**Davide Carini** **(Matricola:890064 – CodicePersona: 10568649)**

# PROVA FINALE

## (Progetto n° 11 di Reti Logiche)

DOCUMENTAZIONE

**Corso di Reti Logiche**

**Anno Accademico 2019/2020**

**Professore: Carlo Brandolese**

# INDICE

# INTRODUZIONE

## Scopo del Progetto

Lo scopo è progettare un divisore intero su 32 bit basato sul metodo di "divisione lunga". Siano N il dividendo, D il divisore, Q il quoziente , R il resto e i la dimensione delle parole, l'algoritmo è descritto dal seguente pseudocodice:

```
//Division By zero
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 e la verifica del corretto funzionamento nei diversi casi.

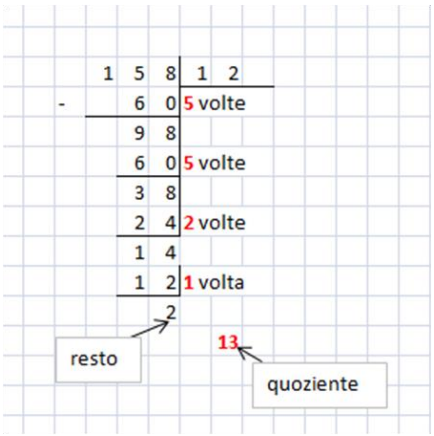



*Figura 1. Esempio di "Lunga Divisione"*

L'algoritmo di "divisione lunga" è l'algoritmo standard più utilizzato per la divisione di numeri espressi in notazione decimale spesso chiamato 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 descritto tramite diagramma di flusso:

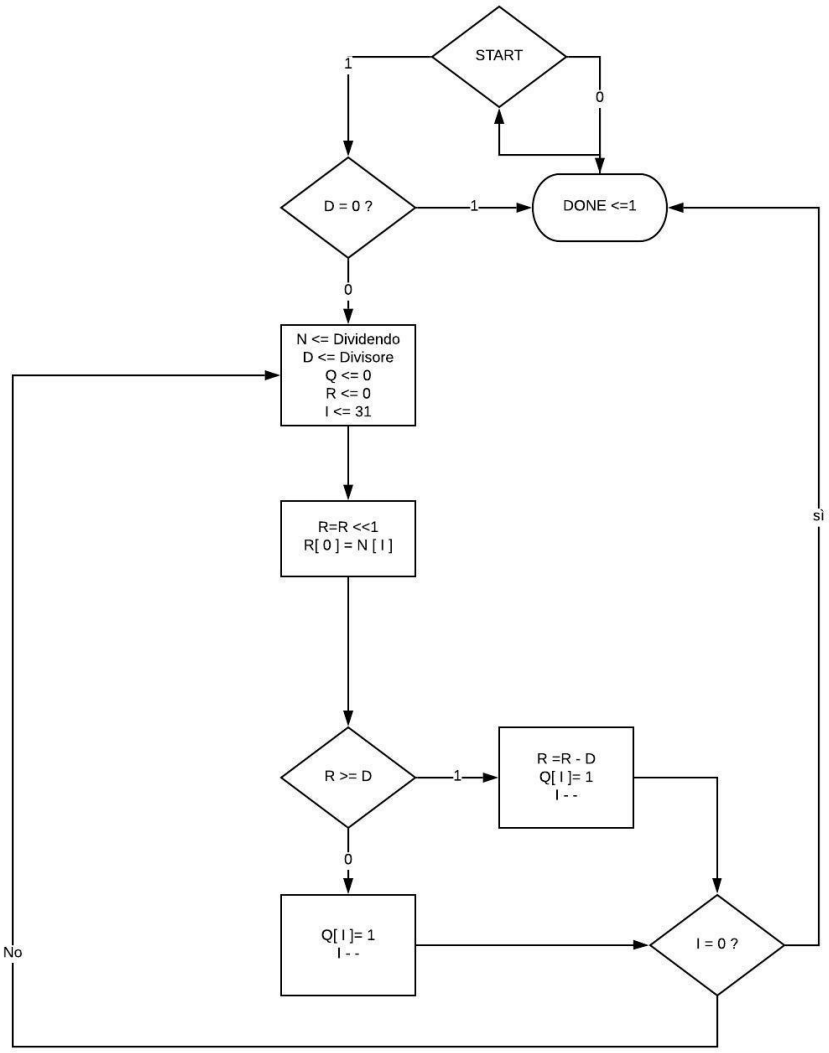



*Figura 2. Diagramma di flusso*

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 parallelamente. Il problema è che a differenza di una cpu che ad ogni ciclo di pipeline carica una nuova istruzione , in questa 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.

Ho pensato quindi di risolvere l'algoritmo utilizzando un divisore descritto in modo strutturale che spiegherò dopo l'esempio di implementazione del ciclo for.

## Esempio ( con n=8 bit )

Dimostro tramite un esempio 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). 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

$R=1000_2$ -> $8_{10}$

$Q=00001111_2$ -> $15_{10}$

La divisione è stata fatta 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.

# DESIGN ARCHITETTURA

## Interfaccia del componente (top level)

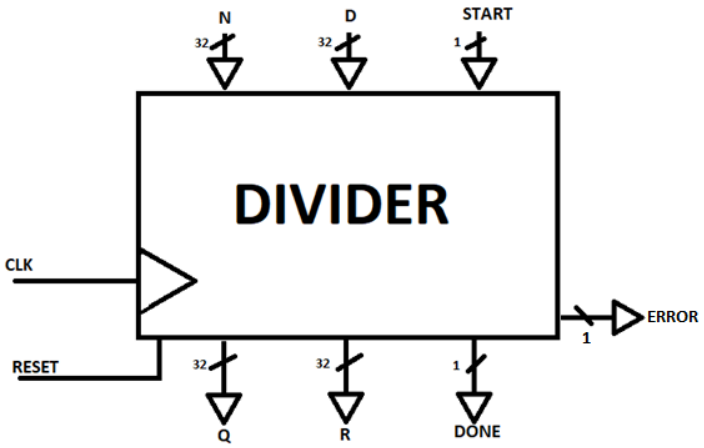
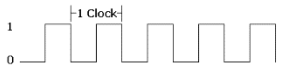



*Figura 3. Componente top level con segnali di interfaccia*

Il divisore è composto da 5 segnali di ingresso e 4 segnali di 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 corrisponde all'inizio dell' algoritmo ;
- CLK rappresenta il clock utile a sincronizzare i dispositivi interni al divisore ;




- RESET è il segnale che consente il ripristino dei valori dei segnali interni al dispositivo .

Le 4 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.

Il divisore (componente top level) mi servirà per mappare i sottocomponenti e quindi permettere la corretta comunicazione tra essi . All'interno del divisore dovrò inserire le diverse istanze dei componenti.

In totale i segnali d'interfaccia saranno dunque 9.

Nel divisore ci dovrà essere un process con all'interno tutte le istruzioni che verranno eseguite in modo sequenziale. Nella sensitivity list ci saranno il clock e il reset e il processo verrà stimolato solo all' attivazione di un evento su uno dei due segnali nella lista.

## Entity principali

I principali sotto componenti che ho pensato di utilizzare sono :

- Left shifter
- Full subtractor
- Contatore
- Comparatore

Tali entity sono tutte reti combinatorie(ovvero non contengono il clock) tranne il contatore che è sequenziale.

Vediamo i seguenti componenti nel dettaglio:

# LEFT SHIFTER

Lo shifter è utilizzato per effettuare il left shift del resto nella prima istruzione del ciclo. Al resto viene troncato il MSB e viene aggiunto uno 0 come LSB. Questo risultato sarà quindi il nuovo valore del resto che dovrà essere utilizzato nell'istruzione successiva. In VHDL per implementare lo shift dovrò fare riferimento al costrutto di slicing dove uno slice rappresenta una porzione di vettore e all' operatore di concatenamento.


*Figura 5. Entity con segnali di interfaccia*

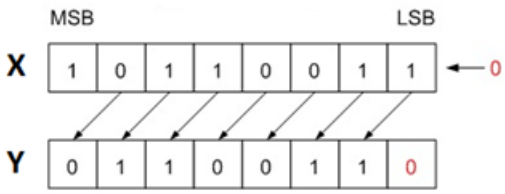



*Figura 4. Esempio di left shift*

Segnali:

- R è un segnale a 32 bit che rappresenta il resto da shiftare;
- S è un bit che rappresenta il valore i-esimo del dividendo in un determinato istante;
- RS è un segnale a 32 bit che rappresenta il resto shiftato e con il primo bit avente il valore di N[i] ad ogni istante.

Ho aggiunto la modifica del primo bit di R shiftato nello shifter in quanto altrimenti ho avuto problemi a gestirlo nel process del componente top level. In questo modo RS contiene il valore corretto che rispecchia le prime due istruzioni del ciclo for.

# COMPARATORE a 32 bit

Il comparatore è un circuito combinatorio che prende come ingressi il resto e il divisore. Questo componente mi sarà utile per verificare la condizione del costrutto if del ciclo.

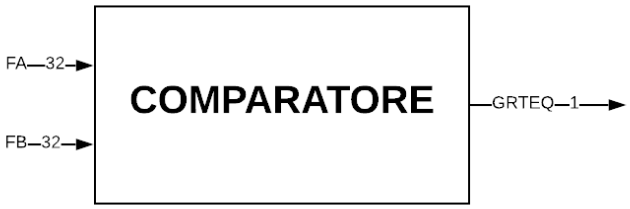



*Figura 6.Entity con segnali di interfaccia*

Segnali:

- FA è un segnale a 32 bit che rappresenta il resto shiftato e con il primo bit modificato( è il valore di uscita dallo shifter);
- FB è un segnale a 32 bit che corrisponde al divisore;
- GRTEQ è un bit che assume il valore 1 quando FA>= FB mentre assume il valore 0 quando FA < FB.

# SOTTRATTORE a 32 bit

Per descrivere il sottrattore ho preso spunto dal full adder. Il full subtractor a 32 bit conterrà 32 subtractor basici. La sottrazione avrà sempre come risultato un numero positivo in quanto la sottrazione verrà fatta solo se è rispettata la condizione: R >=D.

Il sottrattore prende come ingressi R e D e restituirà il nuovo valore di R.

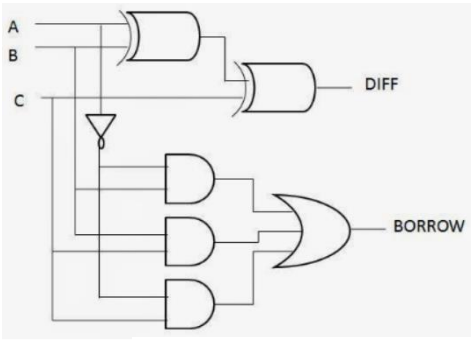



*Figura 8.Circuito logico sottrattore*


*Figura 7. Sottrattore con segnali d'interfaccia*

Segnali:

- A corrisponde ad un segnale a 32 bit che rappresenta il valore in uscita dallo shifter;
- B è un segnale a 32 bit che rappresenta il divisore;
- C è un bit che avrà come ingresso 0;
- D è un segnale a 32 bit che rappresenta la differenza tra i valori in ingresso;
- B0 è un bit che indica il prestito.

# CONTATORE

Il contatore viene utilizzato per effettuare le 32 iterazioni del ciclo for. Il contatore sarà 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, saranno necessari 5 bit.

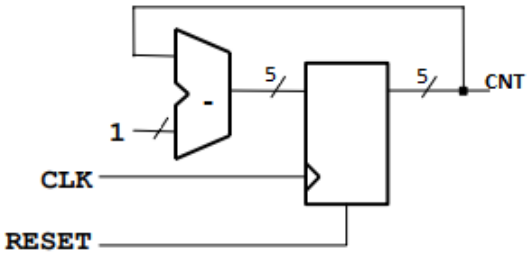



*Figura 9. Rappresentazione logica contatore*

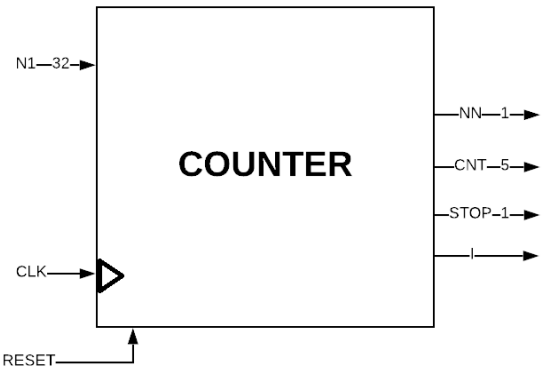



*Figura 10. Segnali contatore*

Segnali:

- CLK rappresenta il clock( lo stesso che è in ingressi al divisore);
- RESET rappresenta il segnale che consente il ripristino del contatore;
- N1 è un segnale a 32 che rappresenta il dividendo( è necessario per poter calcolare il valore N[i] da passare allo shifter);
- NN rappresenta N[i];
- CNT è un segnale a 5 bit che rappresenta il conteggio in binario;
- STOP è il segnale che viene settato ad 1 quando il segnale interno al contatore raggiunge il valore “00000”;
- I è un integer che rappresenta il valore del conteggio convertito in intero.

## Interazione componenti

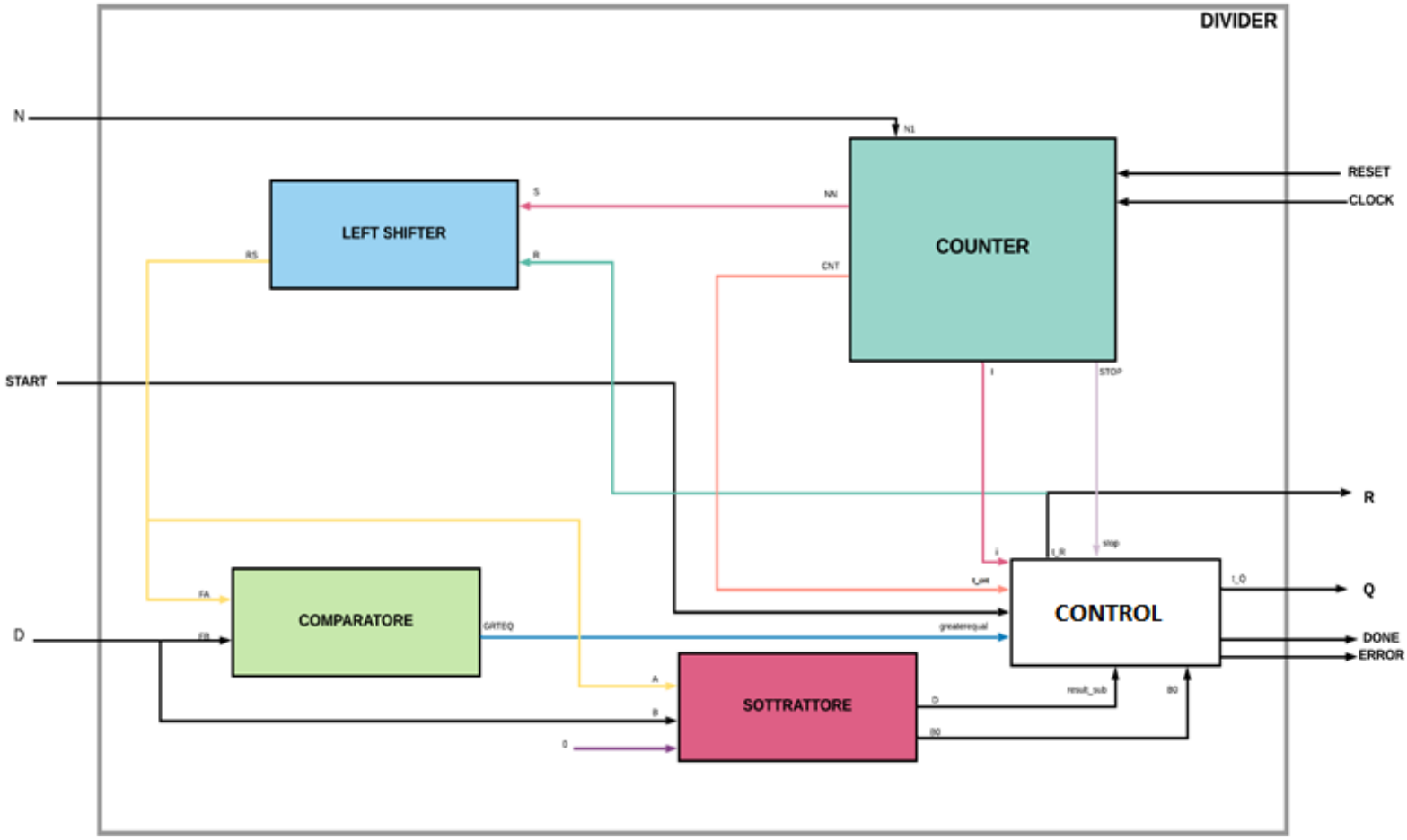



*Figura 11.Schema a blocchi architettura*

Il control rappresenta quello che viene svolto nel divisore e mi serve per collegare i vari componenti tra di loro. Tutti i segnali collegati al control sono segnali interni al divisore.

Ad ogni ciclo di clock il contatore decrementa il conteggio e calcola N[i]. Lo shifter shifta appunto il resto ottenuto dal ciclo precedente e modifica il primo bit. Sempre nello stesso ciclo di clock viene fatto il confronto tra il resto shiftato e il divisore. Se R>= D il resto viene mandato al sottrattore e viene modificato il bit i-esimo di t_Q altrimenti viene solo modificato il valore i-esimo di t_Q. Viene poi aggiornato il valore di t_R per il ciclo di clock successivo.

Quando il conteggio raggiunge il valore 0, il counter attiva il segnale di stop e viene messo a 1 il segnale di DONE.

# TEST BENCH

Per verificare la correttezza del progetto mi basterà provare creare un test bench sul divisore ed osservare i diversi risultati al variare degli ingressi. Dati i 32 bit, mi conviene utilizzare delle variabili di appoggio integer che posso poi convertire in std_logic_vector.

Inizialmente il reset sarà ad 1 in modo tale da inizializzare i segnali interni. Il clock avrà un tempo di 10 ns.

Dovrò testare la correttezza dell'algoritmo anche quando il divisore è posto a 0.

Come primo esempio di test bench partirò con l' esempio descritto in precedenza in modo tale  che potrò verificare passo per passo la correttezza.

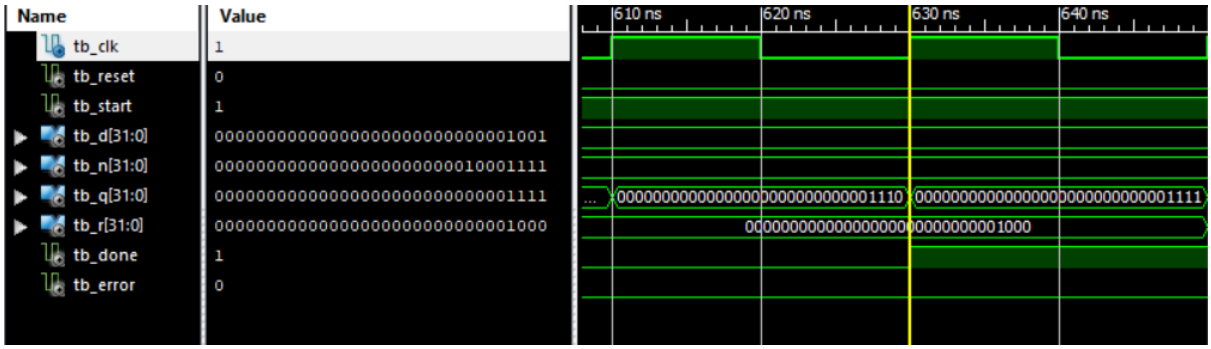



*Figura 12.Esempio di divisione(143/9)*

# RIFERIMENTI


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/Addizionatori%20v1.01.pdf

Introduzione al linguaggio VHDL (Carlo Brandolese)


## Note:

A differenza del componente(XC3S50) visto a lezione, ho dovuto utilizzare quello con più porte ovvero il XC3S200 poiché altrimenti avevo degli errori in fase di mapping. 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.

Utilizzando il componente XC3S200 gli errori si sono risolti.