1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

DISPEA
DIPARTIMENTO DI
SCIENZE PURE E
APPLICATE

Relazione Progetto d'esame
Reti Logiche

2023/2024

AUTORI

Mattia Gasperoni
matricola: 329235

Andrea Marchionni
matricola: 326970

Claudio Valentin Gaspar
matricola: 328833

1 Specifica

1.1 Scopo del progetto

Lo scopo di questo progetto è la costruzione di un circuito attraverso l'utilizzo del simulatore circuitale TkGate che deve essere in grado di risolvere la seguente espressione aritmetica:

$$f : (x1 * c1) + (x2 * c2) + (x3 * c3) - (x4 * c4)$$

1.2 Specifica funzionale

Essendo il progetto strutturato con input di 4 bit verranno considerati validi solamente quei valori (decimali) esprimibili con 4 bit. Utilizzando le regole della notazione posizionale per i numeri interi senza segno, con la formula $b^n - 1$, dove b è la base del sistema posizionale di partenza e n il numero di posizioni avremo $2^4 - 1 = 16 - 1 = 15$; potremmo codificare valori interi decimali da 0 a 15 quindi un totale di 16 valori decimali.

Il primo modulo che troviamo è la moltiplicazione di due valori a 4bit e restituisce un valore a 8bit.

Il calcolo dei bit necessari per un prodotto è semplicemente deducibile dalla seguente formula $n_a + n_b = n_p$ dove n_a sono i numero dei bit del moltiplicatore, n_b sono il numero dei bit del moltiplicando e n_p è il numero massimo di bit necessari per rappresentare il prodotto.

Dopo aver effettuato le prime due moltiplicazioni si effettuerà la loro addizione e successivamente verrà effettuata l'addizione della terza moltiplicazione e infine ci sarà la sottrazione della quarta moltiplicazione con il resto dell'espressione, verrà quindi generato il risultato finale e si potrà visualizzare tramite un led a 7 segmenti di tipo decimale.

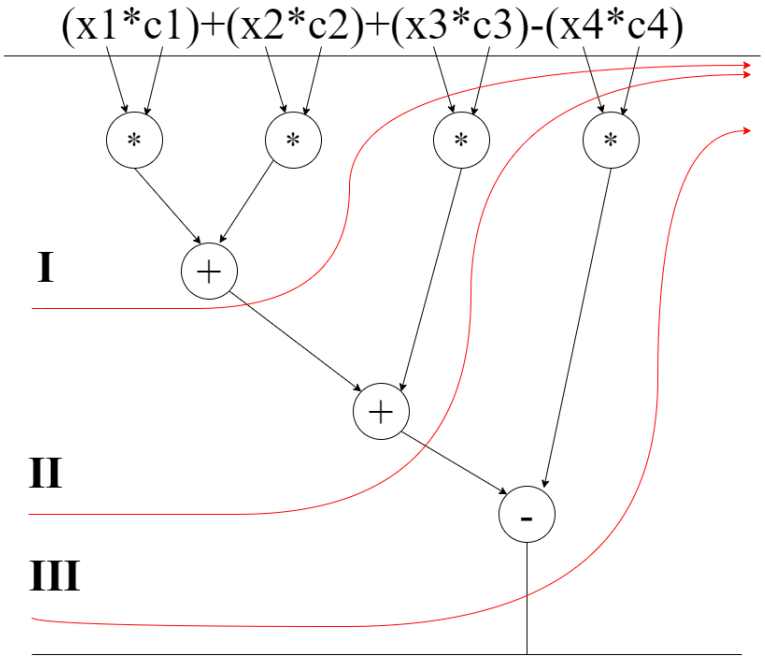
1.3 Specifica parametrica

Abbiamo deciso di suddividere l'esecuzione dell'espressione in 3 cicli di clock, abbiamo anche implementato il

Resource Sharing di 2 addizioni e di 2 moltiplicazioni.

2 Impostazione del progetto a livello RT

2.1 Data Flow Graph



Come si vede dall'immagine abbiamo diviso l'esecuzione dell'espressione in 3 cicli di clock.
Il Data Flow Graph mostra che vi sono alcuni elementi che possono essere riutilizzabili in un ottica di Resource Sharing.

Questi elementi sono:

1. Due addizionatori indicati con $+$.
2. Due moltiplicatori indicati con $*$.

2.2 Risorse

Le risorse utilizzate per lo sviluppo di questa espressione aritmetica sono:

- 10 Registri a 4bit
- 1 Registro a 8bit
- 1 Multiplexer a 8bit e 2 Multiplexer a 4bit
- 1 Control Unit (CU)
- 5 Macro aritmetiche:
  - 3 MUL a 4bit
  - 1 ADD a 8bit
  - 1 SUB a 8bit

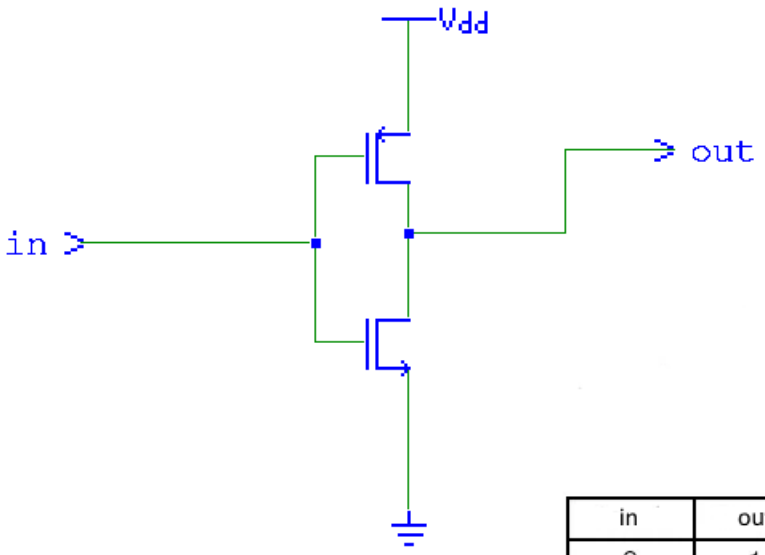
3 Progetto delle risorse a livello gate

3.1 Porte logiche elementari

Le porte logiche elementari sono le porte con le quali si può costruire ogni altra porta logica, inclusi i componenti complessi, esse sono *NOT*, *NAND* e *NOR*, ognuna di essa è composta da transistor P-MOS e da N-MOS.

3.1.1 INV

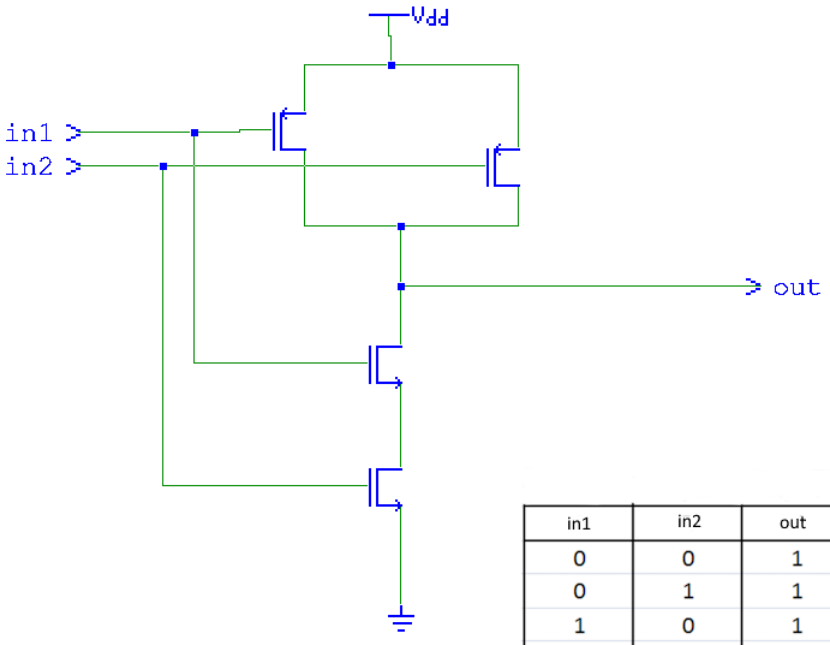
La porta logica *INV* o inverter, è l'unica porta logica che riceve un solo segnale in ingresso, implementa l'operatore logico *NOT*, producendo come segnale d'uscita *out* la negazione, ovvero l'opposto del segnale d'ingresso *in*.



in	out
0	1
1	0

3.1.2 NAND

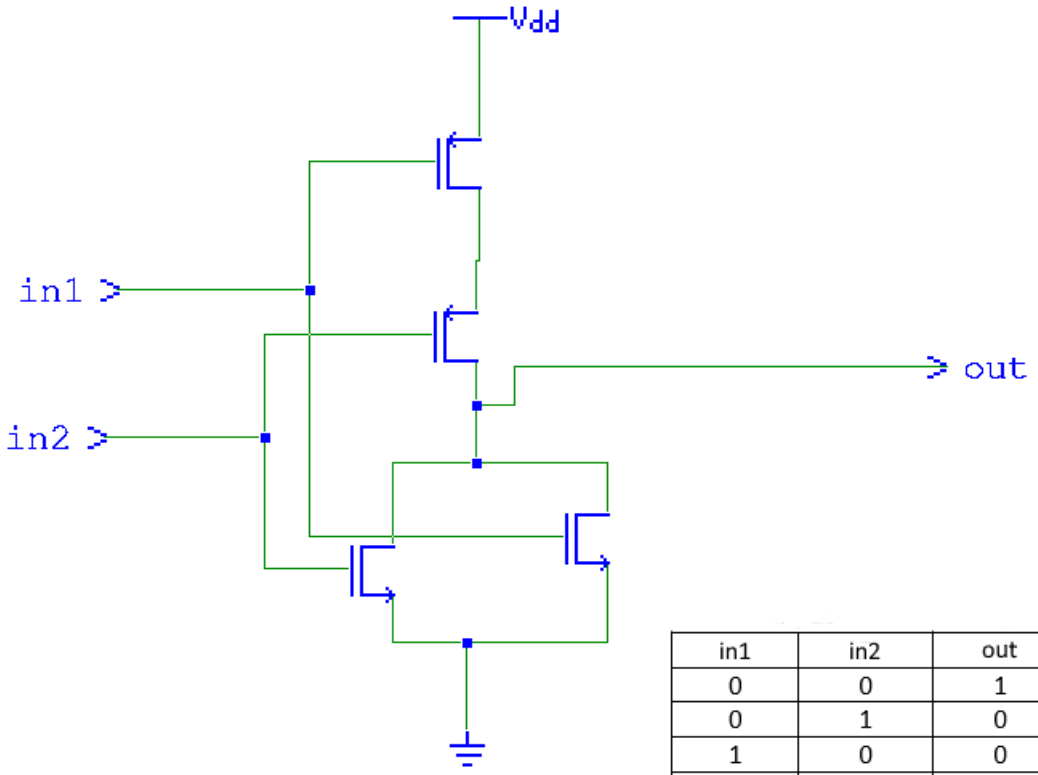
La porta logica *NAND* riceve in entrata due input e restituisce un solo output, effettua la negazione di una porta logica *AND*, perciò restituisce 0 solo quando entrambi i valori di input sono 1.



in1	in2	out
0	0	1
0	1	1
1	0	1
1	1	0

3.1.3 NOR

La porta logica *NOR* riceve in entrata due input e restituisce un solo output, effettua la negazione di una porta logica *OR*, perciò restituisce 1 solo quando entrambi i valori di input sono 0.



in1	in2	out
0	0	1
0	1	0
1	0	0
1	1	0

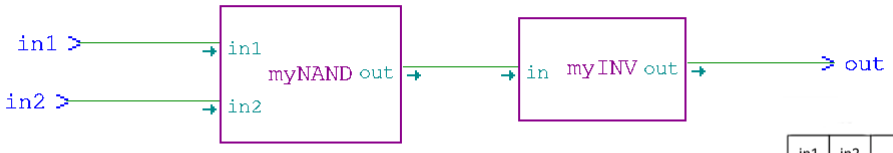
3.2 Porte logiche composte

3.2.1 AND

La porta logica *AND* è composta da un *NAND* che restituisce un valore che viene invertito dall'*INV* prima di essere restituito.

La sua funzione infatti è: (AB) in questo caso però è più correttamente descrivibile con la formula $((AB)')'$.

Nel caso in cui entrambi gli ingressi siano 1, otterremo un valore di uscita pari a 1, mentre se uno o entrambi gli ingressi valgono 0 il risultato dell'*AND* sarà 0.



in1	in2	out
0	0	0
0	1	0
1	0	0
1	1	1

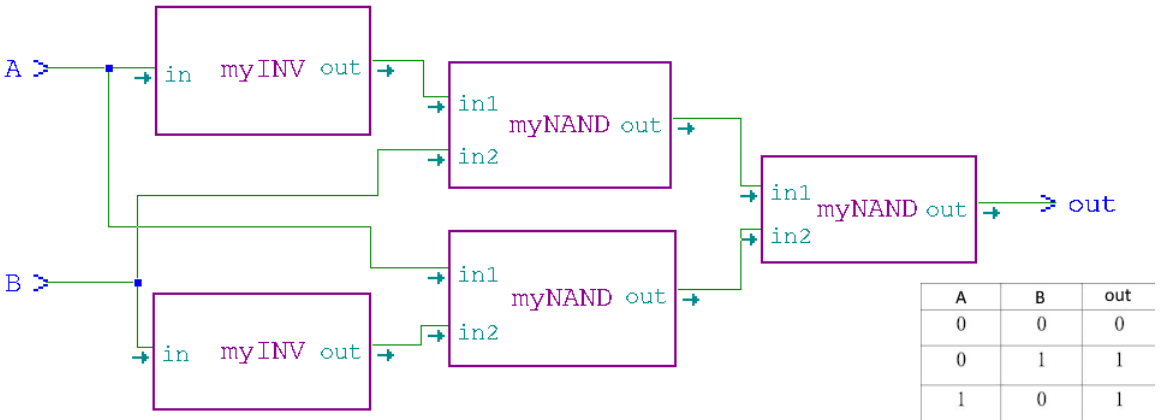
3.2.2 EXOR

La porta logica *EXOR* è composta da due inverter e tre nand ed è stata utilizzata per la creazione del Full Adder.

La sua funzione logica è $(A \oplus B)$, però in questo caso specifico è più corretto esprimerla come $((AB')'(A'B)')'$.

Gli input vengono entrambi inseriti in due *NAND*: uno con il primo ingresso negato ottenendo la parte di funzione $(A'B)'$, e l'altro *NAND* in cui ad essere negato questa volta sarà il secondo ingresso, ottenendo la seconda parte della funzione $(AB')'$, in questo momento quindi la funzione è divisa nelle due parti come $(AB')'\&(A'B)'$.

Successivamente i due output dei *NAND* verranno inseriti a loro volta in terzo *NAND*, unendo le due metà della funzione con un *NAND* otteniamo quindi la funzione $((AB')'(A'B)')'$.



A	B	out
0	0	0
0	1	1
1	0	1
1	1	0

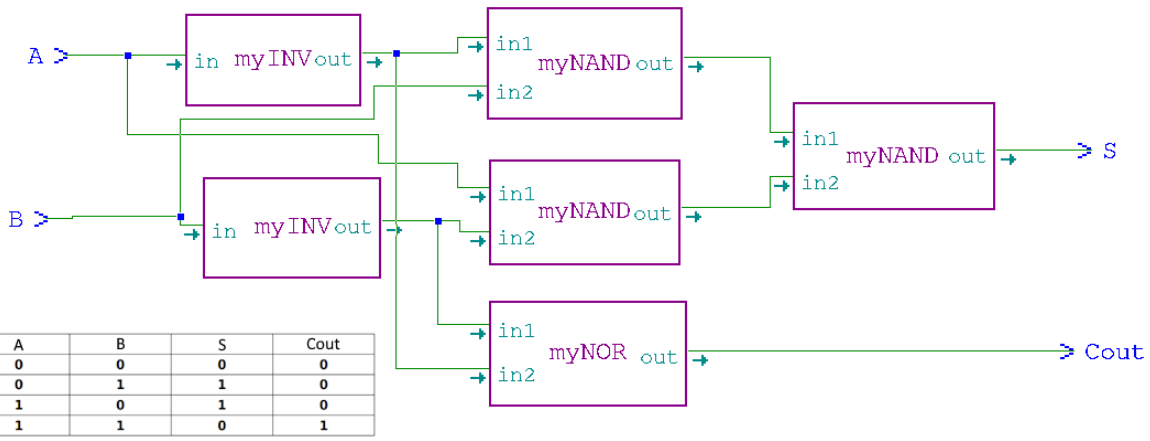
Tempi di Propagazione - Contaminazione - Area

Porte logiche Semplici e Complesse

PORTA LOGICA	TP	TC	AREA
INV	**1**	**1**	**1**
NAND	**1**	**1**	**1**
NOR	**1**	**1**	**1**
AND	**2**	**2**	**2**
OR	**2**	**2**	**2**
EXOR	**3**	**2**	**5**

3.2.3 Half Adder

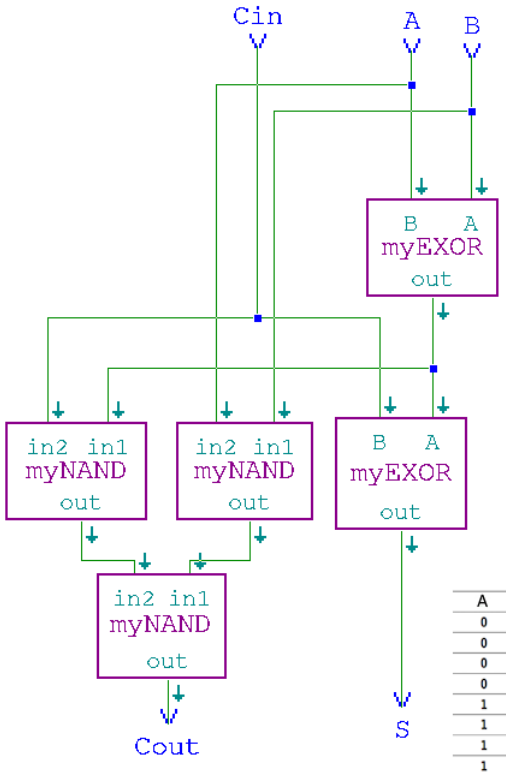
L'Half Adder calcola la somma dei due input di ingresso e la presenza di un eventuale riporto ($Cout$), esso però non può utilizzare un riporto fornitogli in ingresso, ha quindi due ingressi (i due valori A e B) e due uscite (la somma e l'eventuale riporto).



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

3.2.4 Full Adder

Il Full Adder, diversamente dall'Half Adder, ha tre ingressi e due uscite, come output fornirà gli stessi dati dell'Half Adder, ovvero la somma tra i due input, e la presenza di un eventuale riporto proveniente dall'addizione; tuttavia come input aggiuntivo gli deve essere fornito l'eventuale riporto proveniente dall'Half-Adder (o Full-Adder) precedente.



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

3.3 Registri

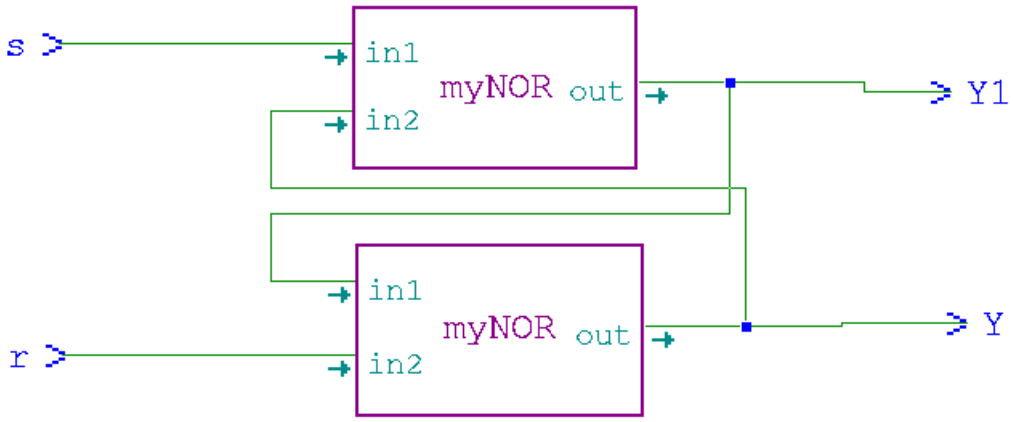
Nel nostro circuito utilizziamo dei registri a 4 bit e, come vedremo successivamente, uno a 8 bit.
Un registro viene impiegato quando vi è la necessità di memorizzare un valore, per esempio un risultato oppure un operando.
Un registro è costituito da una batteria di elementi, detti Flip Flop, che consentono il campionamento di n bit di informazione.
Ne consegue che per un registro a 4 bit avremo 4 Flip Flop, per 8 bit ne avremo 8 e così via.
Questi Flip Flop appartengono alla classe dei bistabili, in quanto sono caratterizzati da due stati stabili: valore logico 0 stato 0 e valore logico 1 stato 1.
Il passaggio da uno stato all'altro dipende da un evento esterno, in assenza del quale il valore memorizzato viene preservato senza variazioni fintanto che' il bistabile rimane alimentato.

3.3.1 Latch SR

Il latch è un circuito elettronico bistabile, quindi caratterizzato da almeno due stati stabili, in grado di memorizzare un bit di informazione.
Il suo funzionamento è cosi spiegato:

- s=1, r=0: avremo una condizione di set, ovvero l'uscita Y varrà 1.
- s=0, r=1: avremo una condizione di reset, ovvero l'uscita Y varrà 0.
- s=0, r=0: avremo una condizione di hold, ovvero l'uscita dipende dal valore dei risultati precedenti e in particolare si manterrà lo stato del circuito.

Tp = Nor = 1 — Tc = Nor = 1 — A = 2(Nor) = 2

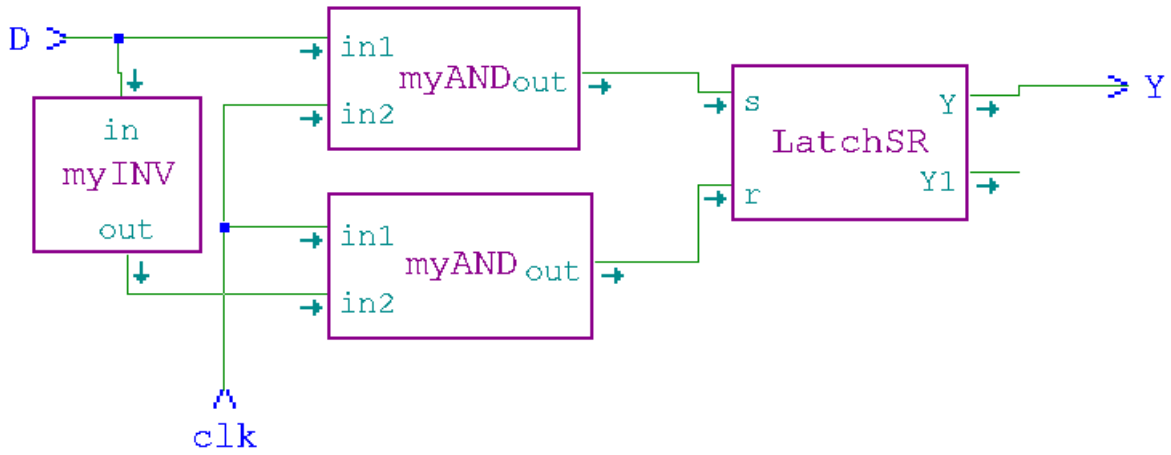


3.3.2 Flip Flop tipo D Level Sensitive

Il Flip Flop tipo D level sensitive ha un unico ingresso che va a gestire entrambi i valore di s di r del *LatchSR*, quando il segnale *clk* è uguale a 0 il FFDls si mantiene in hold mentre quando il segnale *clk* è uguale a 1 si potranno verificare queste due condizioni:

- D = 1, Y assume valore 1 indipendentemente dal valore precedente dello stato.
- D = 0, Y assume valore 0 indipendentemente dal valore precedente dello stato.

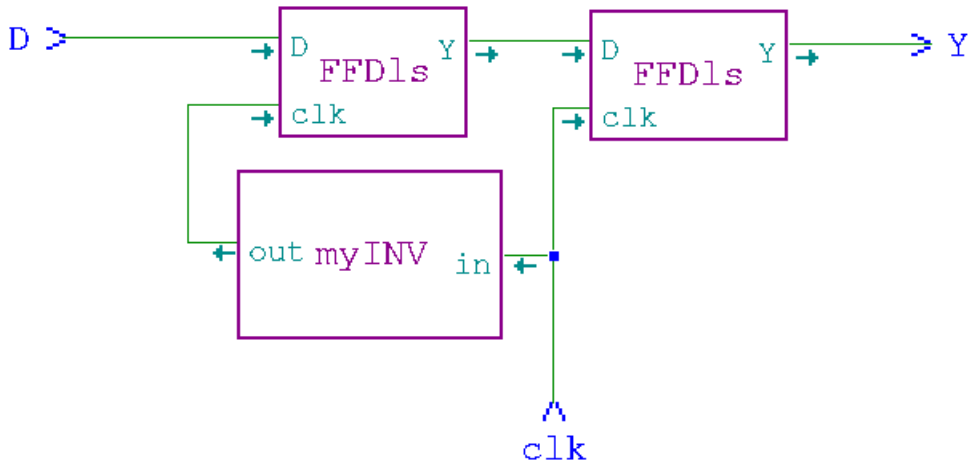
Tp = Inv + And + LatchSR = 4 — Tc = And + LatchSR = 3 — A = Inv + 2(And) + LatchSR = 7



3.3.3 Flip Flop tipo D Edge Triggered

Il Flip Flop tipo D edge triggered può controllare i cambiamenti di valore anche quando il segnale di *clk* vale 0 per farlo si usano due FFDls, uno con il *clk* in forma normale e l'altro con il *clk* in forma negata.
Il FFDet è quello che sarà alla base per la costruzione dei registri.
Infatti un registro può essere definito come una batteria di FFDet disposti in parallelo.

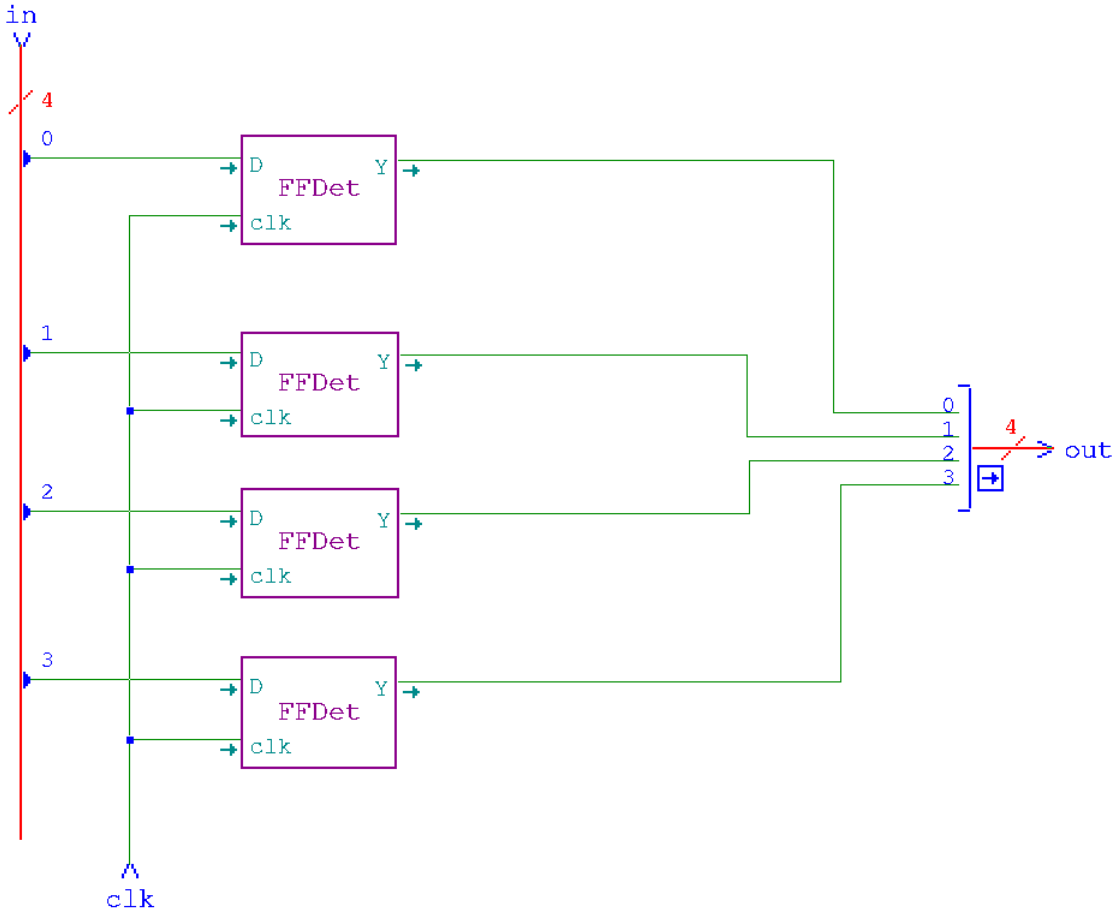
Tp = Inv + 2(FFDls) = 9 — Tc = FFDls = 3 — A = Inv + 2(FFDls) = 15



3.3.4 Registro a 4bit

Servono per memorizzare una valore binario di ampiezza massima di 4bit.
Li abbiamo utilizzati per il campionamento degli ingressi prima di effettuare le moltiplicazioni.

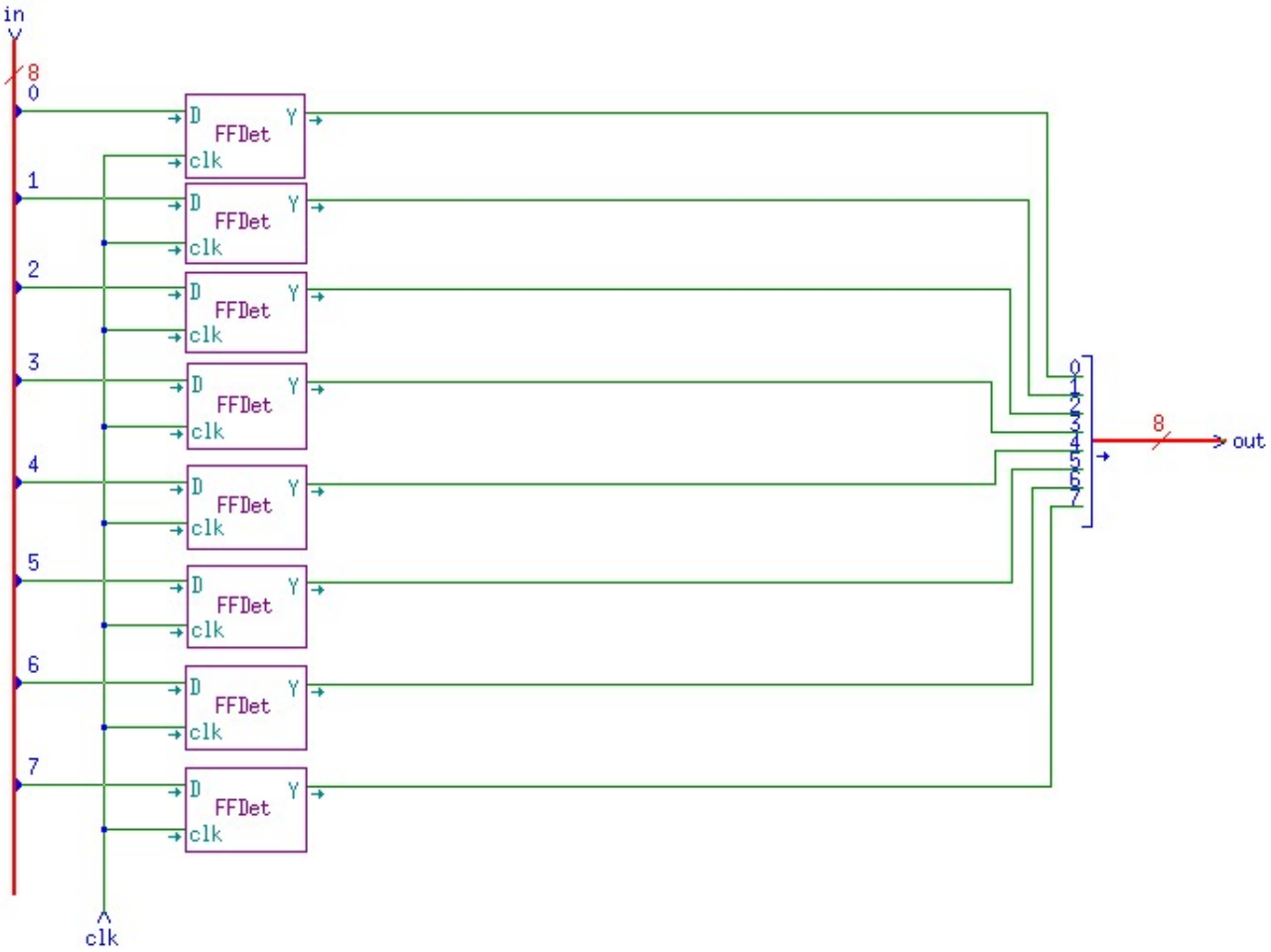
Tp = FFDet = 9 — Tc = FFDet = 3 — A = 4(FFDet) = 60



3.3.5 Registro a 8bit

Servono per memorizzare una valore binario di ampiezza massima di 8bit.
Li abbiamo utilizzati per campionamento del risultato dell'addizione e per effettuare il Resource Sharing.

Tp = FFDet = 9 — Tc = FFDet = 3 — A = 8(FFDet) = 120

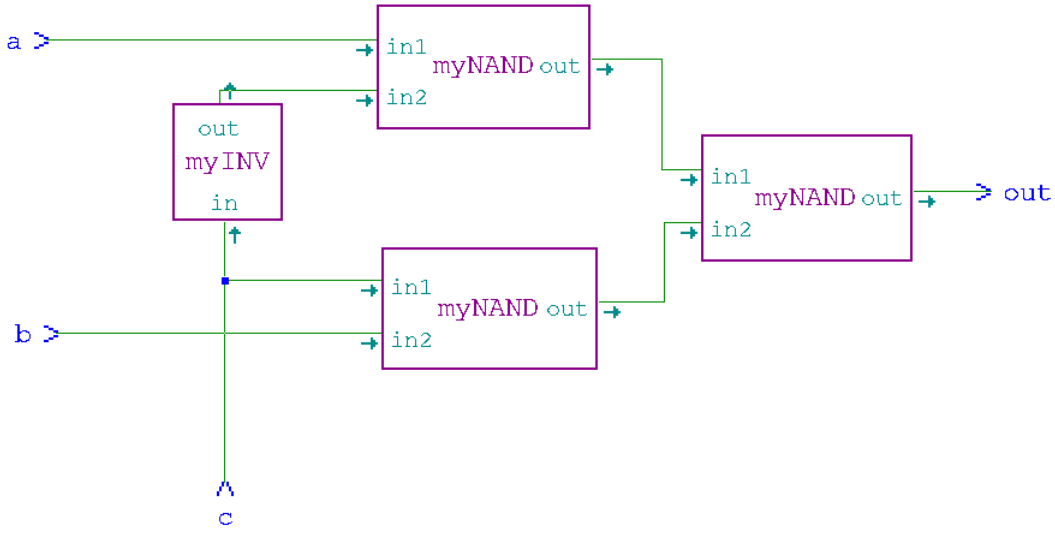


3.4 Multiplexer

Il modulo del multiplexer (MUX) e composto da 3 ingressi e 1 uscita.
La funzione di questo componente e quella di riportare come output uno dei due valori di input a seconda del valore di un controllo.
In particolare se:

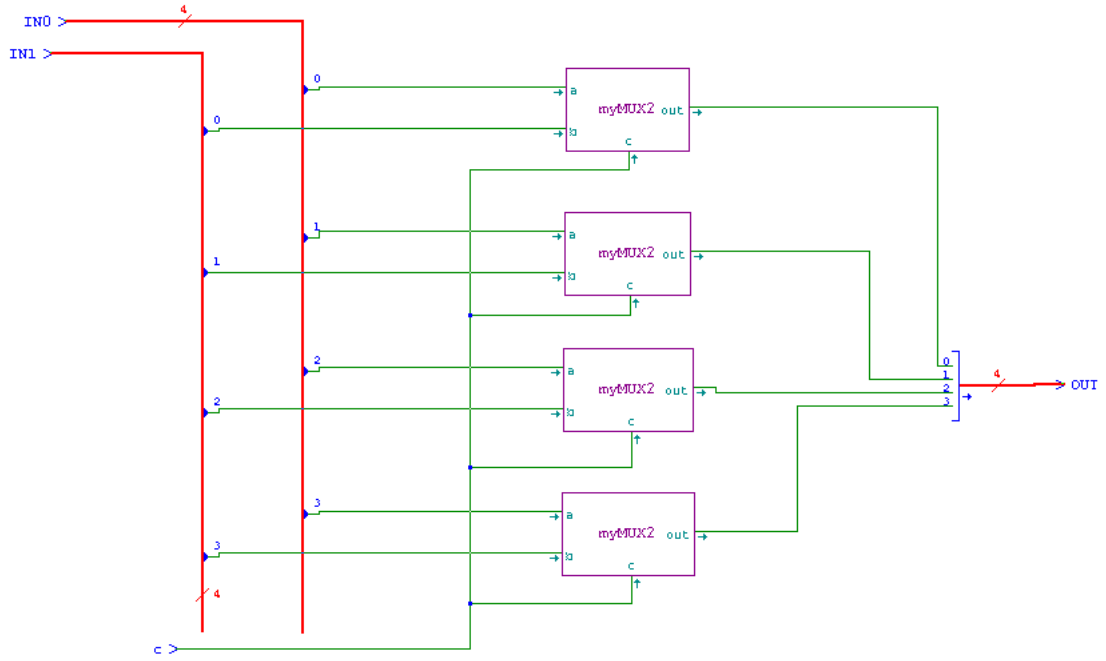
- c = 0, l'output sarà a
- c = 1, l'output sarà b

Tp = Inv + 2(Nand) = 3 — Tc = 2(Nand) = 2 — A = Inv + 3(Nand) = 4



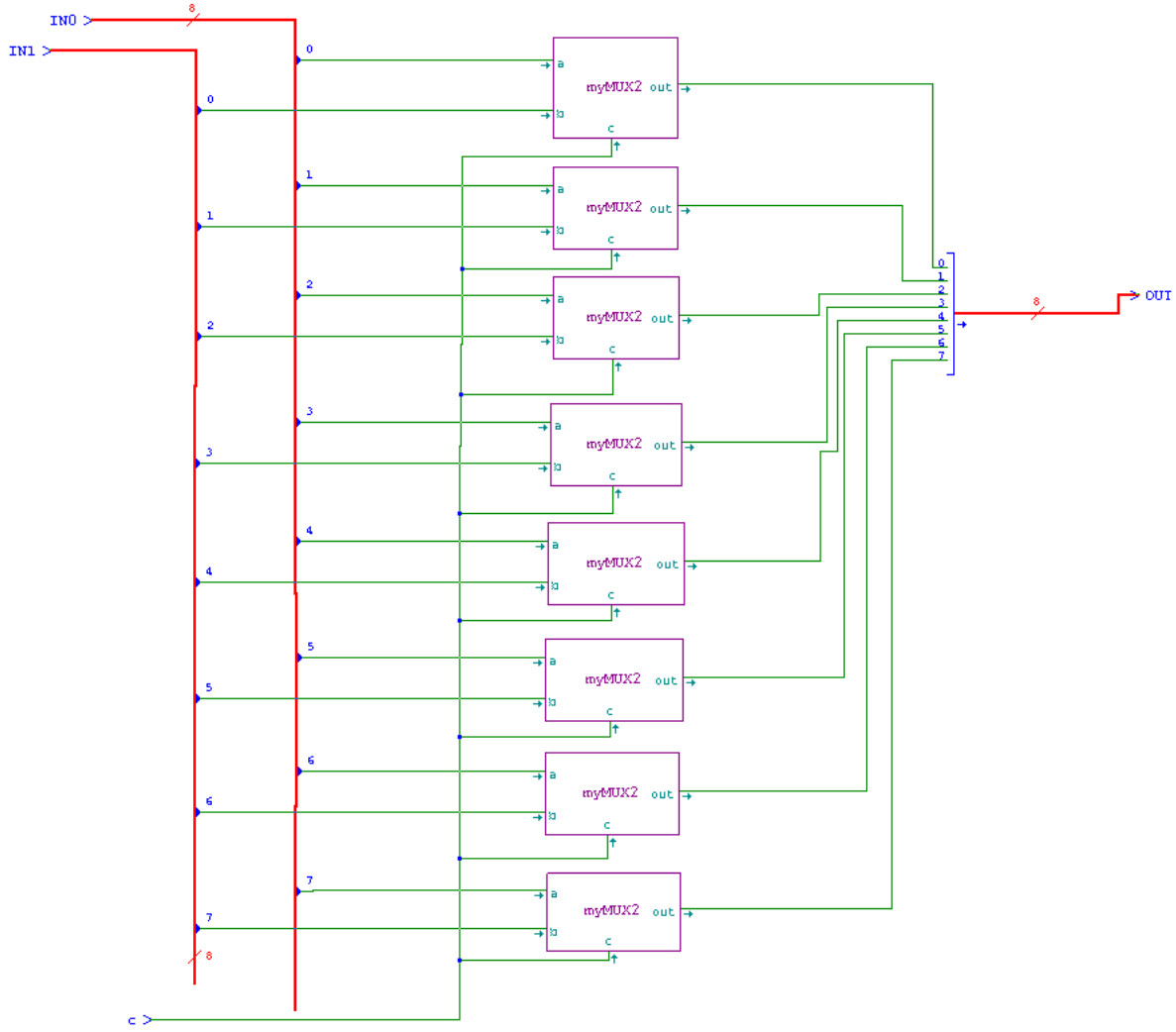
3.4.1 MUX2x4

I MUX2x4 sono posizionati prima di una moltiplicazione in modo da poter usufruire del Resource Sharing, ossia la possibilità di eseguire due moltiplicazioni utilizzando un solo moltiplicatore.
Tp = MUX = 3 — Tc = MUX = 2 — A = 4(MUX) = 16



3.4.2 MUX2x8

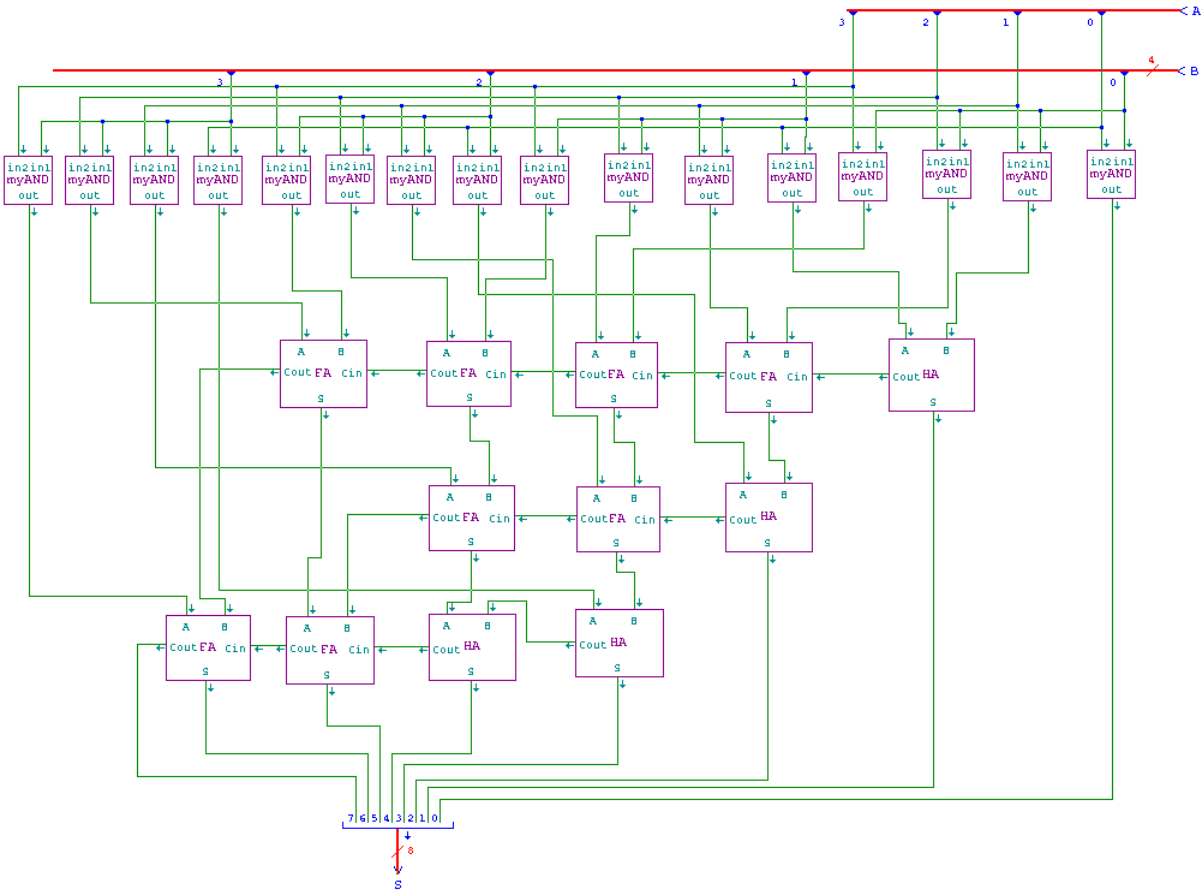
Il MUX2x8 è posizionato prima della somma e permette di effettuare il Resource Sharing dell'addizione andandola ad effettuare due volte usando un solo addizionatore.
Tp = MUX = 3 — Tc = MUX = 2 — A = 8(MUX) = 32



3.5 Moltiplicatore

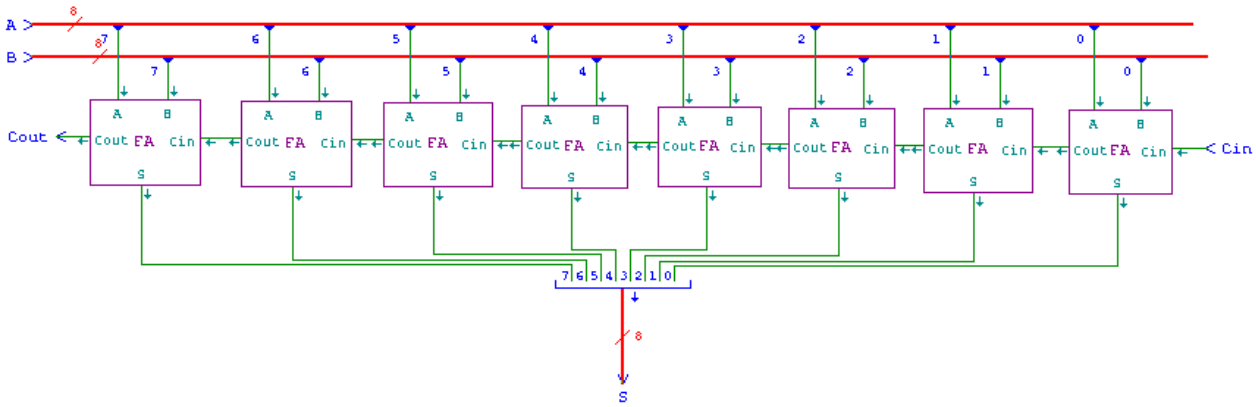
Il moltiplicatore MUL si occupa di eseguire la moltiplicazione dei suoi operandi in ingresso di 4 bit e genera un risultato S di massimo 8bit, è composto dalle seguenti porte logiche:

- AND
- Half Adder
- Full Adder



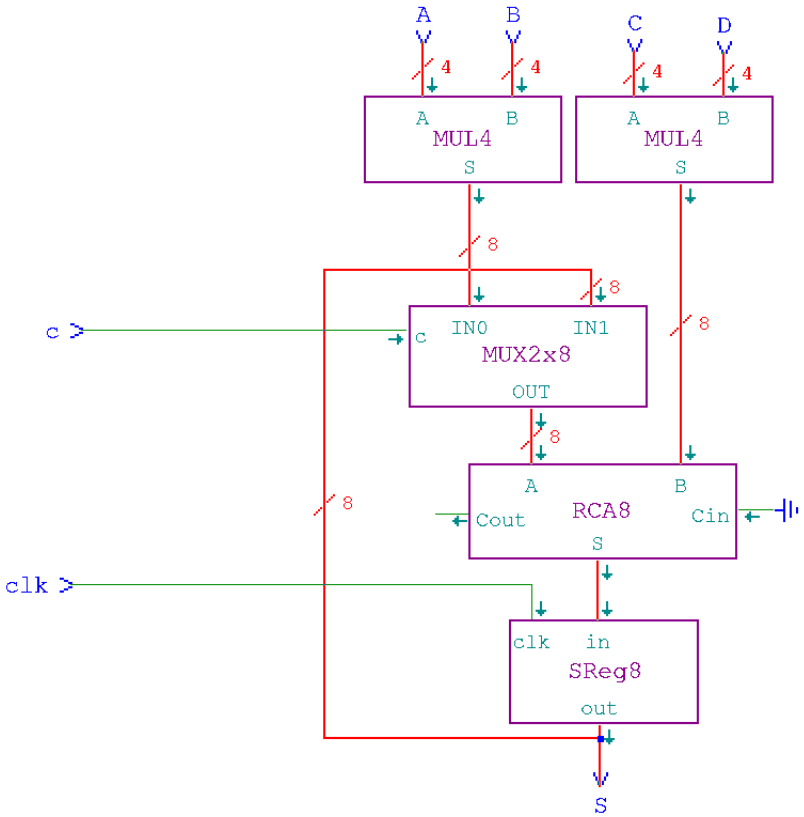
3.6 Addizionatore

L'addizionatore RCA8 si occupa di eseguire la somma dei suoi operandi in ingresso ed il riporto in ingresso indicato come *Cin*.
Nella sua uscita vi è quindi il risultato della somma e l'eventuale riporto indicato come *Cout*.
L'elemento di base necessario per sua la costruzione è il Full Adder (FA) che prende i due bit da sommare e l'eventuale riporto e genera la somma e l'eventuale riporto.



3.7 MUL_ADD

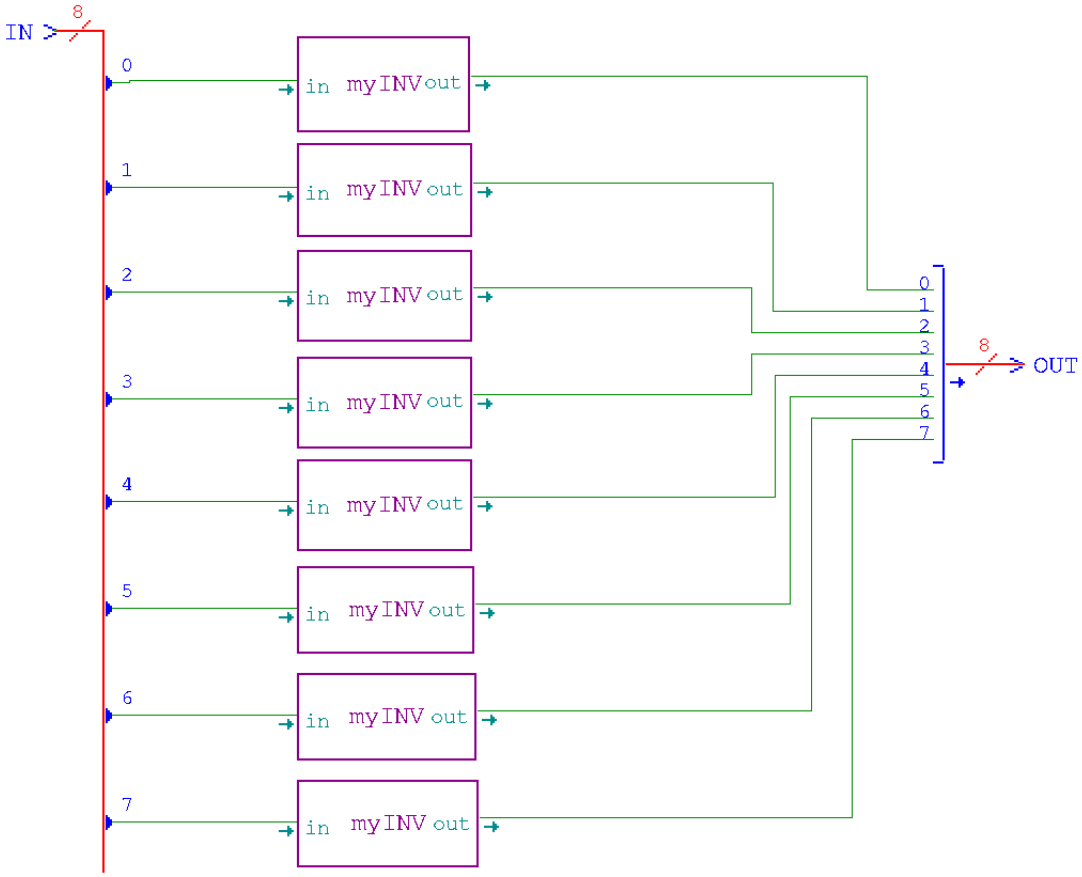
Abbiamo implementato questo modulo per incrementare la leggibilità del nostro circuito, è composto da 4 ingressi A,B,C,D,c e clk e un'unica uscita S. Al suo interno troviamo i moduli per effettuare le moltiplicazioni e le addizione in Resource Sharing.



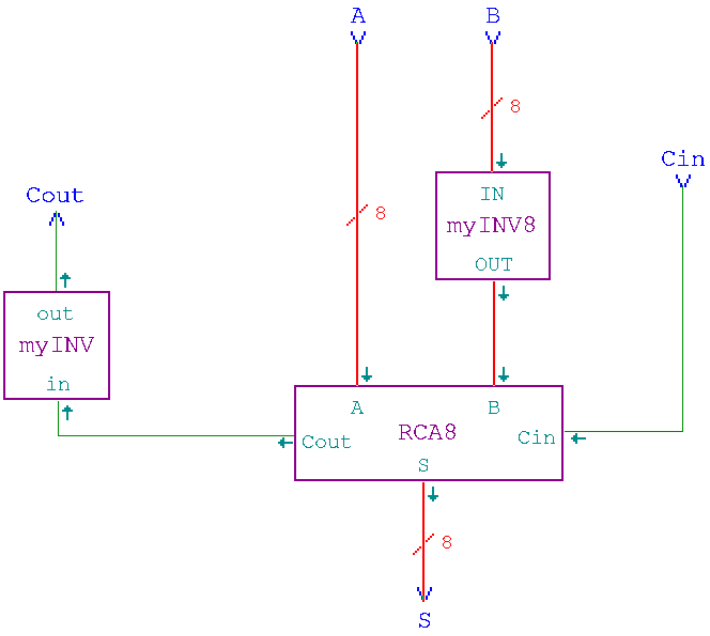
3.8 Sottrattore

Il sottrattore SUB8 si occupa di eseguire la differenza dei suoi operandi in ingresso.
E' composto da un inverter a 8bit e da un addizionatore a 8bit, abbiamo pensato di inserire un led che normalmente rimane spento ma quando si ottiene un valore da interpretare si accende.
Per valori da interpretare intendiamo i valori ottenuti quando il minuendo è minore del sottraendo.

3.8.1 INV 8 bit

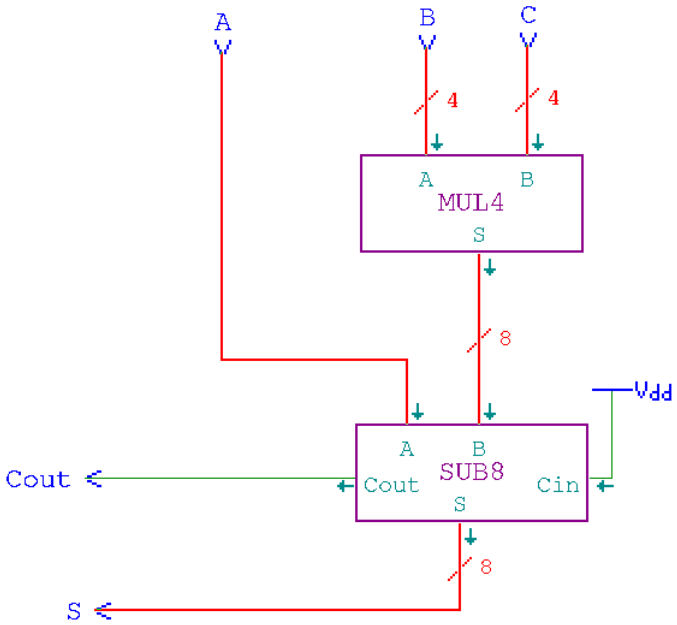


3.8.2 SUB 8 bit

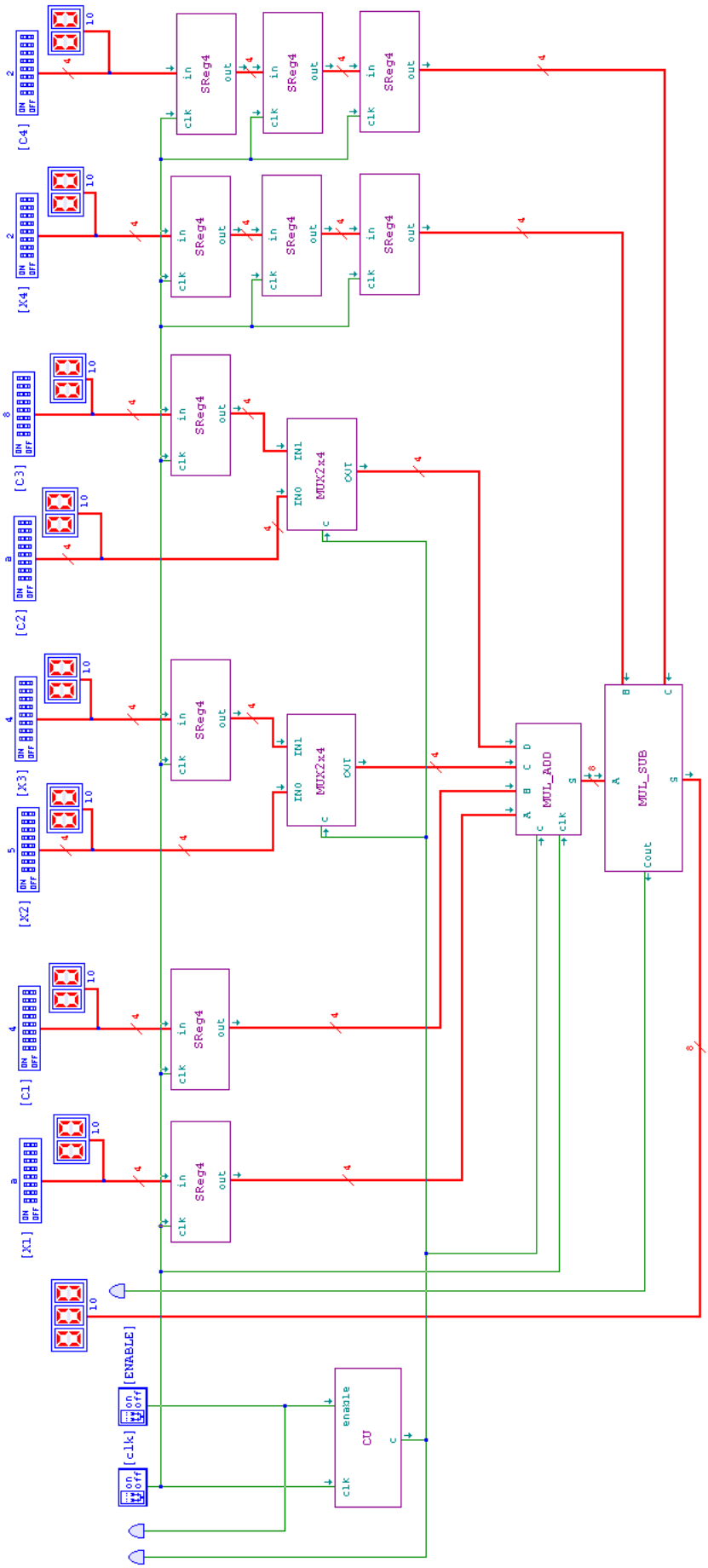


3.9 MUL_SUB

Abbiamo implementato questo modulo per incrementare la leggibilità del nostro circuito, è composto da 3 ingressi A,B,C, e due uscite S e Cout. Al suo interno troviamo i moduli per effettuare l'ultima moltiplicazione e la sottrazione finale.



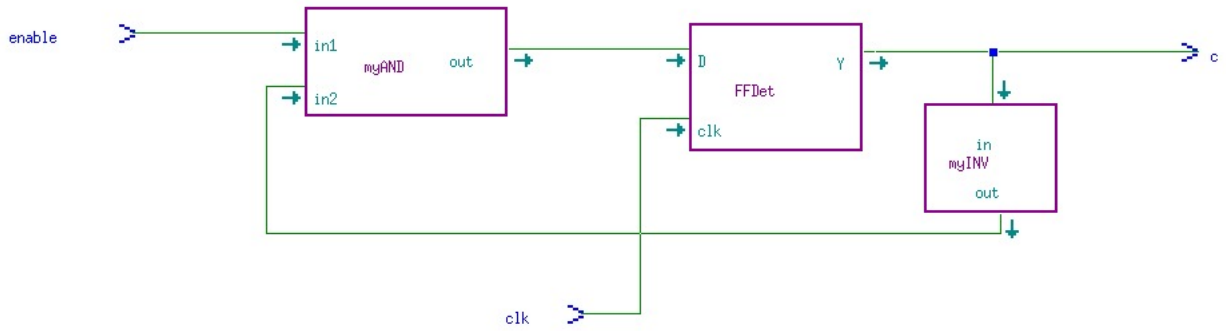
4 Data Path


[C4]
[X4]
[C3]
[C2]
[X3]
[X2]
[C1]
[X1]
SReg4
in
clk
out
MUX2x4
IN0
IN1
OUT
MUL_ADD
MUL_SUB
cout
[ENABLE]
[clk]
on
off
CU
enable

5 Control Unit

5.1 Specifica

Il modulo della Control Unit (CU) è composto da un ingresso chiamato ENABLE, munito di un interruttore per poter essere attivato, e un ingresso per il segnale di clock.
La CU è composta da un AND, un FFDet e un inverter.
Il segnale di uscita è il segnale di controllo che viene collegato ai vari MUX e serve a stabilire quale sarà l'uscita del MUX.



Se $S = 0 \longrightarrow Snext = 1 \longrightarrow c = 0$

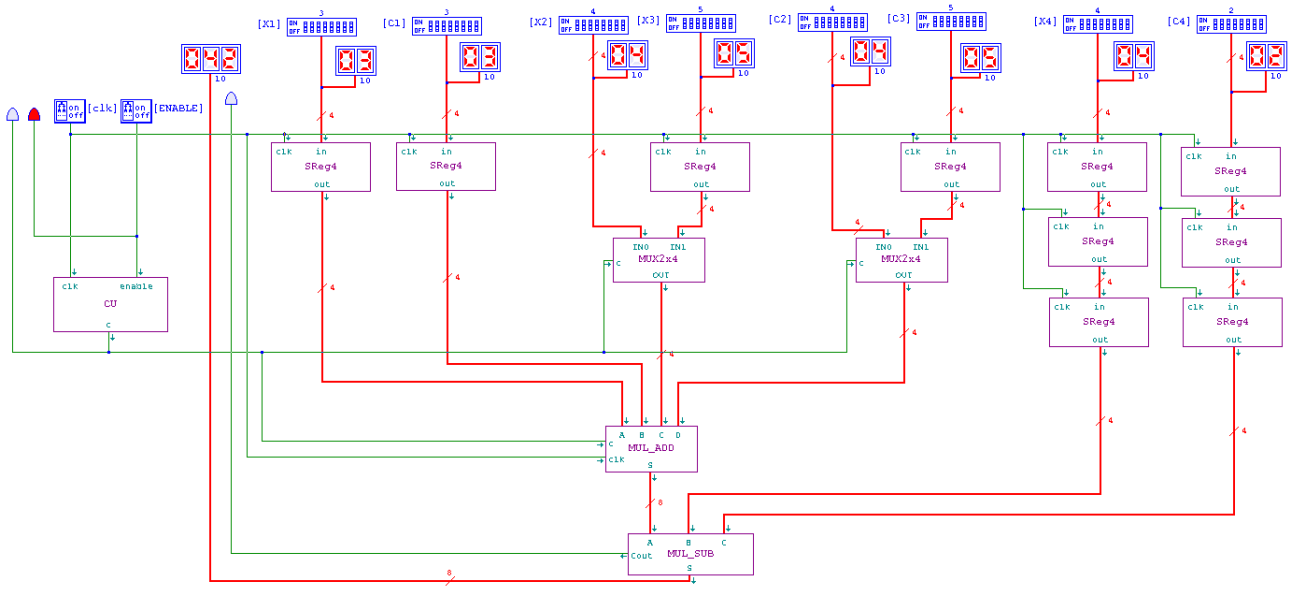
Se $S = 1 \longrightarrow Snext = 0 \longrightarrow c = 1$

S rappresenta lo stato corrente, $Snext$ rappresenta lo stato futuro e c rappresenta il valore dell'output della Control Unit.
Come possiamo osservare c è uguale ad S mentre Snext è il valore invertito di S.
Per quanto riguarda l'and, il suo compito è quello di inizializzare il valore della control unit a 0 nel primo ciclo di clock.

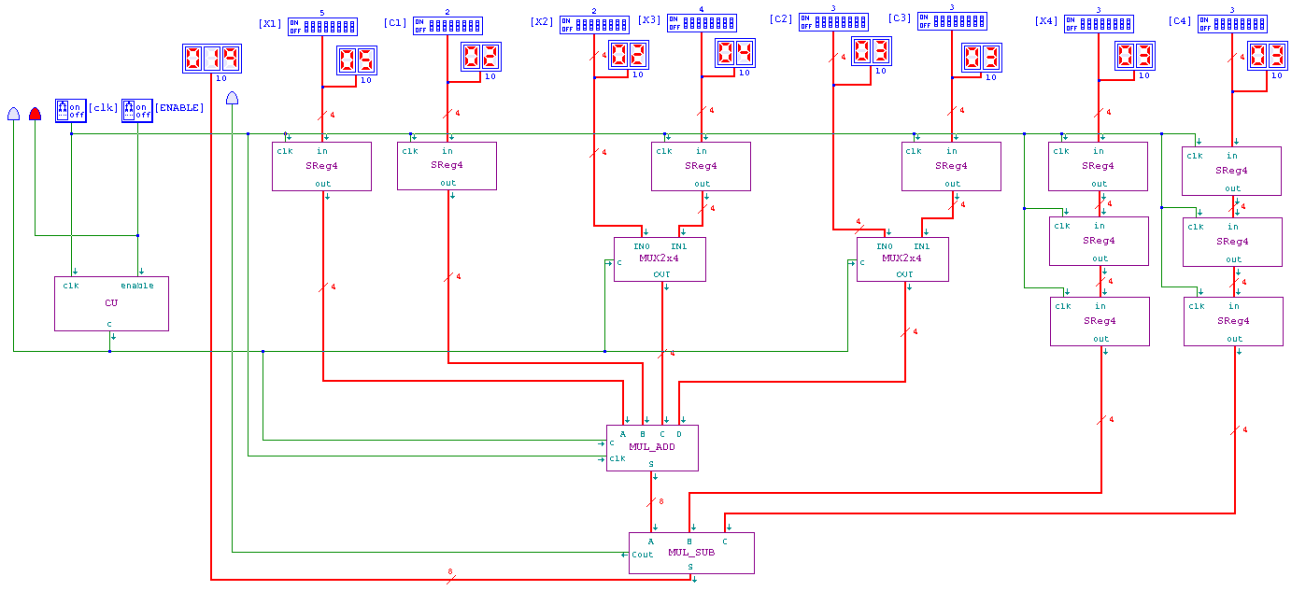
6 Simulazione e analisi del progetto

6.1 Verifica funzionale

6.2 Test1

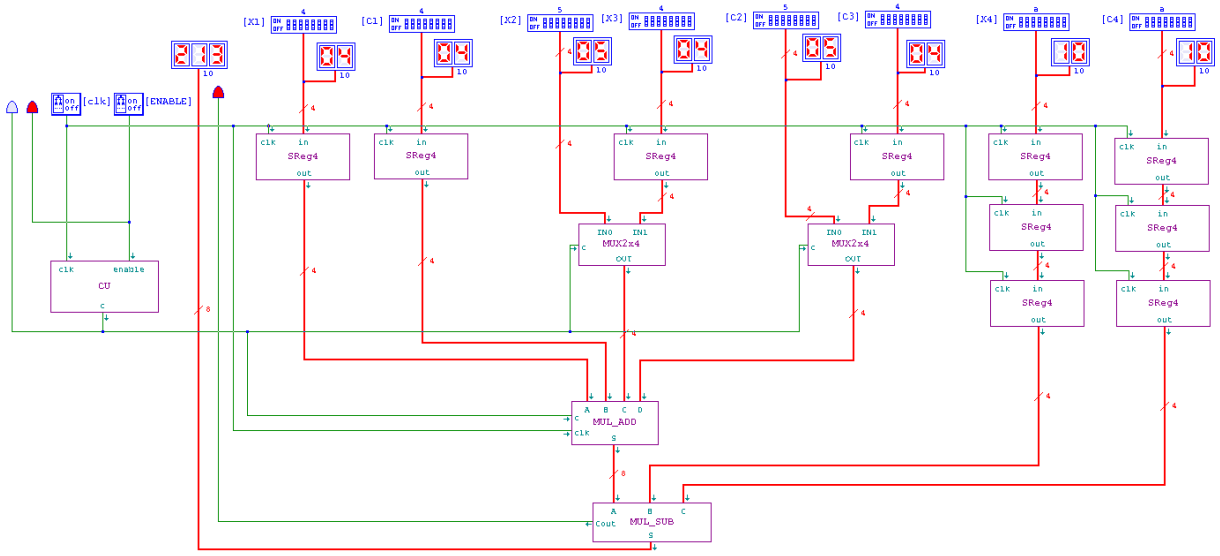


6.2.1 Test2

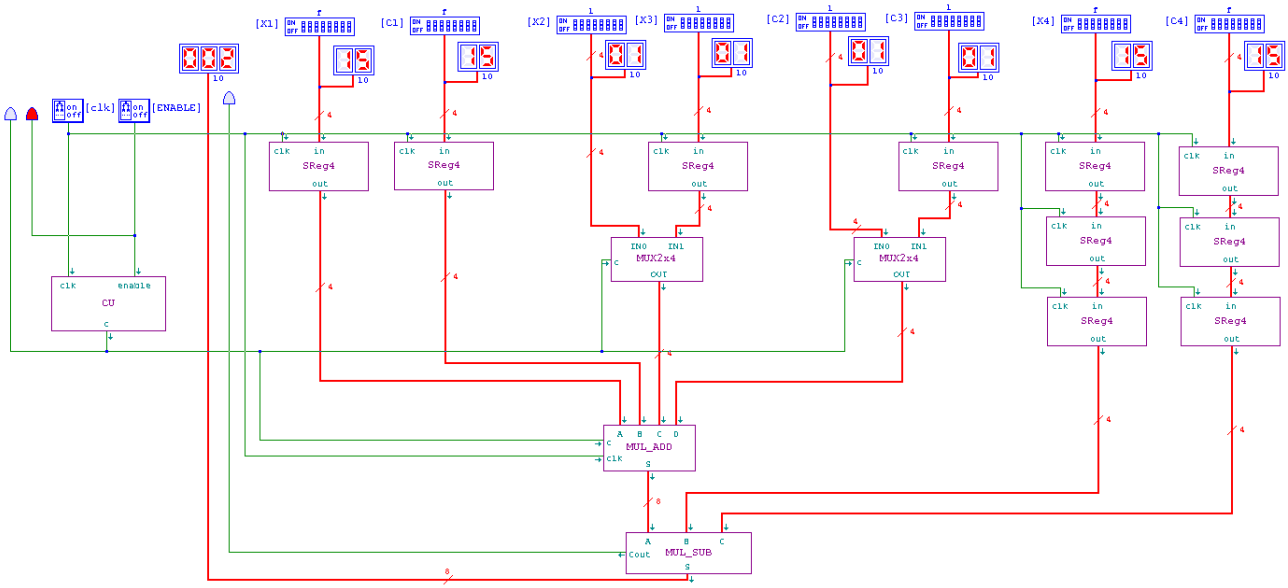


6.2.2 Test3

Caso di risultato da interpretare:



6.2.3 Test4



7 Valutazione di prestazioni e complessità

7.1 Stima di complessità circuitale e prestazioni

In questa sezione verranno descritte le varie complessità dei vari componenti utilizzati.
Con "*tp*" intendiamo il tempo di propagazione o latenza, ovvero il tempo massimo necessario affinché l'uscita si stabilizzi a fronte di una modifica dei segnali di input.
Invece con "*tc*" si intende il tempo di contaminazione, ovvero il tempo entro il quale un cambiamento a livello di segnali di input si ripercuote sulle uscite senza che questo sia necessariamente il segnale corretto.
Per quanto riguarda l'area (A) consideriamo ciascuna porta logica elementare come un'unità ovvero 1.

And:

- Tp = 1Nand + 1Inv = 2
- Tc = 1Nand + 1Inv = 2
- A = 1Nand + 1Inv = 2

Exor:

- Tp = 1Inv + 2Nand = 3
- Tc = 2Nand = 2
- A = 2Inv + 3Nand = 5

Half Adder:

- Tp = 1Inv + 2Nand = 3
- Tc = 2Nand = 2
- A = 2Inv + 3Nand + 1Nor = 6

Full Adder:

- Tp = 2(Exor) = 6
- Tc = 2Nand = 2
- A = 2Exor + 3Nand = 13

MUL

- Tp = And + 3(Full Adder) = 20
- Tc = And = 2
- A = 16(And) + 4(Half Adder) + 8(Full Adder) = 160

RCA8:

- Tp = Full Adder = 6
- Tc = Full Adder = 2
- A = 8(Full Adder) = 104

MUL_ADD:

- Tp = MUL + MUX + RCA8 + SReg = 37
- Tc = MUL + RCA8 + SReg = 7
- A = 2(MUL) + MUX + RCA8 + SReg8 = 576

INV8:

- Tp = Inv = 1
- Tc = Inv = 1
- A = 8(Inv) = 8

SUB8:

- Tp = INV8 + RCA8 + Inv = 8
- Tc = RCA8 = 2
- A = INV8 + RCA8 + Inv = 113

MUL_SUB:

- Tp = MUL + SUB8 = 28
- Tc = MUL + SUB8 = 4
- A = MUL + SUB8 = 273