



Università degli Studi di Udine

DIPARTIMENTO POLITECNICO DI INGEGNERIA E ARCHITETTURA

Corso di Laurea Magistrale in Ingegneria Elettronica

RELAZIONE PROGETTO: DIVIDER e MCD

Gruppo:

Simone Dotto

Riccardo Marcon

ANNO ACCADEMICO 2021/2022

Sommario

1	Introduzione	4
2	Progettazione	4
2.1	DIVIDER.....	4
2.1.1	Algoritmo	5
2.1.2	ASM chart	6
2.1.3	Datapath	9
2.1.4	Testbench	10
2.2	MCD	11
2.2.1	Algoritmo	12
2.2.2	ASM chart	13
2.2.3	Datapath	16
2.2.4	Testbench	17
3	Simulazione pre-sintesi.....	18
3.1	DIVIDER.....	18
3.2	MCD	19
4	Sintesi	20
5	Simulazione post-sintesi	21
5.1	DIVIDER.....	21
5.2	MCD	22
6	Simulazione post-sintesi con T_{CLK} minimo	23

1 Introduzione

In questa relazione esporremo la progettazione in VHDL (a livello RT) e simulazione di un dispositivo sincrono che calcola il massimo comun divisore tra due numeri. Per realizzare questo dispositivo progetteremo e simuleremo anche un sottodispositivo (divisore) necessario al nostro fine. In seguito andremo sintetizzare il dispositivo complessivo per una FPGA Xilinx Spartan-6.

Nota: abbiamo supposto che non sia mai richiesto il calcolo del massimo comun divisore tra due valori dei quali uno dei due o entrambi siano nulli, quindi anche il problema della divisione per zero viene meno. Di conseguenza non ci siamo preoccupati di trattare questi casi particolari nell'analisi degli algoritmi.

2 Progettazione

Per la progettazione un dispositivo sintetizzabile su un circuito target abbiamo adottato il seguente flusso di lavoro:

- 1) Progettazione dell'algoritmo ad alto livello e simulazione del suo funzionamento in C
- 2) Traduzione dell'algoritmo in ASM chart
- 3) Deduzione della Control Unit
- 4) Deduzione del Datapath
- 5) Stesura del Toplevel
- 6) Stesura del Testbench

2.1 DIVIDER

Dalle specifiche del progetto sappiamo che è necessario creare un divisore che accetti in ingresso due numeri A e B (di 32 bit) e restituisca in uscita il quoziente e il resto (di 32 bit).

L'interfaccia del divisore sarà la seguente:

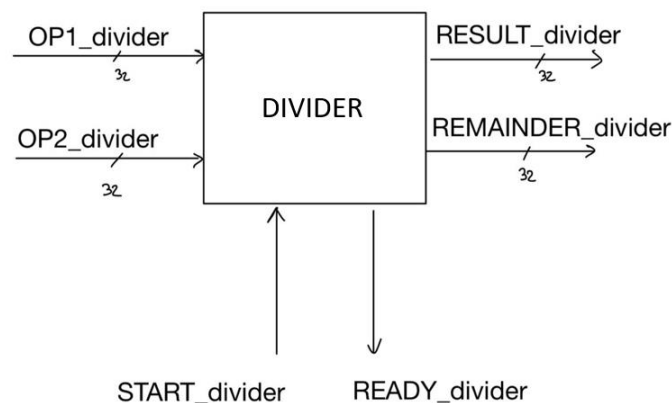


Figura 1

Esso è caratterizzato da:

- un ingresso dati OP1_divider da 32 bit che corrisponde al dividendo
- un ingresso dati OP2_divider da 32 bit che corrisponde al divisore
- un ingresso START_divider con cui si segnala al dispositivo di iniziare la computazione sui dati che ha ricevuto
- un'uscita READY_divider con cui il dispositivo segnala che ha terminato la computazione e che le uscite sono valide
- un'uscita dati RESULT_divider da 32 bit che corrisponde al quoziente della divisione
- un'uscita dati REMAINDER_divider da 32 bit che corrisponde al resto della divisione

Abbiamo scelto di definire le dimensioni dei segnali come dei generic che andremo poi ad assegnare.

2.1.1 Algoritmo

Abbiamo progettato l'algoritmo della divisione tra due numeri binari A, B nel seguente modo:

1. inizializzo due variabili a zero: CNT=0, RES=0
2. finché $A > B$ moltiplico B per 2 e tengo traccia di quante volte ho fatto questa divisione incrementando CNT ad ogni passo
3. moltiplico RES per 2
 - a. se $A \neq 0$ & $A \geq B$, allora $A = A - B$ e incremento RES
 - b. divido B per 2
 - c. decremento CNT
4. Ripeto il punto 3. finché $CNT \geq 0$
5. A questo punto in RES c'è il quoziente e in A c'è il valore del resto

Ricordiamo che:

- lo shift a destra ($>> 1$) di una posizione di un numero binario equivale a dividere tale numero per 2
- lo shift verso sinistra ($<< 1$) di una posizione di un numero binario equivale a moltiplicare tale numero per 2

2.1.2 ASM chart

Osservando l'algoritmo abbiamo eseguito una prima traduzione "immediata" in ASM chart:

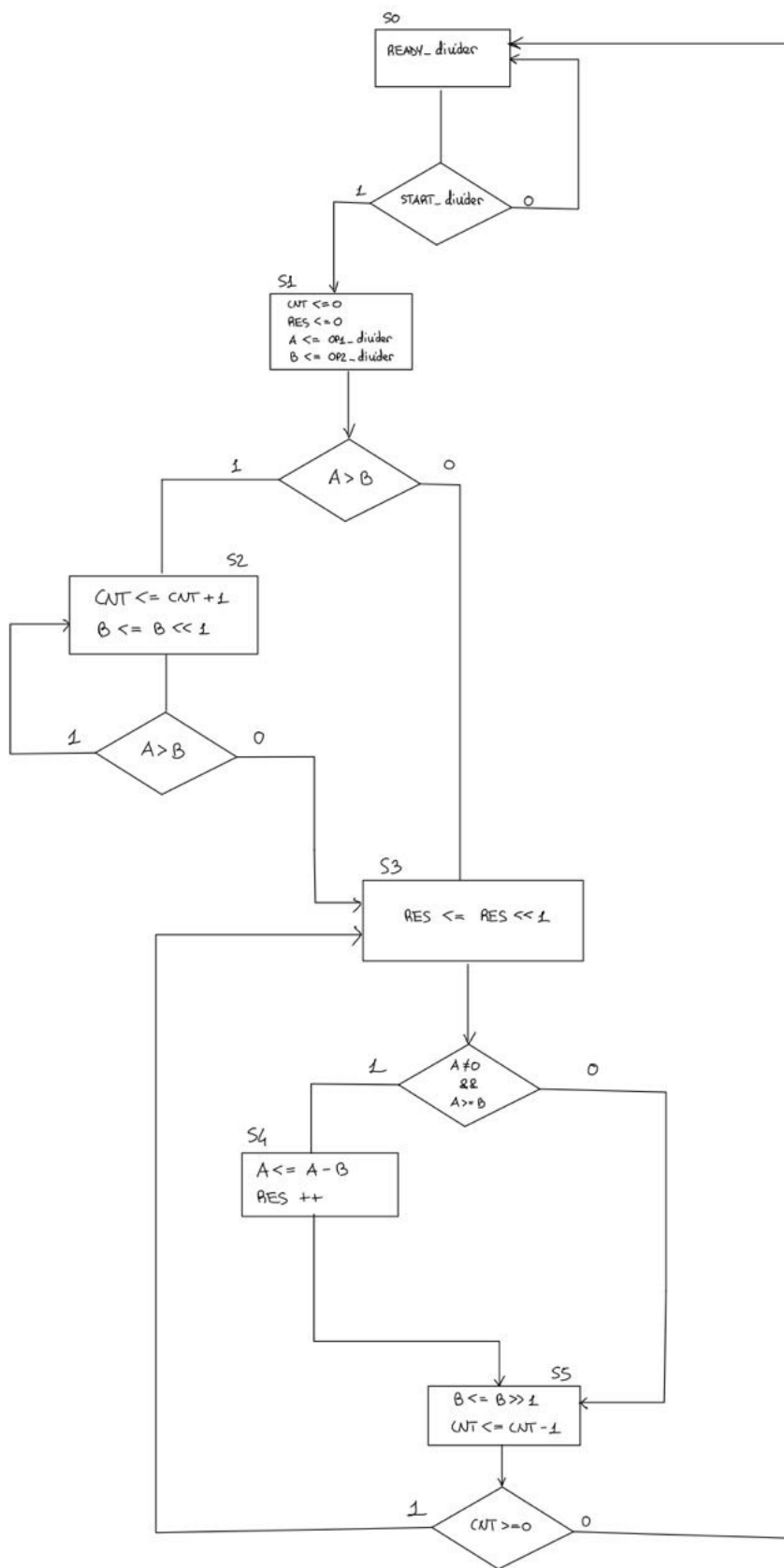


Figura 2

Modifiche:

Innanzitutto, sappiamo che quando in uno stato aggiorniamo l'ingresso di un registro, la sua uscita assumerà il nuovo valore solamente al ciclo di clock successivo. Quindi nel caso dell'ASM chart sopra dobbiamo inserire degli stati di attesa subito dopo a:

- S1
- S2
- S5

In modo tale da consentire ai registri di assumere i nuovi valori che poi andremo a testare.

Quindi l'ASM chart finale del DIVIDER è descritto nella seguente figura (*Figura 3*).

Dall'ASM chart abbiamo dedotto direttamente la Control Unit, scegliendo di implementarla come una FSM di Moore.

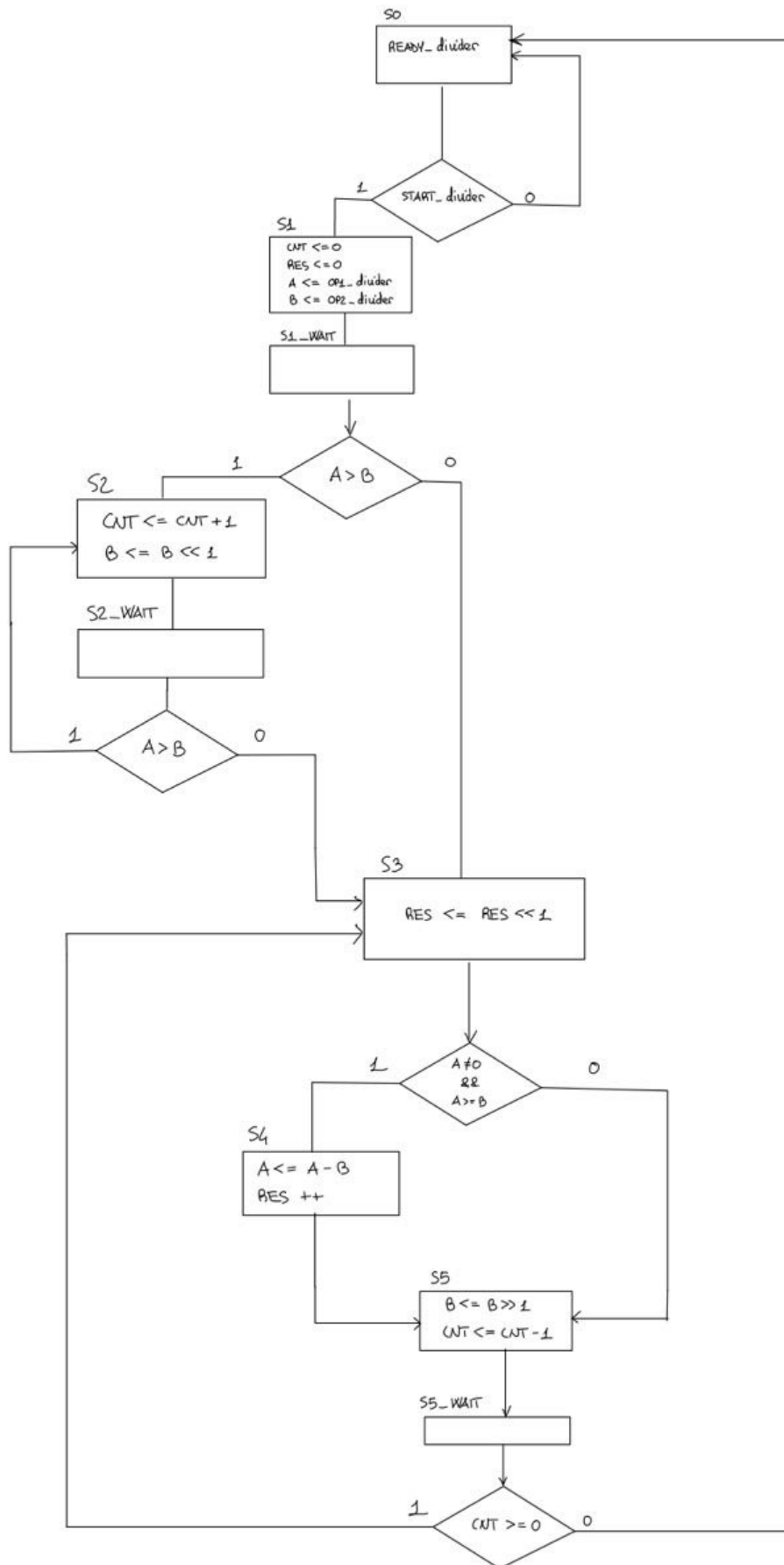


Figura 3

2.1.3 Datapath

Analizzando l'ASM chart abbiamo direttamente disegnato la struttura del Datapath. Abbiamo supposto che le risorse disponibili comprendessero: un sommatore, un sottrattore, tre comparatori di uguaglianza, due comparatori di maggioranza.

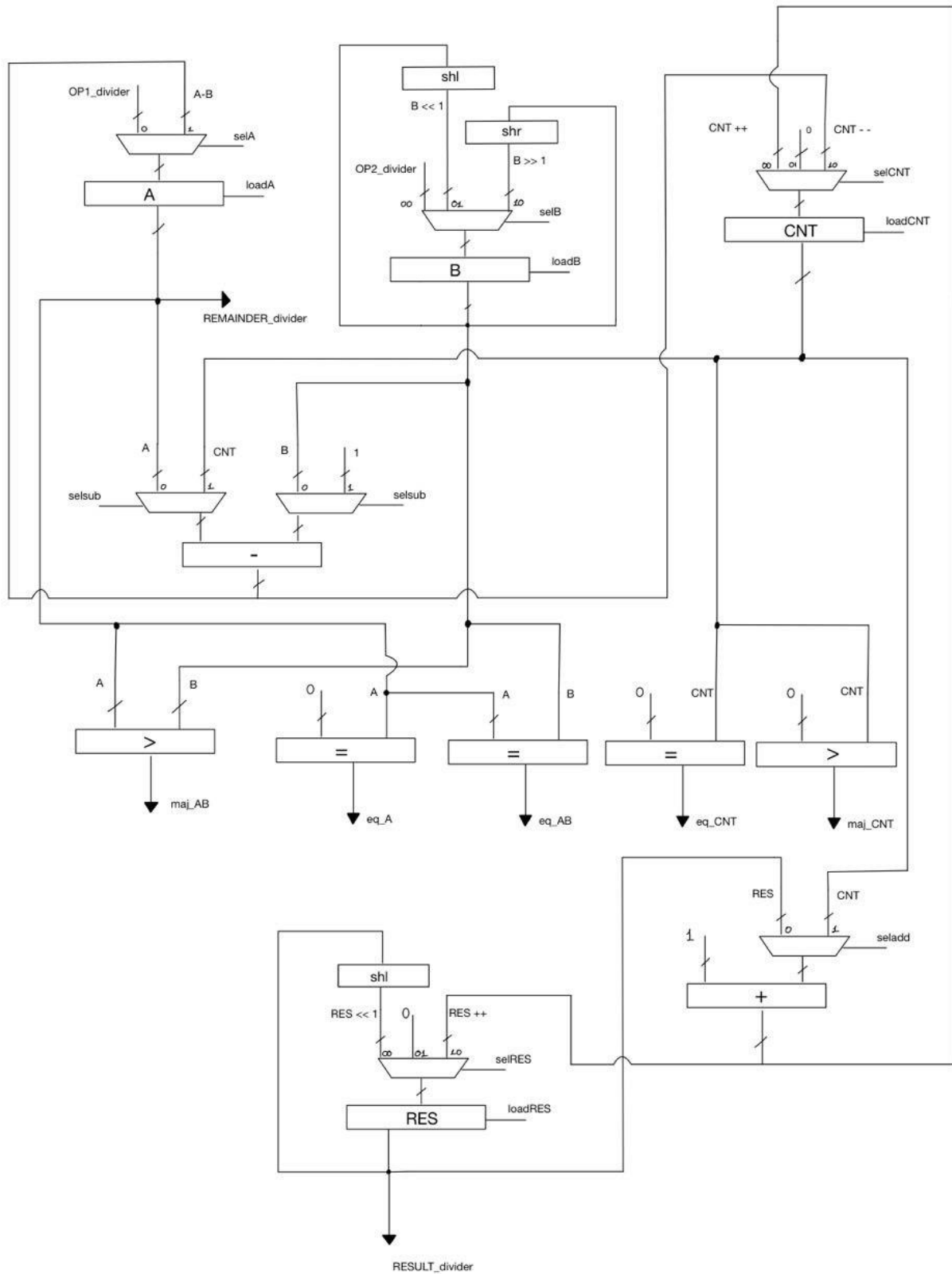


Figura 4

2.1.4 Testbench

Una volta che abbiamo scritto il codice vhdI per la Control Unit, Datapath e Toplevel abbiamo creato il testbench per il DIVIDER.

Il testbench legge i dati da un file di testo, li sottopone al DIVIDER e controlla automaticamente la correttezza dei risultati. Il file di testo ha la seguente struttura:

```
dividendo1 divisore1 risultato_atteso1 resto_atteso1
dividendo2 divisore2 risultato_atteso2 resto_atteso2
```

.....

Se vi fossero errori nel codice, il seguente messaggio apparirebbe nella console di ModelSim:

```
# -----> Error in DIVIDER(13,4)=3 rem 1, expecting 3 rem 10. Data counter is 3 <-----
# ** Error: Assertion violation.
#   Time: 1600 ns Iteration: 1 Instance: /tes
# ** Failure: ERROR
```

Figura 5

Segnaliamo in quale test è avvenuto l'errore (Data counter), i risultati ottenuti e quelli invece attesi (nella figura riportato è un esempio "fittizio", i dati del file erano sbagliati).

Se invece tutti i risultati sono corretti, abbiamo scelto questo messaggio di conferma:

```
# End simulation - Cycle counter is 246
# ** Error: Assertion violation.
#   Time: 9840 ns Iteration: 2 Instance: /tes
# ** Failure: ALL OK, END OF SIMULATION
```

Figura 6

Nella directory "DIVIDER" (che si trova all'interno della directory MCD) sono presenti tutti i file a cui ci siamo riferiti finora:

- Control Unit: CTRLUNIT_div.vhdI
- Datapath: DATAPATH_div.vhdI
- Toplevel: DIVIDER.vhdI
- Testbench: TES.vhdI
- File dati: data_div.txt

2.2 MCD

Abbiamo progettato l'MCD in modo gerarchico: il suo Datapath fa uso del sottodispositivo DIVIDER che viene istanziato come componente esterno.

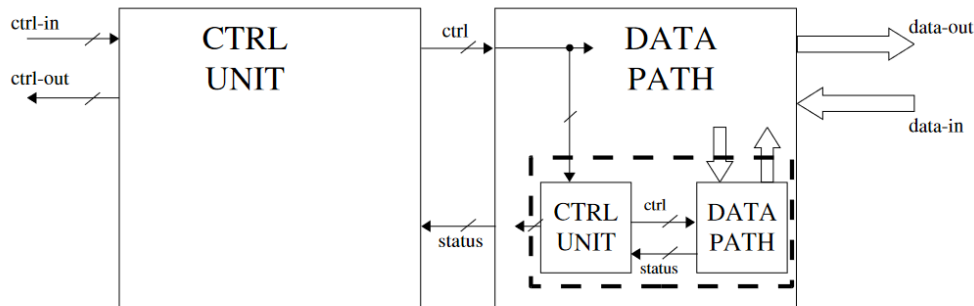


Figura 7

Quindi l'interfaccia del MCD sarà la seguente:

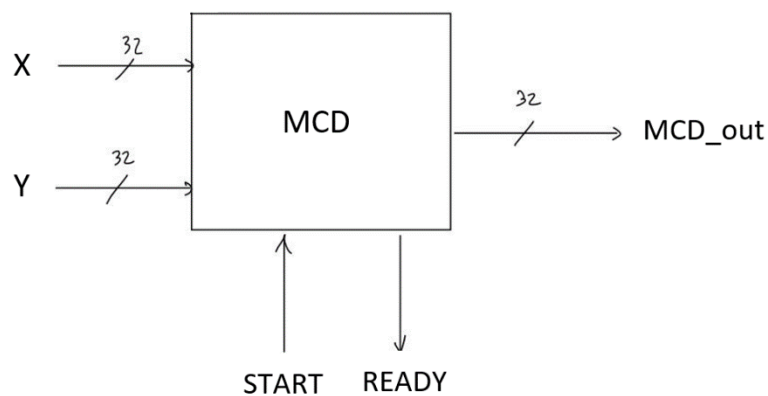


Figura 8

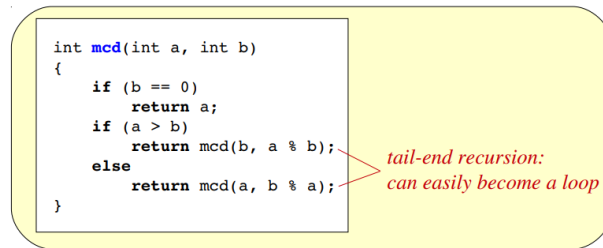
In particolare il dispositivo avrà:

- un ingresso dati X da 32 bit
- un ingresso dati Y da 32 bit
- un ingresso START con cui si segnala al dispositivo di iniziare la computazione sui dati che ha ricevuto
- un'uscita READY con cui il dispositivo segnala che ha terminato la computazione e che l'uscita è valida
- un'uscita dati MCD_out da 32 bit con il massimo comune divisore tra X e Y

Anche in questo caso abbiamo definito le dimensioni dei segnali come dei generic, ai quali abbiamo infine assegnato il valore desiderato nel testbench.

2.2.1 Algoritmo

Siamo partiti dall'algoritmo di Euclide in forma ricorsiva di seguito riportato.



```
int mcd(int a, int b)
{
    if (b == 0)
        return a;
    if (a > b)
        return mcd(b, a % b);
    else
        return mcd(a, b % a);
}
```

*tail-end recursion:
can easily become a loop*

Figura 9

Ne abbiamo poi dedotto il corrispettivo in forma iterativa:

1. se $A < B$ allora scambio i loro valori, in modo che il primo numero (A) sia sempre il maggiore
2. divido A per B e assegno il valore del resto alla variabile R
 - a. se $R=0$ allora ho terminato, in B c'è il valore finale dell'mcd
 - b. se $R>0$ allora: scambio A e B, scambio B ed R, ripeto dal punto 2.

2.2.2 ASM chart

Abbiamo tradotto l'algoritmo di Euclide iterativo in ASM:

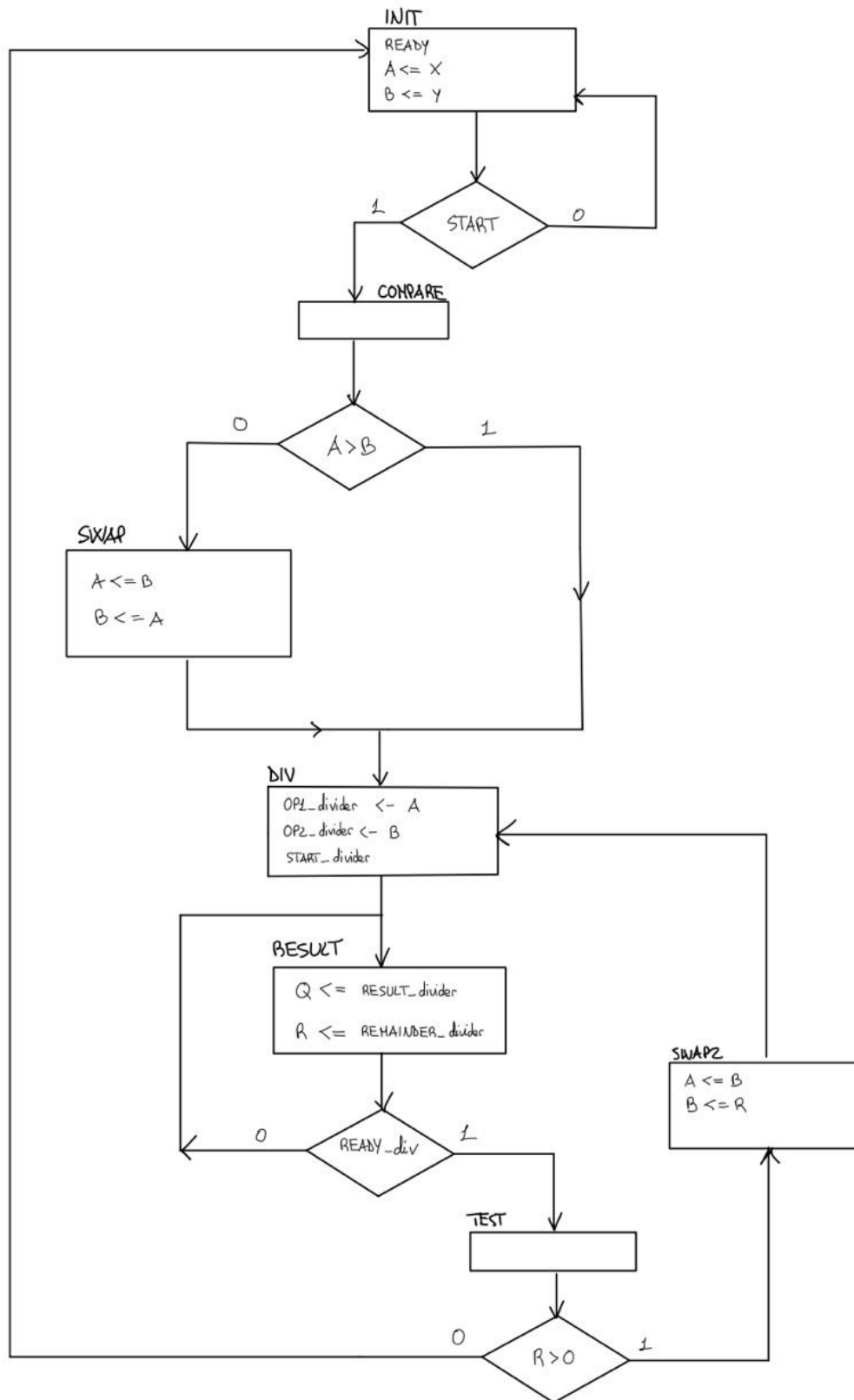


Figura 10

Modifiche:

Abbiamo pensato di apportare un miglioramento che consiste nell'eliminare lo stato SWAP2 e inserire le assegnazioni $A \leftarrow B$ e $B \leftarrow R$ direttamente nello stato TEST. In questo modo tuttavia bisogna fare attenzione che se nella versione iniziale dell'ASM chart, nel Datapath dovevamo collegare l'uscita MCD_out al registro B, ora dobbiamo collegare l'uscita MCD_out al registro A.

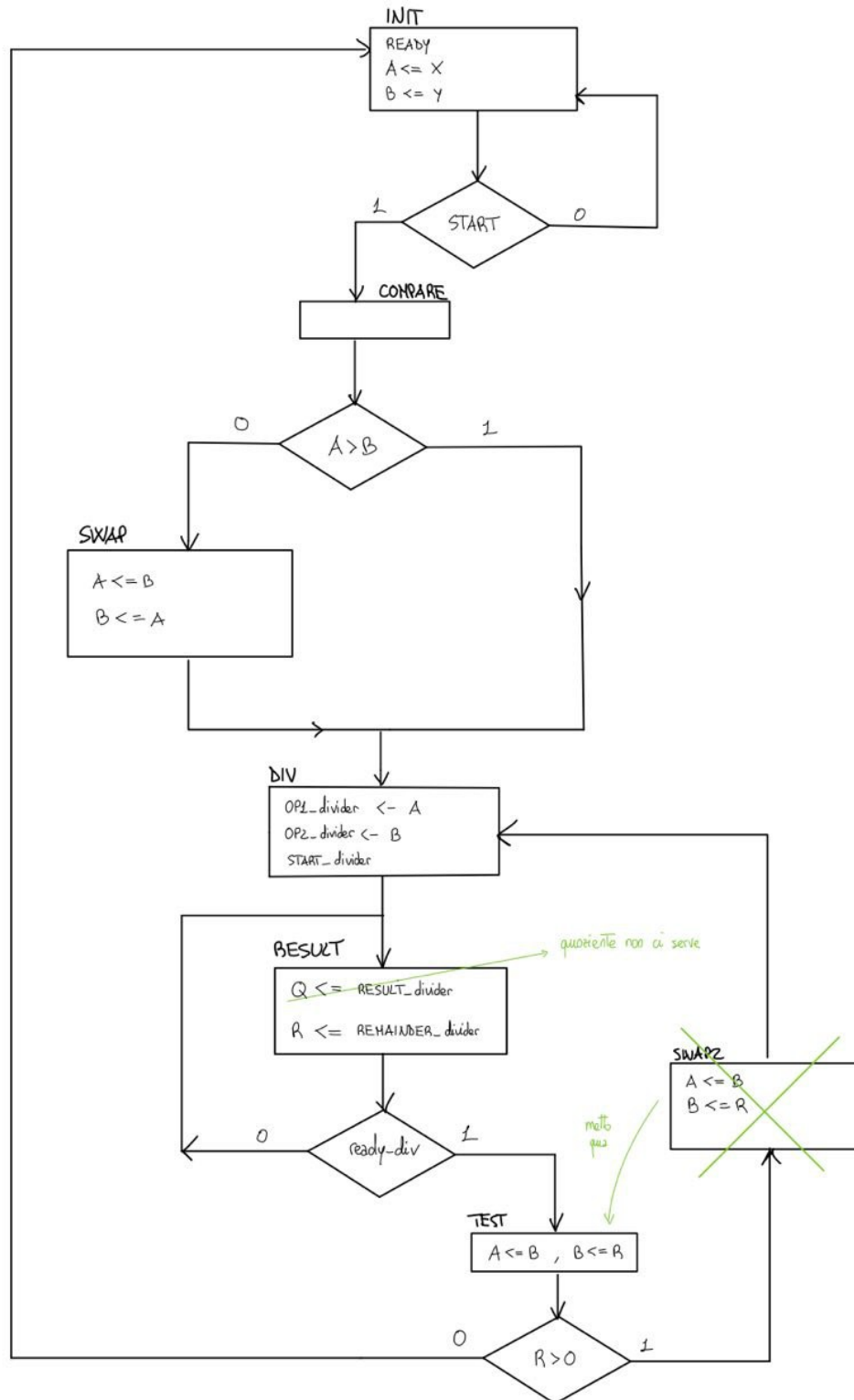


Figura 11

La versione finale dell'ASM chart è la seguente, dalla quale poi abbiamo dedotto la Control Unit descrivendola come FSM di Moore.

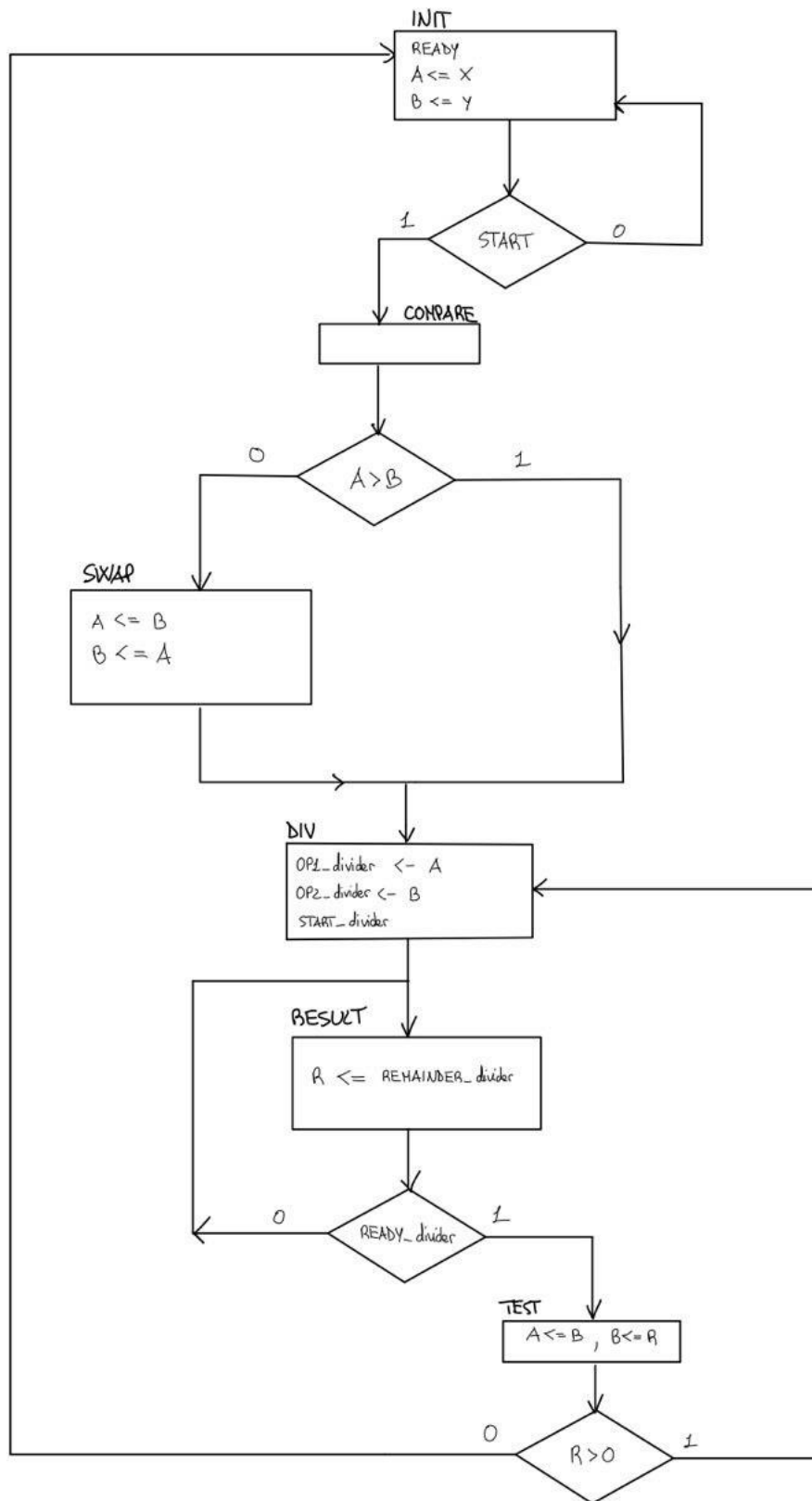


Figura 12

2.2.3 Datapath

Dall'ASM chart abbiamo direttamente disegnato la struttura del Datapath. Abbiamo supposto che le risorse disponibili per il progetto comprendessero altri due comparatori di maggioranza, oltre a quelli già considerati per il DIVIDER.

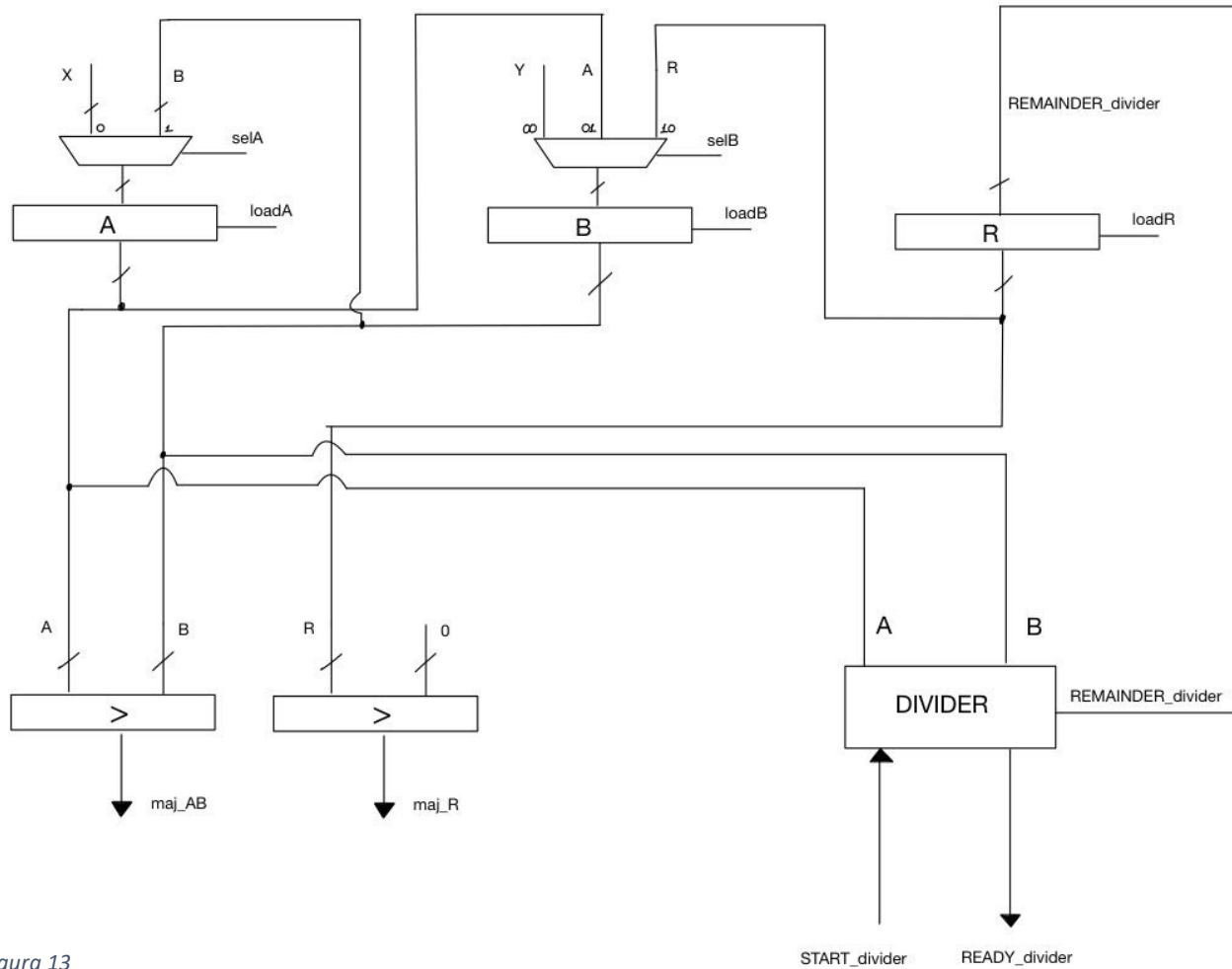


Figura 13

2.2.4 Testbench

Una volta che abbiamo scritto il codice VHDL per la Control Unit, Datapath e Toplevel dell'MCD abbiamo creato il suo testbench.

Analogamente al caso del DIVIDER, anche qui il testbench legge i dati da un file di testo, li sottopone all'MCD e controlla automaticamente la correttezza del risultato. Il file di testo ha la seguente struttura:

X1 Y1 risultato_atteso1

X2 Y2 risultato_atteso2

.....

Eventuali errori nei test e conferme di simulazioni corrette vengono riportati analogamente a quanto accadeva nel testbench del DIVIDER.

Nella directory "MCD" sono presenti tutti i file a cui ci siamo riferiti finora:

- Control Unit: CTRLUNIT.vhdl
- Datapath: DATAPATH.vhdl
- Toplevel: MCD.vhdl
- Testbench: TESTBENCH.vhdl
- File dati: data.txt
- Directory divider: DIVIDER

3 Simulazione pre-sintesi

Dopo aver verificato il corretto funzionamento dei testbench, siamo passati alla simulazione pre-sintesi dei due dispositivi su ModelSim. Abbiamo continuato a simulare e a correggere alcuni piccoli errori fino ad ottenere due dispositivi correttamente funzionanti. Il periodo di clock richiesto è di 10 ns.

Name	Status	Type
TESTBENCH.vhdl	✓	VHDL
MCD.vhdl	✓	VHDL
DIVIDER.vhdl	✓	VHDL
CTRLUNIT_div.vhdl	✓	VHDL
CTRLUNIT.vhdl	✓	VHDL
TES.vhdl	✓	VHDL
DATAPATH_div.vhdl	✓	VHDL
DATAPATH.vhdl	✓	VHDL

Figura 14

3.1 DIVIDER

Riportiamo qualche esempio del corretto comportamento del DIVIDER. Vediamo ad esempio che intorno ai 225 ns vengono caricati gli operandi 13 e 4 ed il segnale START_divider diventa alto, facendo partire il DIVIDER. Ad un certo punto il segnale READY_divider diventa alto e come risultato e resto otteniamo 3 ed 1. Analogamente accade con la divisione tra 144 e 12 che parte immediatamente dopo e così via per altri dati di test.

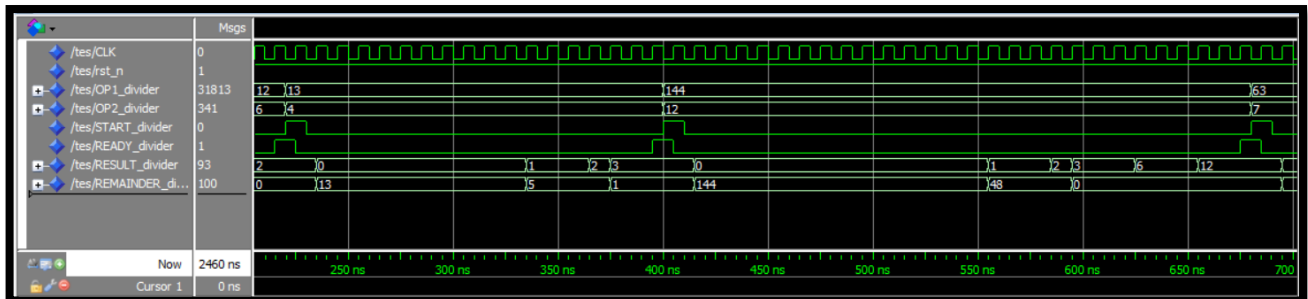


Figura 15

Tutti i test hanno buon fine ed infatti otteniamo il seguente messaggio a fine simulazione:

```
# End simulation - Cycle counter is 246
# ** Error: Assertion violation.
#   Time: 2460 ns Iteration: 2 Instance: /tes
# ** Failure: ALL OK, END OF SIMULATION
#   Time: 2460 ns Iteration: 2 Process: /tes/done_process File: C:/Users/rikma/Desktop/MCD/DIVIDER/TES.vhdl
# Break in Process done_process at C:/Users/rikma/Desktop/MCD/DIVIDER/TES.vhdl line 165
VSIM 11>
```

Figura 16

3.2 MCD

Riportiamo qualche esempio del corretto comportamento dell'MCD. Vediamo ad esempio che intorno ai 6000 ns vengono caricati gli operandi 194 e 29934 ed il segnale START diventa alto, facendo partire l'MCD. Ad un certo punto il segnale READY diventa alto e come risultato otteniamo 2. Analogamente accade per altri dati di test.

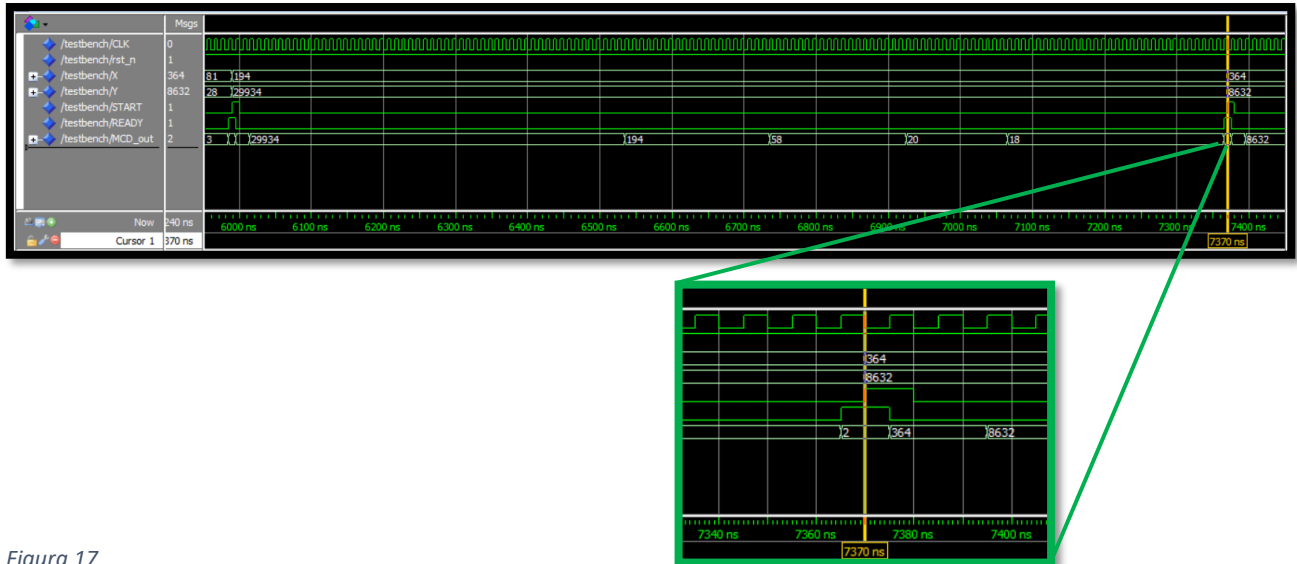


Figura 17

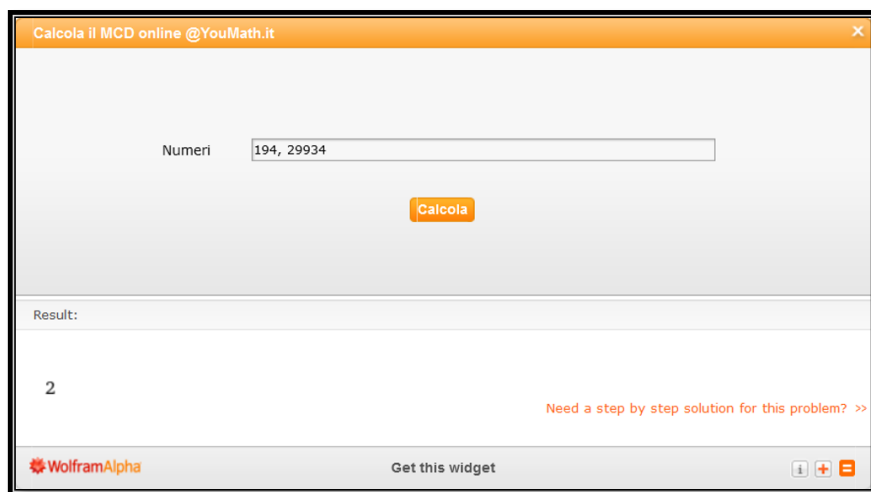


Figura 18

Anche in questo caso tutto funziona correttamente:

```
# End simulation - Cycle counter is 824
# ** Error: Assertion violation.
# Time: 8240 ns Iteration: 2 Instance: /testbench
# ** Failure: ALL OK, END OF SIMULATION
# Time: 8240 ns Iteration: 2 Process: /testbench/done_process File: C:/Users/rikma/Desktop/MCD/TESTBENCH.vhdl
# Break in Process done_process at C:/Users/rikma/Desktop/MCD/TESTBENCH.vhdl line 159
VSIM 21>
```

Figura 19

4 Sintesi

Abbiamo sintetizzato il dispositivo completo grazie al tool ISE. Durante il processo, tutto è andato a buon fine, ottenendo soltanto la notifica di alcuni warnings poco importanti.

```
=====
HDL Elaboration
=====
Elaborating entity <MCD> (architecture <struct>) with generics from library <work>.
Elaborating entity <CTRLUNIT> (architecture <behav>) from library <work>.
INFO:HDLCompiler:679 - "/home/yes/Desktop/MCD/CTRLUNIT.vhdl" Line 76. Case statement is complete. others clause is never selected
Elaborating entity <DATAPATH> (architecture <ss>) with generics from library <work>.
WARNING:HDLCompiler:871 - "/home/yes/Desktop/MCD/DATAPATH.vhdl" Line 61: Using initial value "00000000000000000000000000000000" for allzeros since it is never assigned
Elaborating entity <DIVIDER> (architecture <struct>) with generics from library <work>.
Elaborating entity <CTRLUNIT_div> (architecture <behav>) from library <work>.
INFO:HDLCompiler:679 - "/home/yes/Desktop/MCD/DIVIDER/CTRLUNIT_div.vhdl" Line 91. Case statement is complete. others clause is never selected
Elaborating entity <DATAPATH_div> (architecture <ss>) with generics from library <work>.
WARNING:HDLCompiler:871 - "/home/yes/Desktop/MCD/DIVIDER/DATAPATH_div.vhdl" Line 54: Using initial value "0000000000000000000000000000000000000000000000000000000000000000" for allzeros since it is never assigned
=====
```

Figura 20

Il report completo della sintesi si trova in uno dei file allegati.

L'unico constraint richiesto è stato il periodo di clock, pari a 10 ns. Nel "Timing Summary" della sintesi viene indicato che il minimo periodo a cui si può arrivare è di 4.130 ns (Figura 21), quindi tale constraint viene rispettato.

```
Timing Summary:
-----
Speed Grade: -3

Minimum period: 4.130ns (Maximum Frequency: 242.116MHz)
```

Figura 21

Alla fine della simulazione otteniamo il file binario con il quale è possibile configurare l'FPGA.

In seguito abbiamo generato il file vhd1 a livello gate del dispositivo e il relativo file dei ritardi, che useremo per la simulazione post-sintesi.

Nella directory "Sintesi_MCD" sono presenti i file a cui ci siamo riferiti finora, insieme ad altri:

- Vhdl gate-level: MCD_timesim.vhd
- Delays info: MCD_timesim.sdf
- Synthesis report: MCD.syr
- FPGA bitstream file: MCD.bit
- Place and Route report: MCD.par
- Timing analysis report: MCD.twr

Nota: abbiamo eseguito la sintesi anche per il DIVIDER soltanto, in quanto il suo vhd1 gate-level è stato utile. Abbiamo creato una cartella "Sintesi_DIVIDER" analoga a quella appena descritta.

5 Simulazione post-sintesi

Infine, abbiamo effettuato la simulazione post-sintesi dell'intero dispositivo MCD descritto a livello gate, utilizzando anche le informazioni sui ritardi contenute nel file sdf (specificando, in ModelSim, che tali ritardi fanno riferimento soltanto al dispositivo under test istanziato nel testbench). Per poter effettuare correttamente questa simulazione abbiamo modificato leggermente il testbench:

- Abbiamo impostato un tempo di reset pari a 200 ns, necessario per il corretto funzionamento dei componenti di libreria
- Abbiamo ritardato di un ciclo il test del segnale READY dopo aver effettuato una lettura da file. Questo è stato necessario in quanto, simulando il dispositivo con i ritardi descritti nel file sdf, dopo la lettura di una nuova coppia di dati il segnale READY diventa basso con un certo ritardo (circa $\frac{3}{4}$ di ciclo) rispetto alla simulazione ideale, senza ritardi. Questa necessità comunque non incide sulla performance del dispositivo in quanto il vero risultato non sarà mai pronto dopo soltanto un ciclo di clock.

Prima di avviare la simulazione abbiamo compilato le librerie di Xilinx che vengono dichiarate nel file gate-level.

Abbiamo effettuato la sintesi e quindi simulazione gate-level anche del DIVIDER, adottando gli stessi accorgimenti descritti sopra.

5.1 DIVIDER

Possiamo notare che ora sono presenti dei ritardi rispetto alla simulazione pre-sintesi. I risultati sono sempre corretti.

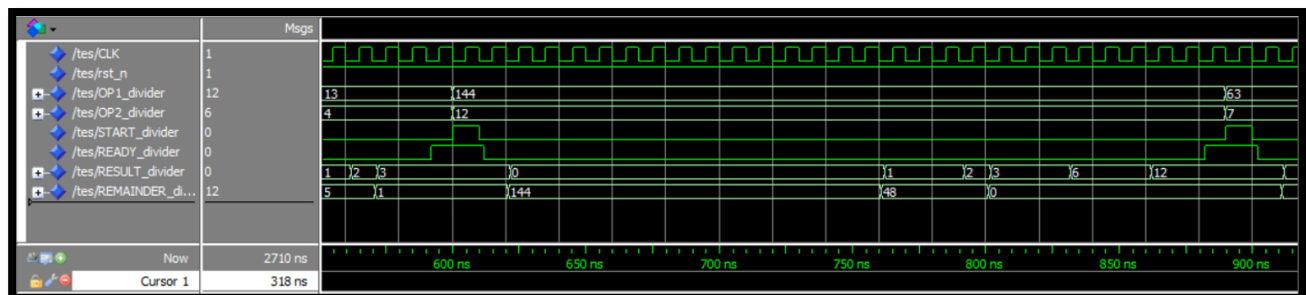


Figura 22

```
# End simulation - Cycle counter is 271
# ** Error: Assertion violation.
#   Time: 2710 ns  Iteration: 2  Instance: /tes
# ** Failure: ALL OK, END OF SIMULATION
#   Time: 2710 ns  Iteration: 2  Process: /tes/done_process File: C:/Users/rikma/Desktop/DIVIDER_gatelevel/TES.vhdl
# Break in Process done_process at C:/Users/rikma/Desktop/DIVIDER_gatelevel/TES.vhdl line 169

VSIM 23>
```

Figura 23

5.2 MCD

Anche per il gate-level dell'MCD otteniamo l'esito positivo dei test.

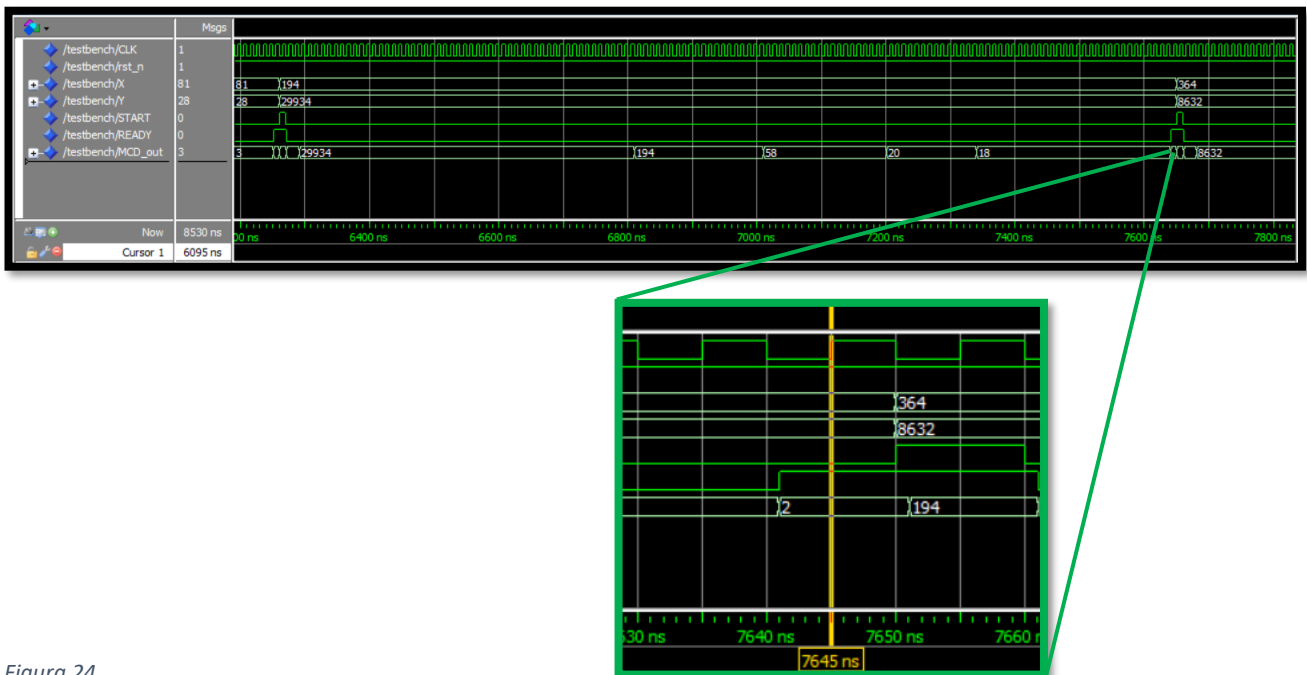


Figura 24

```
# End simulation - Cycle counter is 853
# ** Error: Assertion violation.
#   Time: 8530 ns Iteration: 2 Instance: /testbench
# ** Failure: ALL OK, END OF SIMULATION
#   Time: 8530 ns Iteration: 2 Process: /testbench/done_process File: C:/Users/rikma/Desktop/MCD_gatelevel/TESTBENCH.vhdl
# Break in Process done_process at C:/Users/rikma/Desktop/MCD_gatelevel/TESTBENCH.vhdl line 162
VSIM 10>
```

Figura 25

Abbiamo la conferma che tutto il codice scritto è corretto e il dispositivo è funzionante.

Tutti i file per le simulazioni gate-level, testbench adattati compresi, si trovano nelle cartelle "MCD_gatelevel" e "DIVIDER_gatelevel" che contengono:

- Vhdl gate-level: MCD_timesim.vhd – DIVIDER_timesim.vhd
- Delays info: MCD_timesim.sdf – DIVIDER_timesim.sdf
- Testbench: TESTBENCH.vhdl – TES.vhdl
- File dati: data.txt – data_div.txt

6 Simulazione post-sintesi con T_{CLK} minimo

Abbiamo effettuato un'ultima simulazione dell'MCD gate-level adottando un periodo di clock più prossimo possibile a quello minimo (che ricordiamo essere pari a 4.130 ns, ed in particolare è dettato dal DIVIDER), cioè di 6 ns.

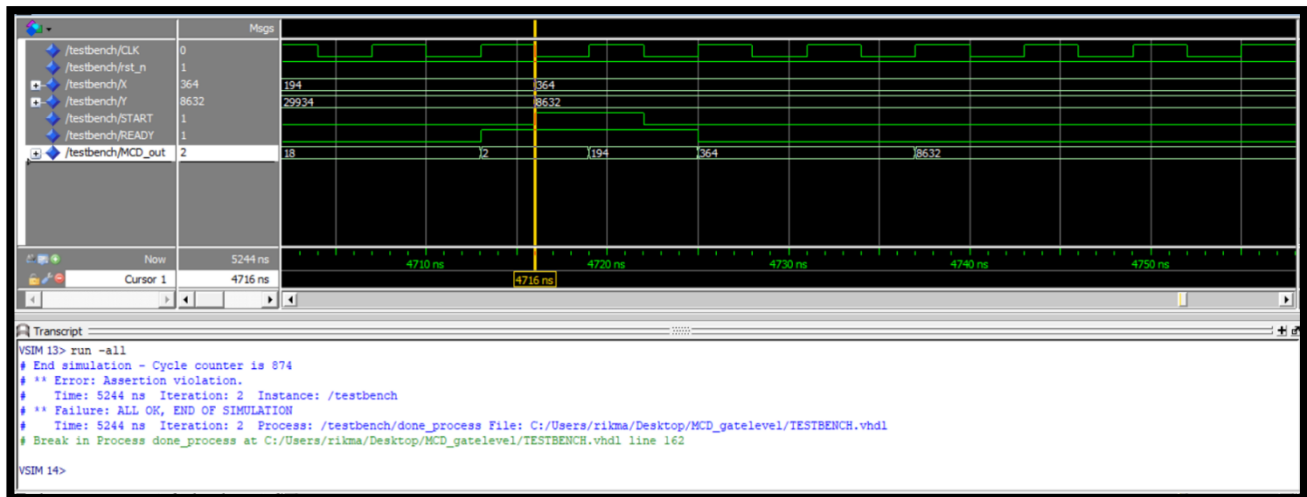


Figura 26

Anche con questo periodo di clock il dispositivo funziona. Ripetendo la procedura con un periodo di 4 ns, la simulazione fallisce, in quanto il DIVIDER non riesce a funzionare correttamente. Infatti simulando il DIVIDER gate-level con un periodo di clock di 4 ns viene riportato il malfunzionamento (Figura 27).

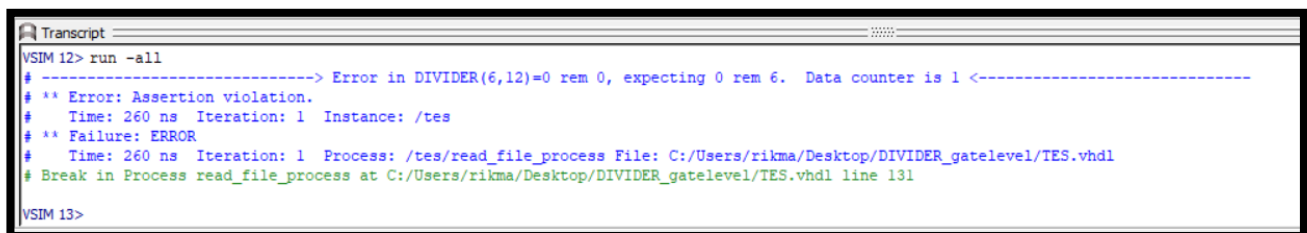


Figura 27