1506
UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO
DISPEA
DIPARTIMENTO DI SCIENZE PURE E APPLICATE

Calcolo del delta di un'equazione di secondo grado.

Progetto d'esame di Reti Logiche A.A. 2019/2020

Matteo Marco Montanari, 299166
Giacomo Di Fabrizio, 301591

Specifica

Scopo del progetto

La specifica che si vuole presentare consiste nel progettare un circuito che calcoli il valore del *Delta* di una equazione di secondo grado ($a*x^2+b*x+c = 0$), avendo come parametri i valori di a, b e c numeri interi positivi.

Specifica funzionale

Questo circuito pertanto calcola il valore della seguente funzione aritmetica:
$f = b^2 – 4*a*c$.

Specifica parametrica

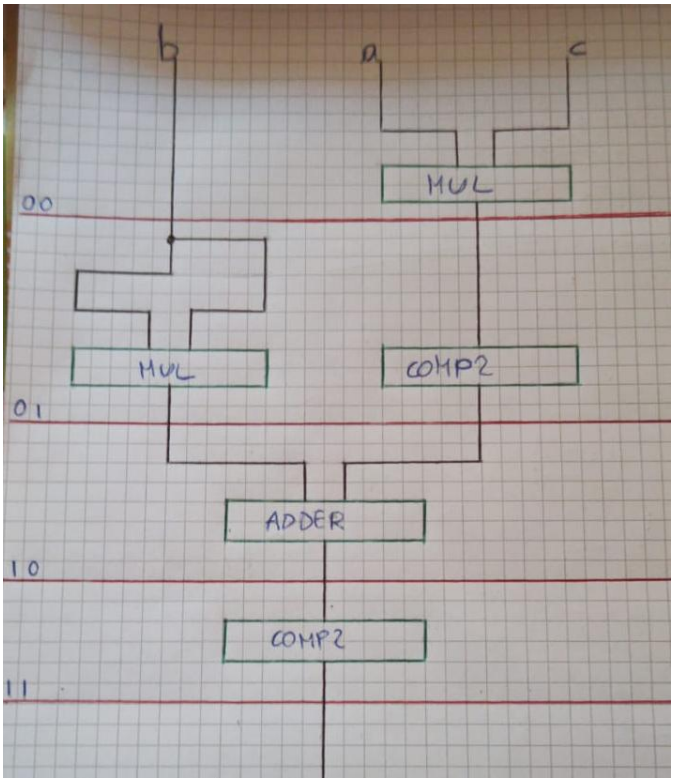
Per questo progetto abbiamo utilizzato valori in input di 8 bit per rappresentare a, b e c, pertanto b^2 sarà rappresentabile correttamente con 8+8=16 bit, mentre 4*a*c con 2+8+8=18 bit. Il risultato sarà perciò costituito da 18 bit in modo da non introdurre ulteriori limitazioni o troncamenti.
Pertanto i valori assegnabili alle variabili a, b e c sono i numeri interi positivi compresi tra 0 e $2^8 – 1=255$, compresi gli estremi.

Impostazione del progetto a livello RT

Data Flow Graph

Si è scelto di suddividere la computazione in 4 cicli di clock, in linea con la progettazione in Pipelining, per aumentare la frequenza di lavoro del circuito.

Il flusso dei dati che permette il calcolo della funzione data è schematizzato dal seguente grafo:

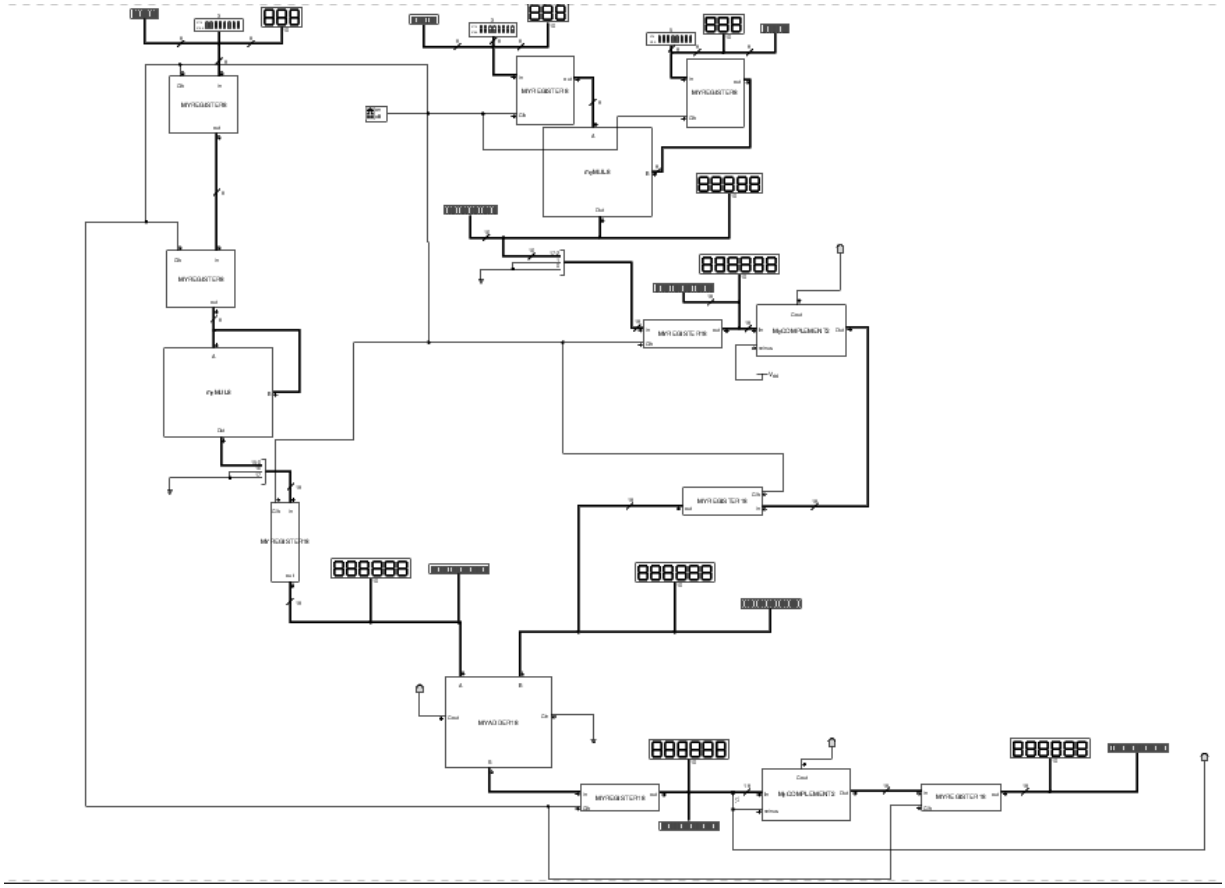


Al primo ciclo di clock viene calcolato il prodotto 4*a*c, moltiplicando a e c con un moltiplicatore e aggiungendo due zeri (da destra) al risultato ottenuto.
Al secondo ciclo di clock viene moltiplicato b per b con un altro moltiplicatore in modo da ottenere b^2, e viene complementato a 2 il valore di 4*a*c in modo da renderlo negativo.
Al terzo ciclo di clock vengono sommati b^2 e -4*a*c (complementato) con un addizionatore.
Al quarto ciclo di clock viene complementato a 2 il risultato (nel caso risulti negativo) per ottenere il suo modulo.

Risorse

Le risorse necessarie per realizzare il circuito sono:

- Registri (4 reg. a 8bit, 5 reg. a 18 bit),
- Macro aritmetiche (1 Ripple Carry Adder a 18 bit, 2 moltiplicatori a 8 bit, 2 complementatori a 2 a 18 bit, 1 switch)

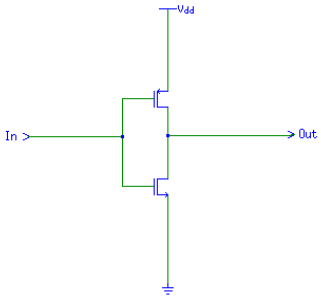


1506
UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO
DISPEA
DIPARTIMENTO DI SCIENZE PURE E APPLICATE

Progetto delle risorse a livello gate

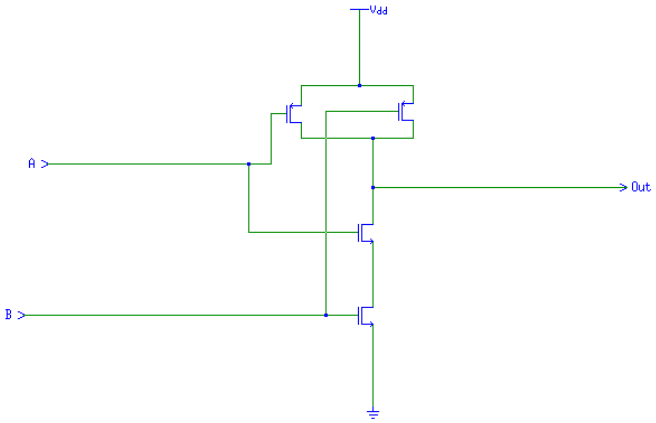
Porte logiche elementari

Porta NOT:
La porta logica NOT (conosciuta anche come inverter) prende in input un singolo bit e come risultato restituisce lo stesso bit in forma negata.



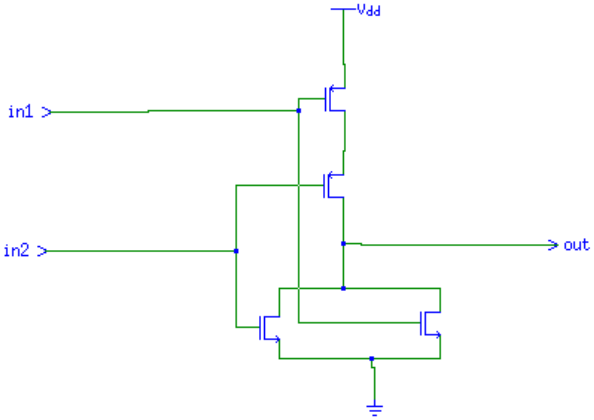
INPUT	OUTPUT
A	NOT A
0	1
1	0

Porta NAND:
La porta logica NAND da noi utilizzata prende in ingresso due input e restituisce in uscita un solo output.
Essa restituisce 1 se almeno uno dei due input è 0, mentre restituisce 0 solo nel caso in cui entrambi gli input siano 1. (In logica corrisponde all'AND negato)



INPUT		OUTPUT
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

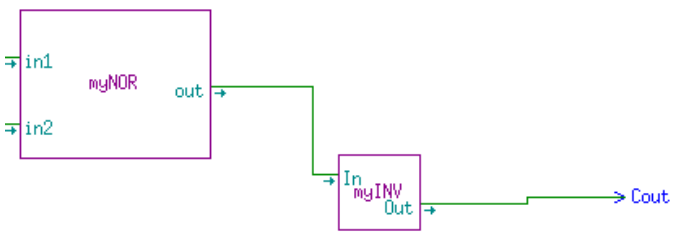
Porta NOR:
La porta logica NOR fornisce 2 ingressi e una sola uscita; se in input riceve almeno un 1 il suo output sarà 0, altrimenti l'output risulterà 1 (In Logica equivale a un OR negato)



INPUT		OUTPUT
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

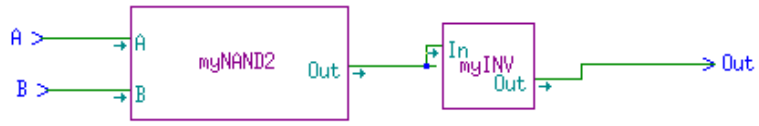
Porte logiche composte

Porta OR:
La porta logica OR implementata nel progetto è una combinazione della porta NOR C-Mos concatenata con un NOT detto anche Inverter.
In Logica la porta OR ha due ingressi (gli stessi del NOR in C-MOS) ed una sola uscita (Come nel NOR) ma essa invece di essere collegata all'output entra nell'input del inverter così da avere il risultato negato rispetto al NOR visto in precedenza.
Quindi avremo una risposta 0 solamente quando l'input di entrambi gli ingressi è 0 altrimenti avremmo sempre 1.
L'OR è stato utilizzato nel progetto per la funzione di riporto all'interno del Full-Adder; la funzione necessaria al progetto era (A + B), ma con tecnologia C-Mos in questo caso specifico, la funzione sarà ((A + B)')', essendo come abbiamo visto precedentemente un NOR il cui risultato viene negato con un NOT.
Queste due funzioni sono equivalenti ovvero producono le stesse tabelle di verità.



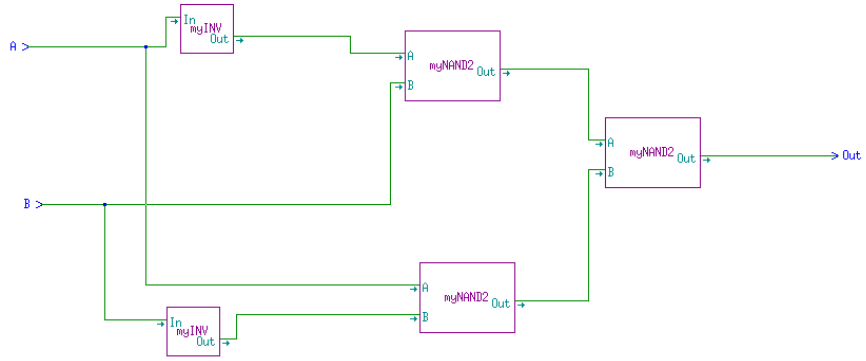
INPUT		OUTPUT
A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

Porta AND:
L'AND utilizzato nel progetto è stato implementato analogamente all'OR, ovvero con l'utilizzo della porta NAND in C-Mos negata con l'inverter (NOT).
La funzione di un AND è (AB) che nel caso del progetto è stata realizzata con la funzione equivalente ((AB)')'.
L'AND è utilizzato per effettuare il prodotto nel modulo della moltiplicazione, perché rispecchia esattamente questo tipo di operazione nel sistema binario.
Nel caso in cui entrambi gli input di ingresso siano 1, otterremo un valore di uscita pari a 1, equivalente a 1 * 1 nel sistema binario che da come risultato 1.
Nel caso in cui uno dei due ingressi sia 0 o entrambi 0 avremo come risultato 0 come nella moltiplicazione binaria (1 * 0 = 0; 0 * 1 = 0; 0 * 0 = 0)



INPUT		OUTPUT
A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Porta EXOR:
La porta logica EXOR è stata utilizzata per la creazione del modulo Half-Adder.
Rappresenta precisamente il risultato di un'operazione di somma di due cifre binarie senza considerare il riporto (1+1 = 0 con riporto di 1).
Ricordiamo inoltre che essa è stata una porta logica veramente molto importante per effettuare il complemento a 2.
La porta EXOR in tecnologia C-Mos è realizzata con 3 porte NAND e 2 inverter concatenate come in figura.
La funzione logica dell'EXOR è (AB' + A'B) equivalente alla funzione ((AB')' (A'B)')' risultante dalle combinazioni delle porte C-Mos utilizzate in questo modulo creato con TKGATE.
In figura si vede come entrambi gli input vengano inseriti in due NAND ma in modo alternato uno dei due ingressi viene invertito con un NOT, ottenendo così nel primo NAND, con ingresso B invertito, la funzione (AB')' e nel secondo NAND, con l'ingresso A invertito otteniamo (A'B)'.
Il risultato dei singoli NAND entrano in un ulteriore NAND e otteniamo la funzione ((AB')'(A'B)')' corrispondente alla EXOR che ci interessa per il progetto.



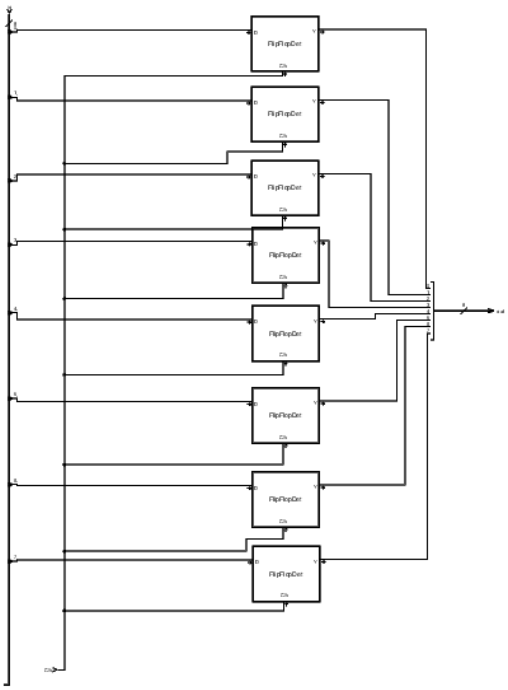
INPUT		OUTPUT
A	B	$A\bar{B} + \bar{A}B$
0	0	0
0	1	1
1	0	1
1	1	0

Data Path

Registri

I registri che abbiamo utilizzato per gestire il data flow in 4 cicli di clock sono costruiti come visto a lezione. I registri a 8 (o 18) bit sono costituiti da una sequenza di 8 (o 18) Flip Flop D Edge Triggered, i quali a loro volta sono costituiti ciascuno da 2 Flip Flop D Level Sensitive a formare un master-slave. Ogni Flip Flop D Level Sensitive è a sua volta costituito da due porte AND, una porta NOT e un latch di tipo SR (circuito sequenziale costituito da due porte NOR).

Ogni volta che il segnale di clock passa da 0 a 1 i registri campionano i dati e si passa al ciclo di clock successivo.

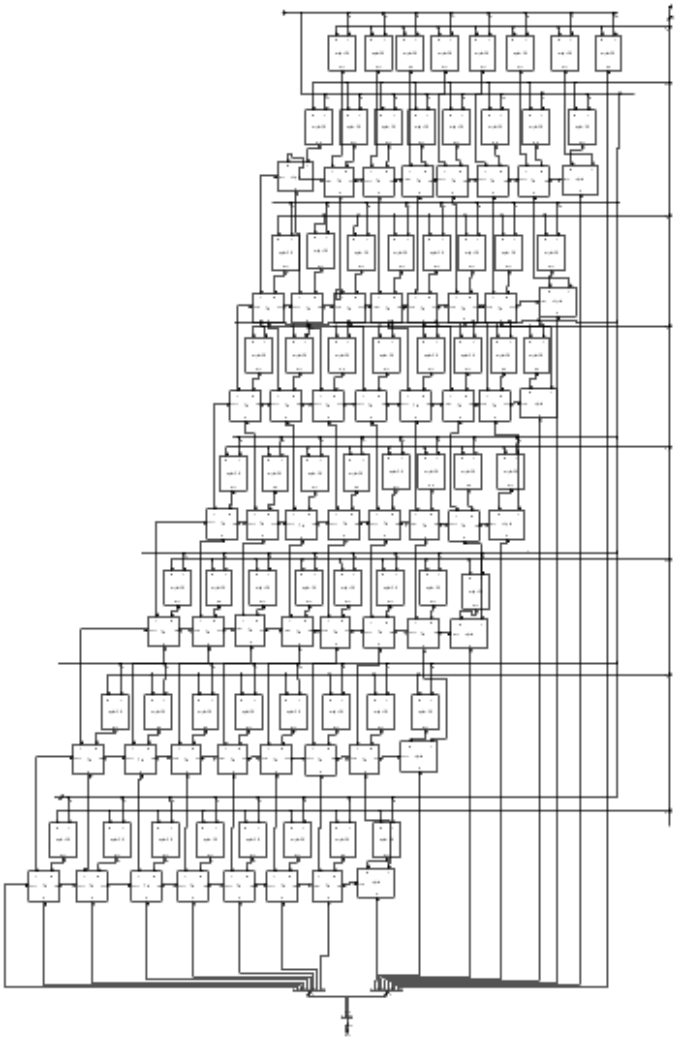


Moltiplicatore

Il moltiplicatore che abbiamo realizzato implementa l'algoritmo matematico della moltiplicazione di due numeri in colonna ed è diviso in due momenti:

1) *Primo passo, dei prodotti parziali*: si mette in AND ciascun bit del moltiplicando con il bit di posto zero del moltiplicatore, in modo da ottenere il primo prodotto parziale. Il secondo prodotto parziale è ottenuto come il primo, agendo sul secondo bit del moltiplicatore. Si continua così fino a terminare i bit del moltiplicatore. Viene utilizzata la porta logica AND per fare i prodotti in quanto corrisponde proprio alla funzione aritmetica che esprime il prodotto tra due cifre binarie.

2) *Secondo passo, le somme*: ottenuti tutti i prodotti parziali si effettua la somma in colonna degli stessi nell'ordine in cui sono stati calcolati, dopo essere stati allineati in modo tale che ogni prodotto parziale (eccetto il primo) sia spostato a sinistra di una cifra rispetto al prodotto che lo precede. Le cifre mancanti sono da considerare ai fini della somma come degli zeri. Le somme vengono effettuate bit a bit seguendo lo schema della somma con propagazione di riporto attraverso dei Full Adder e Half Adder.

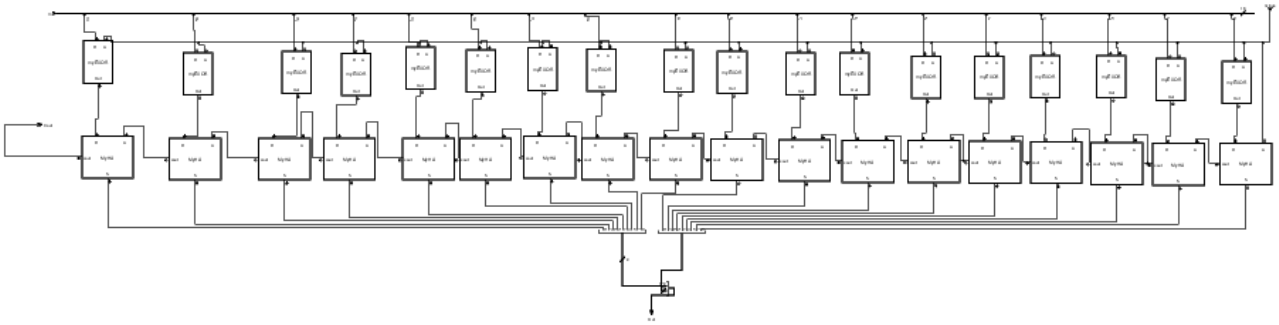


Complementatore a 2

Il complemento a due di un numero binario positivo viene utilizzato per renderlo negativo e viceversa. Il bit più significativo è il bit di segno, 1 se è negativo 0 altrimenti. Per sapere il modulo di un numero negativo in complemento a 2 occorre prima renderlo positivo ricomplementandolo.

Per complementare a due un numero binario basta negare ogni suo bit e sommare 1. Nella nostra implementazione l'operazione di negazione è affidata a una porta EXOR collegata a un segnale di controllo. Tale segnale è anche utilizzato per sommare una unità al risultato tramite degli *Half Adder* a propagazione di riporto.

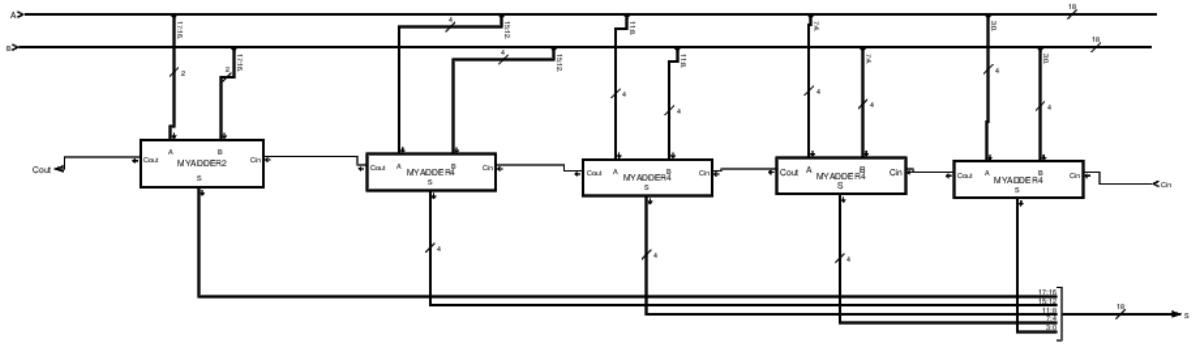
Se il segnale di controllo è 0 il numero rimane inalterato altrimenti viene eseguito il complemento a 2. Se il numero in complemento a 2 eccede il numero di bit assegnati si ha un errore di overflow segnalato da un LED.



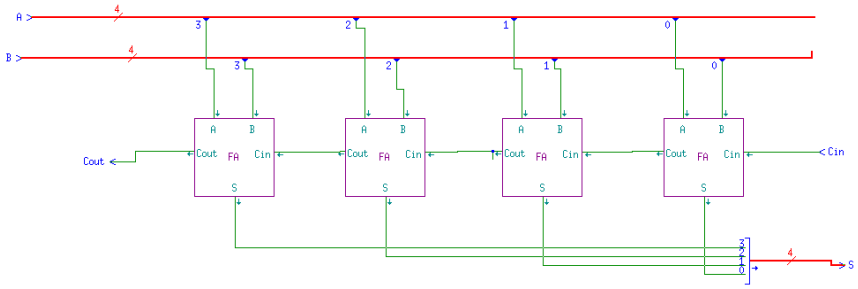
Addizionatore (RCA18):

L'addizionatore che abbiamo realizzato implementa l'algoritmo matematico dell'addizione di due numeri in colonna.
L'addizionatore a 18 bit utilizzato nel progetto è un esempio di Ripple-Carry Adder.
RCA è il metodo più diretto per realizzare un addizionatore a n bit ed esso prende anche il nome di addizionatore a propagazione di riporto.
L'addizionatore RCA non fa nient'altro che calcolare la somma così come verrebbe calcolata a mano.
Per realizzare ciò abbiamo utilizzato componenti veramente essenziali che sono Half adder e Full adder (a loro volta composti da due Half adder).
Ogni Half adder esegue la somma di 2 bit e restituisce il risultato della somma e il riporto senza tenere conto di un riporto precedente.
Ogni Full adder esegue la somma di 2 bit e restituisce il risultato della somma e il riporto tenendo conto di un riporto precedente, quindi, di conseguenza, la somma di un numero a n bit risulta la concatenazione di n somme a 1 bit.

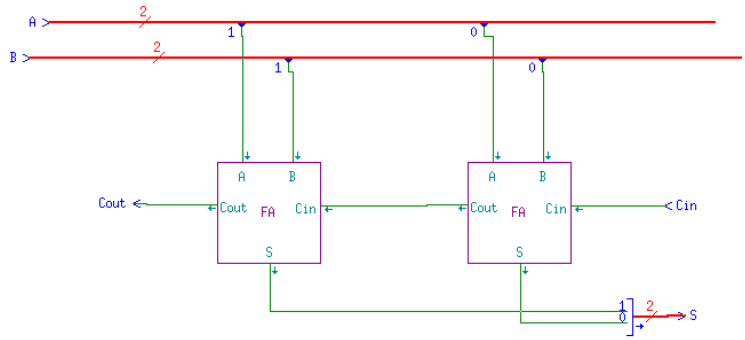
Addizionatore a 18 bit:



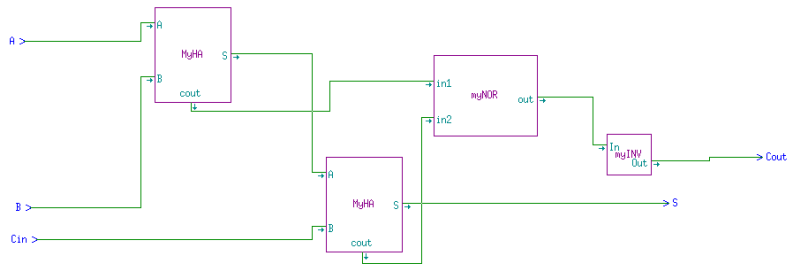
Addizionatore a 4 bit:



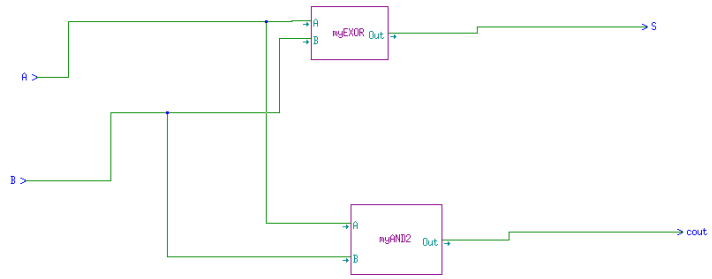
Addizionatore a 2 bit:



Full-Adder:



Half-Adder:



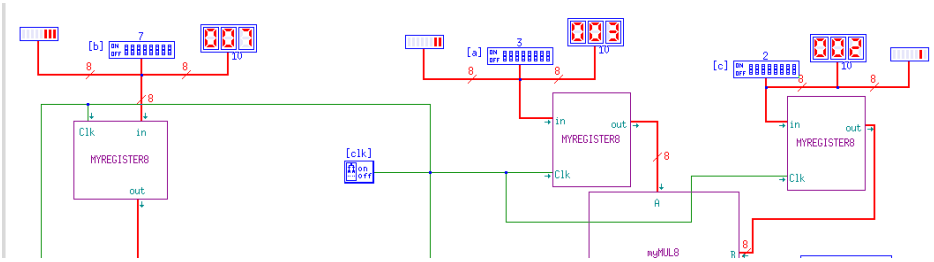
Control Unit

Il nostro progetto non necessita di control unit poiché non è stato necessario utilizzare alcun Multiplexer per gestire il flusso dei dati.

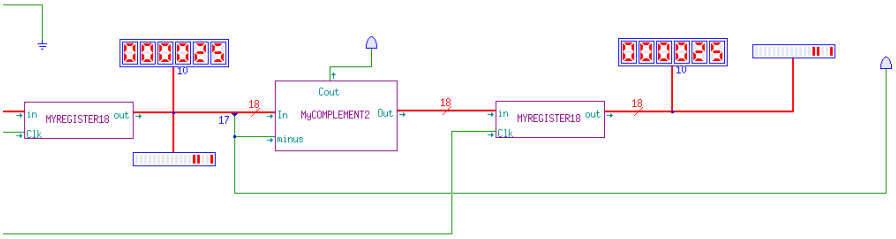
Simulazione e analisi del progetto

Verifica funzionale

Delta positivo:

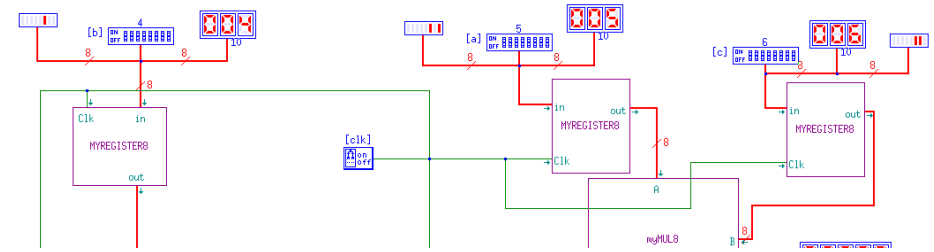


INPUT

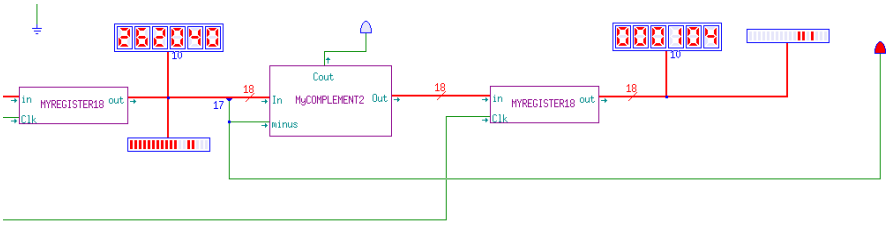


RISULTATO

Delta negativo:

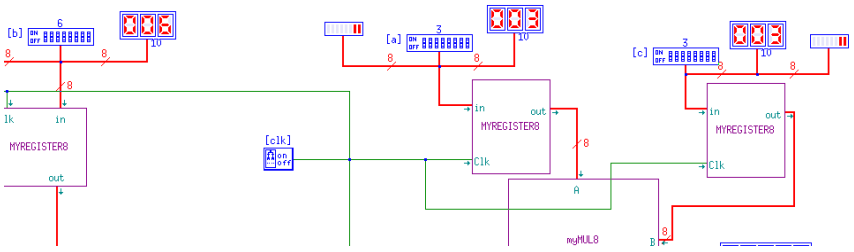


INPUT

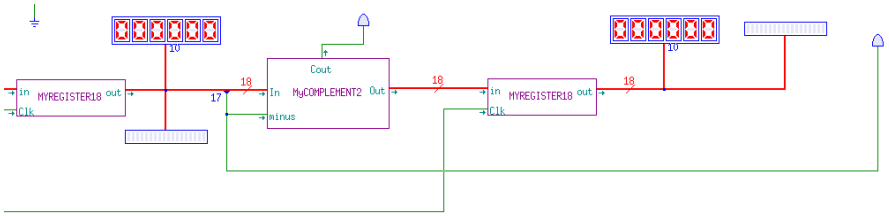


RISULTATO

Delta nullo:



INPUT

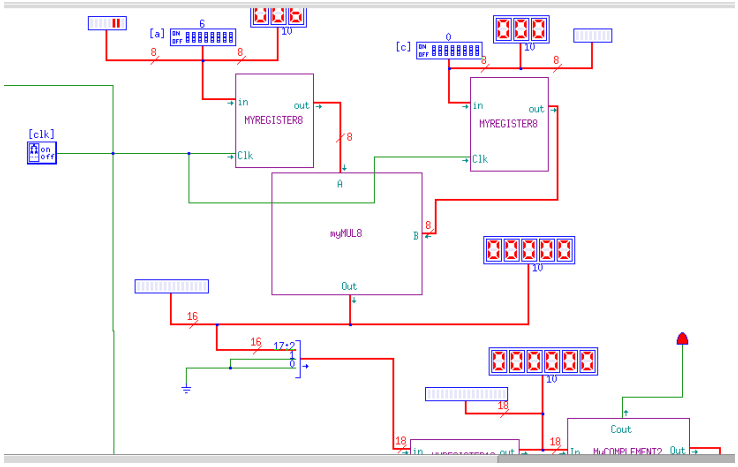


RISULTATO

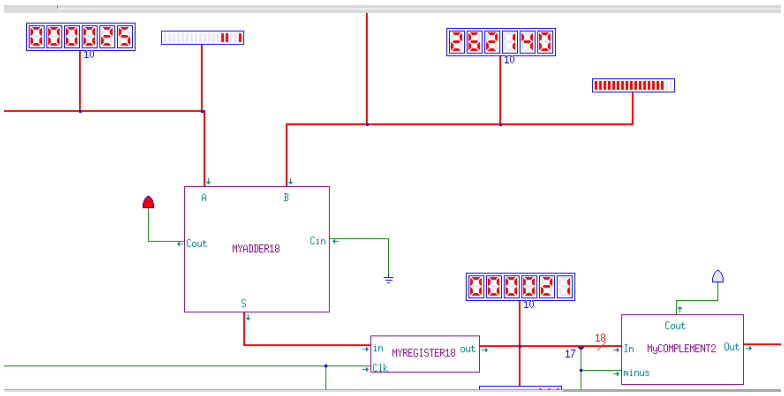
In seguito a vari test, abbiamo concluso che il circuito descritto presenta un errore di overflow(inevitabile) nel caso in cui il prodotto di a*c sia zero per via del complemento a 2. Questo risultato tuttavia non compromette il funzionamento del circuito in quanto il complemento a 2 di zero rimane invariato a zero e il bit di overflow è scartato.

Nel caso in cui il modulo di 4*a*c sia minore di b^2, e dunque il delta risulti positivo, l'adder va in overflow poiché, in modulo, il valore complementato di 4*a*c è un numero che sommato a b^2 eccede il numero di bit assegnati. Il risultato è comunque corretto, in quanto, il complemento a 2 gioca sulla lunghezza fissa di 18 bit della rappresentazione e permette di ignorare il riporto di 1 che causa l'overflow. Non è necessario ricomplementare il risultato in quanto esso è un valore positivo.

Overflow del complemento a 2:



Overflow dell' addizionatore:



Valutazione di prestazioni e complessità

Dall'impostazione del progetto a livello RT, risultano le seguenti metriche:

Area o Complessità circuitale: A = 2*A(MUL) + A(ADD) + 2*A(C2) + 4*A(Reg8) + 5*A(Reg18)

Tempo di contaminazione: Tc = Tc(MUL) + Tc(ADD) + Tc(C2)

Tempo di propagazione: Tp = Tp(MUL) + Tp(ADD) + 2*Tp(C2)

Rate: fr = 1/Tp(MUL) in Pipelining.

Stima di complessità circuitale e prestazioni

Tempo propagazione per ogni componente (con modello di ritardo unitario):

- Addizionatore a 18 bit:
  Tp(ADDER18) =
  4*Tp(ADDER4) + Tp(ADDER2) = 4*4*Tp(FA) + 2*Tp(FA) = 16*6 + 2*6 = 108u

  Poiché:
  Tp(FA) = max(2*Tp(HA), Tp(HA) + Tp(NOR) + Tp(INV)) = max(2*Tp(HA), Tp(HA) +2)
  = max(6, 5) = 6u,

  Tp(HA) = max(Tp(EXOR), Tp(AND))= max(3, 2) = 3u

- Complementatore a 18 bit:
  Tp(COMP2) = 18*Tp(HA) + Tp(EXOR) = 18*3 + 3 = 57u

- Moltiplicatore a 8 bit:
  Tp(MUL8) = Tp(AND) + Tp(HA) + 7*(2* Tp(FA)) + 5*Tp(FA) = 1 + 3 + 7*2*6 + 5*6 = 118u

- Registri:
  Tp(REG) = Tp(REG8) = Tp(REG18) = Tp(FFDET) = 1 + 2* Tp(FFDLS)
  = 1 + 2*(1 + Tp(FFSR)) = 1 + 2*(1 + Tp(AND) + Tp(LATCHSR)) = 1 + 2*(1+2+1) = 9u

- Intero progetto:
  Tp(DELTA) = Tp(REG) + Tp(MUL) + Tp(REG) + Tp(COMP2) + Tp(REG) + Tp(ADDER18) + Tp(REG)
  + Tp(COMP2) + Tp(REG) = 9+118+9+57+9+108+9+57+9 = 385u

- Primo stadio:
  Tp(DELTA1CLK) = Tp(REG) + Tp(MUL8) + Tp(REG) = 9+118+9 = 136u

- Secondo stadio:
  Tp(DELTA2CLK) = max(Tp(MUL8), Tp(COMP2)) + 2*Tp(REG) = max(118, 57) + 18 = 136u

- Terzo stadio:
  Tp(DELTA3CLK) = Tp(ADDER18) + 2*Tp(REG) = 108+18 = 126u

- Quarto stadio:
  Tp(DELTA4CLK) = Tp(COMP2) + 2*Tp(REG) = 57 + 18 = 75u

Pertanto la durata minima del periodo di clock a cui il circuito può funzionare è:

Tclk > max (Tp(DELTA1CLK), Tp(DELTA2CLK), Tp(DELTA3CLK), Tp(DELTA4CLK)) =
= Tp(DELTA1CLK) = Tp(DELTA2CLK) = 136u

Termine della relazione.