



**Davide Carini**

**Matricola: 890034 - Codice Persona: 10568649**

# **PROVA FINALE**

# **DI RETI LOGICHE**

**(PROGETTO 11)**

**Corso di Reti Logiche**

**Anno Accademico 2020/2021**

**Professore: Carlo Brandolese**



# INDICE

<b>INTRODUZIONE .....</b>	<b>3</b>
Scopo del Progetto.....	3
Esempio .....	5
Esempio con errore .....	6
<b>DESIGN ARCHITETTURA.....</b>	<b>7</b>
Interfaccia del componente top level.....	7
Entity principali .....	8
Interazione componenti .....	14
Report utilizzo divider .....	16
<b>TEST BENCH .....</b>	<b>17</b>
<b>RIFERIMENTI.....</b>	<b>18</b>

# INTRODUZIONE

## Scopo del Progetto

Lo scopo del progetto è quello di realizzare un divisore intero a 32 bit basato sul metodo di “**divisione lunga**”. Siano N il dividendo, D il divisore, Q il quoziente, R il resto ed n la dimensione delle parole, l’algoritmo è descritto dal seguente pseudocodice:

```
if( D == 0 ) {  
    error();  
}  
Q = 0;  
R = 0;  
for(i = n-1; i >= 0; i-- ) {  
    R = R << 1;  
    R[0] = N[i] ;  
    if( R ≥ D ) {  
        R = R - D ;  
        Q[i] = 1 ;  
    } else {  
        Q[i] = 0 ;  
    }  
}  
}
```

Deve essere realizzata una **rete sequenziale** che implementi il divisore basato su tale algoritmo. Una volta realizzato il componente, è richiesto di realizzare un test-bench per la simulazione(sia a livello behavioral sia a livello post-route) e la verifica del corretto funzionamento nei diversi casi possibili.

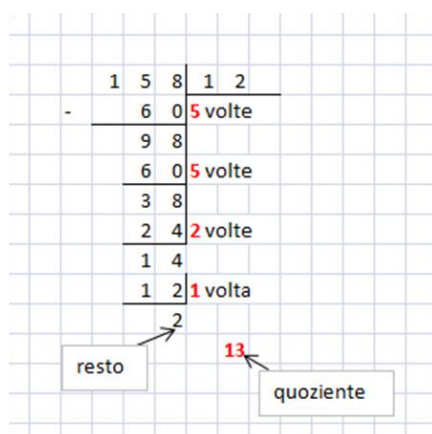


Figura 1: Esempio di "Lunga Divisione"

L'algoritmo di “**divisione lunga**” è l'algoritmo standard più utilizzato per la divisione di numeri espressi in notazione decimale conosciuto come **divisione in colonna**.

Come primo approccio ho pensato a capire lo scopo del progetto e comprendere il problema da un punto di vista generale più ad ampia veduta rispetto ad una descrizione subito dettagliata dei sotto problemi presenti.

Per fare ciò, ho tradotto il ciclo for in una serie di processi in ordine temporale descritti tramite **diagramma di flusso** che aiuta meglio a comprendere il procedimento nel suo insieme.

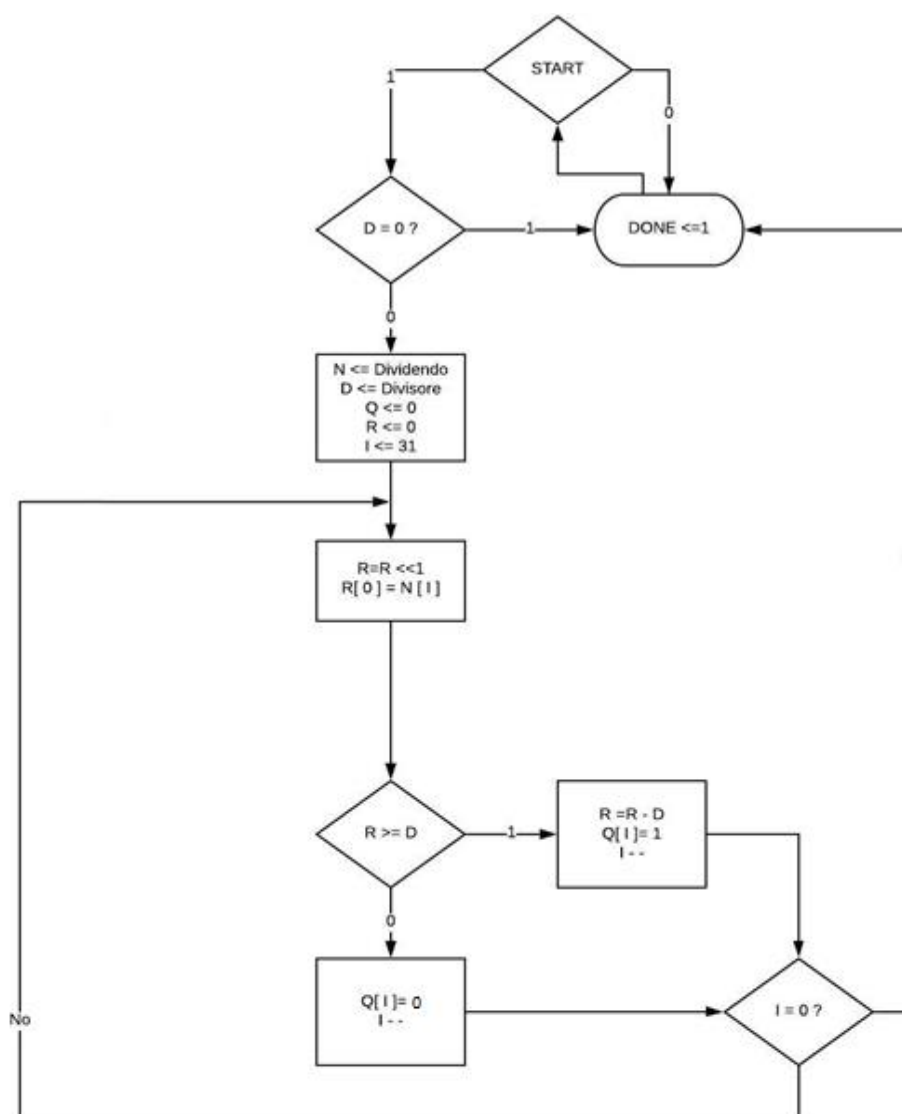


Figura 2: Diagramma di flusso rappresentante l'algoritmo

Essendo un ciclo, le istruzioni devono essere eseguite in modo sequenziale. Inizialmente ho pensato ad un possibile utilizzo di una pipeline che potesse eseguire le diverse istruzioni in parallelo. Il problema è che, a differenza di una CPU che ad ogni ciclo di pipeline carica una nuova istruzione, in questo caso il risultato dell'istruzione corrente verrà utilizzato in quella successiva. Questo mi ha portato all' **esclusione dell'utilizzo della pipeline** che in questo problema perderebbe di significato.

Il diagramma di flusso mi fornisce una descrizione dal punto di vista SW e quindi necessito di un datapath per la risoluzione **HW** del problema.

Ho pensato di risolvere l'algoritmo utilizzando un divisore descritto ad un livello di astrazione VHDL **RTL** (Register transfer level). Questo è il livello intermedio meno astratto del livello behavioral e meno a basso livello di quello strutturale.

## Esempio

Dimostro, tramite due esempi, il corretto funzionamento dell'algoritmo.

Pongo  $N=10001111_2$  ( $143_{10}$ ) e  $D=00001001_2$  ( $9_{10}$ ). Verifico inizialmente che il divisore sia diverso da 0 (condizione rispettata in questo caso). Dopo inizializzo quoziente e resto a 0.

### CICLO FOR

$i=n-1=7$

R= 00

R=01

Condizione if false

Q=0

$i=6$

R=010

R=010

Condizione if false

Q=00

$i=5$

R=0100

R=0100

Condizione if false

Q=000

$i=4$

R=01000

R=01000

Condizione if false

Q=0000

$i=3$

R=010000

R=010001

Condizione if true

R=1000

Q=00001

$i=2$

R=10000

R=10001

Condizione if true

R=1000

Q=000011

$i=1$

R=10000

R=10001

Condizione if true

R=1000

Q=0000111

$i=0$

R=10000

R=10001

Condizione if true

**R=1000<sub>2</sub> -> 8<sub>10</sub>**

**Q=00001111<sub>2</sub> -> 15<sub>10</sub>**

L'esempio di divisione è stato fatto con parole di 8 bit in modo tale da semplificare il ciclo iterativo. Con parole di 32 bit, il ciclo for avrà 32 iterazioni differenti.

## Esempio con errore

Pongo  $N=10001111_2$  ( $143_{10}$ ) e  $D=00000000_2$  ( $0_{10}$ ). Inizializzo quoziente e resto a 0.

### CICLO FOR

$i=n-1=7$

R= 0

R= 1

Condizione if true

R=1

Q=1

$i=6$

R=10

R=10

Condizione if true

R=10

Q=11

$i=5$

R=100

R=100

Condizione if true

R=100

Q=111

$i=4$

R=1000

R=1000

Condizione if true

R=1000

Q=1111

$i=3$

R=10000

R=10001

Condizione if true

R=10001

Q=11111

$i=2$

R=100010

R=100011

Condizione if true

R=100011

Q=111111

$i=1$

R=1000110

R=1000111

Condizione if true

R=100011

Q=1111111

$i=0$

R=10001110

R=10001111

Condizione if true

**R=10001111**

**Q=11111111**

Quando il divisore è nullo, il resto è sempre maggiore o uguale del divisore e quindi la condizione del costrutto if sarà sempre vera. Il resto diventa uguale al dividendo mentre il quoziente è composto da tutti '1' (osservazione molto utile).

# DESIGN ARCHITETTURA

## Interfaccia del componente top level

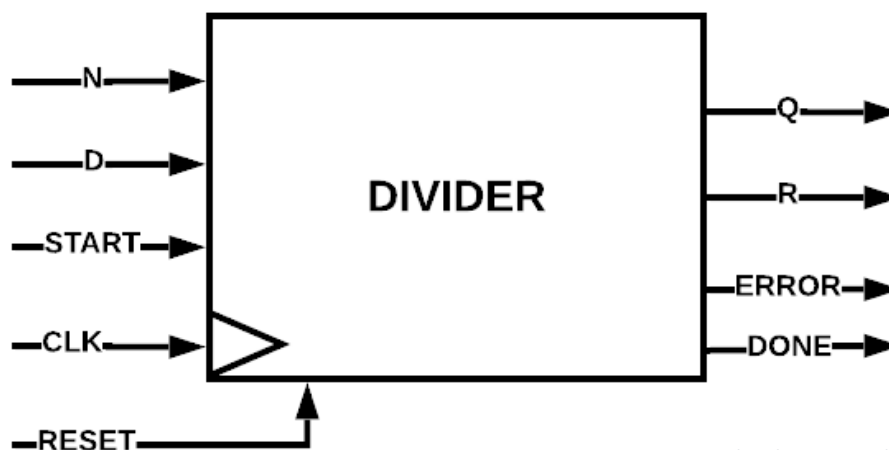


Figura 3: Componente top level con segnali d'interfaccia

Il divisore ha 5 segnali in ingresso e 4 segnali in uscita.

Gli ingressi rappresentano:

- **N** rappresenta il dividendo a 32 bit;
- **D** rappresenta il divisore a 32 bit;
- **START** è il segnale di ingresso che quando vale 1 resetta resto e quoziente e dà il via alla divisione;
- **CLK** rappresenta il segnale di clock utile a sincronizzare i componenti interni al divisore;
- **RESET** è il segnale che inizializza il componente pronto per ricevere il segnale di START.

Le uscite invece rappresentano:

- **Q** corrisponde al quoziente a 32 bit della divisione ;
- **R** rappresenta il resto a 32 bit della divisione ;
- **ERROR** è il segnale che esprime la condizione di errore ovvero quando il divisore è nullo;
- **DONE** è il segnale che viene settato ad 1 quando il risultato della divisione ovvero il quoziente ed il resto sono pronti (dopo le 32 iterazioni di ciclo for).

Nel divisore vengono mappati e istanziati i vari componenti interni e i segnali temporanei che permettono il corretto collegamento tra di essi.

Il divisore può essere visto come un insieme di componenti di logica combinatoria e di registri atti a salvare risultati di elaborazioni intermedie.

## Entity principali

I principali componenti sviluppati ed utilizzati sono:

### SHIFTER

Lo shifter prende in ingresso il valore del **resto** presente nel registro, **lo shifta a sinistra di una posizione e restituisce in uscita 2 valori**. Il primo rappresenta il resto shiftato con il primo valore messo a 0 mentre il secondo rappresenta il resto shiftato con il primo valore ad 1. La presenza delle due uscite mi permette di avere entrambe le possibilità per il successivo assegnamento del valore i-esimo del dividendo al primo bit del resto.



Figura 4: Shifter con i segnali d'interfaccia

Segnali:

#### INPUT

- **R** rappresenta il resto;

#### OUTPUT

- **R\_SHIFTED\_0** rappresenta il resto in ingresso shiftato di una posizione con il primo bit a 0;
- **R\_SHIFTED\_1** rappresenta il resto in ingresso shiftato di una posizione con il primo bit a 1.

### MUX\_2\_1

Questo componente rappresenta un **multiplexer** con due ingressi, un' uscita e un segnale di selezione. Saranno 2 i multiplexer di questo tipo utilizzati nell' architettura. Uno sarà utilizzato per prendere il giusto valore in uscita dallo shifter mentre l'altro servirà per salvare nel registro il corretto valore del resto. Il segnale di selezione del primo mux sarà il valore i-esimo del dividendo mentre il segnale di selezione dell' ultimo mux sarà il prestito in uscita dal sottrattore che vale 0 quando il resto è maggiore o uguale al divisore e vale 1 altrimenti.

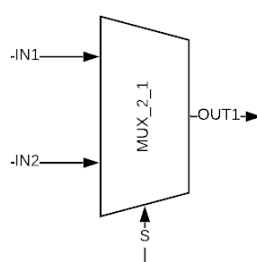


Figura 5: Rappresentazione MUX\_2\_1



## SOTTRATTORE (FS\_Nbit)

Per descrivere il sottrattore ho preso spunto dal full adder visto e spiegato in classe. Il full subtractor a N bit conterrà N sottrattori basici.

Ho ipotizzato che i valori in ingresso al sottrattore siano positivi.

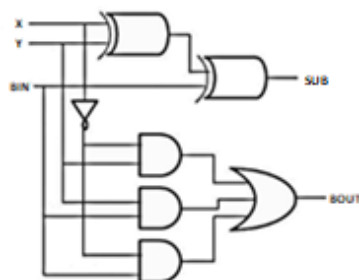


Figura 7: Circuito logico Sottrattore a 1 bit

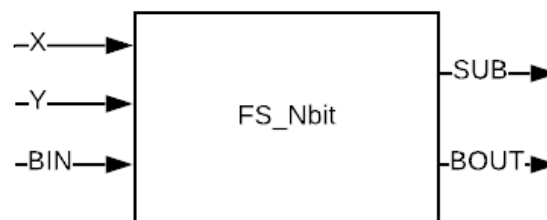


Figura 6: Sottrattore con segnali d'interfaccia

Segnali:

### INPUT

- **X** corrisponde ad un segnale che rappresenta il minuendo;
- **Y** è un segnale che rappresenta il sottraendo;
- **BIN** è un bit che indica il prestito in ingresso ed inizialmente vale 0;

### OUTPUT

- **SUB** è un segnale che rappresenta la differenza tra i valori in ingresso;
- **BOUT** è un bit che indica il prestito in uscita.

Queste sono le 2 equazioni logiche che rappresentano la sottrazione binaria:

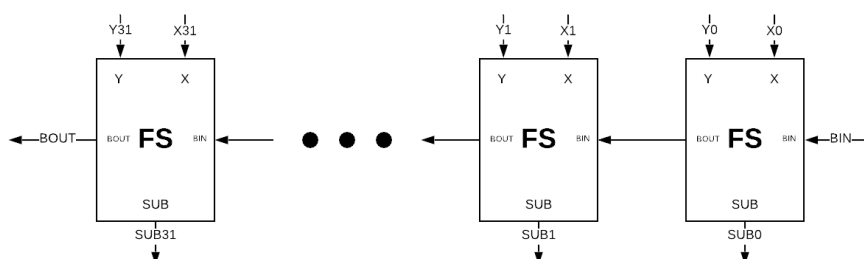
$$SUB = X \text{ XOR } Y \text{ XOR } BIN$$

$$BOUT = (X' * Y) + (X' * BIN) + (Y * BIN)$$

Si può notare la somiglianza con l'equazione utilizzata per il full adder.

Questo sottrattore viene utilizzato anche per decrementare il valore di conteggio, salvato nel registro del contatore, di un'unità.

Per effettuare l'operazione di sottrazione tra resto e divisore, vengono mappati internamente 32 full subtractor basici mentre nel sottrattore utilizzato per il conteggio vengono mappati 5 full subtractor basici e uno dei due ingressi corrisponderà al numero binario "00001".



## REGISTRO PIPO generico con reset ed enable sincroni

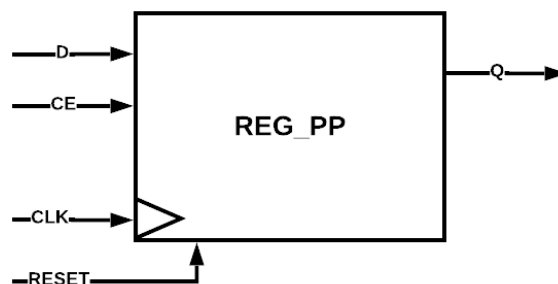


Figura 8: Registro PIPO con segnali d'interfaccia

Questo registro è un registro PIPO generico a n bit con reset e clock enable sincroni. E' utilizzato per il salvataggio di quoziente e resto e quindi ne saranno presenti 2 all'interno dell'architettura. Il reset e lo start mi permettono di inizializzare il valore salvato nel registro a 0. Il segnale di CE è utilizzato per permettere il salvataggio di nuovi valori solo quando è alto.

Segnali:

### INPUT

- **CLK** rappresenta il clock (lo stesso che è in ingresso al divisore);
- **RESET** rappresenta il segnale che consente di inizializzare quoziente e resto;
- **CE** è un bit se vale 1 permette il salvataggio del valore in ingresso;
- **D** è un segnale che rappresenta il valore da salvare nel registro.

### OUTPUT

- **Q** è un segnale che rappresenta il valore in uscita che verrà salvato al ciclo di clock successivo.

## MUX\_32\_1

Questo multiplexer viene utilizzato per estrarre dal dividendo il valore del bit in posizione i-esima (data dal conteggio). Il segnale di selezione sarà il segnale a 5 bit in uscita dal contatore. In uscita ci sarà il bit che andrà in ingresso al mux visto in precedenza per scegliere il corretto valore in output dallo shifter.

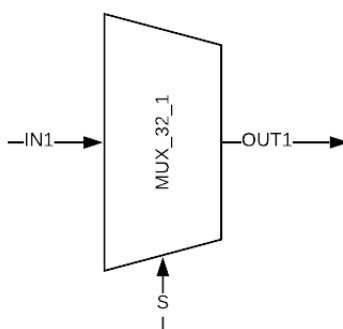


Figura 9: Rappresentazione MUX\_32\_1

## DECODER\_5\_32

Questo componente è un decoder che permette di trasformare un valore scritto in codice binario naturale in codice one-hot. Questo è necessario per poter poi successivamente fare l'assegnamento al valore i-esimo del quoziente.

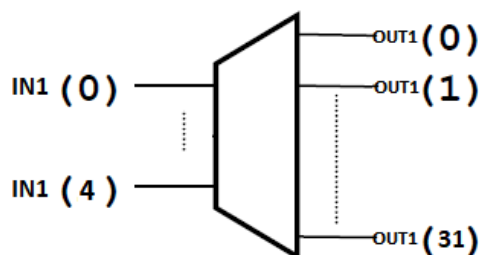


Figura 10: Rappresentazione Decoder\_5\_32

## FFD (FLIP FLOP D)

Questo componente è utilizzato per ritardare di 1 ciclo di clock la commutazione del segnale di DONE a 1, in modo tale da avere il DONE a 1 quando il quoziente ed il resto sono pronti.

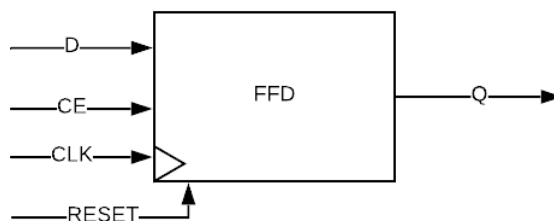


Figura 11: FFD con segnali d'interfaccia

## BLOCK\_ERROR\_CHECK

Questo componente è utilizzato per calcolare la possibile presenza/assenza di errore. Come già visto nell'esempio precedente, quando il divisore è nullo, il resto diventa uguale al dividendo mentre il quoziente prende tutti 1.

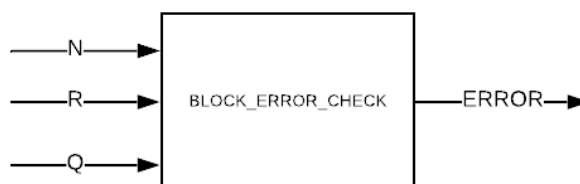


Figura 12: BLOCK\_ERROR\_CHECK con segnali d'interfaccia

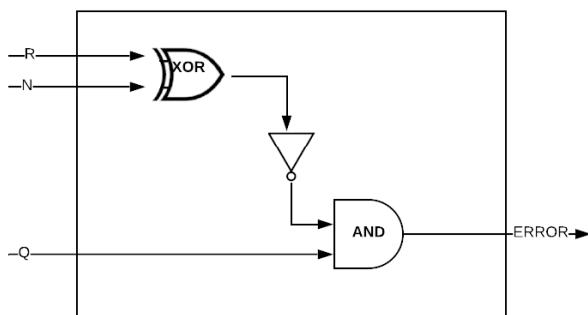


Figura 13: Rappresentazione interna BLOCK\_ERROR\_CHECK

Qui a fianco è rappresentata la struttura circuitale interna del componente. Tramite la porta XOR si verifica l'uguaglianza tra il resto e il dividendo. Se resto e dividendo sono uguali la XOR restituisce un segnale composto da tutti 0. Proprio per questo il segnale viene negato e dopo messo in ingresso alla porta AND insieme al quoziente. Se la porta AND restituisce un segnale di uscita composto da tutti 1, viene settato ad 1 il segnale di ERROR mentre prende il valore 0 altrimenti.

## BLOCK\_ASSIGNMENT\_Q

Questo componente è utilizzato per assegnare all'i-esimo valore del quoziente il valore corretto.

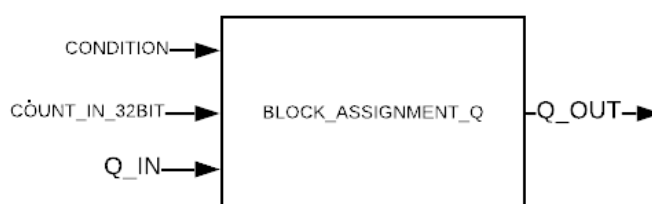


Figura 14: BLOCK\_ASSIGNMENT\_Q con segnali d'interfaccia

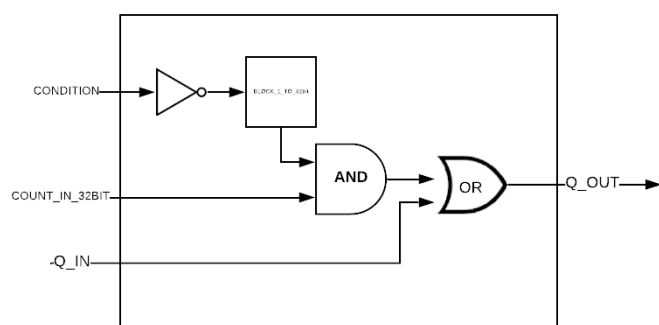


Figura 15: Rappresentazione interna BLOCK\_ASSIGNMENT\_Q

La figura a fianco rappresenta la struttura interna del componente. Il segnale CONDITION rappresenta il prestito in uscita dal sottrattore che viene negato poiché vale 1 quando al valore i-esimo di Q viene assegnato il valore 0 e viceversa. Questo segnale negato viene messo in AND con il codice one-hot del conteggio. Il segnale in uscita dalla porta AND viene messo in OR con il quoziente dell'iterazione precedente. Il segnale Q\_OUT di uscita rappresenta il nuovo valore del quoziente che viene portato in uscita dal divisore.

## FDD\_DETECT\_LEVEL

Questo componente è utilizzato quando il segnale di START va a 1 e genera un segnale di uscita ad 1 per un solo ciclo di clock. Questo meccanismo permette alla nostra architettura di poter avere il corretto funzionamento anche quando il segnale di START in input dura per più cicli di clock. Questo flip flop è utilizzato quindi come filtro. Il segnale PULSE è quello utilizzato come LOAD nel contatore e come reset nei 2 registri.

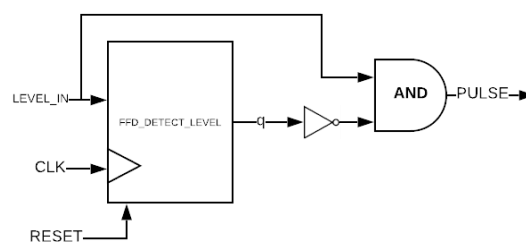


Figura 16: Architettura FFD\_DETECT\_LEVEL

## COUNTER

Il contatore viene utilizzato per effettuare le 32 iterazioni del ciclo for. Il contatore è un contatore down ovvero opera in decremento. Inizialmente il counter sarà a "11111" e dopo verrà decrementato fino a diventare "00000". Può essere utilizzato un contatore modulo  $2^n$  che lavora al contrario. Poiché dobbiamo rappresentare 32 iterazioni differenti, sono necessari 5 bit.

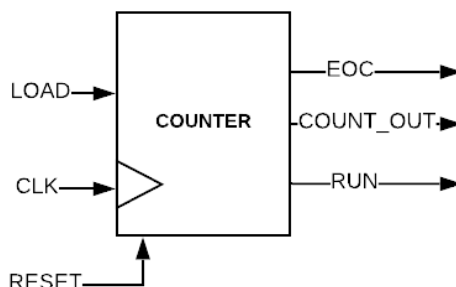


Figura 17: COUNTER con segnali d'interfaccia

Segnali:

### INPUT

- **CLK** rappresenta il clock;
- **RESET** rappresenta il segnale che consente il ripristino del segnale di conteggio interno al contatore;
- **LOAD** è un bit che quando è alto (è dato dal PULSE in uscita dal FFD\_detect\_level) permette l'aggiornamento di conteggio.

### OUTPUT

- **COUNT\_OUT** è un segnale a 5 bit che rappresenta il conteggio in uscita in binario;
- **EOC** è un bit che quando è 1 rappresenta la fine del conteggio e quindi mette ad 1 il segnale di DONE (dopo averlo ritardato);
- **RUN** è un bit che vale 1 durante il conteggio e rappresenta il CE dei registri di Q e di R.

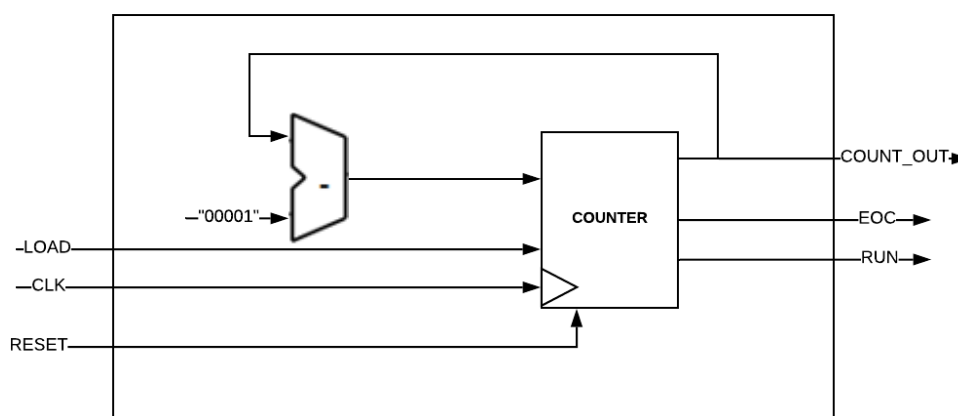


Figura 18: Struttura interna COUNTER

## Interazione componenti

Nella sezione seguente è rappresentato l'intero schema contenente tutti i componenti utilizzati e con disegnati tutti i segnali interni.

Il valore iniziale contenuto nel registro del resto viene messo in ingresso allo shifter il quale lo shifta di una posizione e restituisce due valori. Uno rappresenta il resto shiftato con il primo valore a zero mentre il secondo il resto shiftato con il primo valore ad 1. Questi due valori sono in input ad un multiplexer che seleziona il valore in base all'  $i$ -esima cifra del dividendo. Per restituire tale cifra viene utilizzato un multiplexer che prende in ingresso un numero binario a 32 bit (in questo caso il dividendo) e restituisce la cifra corrispondente la posizione indicata dal conteggio. Il valore binario di conteggio rappresenta per questo mux il valore in ingresso di selezione.

Tale valore di resto viene messo in ingresso al sottrattore insieme al divisore e viene svolta in tutti i casi la sottrazione indipendentemente che il resto sia maggiore, minore oppure uguale del divisore. In uscita dal sottrattore si trova un mux che prende in ingresso il risultato della sottrazione e il precedente valore di resto (lo stesso che andava in ingresso al sottrattore). In questo caso il valore di selezione è rappresentato dal valore del prestito in uscita dal sottrattore. Se questo vale 0 significa che il risultato della sottrazione è positivo o nullo mentre se vale 1 significa che il risultato ha prodotto un valore negativo. Utilizzando questo metodo si evita di dover aggiungere un ulteriore componente per effettuare il confronto tra resto e divisore, ma si usa direttamente il sottrattore. Il valore in uscita da questo ultimo mux viene salvato all'interno del registro di resto. Il valore in uscita da tale registro sarà lo stesso valore in uscita dal divisore.

Per trovare il corretto valore del quoziente, entrano nel BLOCK\_ASSIGNMENT\_Q il valore del quoziente salvato nel registro, il codice one-hot del conteggio e il prestito in uscita dal sottrattore. Il componente restituisce il nuovo valore del quoziente che viene messo in uscita dal divisore e viene salvato nel registro.

I registri di resto e quoziente sono abilitati al salvataggio solo quando il contatore sta effettuando l'operazione di conteggio (bit RUN alto). I 2 registri vengono inizializzati sia quando il reset è alto e sia quando il segnale di START è alto.

Quando il conteggio raggiunge il valore 0, il segnale di EOC del contatore viene ritardato di un ciclo di clock e dopo viene settato a 1 il segnale di DONE.

Per il controllo dell' errore, i segnali di resto, dividendo e quoziente vengono portati in ingresso al componente BLOCK\_ERROR\_CHECK che setta il segnale di ERROR al corretto valore.

---

Ho utilizzato i generic nell'implementazione dei componenti. Il valore è stato assegnato alle costanti durante la mappatura dei componenti all'interno del divisore.

Per trovare il possibile errore all'interno della divisione poteva anche essere utilizzato un approccio alternativo ovvero si poteva subito controllare se il divisore fosse uguale a 0 oppure no tramite una porta xor. La mia scelta è stata fatta per avere il segnale di ERROR alto solo dopo le 32 iterazioni.

L' utilizzo del reset sincrono nei registri è stata una scelta non vincolata in quanto conosco il comportamento della macchina e non mi aspetto mai di trovarmi in una situazione in cui il clock non è presente.

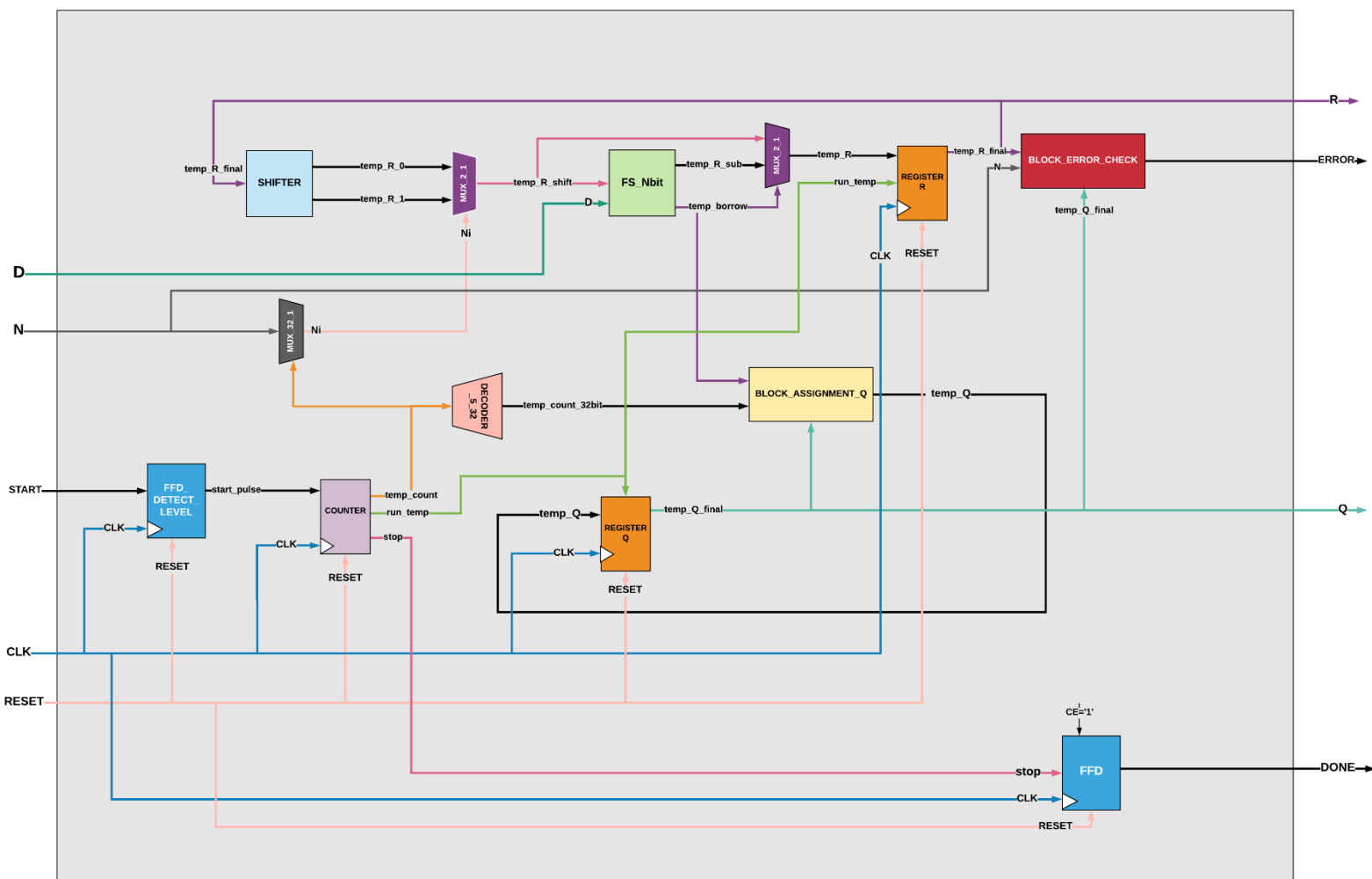



Figura 19: Struttura interna Divider

## Report utilizzo divider

Qui di seguito viene mostrato il report di utilizzo dispositivo:

Device Utilization Summary					
Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Flip Flops	75	3,840	1%		
Number of 4 input LUTs	262	3,840	6%		
Number of occupied Slices	140	1,920	7%		
Number of Slices containing only related logic	140	140	100%		
Number of Slices containing unrelated logic	0	140	0%		
Total Number of 4 input LUTs	262	3,840	6%		
Number of bonded <a href="#">IOBs</a>	133	141	94%		
Number of BUFMUXs	1	8	12%		
Average Fanout of Non-Clock Nets	4.13				

Dei 75 Flip Flops, 32 sono utilizzati per il resto, 32 per il quoziente, 7 per il contatore, 1 che rappresenta il FFD delay e 1 che rappresenta il FFD\_detect\_level. Questi flip flops sono 73 più 2 che vengono aggiunti in fase di ottimizzazione.

FlipFlop inst\_COUNTER/inst\_COUNTER\_CONTROLLER/t\_count\_0 has been replicated 1 time(s)

FlipFlop inst\_COUNTER/inst\_COUNTER\_CONTROLLER/t\_count\_1 has been replicated 1 time(s)

### Note:

A differenza del dispositivo XC3S50, visto a lezione, ho dovuto utilizzare il dispositivo **XC3S200** poiché altrimenti c'erano errori in fase di mapping(maggiore grandezza poiché divisione a 32 bit). Nello specifico questi erano i 2 errori riportati:

**ERROR:Pack:2309 - Too many bonded comps of type "IOB" found to fit this device.**

**ERROR:Pack:18 - The design is too large for the given device and package.**

Number of bonded <a href="#">IOBs</a>	133	124	107%	OVERMAPPED
---------------------------------------	-----	-----	------	------------

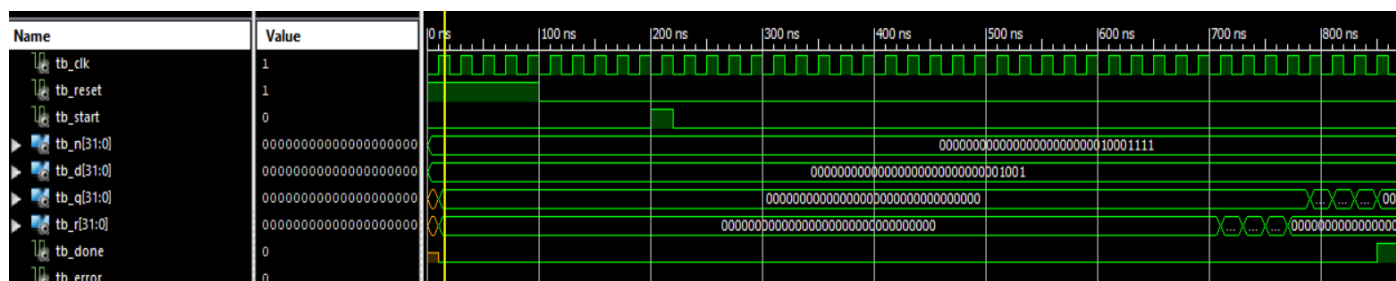
Utilizzando il componente XC3S200 gli errori si sono risolti poiché il numero di IOBs (numero di pin) forniti dal componente è 141 a differenza dei 124 forniti dal dispositivo XC3S50.



# TEST BENCH

Come primo test bench ho deciso di testare l'esempio presentato in precedenza. Il dividendo è rappresentato dal numero decimale 143 mentre il divisore dal numero 9.

Inizialmente il segnale di RESET vale 1 in modo tale da inizializzare quoziente e resto. Il divisore è quindi pronto a ricevere il segnale alto di START che dà inizio alle iterazioni. Il segnale di START può durare anche più cicli di clock tanto viene filtrato dal FFD\_detect\_level. Dopo le 32 iterazioni, resto e quoziente sono pronti e viene posto a 1 il segnale di DONE.



Un secondo test bench provato è stato quello con il **divisore nullo** per verificare la correttezza del segnale ERROR. Tale segnale si alza in corrispondenza del segnale DONE quando il quoziente è composto da tutti 1 mentre il resto coincide con il dividendo.

Oltre ad avere provato questi test bench a livello behavioral, sono stati testati anche a livello post-route il tutto con un tempo di ciclo di clock impostato a 20 ns.

Si può verificare che il risultato sia corretto verificando che il dividendo sia uguale alla somma tra il resto e il prodotto tra divisore e quoziente. Questo controllo deve essere fatto quando il segnale di DONE è alto.

Altri test bench utili implementati sono la **divisione perfetta** ovvero quando il resto è nullo, la divisione quando il dividendo è minore del divisore (**N<D**) e quindi il quoziente è nullo ed il resto è uguale al dividendo e la divisione quando il dividendo è uguale a divisore dove il quoziente diventa 1 con il resto di 0 (**N=D**).

Per maggiore chiarezza, dividendo e divisore vengono scritti in esadecimale( 8 cifre) anziché dover scrivere 32 cifre in codice binario.

Per maggiori dettagli, avviare il test bench da xilinx ise webpack.



# RIFERIMENTI

Qui di seguito sono riportati i **link** da cui ho ricavato informazioni utili ai fini dello svolgimento dell'intero progetto:

<https://www.allaboutcircuits.com/technical-articles/how-to-vhdl-description-of-a-simple-algorithm-the-control-path/>

<https://staff.emu.edu.tr/muhammedsalamah/Documents/CMPE224/labs/Exp7.pdf>

<https://www.allaboutcircuits.com/technical-articles/implementing-a-finite-state-machine-in-vhdl/>

<http://unina.stidue.net/Architettura%20dei%20Sistemi%20di%20Elaborazione/Materiale/Ad dizionatori%20v1.01.pdf>

[Introduzione al linguaggio VHDL \(Carlo Brandolese\)](#)

[https://people.unica.it/massimobarbaro/files/2014/06/07didel\\_rtl.pdf](https://people.unica.it/massimobarbaro/files/2014/06/07didel_rtl.pdf)

[http://users.encs.concordia.ca/~asim/COEN\\_6501/project\\_Giovanni\\_D'Aliesio.pdf](http://users.encs.concordia.ca/~asim/COEN_6501/project_Giovanni_D'Aliesio.pdf)

<https://www.xilinx.com/support/documentation/university/Vivado-Teaching/HDL-Design/2015x/VHDL/docs-pdf/lab6.pdf>