Davide Carini

Matricola: 890064 – Codice Persona: 10568649

PROVA FINALE

(Progetto n° 11 di Reti Logiche)

Corso di Reti Logiche

Anno Accademico 2019/2020

Professore: Carlo Brandolese

INDICE

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:

```
//Division By zero
if( D == 0 ) {
        error();
}
Q = 0;
R = 0;
for(i = n-1; i >= 0; i-- ) {
        R = R << 1;   //Left shift
        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.

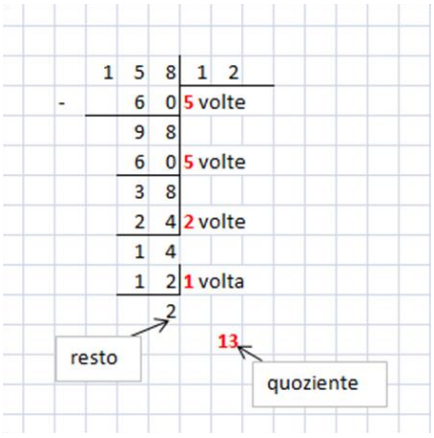


Figura 1: Esempio di "Lunga Divisione"

L'algoritmo di "**divisione lunga**" è l'algoritmo standard più utilizzato per la divisione di numeri espressi in notazione decimale 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 sottoproblemi 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.

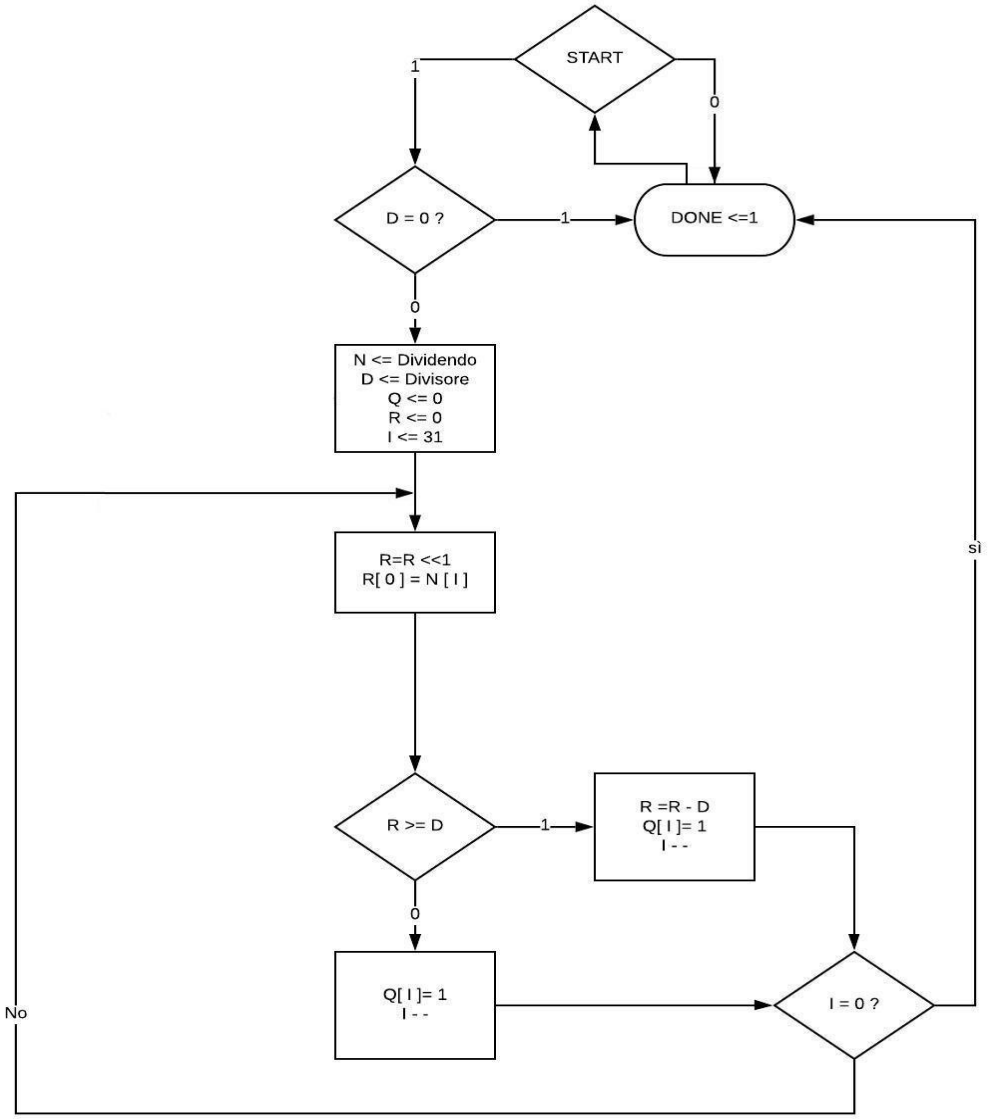


Figura 2: Diagramma di flusso rappresentante l' algoritmo

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 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.

Il diagramma di flusso mi fornisce una descrizione dal punto di vista SW e quindi necessito di un data path 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) che verrà mostrato e spiegato 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 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_2 -> 8_{10}

Q=00001111_2 -> 15_{10}

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.

DESIGN ARCHITETTURA

Interfaccia del componente (top level)

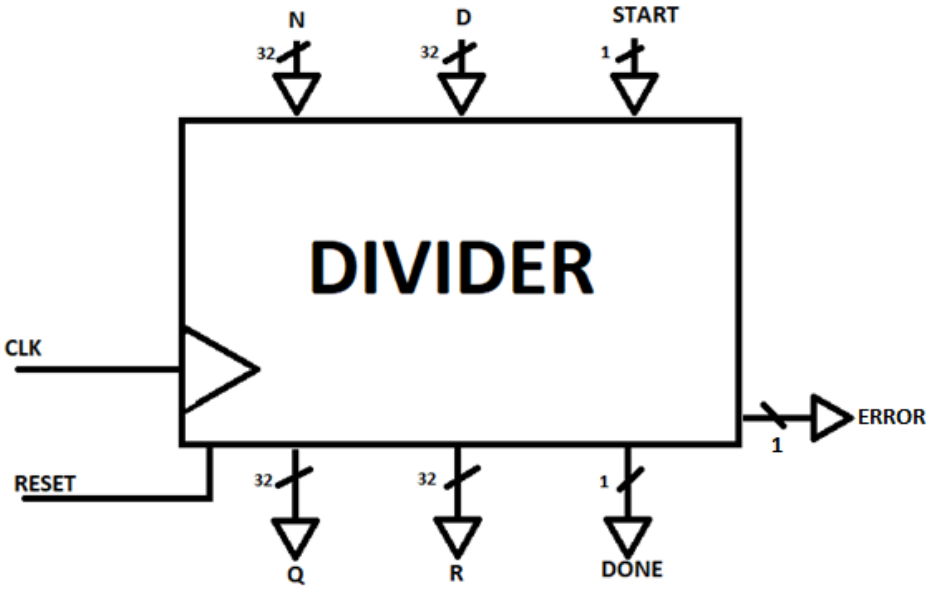


Figura 3: Componente top level con segnali di interfaccia

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.

Nel divisore (componente top level) vengono mappati i componenti interni.

All'interno del divisore sono inserite le varie istanze dei componenti ed i segnali di controllo che permettono la corretta comunicazione ed interazione tra i sottocomponenti. I segnali di controllo sono pilotati dalla macchina a stati(descritta in dettaglio successivamente).

Il divisore può essere visto come un insieme di componenti di logica combinatoria e di registri atti a salvare risultati di elaborazioni intermedie. Di seguito viene mostrato un diagramma a blocchi i cui componenti verranno spiegati dettagliatamente.

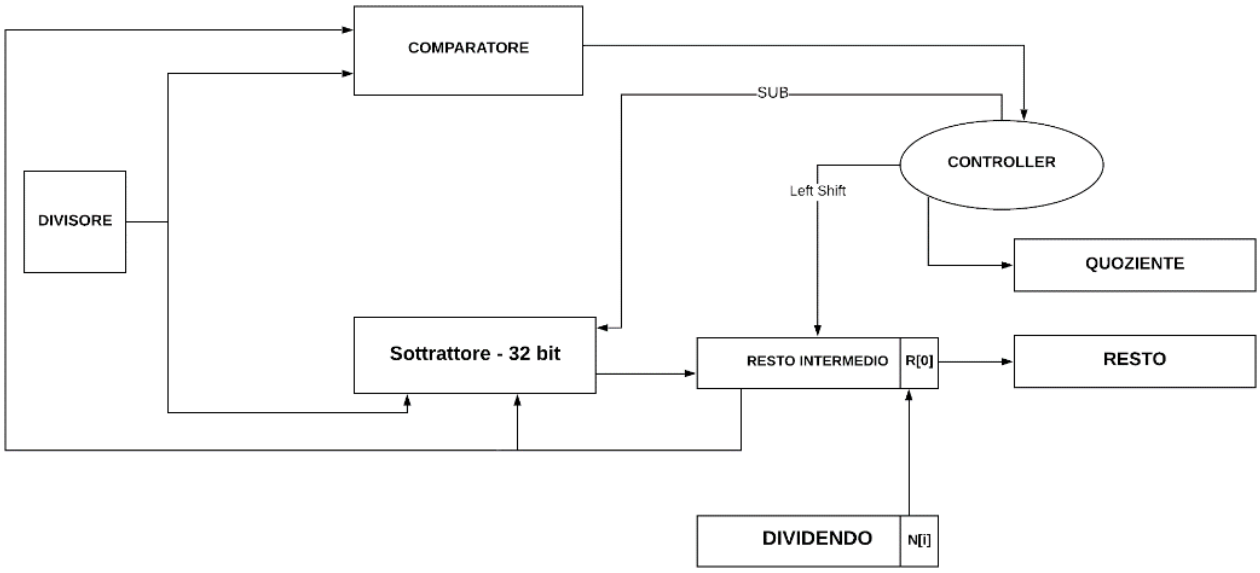


Figura 4: Diagramma a blocchi rappresentante il Datapath

Questo diagramma è una rappresentazione semplificata rispetto a quello che è stato sviluppato realmente. Tale schema ci permette però di avere una visione chiara di quello che sarà implementato e dei componenti che ci serviranno per effettuare la divisione.

Il datapath contiene permette l'interazione tra i registri e la logica combinatoria presente nel nostro sistema.

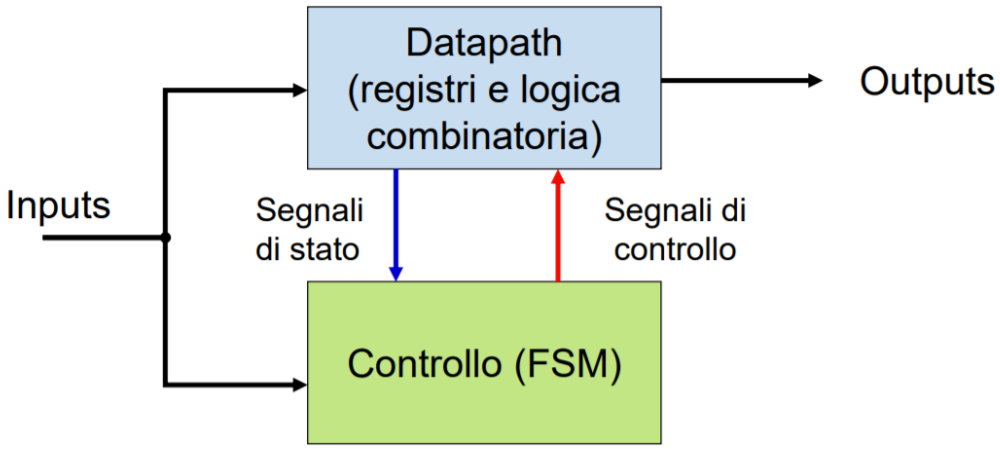


Figura 5: Interazione tra FSM e Datapath

Lo schema rappresenta proprio quello che verrà mostrato successivamente e che già nel diagramma a blocchi viene in parte mostrato. Il controller appunto controlla la rete sequenziale in modo tale che segua tutti gli step utili alla corretta implementazione dell'algoritmo.

Entity principali

I principali componenti sviluppati ed utilizzati sono :

- **Sottrattore a 32 bit**
- **Contatore**
- **Comparatore a 32 bit**
- **3 Registri generici a 32 bit**

Sottrattore e comparatore sono reti combinatorie mentre i registri e il contatore sono reti sequenziali.

Vediamo i seguenti dispositivi nel dettaglio:

COMPARATORE a 32 bit

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

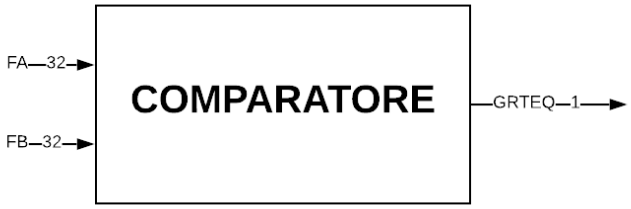


Figura 6: Comparatore con segnali di interfaccia

Segnali:

- **FA** è un segnale a 32 bit che rappresenta il resto shiftato e con il primo bit modificato;
- **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 visto in classe. Il full subtractor a 32 bit conterrà 32 sottrattori 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 il resto e il divisore e restituisce il nuovo valore del resto.

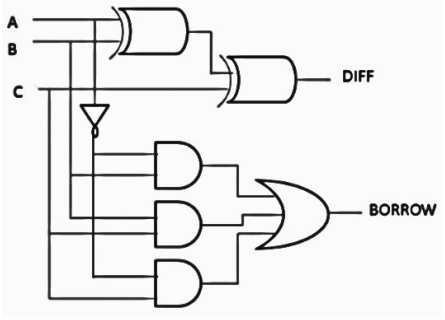


Figura 8: Circuito logico sottrattore

Figura 7: Sottrattore con segnali d'interfaccia

Segnali:

- **A** corrisponde ad un segnale a 32 bit che rappresenta il valore del resto;
- **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 a 5 bit

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. Il counter ha una funzione di control unit dell'intero sistema e quindi implementa una macchina a stati che genera i segnali di controllo per il datapath.

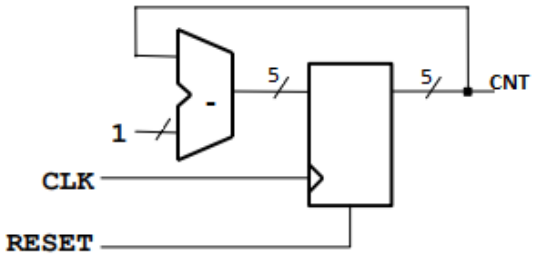


Figura 10: Rappresentazione logica contatore

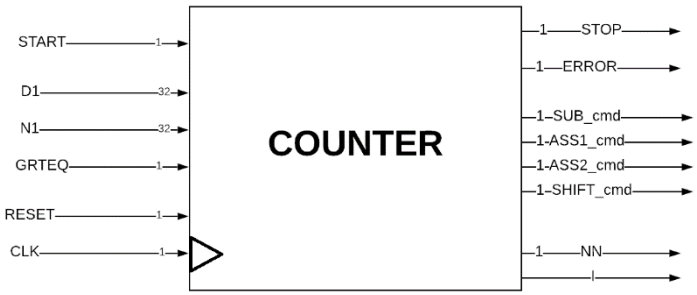


Figura 9: Contatore con segnali d'interfaccia

Segnali:

- **CLK** rappresenta il clock(lo stesso che è in ingresso al divisore);
- **RESET** rappresenta il segnale che consente il ripristino del contatore;
- **N1** è un segnale a 32 bit che rappresenta il dividendo(è necessario per poter calcolare il valore N[i]);
- **D1** rappresenta il divisore (mi serve per controllare che sia diverso da 0);
- **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";
- **START** è un segnale di un bit che regola l'inizio del processo di divisione;
- **NN** rappresenta N[i] ovvero il valore della cifra in posizione i-esima del dividendo ;
- **I** è un integer che rappresenta il valore del conteggio convertito in intero;
- **SUB_cmd** è un bit che vale 1 quando ci si trova nello stato di SUB;
- **SHIFT_cmd** è un bit che vale 1 quando ci si trova nello stato di SHIFT;
- **ASS1_cmd** è un bit che vale 1 quando ci si trova nello stato di ASS1;
- **ASS2_cmd** è un bit che vale 1 quando ci si trova nello stato di ASS2;
- **GRTEQ** è un bit che rappresenta l'uscita del comparatore.

DESCRIZIONE MACCHINA A STATI (FSM)

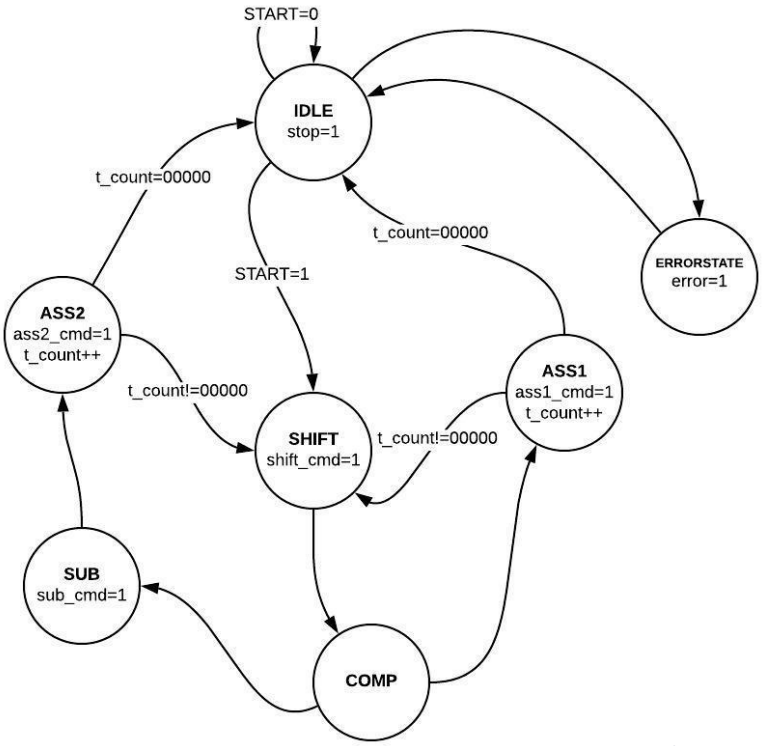


Figura 11: Macchina a stati(FSM)

DESCRIZIONE DEGLI STATI

- **IDLE** rappresenta lo stato iniziale in cui il segnale di stop è alto;
- **ERRORSTATE** è lo stato in cui si passa nel caso in cui il divisore abbia valore nullo (division by zero);
- **SHIFT** rappresenta lo stato in cui deve essere effettuato lo shift del resto e deve essere modificato il suo primo bit;
- **COMP** è lo stato in cui deve essere effettuato il confronto tra il resto e il divisore ed in base al risultato si prendono due strade distinte;
- **SUB** è lo stato in cui viene effettuata l' operazione di differenza tra il resto e il divisore;
- **ASS2** è lo stato successivo a quello di SUB dove avviene l'assegnamento del valore 1 all' i-esimo valore del quoziente;
- **ASS1** è lo stato in cui avviene l' assegnamento del valore 0 all' i-esimo valore del quoziente.

Questi segnali servono per il controllo del datapath.

I 3 registri presentati successivamente sono **registri parallelo-parallelo** a 32 bit generici con reset asincrono.

REGISTRO QUOZIENTE

Questo componente mi serve per il **salvataggio del quoziente**. Oltre a contenere il valore di Q, effettua un' operazione di assegnamento in base al valore degli ingressi ass1_cmd e ass2_cmd.

Il reset, inizialmente attivo, mi permette di inizializzare il registro temporaneo interno a 0.

Figura 12: Registro Q con segnali d'interfaccia

REGISTRO RESTO INTERMEDIO

Questo registro effettua lo shifting del valore del registro temporaneo quando SHIFT_cmd è uguale a 1 oppure assegna al registro temporaneo il valore di uscita proveniente dal sottrattore quando SUB_cmd è uguale a 1. Il valore S rappresenta il valore i-esimo del dividendo in output dal contatore. Questo registro rappresenta dunque uno **shift register**.

Figura 13: Registro intermedio resto con segnali d'interfaccia

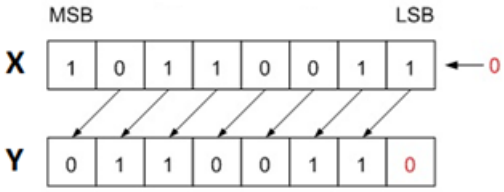


Figura 14: Esempio di left shift

REGISTRO RISULTATO RESTO

Questo registro prende in input il valore in ouput dal registro R e lo porta in uscita al divisore. Il valore salvato all'interno di questo registro rappresenta il **risultato** del resto alla fine di ogni iterazione di ciclo for.

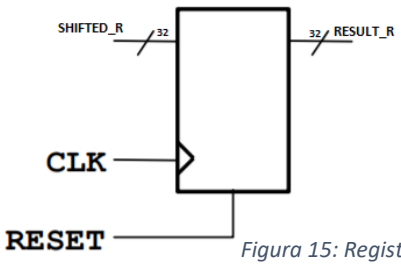


Figura 15: Registro risultato resto con segnali d'interfaccia

Interazione componenti

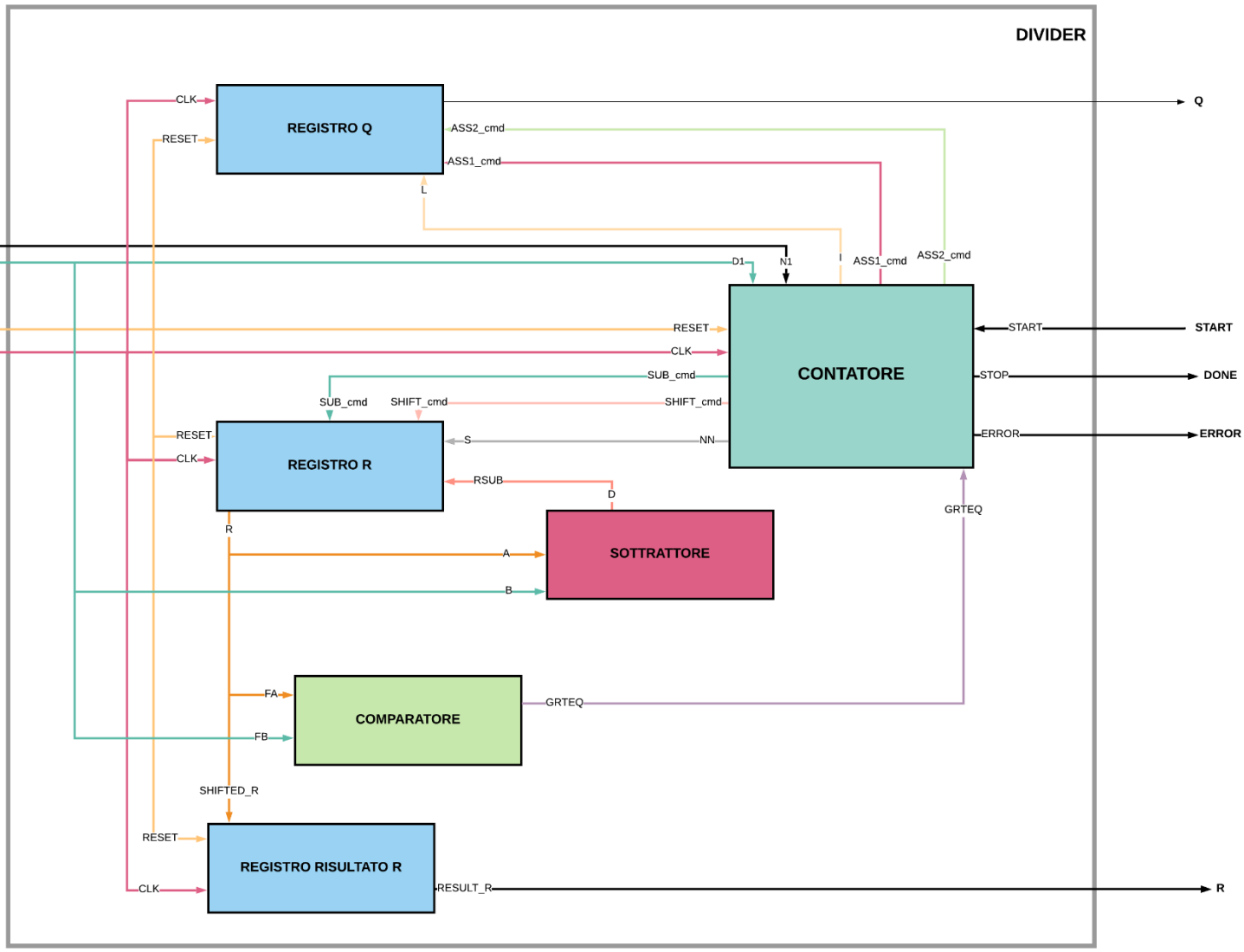


Figura 16: Schema interazione componenti interni al divisore

In questo schema vengono rappresentati i collegamenti interni tra i vari componenti. Come si può notare, è ben chiara la funzione di controllore da parte del contatore il quale tramite segnali, appunto segnali di controllo(con suffisso _cmd) , permette il corretto funzionamento. Questi segnali sono attivi nei vari stati della macchina a stati.

Quando il contatore finisce il conteggio e il segnale di STOP diventa alto, il segnale di DONE diventa 1.

Reset e clock vanno in ingresso ai vari componenti sequenziali interni al divisore in modo da farli lavorare in modo sincrono.

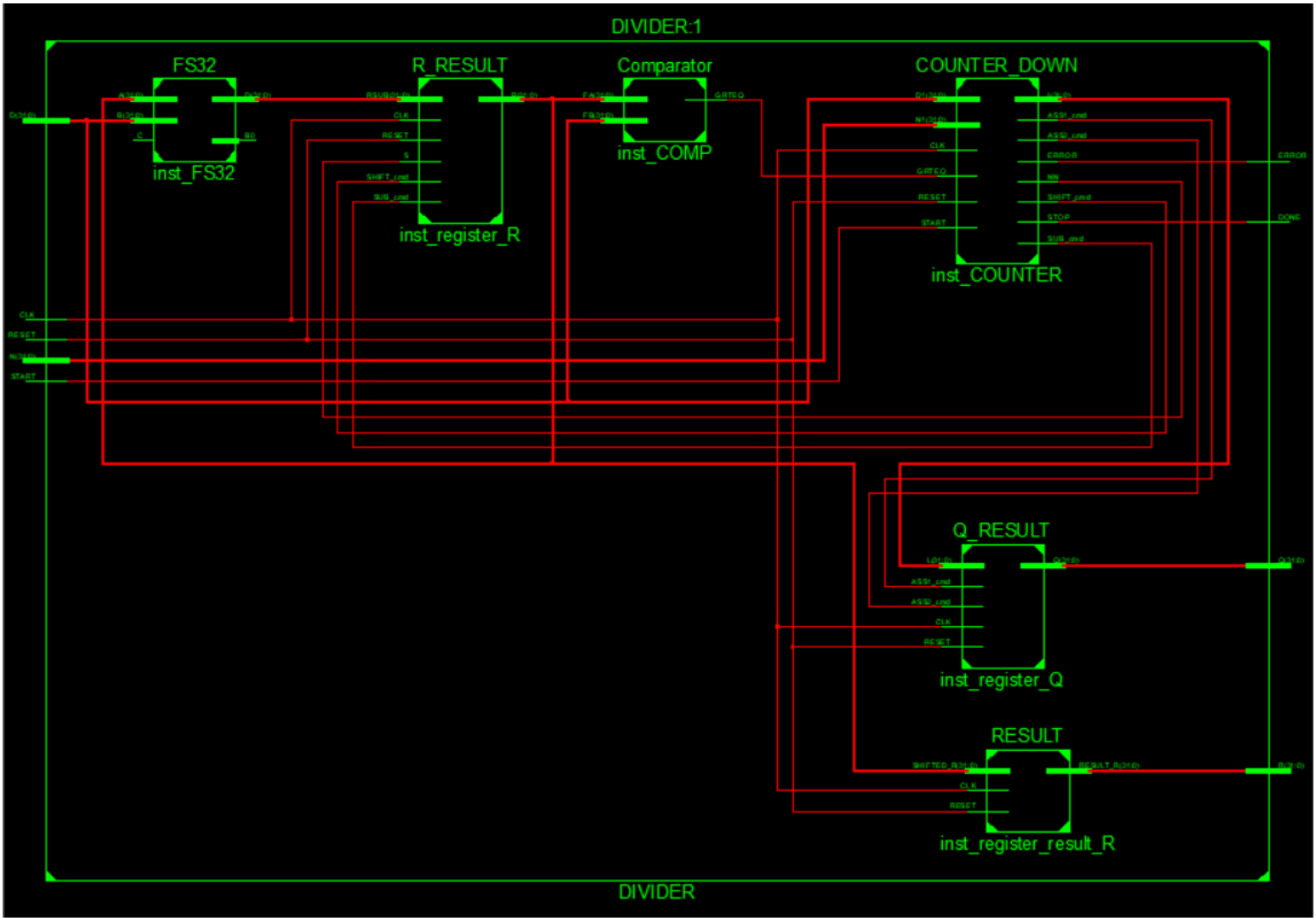


Figura 17: Schema RTL generato tramite XILINX ISE

Questo schema è uguale a quello mostrato in precedenza solo che è stato generato tramite il view schematic RTL di XILINX ISE WEBPACK.

TEST BENCH

Per verificare la correttezza del progetto, ho implementato un test bench sul divisore ed ho osservato il funzionamento del dispostivo al variare degli ingressi. Dati i 32 bit, mi è convenuto utilizzare variabili di appoggio intere che poi ho convertito in std_logic_vector.

Inizialmente il reset è 1 in modo tale da inizializzare i segnali interni. In seguito il segnale di reset viene posto a 0. Il clock è stato impostato su un tempo di 10 ns.

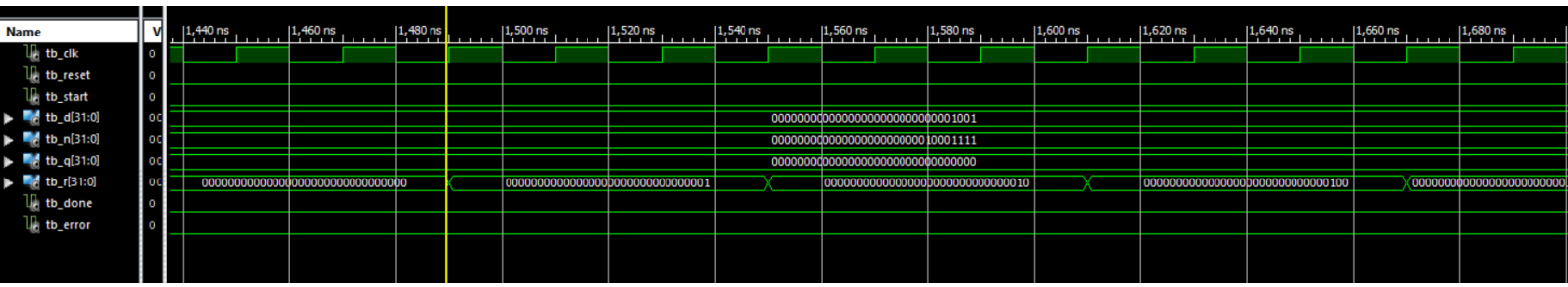
Quando il segnale Done commuta ad 1, il resto ed il quoziente sono pronti.

Ho implementato test bench singoli su comparatore e sottrattore in modo da testare il corretto funzionamento dei singoli dispositivi presenti.

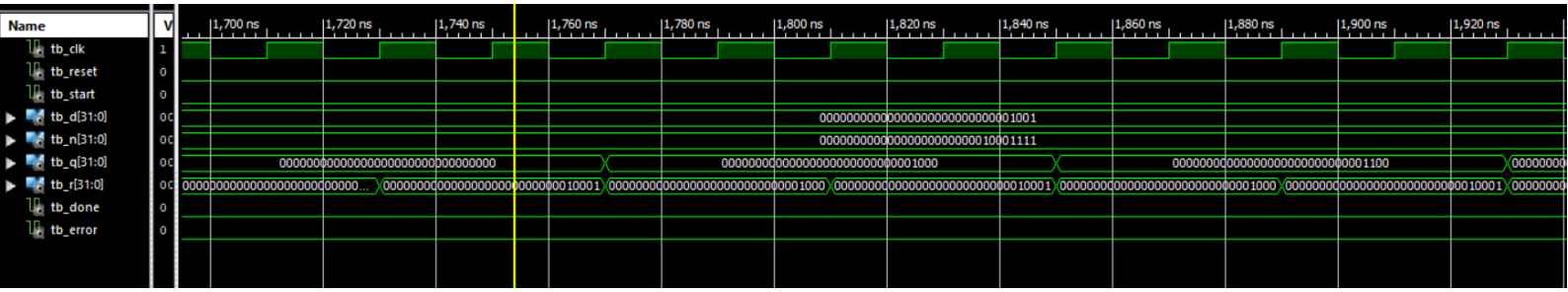
Ho provato la correttezza del codice VHDL anche quando il divisore è posto a 0.

Come primo test bench sono partito dall' esempio presentato all'inizio della documentazione

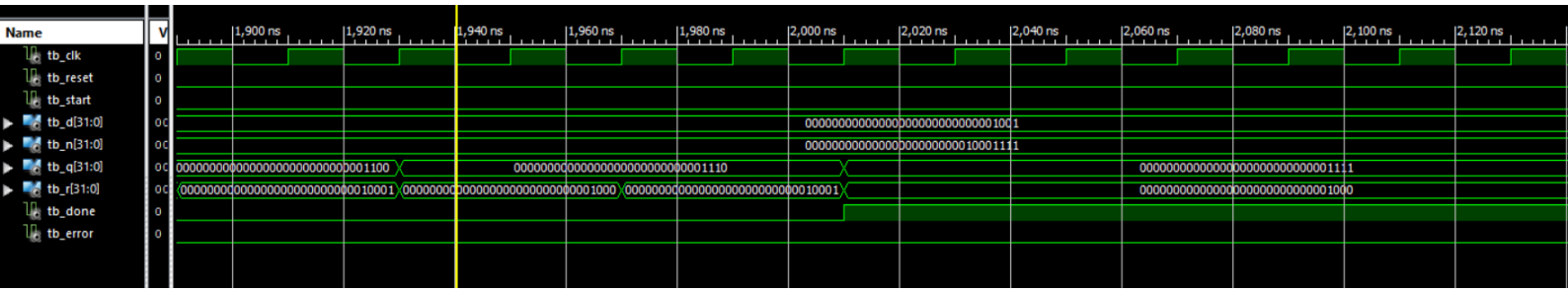
Di seguito sono riportate 3 schermate di ISim catturate durante la divisione di esempio.



Dopo 1,480 ns si ha il primo assegnamento di 1 a R[0].



A 2,010 ns il segnale di DONE diventa a 1 ed il quoziente ed il resto contengono il risultato corretto.



Il test bench produce il risultato corretto sia a livello behavioural sia a livello post-route.

Il codice VHDL darà una migliore visione sul test bench presentato.

RIFERIMENTI

Qui in 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/Addizionatori%20v1.01.pdf

Introduzione al linguaggio VHDL (Carlo Brandolese)

https://people.unica.it/massimobarbaro/files/2014/06/07didel_rtl.pdf

http://users.encs.concordia.ca/~asim/COEN_6501/project_Giovanni_D'Aliesio.pdf

Note:

A differenza del dispositivo(XC3S50) visto a lezione, ho dovuto utilizzare quello con un numero maggiore di porte ovvero il **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.

Utilizzando il componente XC3S200 gli errori si sono risolti.

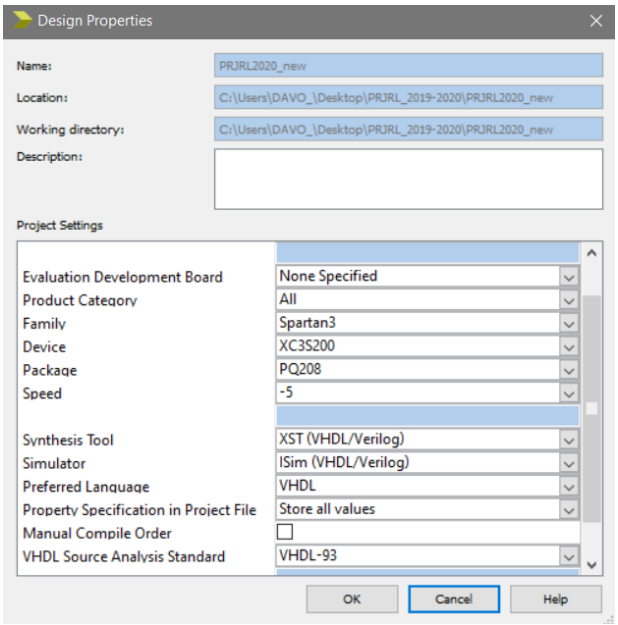


Figura 18: Proprietà del dispositivo Divider