Politecnico di Milano

SCUOLA DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
Corso di Laurea Triennale in Ingegneria Informatica

Progetto Di Reti Logiche

Anno Accademico: 2022/2023

Professore
Prof. Palermo Gianluca

Studente
Codice Persona: **10742803**
Matricola: **963296**

INDICE

Introduzione

Obiettivo

Si tratta della progettazione di un sistema, implementato in Hardware, descritto nel linguaggio di descrizione del hardware, VHDL, che in base alle informazioni lette sui canali d'ingresso al componente, pone sulle opportune uscite, dati letti dalla memoria.
In dettaglio, le indicazioni circa il canale da utilizzare e l'indirizzo di memoria a cui accedere vengono forniti mediante uno degli ingressi primari.

Interfaccia del componente

Il componente presenta un'interfaccia così definita:

```
entity project_reti_logiche is
    port (
        i_clk           : in std_logic;
        i_rst           : in std_logic;
        i_start         : in std_logic;
        i_w             : in std_logic;
        o_z0            : out std_logic_vector(7 downto 0);
        o_z1            : out std_logic_vector(7 downto 0);
        o_z2            : out std_logic_vector(7 downto 0);
        o_z3            : out std_logic_vector(7 downto 0);
        o_done          : out std_logic;
        o_mem_addr      : out std_logic_vector(15 downto 0);
        i_mem_data      : in  std_logic_vector(7 downto 0);
        o_mem_we        : out std_logic;
        o_mem_en        : out std_logic
    );
end project_reti_logiche;
```

In particolare:

- ★ i_clk è il segnale di CLOCK in ingresso generato dal Test Bench;
- ★ i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo
- ★ segnale di START;
- ★ i_start è il segnale di START generato dal Test Bench;
- ★ i_w è il segnale W precedentemente descritto e generato dal Test Bench;
- ★ o_z0, o_z1, o_z2, o_z3 sono i quattro canali di uscita;
- ★ o_done è il segnale di uscita che comunica la fine dell'elaborazione;
- ★ o_mem_addr è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- ★ i_mem_data è il segnale (vettore) che arriva dalla memoria in seguito ad una
- ★ richiesta di lettura;
- ★ o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- ★ o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0.

Dati e memoria

La memoria che è stata usata per il progetto presenta la seguente caratteristica principale:

- Si tratta di una memoria composta da 16 bit (2 byte) di indirizzo;
- E contiene un dato composto da 8 bit (1 byte).

Funzionamento

Il modulo inizia l'elaborazione quando il segnale *i_start* viene posto a 1. Quest'ultimo è garantito rimanere alto finché il segnale *o_done* non viene posto a 1, che oltre ad indicare la terminazione della computazione, indica anche le uscite hanno commutato valori appropriati.
In particolare, mentre l'ingresso *i_start* è alto(= 1), possiamo effettuare la lettura sul canale *i_w*.
Questa operazione viene fatta per un minimo di 2 e per un massimo di 18 cicli di clock, leggendo di conseguenza, 2 bit di intestazione, che identificano il canale d'uscita, e 16 bit che individuano l'indirizzo, composto da questi bit, circa la locuzione in memoria a cui prelevare il dato da porre in uscita al canale adeguato.
Inoltre, il modulo è stato progettato considerando che alla primissima elaborazione, ovvero al primo segnale di *i_start* = 1 (e prima di
richiedere il corretto funzionamento del modulo) verrà sempre dato il segnale di i_rst = 1.
Una seconda o successiva elaborazione con *i_start* = 1 non dovrà invece attendere il reset del modulo.
Ogni volta che viene dato il segnale i_rst = 1, il modulo viene reinizializzato e portato allo stato iniziale.
All'istante iniziale, o meglio, al reset del sistema, le uscite *o_z0, o_z1, o_z2, o_z3* sono poste a 0000 0000, e il segnale *o_done* è 0.
La trasmissione su *i_w*, è caratterizzata da una *trasmissione seriale*[1] da 1 bit, mentre la trasmissione sui 4 canali succitati da 8 bit, gode di una *trasmissione parallela*[2].

Design del componente

Il componente presenta 2 ingressi primari da 1 bit ciascuno e che sono *i_w* e *i_start; 5* uscite primarie, composte da, le prime 4 da 8 bit ciascuno e che sono *o_zo, o_z1, o_z2, o_z3* ed una da un bit, *o_done.*
Oltre a questi ingressi, il modulo presenta il segnale di i_clk e il segnale i_rst, unici per tutto il sistema.
La sequenza di ingresso sul canale *i_w* segue una struttura particolare, abbiamo:

- ➔ i primi 2 bit di intestazione;
- ➔ N bit di indirizzo della memoria.

I primi 2 bit letti, sempre sul fronte di salita del segnale *i_clk,* identificano uno dei canali d'uscita fra le 4 combinazioni di bit possibili avendo a disposizione 2 bit.
Le possibili combinazioni sono:

---

[1] Trasmissione seriale: si tratta di una trasmissione in cui viene trasmesso 1 bit alla volta e vengono ricevuti nello stesso ordine in cui sono stati mandati.
[2] Trasmissione parallela: si tratta di una trasmissione in cui i bit che compongono l'informazione vengono trasferiti in parallelo sui canali separati del componente.

- 00, identifica il canale d'uscita o_z0;
- 01, identifica il canale d'uscita o_z1;
- 10, identifica il canale d'uscita o_z2;
- 11, identifica il canale d'uscita o_z3.

Mentre, gli N bit possono variare da 0 fino ad un massimo di 16 bit(dato che gli indirizzi di memoria sono costituiti da 16 bit).
Dato che per richiedere il dato alla memoria è necessario fornire i 16 bit che fanno riferimento all'indirizzo e non è garantito il fatto che sul canale i_w si legga sempre 2 + 16 bit, bisogna gestire diversamente i casi in cui il numero di bit letti sul canale sia <18 totali.
La soluzione ricorre all'uso del padding, ovvero, ad andare estendere con 0 sui bit più significativi, come segue nei seguenti esempi:

- N = 7 1010111 => 0000000001010111
- N = 16 1110000001010111 => 1110000001010111
- N = 0 0000000000000000 => 0000000000000000

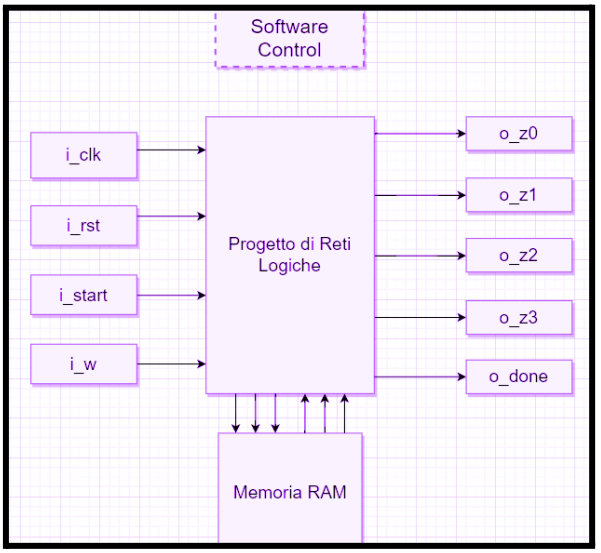

Figura 1: Design del componente.

Esempi di Funzionamento

Nella Figura 2, quando il segnale start = 1 e siamo sul rising-edge del clock, leggiamo l'andamento del segnale i_w;
In modo particolare, in questo esempio, andiamo a leggere un dato, identificato come "D", dall'indirizzo "0000000000010110" e sul canale d'uscita z1, dato che i bit di intestazione sono "01".

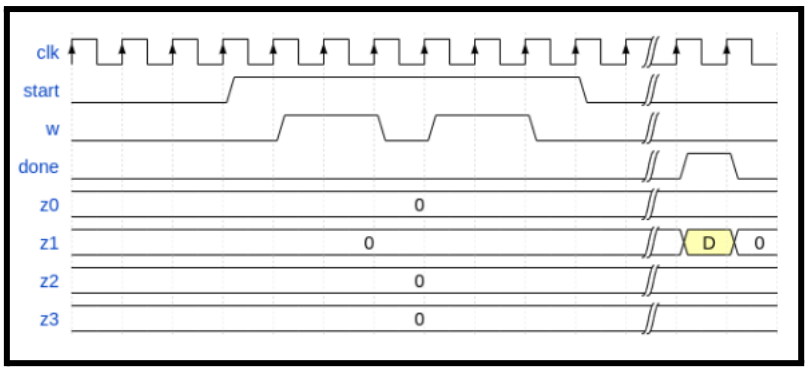

Figura 2: Esempio 1.

Nella figura 3, si può vedere un altro snapshot di funzionamento; in questo esempio,nella prima esecuzione, come si nota dalla figura, il canale prescelto è z2, con la conseguente lettura di “10” e l’indirizzo è “1”(non esteso).
Nel secondo caso invece, il canale prescelto è z1, con la conseguente lettura di “01” mentre l'indirizzo è “0000000101010010”.

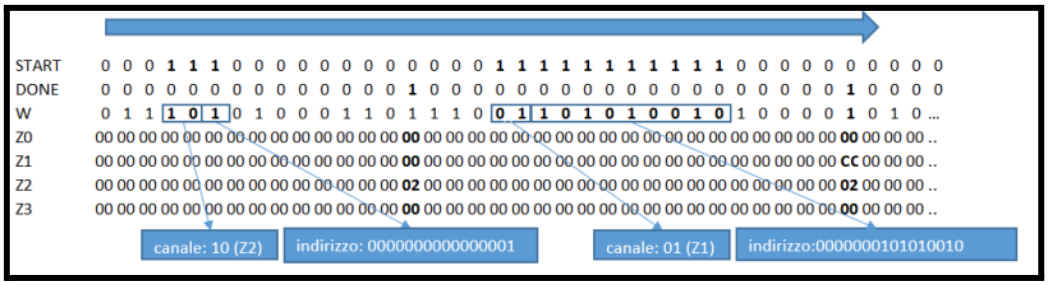

Figura 3: Esempio 2.

Architettura

L’architettura che ho deciso di implementare è composta da 2 processi differenti, di seguito elencati e descritti:

1) Processo di gestione dei segnali *i_clk* e *i_rst,* che rappresenta la parte sequenziale e si occupa della gestione dei registri;
2) Processo della Macchina a Stati finiti (FSM), si occupa, in base allo stato corrente e in base ai segnali, di proseguire allo stato successivo.

I_reset - I_clock

Questo processo ha come compito principale di gestire l’inizializzazione e la commutazione dei registri.
In particolare, quando il segnale i_rst viene posto a 1, i registri necessari riportano il valore di default che per la maggior parte corrisponde a 0 ed inoltre la macchina viene riportata nello stato di IDLE.

Per quanto riguarda i_clk, quando esso individua un fronte di salita (passa da 0 a 1), i registri vengono aggiornati con valori opportuni.

Finite state machine(FSM) - Macchina a stati finiti

Questo processo implementa tutta la logica del progetto e può essere scomposta in 4 fasi:

- ➔ Una fase “zero” che gestisce alcuni registri;
- ➔ Una prima fase che segna la Lettura e la memorizzazione delle informazioni da i_w;
- ➔ Una seconda fase che segna il “dialogo” con la memoria;
- ➔ Una terza ed ultima fase che si occupa della corretta presentazione delle informazioni sulle opportune uscite.

La FSM infine è composta da 8 stati, di seguito elencati e descritti:

Idle

Questo stato è identificato come stato iniziale della FSM; questo stato è raggiungibile quando o_done passa da 1 a 0 oppure se vi è i_rst = 1.
Inoltre, questo stato si occupa anche della lettura del primo bit su i_w.

Lettura_Canale

Questo stato è raggiungibile quando il segnale i_start viene posto a 1 e si occupa della lettura sul canale i_w.
Si rimane in questo stato in attesa che il segnale i_start venga posto a 0 o che siano stati letti 18 bit.

Checker

Questo stato è raggiungibile quando il segnale i_start viene posto a 0 e si occupa della corretta estrazione delle informazioni memorizzate nella variabile “store”.
Una volta terminata la computazione per questo stato, si procede al prossimo stato della FSM.

Set_Ram

Questo stato si occupa del assegnamento dell’indirizzo a o_mem_addr per la fase di lettura dei dati in memoria.

Read_Ram

Questo stato è classificato come stato cuscinetto per la corretta propagazione dei segnali.

Wait_Ram

Questo stato è classificato come stato cuscinetto per la corretta propagazione dei segnali.

Write_OUT

Questo stato è il penultimo stato prima di concludere l'elaborazione e si occupa della corretta scrittura del dato sul registro desiderato.

Done

Questo stato è l'ultimo stato della macchina ed indica la corretta terminazione della computazione da parte della macchina a stati.
Si occupa di portare il segnale o_done a 1 e di scrivere i valori delle varie uscite.
Inoltre si occupa di inizializzare i registri per poter permettere l'inizio di una nuova elaborazione e riporta la macchina allo stato di IDLE.

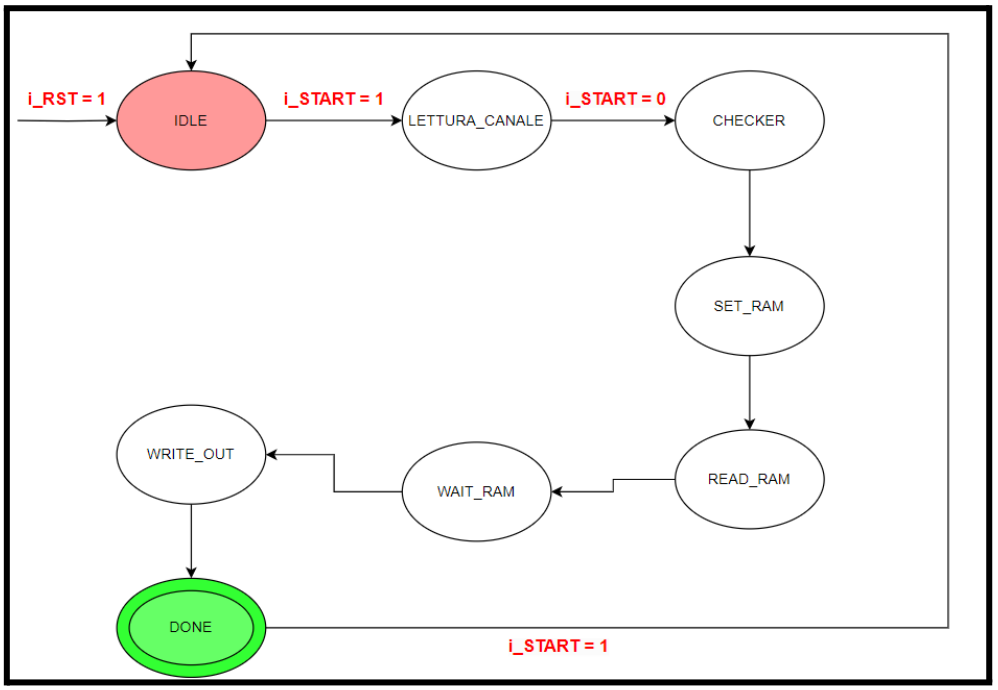

Figura 4: FSM

Scelte progettuali

Come già anticipato per avere una corretta esecuzione e per facilitare il troubleshooting ho deciso di creare 2 process differenti in modo tale da distribuire la logica di funzionamento del programma.
Per quanto riguarda la memorizzazione dei bit letti da i_w, ho deciso innanzitutto di raccogliere tutti i bit in unico std_logic_vector, senza andare a separare bit di intestazione da quelli che compongono l'indirizzo.
Successivamente, in uno stato differente, ho deciso di estrarre i bit di intestazione semplicemente accedendo con un indice costante e assegnando il valore in un altro std_logic_vector.
Per quanto riguarda invece i bit che compongono l'indirizzo ho deciso di implementare un Right Shifter che consiste nello spostamento di bit verso destra di un certo numero di posizioni.

Inoltre, ho preferito l'uso di segnali rispetto alle variabili, in quanto, avendo 2 processi differenti sarebbe stato complicato l'uso delle variabili, in quanto, sono soltanto visibili all'interno del processo in cui sono dichiarate.

Simulazioni

Reset asincrono

Si tratta di un Testbench che verifica che la presenza di un segnale di i_rst ricevuto "casualmente" non alteri la corretta esecuzione del programma.

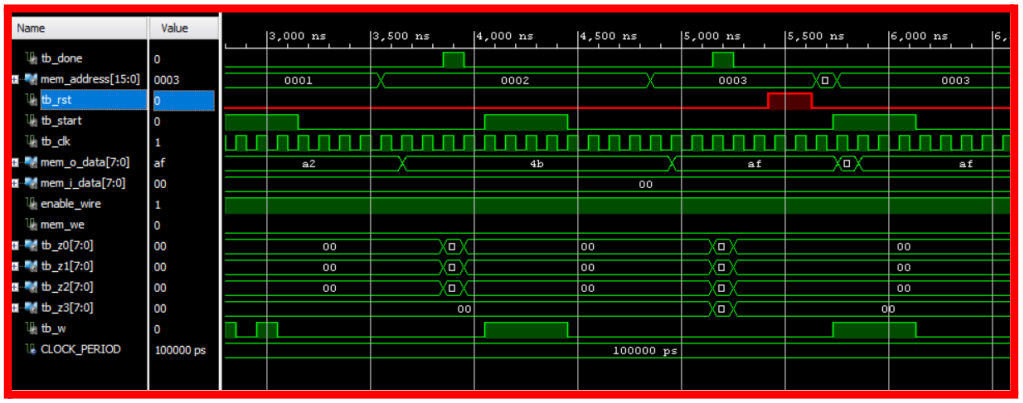

Figura 5: Testbench

Indirizzo vuoto

Si tratta di un Testbench che verifica che a fronte di 0 bit letti per la parte che compone l'indirizzo da 16 bit non blocchi la corretta esecuzione del programma.

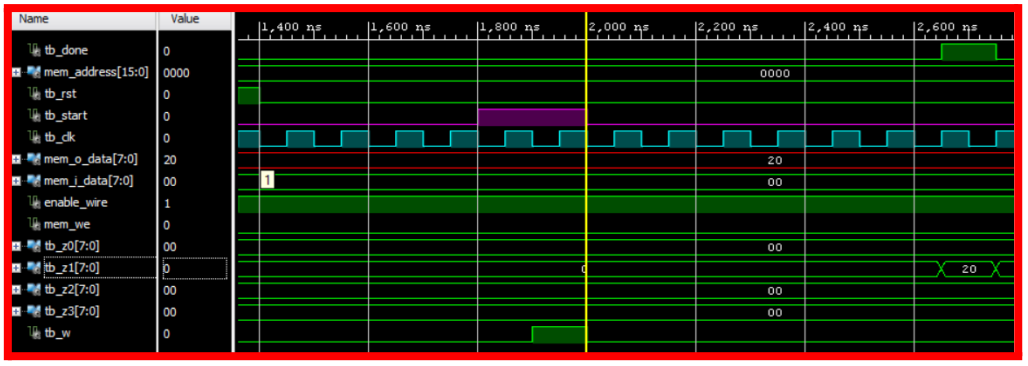

Figura 6: Testbench

Reset e Start := 1

Si tratta di un Testbench che verifica che a fronte di i_rst = 1 e i_start = 1 il programma compie la sua corretta esecuzione. Notare il fatto che state_curr ritorna nello stato di IDLE.

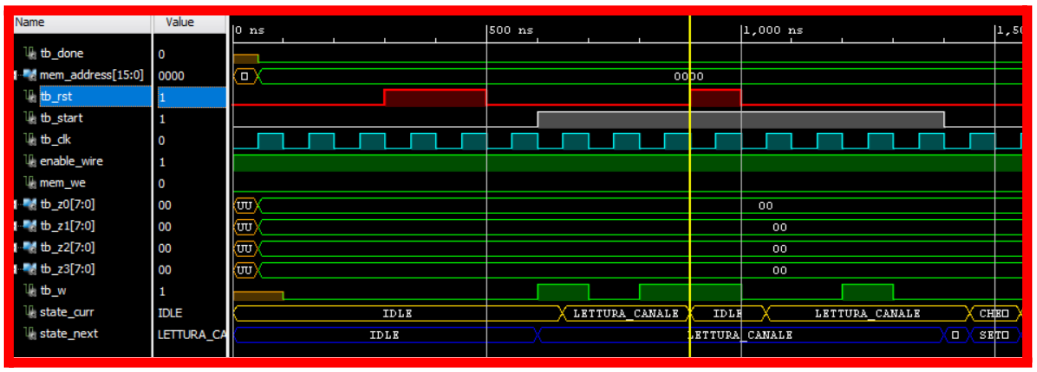

Figura 7: Testbench

Risultati

Il componente ha dimostrato di superare correttamente, con tutti i test forniti dai docenti e quelli sopra citati, la simulazione *Behavioural* e *Post-synthesis functional.*
Di seguito, le immagini riportano alcuni dati sperimentali:

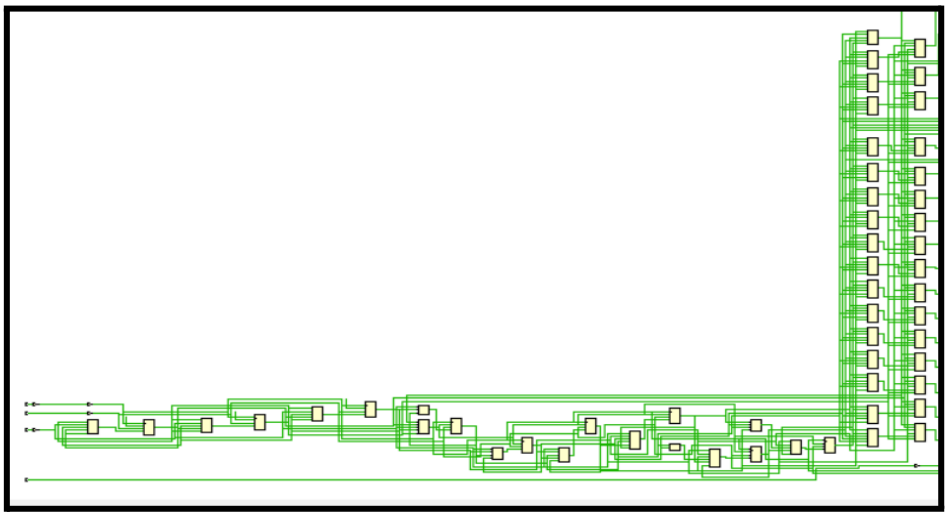

Figura 8: Elaborated Design

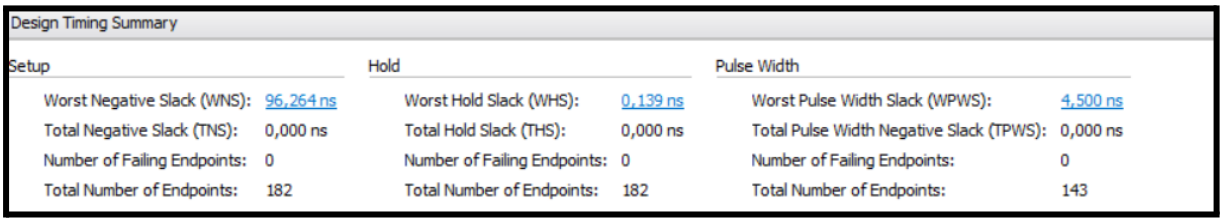

Figura 9: Design Timing Summary

Ottimizzazioni

Le ottimizzazioni principali che ho ottenuto riguardano la riduzione degli stati della FSM.
Inizialmente avevo creato 4 stati differenti, uno per ognuna delle uscite, andando di conseguenza ad aggiornare, passando per 4 stati totali, le relative uscite.
Ho cercato di ridurre questa perdita di cicli di clock, andando a creare uno stato unico, per aggiornare le opportune uscite in contemporanea al segnale di *o_done.*
Un'altra ottimizzazione riguarda l'espansione dell'indirizzo di memoria a 16 bit;
Inizialmente avevo creato uno stato in cui andavo ad aggiungere, ove necessario, i zeri mancanti per completare l'indirizzo.
Ho cercato di ridurre questa sezione di codice andando a dare un valore prefissato (16 bit a zero) alla variabile che memorizza l'indirizzo a 16 bit.
Ho ritenuto importante citare le ottimizzazioni, in quanto, hanno permesso di:

- ➔ Ridurre le dimensioni della FSM, rendendola più semplice e compatta;
- ➔ Riduzione delle risorse hardware necessarie per implementare il circuito;
- ➔ Rendere la FSM più efficiente in termini di spazio e potenza;
- ➔ Riduzione del tempo necessario per la simulazione e la verifica;
- ➔ Ampiamente adattabile essendo di dimensioni ridotte;
- ➔ Velocizzare il funzionamento del circuito stesso.

Conclusioni

In conclusione, il progetto di Reti Logiche è stato un'esperienza significativa che ha permesso di acquisire conoscenza del linguaggio VHDL che non avevo mai visto fino ad ora. Inutile dire che è un linguaggio ben lontano dai soliti linguaggi di programmazione e difatti è un linguaggio usato per la descrizione dell' hardware e dei sistemi digitali.
Grazie alle esercitazioni e al materiale fornito dai docenti del corso, dopo una prima settimana di prove, ho appresso la logica funzionale e progettuale dell'ambiente, in quanto progettazione e l'implementazione con VHDL richiedono un approccio metodico e rigoroso.
All'inizio ho avuto problemi nella stesura del codice in quanto anche il troubleshooting era basato sull'analisi profonda del waveform, un approccio molto diverso rispetto a come ero abituato in precedenza.
Successivamente con l'uso di alcune variabili temporanee e con alcune modifiche alla wave frame, sono riuscito a capire meglio gli errori.
Concludendo, il progetto è stato un'opportunità di crescita professionale e di approfondimento delle conoscenze in un campo a me ignoto.