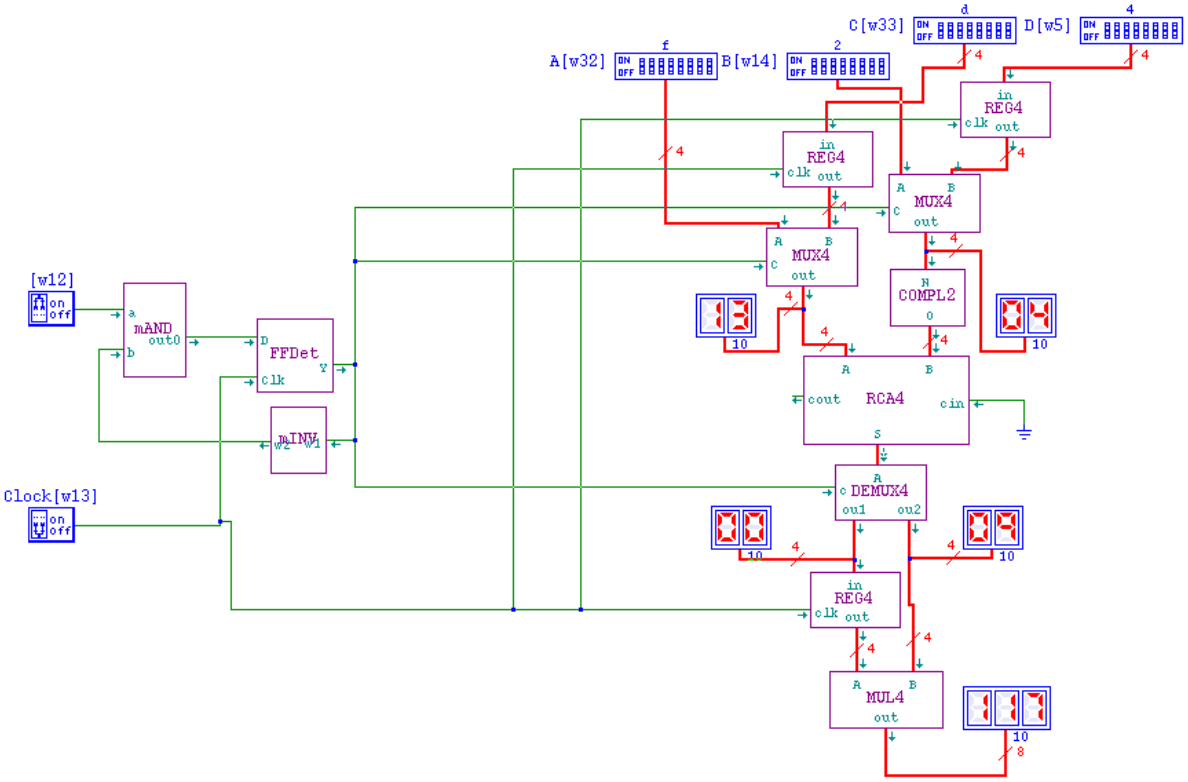


Figura 25: Simulazione 3: ciclo di clock 1

6.2 Stima di complessità circuitale e prestazioni

Per valutare le prestazioni del circuito finale, è necessario introdurre quattro metriche: *complessità circuitale (**Area**), **tempo di propagazione** (il tempo necessario per stabilizzare le uscite), **tempo di contaminazione** (il tempo necessario affinchè si verifichi la prima variazione delle uscite)*, e ***latenza*** *(numero di risultati producibili nell'unità di tempo)*. E' dunque necessario stimare ciascuna di queste metriche per ognuno dei componenti del circuito partendo dal livello gate, dunque dalle porte logiche (semplici e composte). Ognuna di queste metriche viene stimata utilizzando un modello unitario, ovvero ogni porta logica semplice (NOT, NAND e NOR) vale 1. Il tempo di propagazione si calcola stimando il percorso più lungo che i segnali in ingresso devono seguire, il tempo di contaminazione stimando quello più breve, mentre l'area corrisponde semplicemente al numero di porte logiche semplici utilizzate per realizzare il componente.

6.2.1 Prestazioni porte logiche

	NOT	NAND	NOR	AND	OR	EXOR
Area	1	1	1	A(NAND)+A(NOT)=2	A(NOR)+A(NOT)=2	2*A(NOT)+3*A(NAND)=5
Tp	1	1	1	Tp(NAND)+Tp(NOT)=2	Tp(NOR)+Tp(NOT)=2	Tp(NOT)+2*Tp(NAND)=3
Tc	1	1	1	Tc(NAND)+Tc(NOT)=2	Tc(NOR)+Tc(NOT)=2	2*Tc(NAND)=2

6.2.2 Prestazioni multiplexer/demultiplexer

L'area di un multiplexer o demultiplexer a n bit è dipendente linearmente dal numero di bit degli ingressi e delle uscite, in quanto serve un MUX per ciascun bit. Il tempi di propagazione e contaminazione non sono

invece dipendenti dal numero di bit degli ingressi, in quanto i bit delle uscite vengono tutti selezionati in parallelo. Dunque i tempi di propagazione e contaminazione sono gli stessi dei multiplexer/demultiplexer a 1 bit.

	MUX	MUX 4 BIT	DEMUX	DEMUX 4 BIT
Area	2*A(AND)+A(OR)+A(NOT) = 7	4*A(MUX) = 28	2*A(AND)+A(INV)=5	4*A(DEMUX)=20
Tp	Tp(NOT)+Tp(AND)+Tp(OR) = 5	Tp(MUX) = 5	Tp(NOT)+Tp(AND)=3	Tp(DEMUX)=3
Tc	Tc(AND)+Tc(OR) = 4	Tc(MUX) = 4	Tc(AND)=2	Tc(DEMUX)=2

6.2.3 Prestazioni registri

Gli elementi di memoria che memorizzano un solo bit (latch e flip-flop) non hanno dipendendenza dal numero degli operandi. Per i registri parallelo/parallelo, invece, l'area dipende linearmente dal numero di bit da memorizzare. I tempi di propagazione e contaminazione non hanno dipendenza dal numero di bit da memorizzare, in quanto scrittura e lettura avvengono in parallelo per ciascun bit, dunque sia Tp che Tc sono pari a quelli di un singolo FF D edge triggered.

	LatchSR	FFSR	FFDls	FFDet	Registri 4 bit
Area	2*A(NOR) = 2	A(LSR)+2*A(AND)=6	A(FFSR)+A(INV)=7	2*A(FFDls)+A(NOT)=15	4*A(FFDet)=60
Tp	Tp(NOR) = 2	Tp(LSR)+Tp(AND)=4	Tp(NOT)+Tp(FFSR)=5	2*Tp(FFDls)+Tp(INV)=11	Tp(FFDet)=11
Tc	Tc(NOR) = 1	Tc(NOR)+Tc(AND)=3	Tc(FFSR)=3	Tc(FFDls)=3	Tc(FFDet)=3

6.2.4 Prestazioni macro aritmetiche

- **Half Adder e Full Adder:**
  - Il tempo di propagazione dell'HA corrisponde al tempo di propagazione massimo tra quello della porta AND e della porta EXOR, in quanto le due non sono collegate l'una all'altra ma calcolano i risultati in maniera indipendente. Analogamente, il tempo di contaminazione è quello minore tra i tempi di contaminazione delle due porte.
  - Il tempo di propagazione del FA corrisponde al tempo necessario ad attraversare i due HA e la porta OR; mentre il tempo di contaminazione è quello di un singolo HA, in quanto il percorso più breve è quello tra il riporto in ingresso e l'uscita del bit di somma.

	HA	FA
Area	A(AND)+A(EXOR)=7	2*A(HA)+A(OR)=16
Tp	max(Tp(AND), Tp(EXOR))=3	2*Tp(HA)+Tp(OR)=8
Tc	min(Tc(AND),Tc(EXOR))=2	Tc(HA)=2
Rate	1/Tp(HA)	1/Tp(FA)

- **Complementatore a 4 bit:**
  L'area del complementatore è quadraticamente dipendente dal numero degli operandi, in quanto per ogni bit degli ingressi sono necessari un HA e una porta NOT:
  A(C2n) = n*A(HA) + n*A(NOT) = $O(n^2)$. Il percorso più lungo corrisponde al caso in cui il riporto in uscita si propaghi dal primo all'ultimo HA, dunque il tempo di propagazione totale dipende linearmente dal numeri di bit del valore da complementare. Tp(C2n) = Tp(NOT) + n*Tp(HA) = $O(n)$. Il percorso più breve è quello che attraversa un singolo invert e un singolo HA.

	C2
Area	4*A(HA)+4*A(NOT)=32
Tp	4*Tp(HA)+Tp(NOT)=13
Tc	Tc(HA)+Tc(NOT)=3
Rate	1/Tp(C2)

– **RCA4:**
L'area del RCA, così come il tempo di propagazione, dipendono linearmente dal numero di bit degli operandi, in quanto è necessario un FA per ogni coppia di bit. Il tempo di contaminazione è invece sempre pari a quello di un FA.

	RCA4
Area	4*A(FA)=64
Tp	4*Tp(FA)=32
Tc	Tc(FA)=2
Rate	1/Tp(RCA)=0.03

Moltiplicatore:
Il moltiplicatore descritto in precedenza conta in totale 16 porte AND per il calcolo dei prodotti tra singoli bit, 2 HA e 8 FA. L'area corrisponde quindi alla somma delle aree dei componenti elencati. Il percorso più lungo che i segnali in ingresso devono percorrere è quello dei due bit in ingresso al primo HA, nel caso in cui ci sia sempre in riporto in uscita da propagare alla colonna successiva, e dunque il segnale attraversa tutti i full-adder e half-adder. Il percorso più breve è invece quello del prodotto tra i primi due bit di ciascun moltiplicando, che non devono essere sommati ad altri prodotti, e dunque attraversano una sola porta AND.

	MUL4
Area	16*A(AND)+8*A(FA)+2*A(HA)=174
Tp	2*Tp(AND)+2*Tp(HA)+8*Tp(FA)=74
Tc	Tc(AND)=2
Rate	>1/Tp(MUL)=0.01

6.2.5 Circuito Completo

– L'area del circuito completo è costituita dalla somma delle aree di tutti i suoi componenti. **A(TOT)** = 2*A(MUX)+A(C2)+A(DEMUX)+A(RCA4)+A(MUL4)+3*A(REG4) = 526

– Il percorso più lungo all'interno del circuito è quello dei sottraendi, che attraversano tutti i componenti del circuito. Il tempo di propagazione corrisponde dunque alla sommma dei Tp dei suoi componenti. **Tp(TOT)** = Tp(MUX)+Tp(RCA4)+Tp(DEMUX)+Tp(MUL4)+Tp(C2)+Tp(REG4) = 149

– Il percorso più breve è quello dei minuendi, che non attraversano il complementatore.
Tc(TOT) = 2*Tc(MUX)+Tc(DEMUX)+Tc(RCA4)+Tc(MUL4)+3*Tc(REG4) = 32

Quindi la durata minima che il ciclo di clock può assumere è pari a quella necessaria a completare il secondo ciclo di clock, in quanto durante il primo non viene eseguita la moltiplicazione. Il tempo necessario a concludere i calcoli per il secondo ciclo di clock è uguale al tempo di propagazione totale del circuito, più il tempo di propagazione del registro contenente il risultato del primo ciclo di clock.

TClock > Tp(TOT) + Tp(REG4) = 160

I tempi di propagazione e contaminazione sono stati stimati sul circuito completo tramite simulazione:

Osservando l'immagine sopra riportata, e assumendo di far corrispondere il modello unitario precedentemente adottato ad 1ns, è possibile stabilire che i tempi di propagazione e contaminazione corrispondono, con ragionevole approssimazione, a quelli stimati.

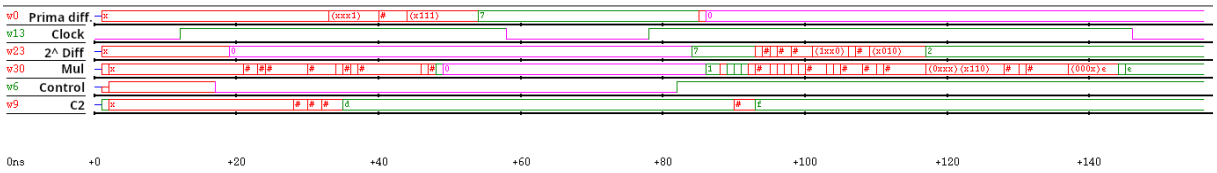


Figura 26: Simulazione dei tempi di propagazione e contaminazione