Prova Finale (Progetto di Reti Logiche)

Docente: Fabio Salice - A.A. 2019/2020

Samuele Negrini (Codice Persona 10539341 - Matricola 866797)

Indice

1 Introduzione

La specifica della Prova Finale (Progetto di Reti Logiche) 2019 è ispirata al metodo di codifica a bassa dissipazione di potenza denominato *Working Zone*.
Tale metodo è pensato per il Bus Indirizzi e si usa per trasformare il valore di un indirizzo, quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working-zone). Una working-zone è definita come un intervallo di indirizzi di dimensione fissa (D_{wz}) che parte da un indirizzo base. All'interno dello schema di codifica possono esistere multiple working-zone (N_{wz}).

1.1 Obiettivo del progetto

Dati gli indirizzi base delle working-zone e l'indirizzo da codificare, si vuole implementare un componente hardware descritto in VHDL in grado di leggere l'indirizzo da codificare e gli indirizzi base delle working-zone, verificare l'appartenenza dell'indirizzo da codificare a tali working-zone e produrre l'indirizzo opportunamente codificato in uscita. In pratica, il modulo da sviluppare si comporta come un **encoder** di indirizzi.

1.2 Specifica generale

Si consideri l'indirizzo da trasmettere *ADDR*. Lo schema di codifica implementato è il seguente:

- se *ADDR* non appartiene a nessuna Working Zone **[WZ MISS]**, verrà trasmesso un bit addizionale $WZ_{BIT} = 0$ concatenato ad *ADDR* (WZ_{BIT} & *ADDR*, dove & è il simbolo di concatenazione);

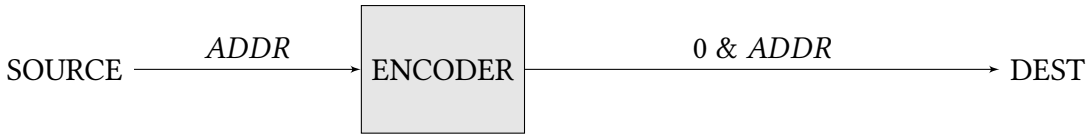


Figura 1: codifica di un indirizzo non appartenente a nessuna WZ.

- se *ADDR* appartiene ad una Working Zone **[WZ HIT]**, verrà trasmesso $WZ_{BIT} = 1$ concatenato a due valori WZ_{NUM} e WZ_{OFFSET}, rappresentanti rispettivamente:
  - il numero della working-zone al quale l'indirizzo appartiene, codificato in binario;
  - l'offset rispetto all'indirizzo di base della working-zone, codificato come one-hot.

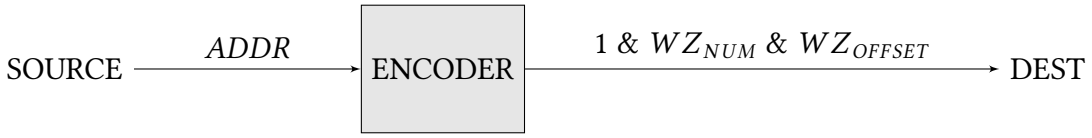
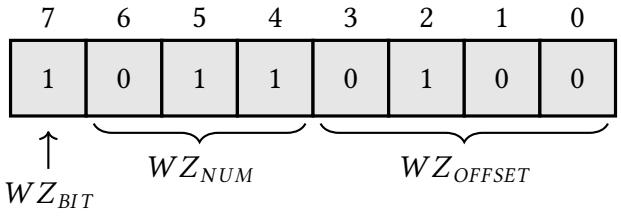


Figura 2: codifica di un indirizzo appartenente ad una WZ.

Si considerino 7 bit per l'indirizzo da codificare (quindi indirizzi validi da 0 a 127).
Il numero di working-zone è 8 (N_{wz} = 8) mentre la dimensione della working-zone è di 4 indirizzi incluso quello base (D_{wz} = 4). Questo comporta che l'indirizzo codificato sarà composto da 8 bit: 1 bit per WZ_{BIT} + 7 bit per *ADDR*, oppure 1 bit per WZ_{BIT}, 3 bit per codificare in binario a quale tra le 8 working-zone l'indirizzo appartiene, e 4 bit per codificare one-hot il valore dell'offset di *ADDR* rispetto all'indirizzo base.

Esempio: codifica di *ADDR*=33, sapendo che la WZ 3 ha come indirizzo di base 31.



Codifica WZ_{OFFSET}	
offset	one-hot
0	000**1**
1	00**1**0
2	0**1**00
3	**1**000

Figura 3: ADDR appartiene alla WZ 3, con offset 2 (in one-hot).

1.3 Interfaccia del componente

Il componente descritto possiede la seguente interfaccia:

```
entity project_reti_logiche is
port (
        i_clk           : in std_logic;
        i_start         : in std_logic;
        i_rst           : in std_logic;
        i_data          : in std_logic_vector(7 downto 0);
        o_address       : out std_logic_vector(15 downto 0);
        o_done          : out std_logic;
        o_en            : out std_logic;
        o_we            : out std_logic;
        o_data          : out std_logic_vector(7 downto 0)
);
end project_reti_logiche;
```

In particolare:

- `i_clk` è il segnale di CLOCK in ingresso generato dal Test Bench;
- `i_start` è il segnale di START generato dal Test Bench;
- `i_rst` è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;

- o_address è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- o_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

1.4 Dati e memoria

Il componente dovrà gestire la comunicazione con una memoria ausiliaria sulla quale saranno presenti i dati in ingresso e andranno salvati quelli in uscita.
I dati, ciascuno di dimensione 8 bit, sono memorizzati sulla memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo da codificare che è da specifica di 7 bit viene memorizzato su 8 bit (il valore dell'ottavo bit sarà sempre zero).

La memoria è così organizzata:

- Le posizioni da 0 a 7 sono usate per memorizzare gli otto indirizzi base delle working-zone;
- La posizione 8 è usata per memorizzare il valore (indirizzo) da codificare (*ADDR*);
- La posizione 9 è usata per scrivere il valore codificato in uscita.

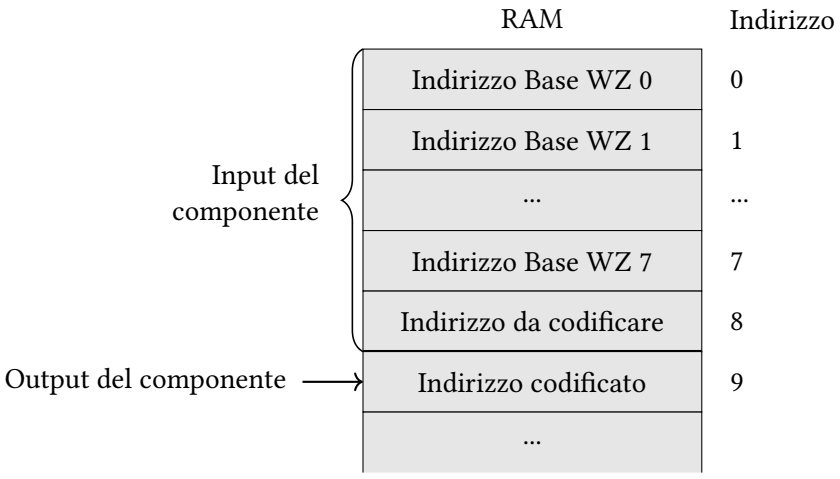


Figura 4: indirizzi della RAM rilevanti.

2 Progettazione

Il modulo partirà nell'elaborazione quando il segnale i_start in ingresso verrà portato a 1. Il segnale i_start rimarrà alto fino a che il segnale o_done non verrà portato alto. Al termine della computazione (e una volta scritto il risultato in memoria), il modulo alza (cioè porta a 1) il segnale o_done che notifica la fine dell'elaborazione. Se a questo punto viene rialzato il segnale i_start, il modulo ripartirà con la fase di codifica.

2.1 Scelte progettuali

Si è scelto di descrivere il componente tramite una macchina a stati finiti (FSM).
In VHDL sono stati utilizzati due process:

- Un process per descrivere la parte sequenziale della FSM, ovvero, rappresentare i registri di stato. Questo process si occupa anche di riportare la macchina nello stato di reset in corrispondenza del valore attivo del segnale i_rst.
- Un process per descrivere la parte combinatoria della FSM, ovvero, determinare lo stato in cui evolve il sistema in funzione dei segnali in ingresso e dello stato corrente.

In particolare il modulo confronta gli indirizzi ADDR e WZ sfruttando l'operazione di **sottrazione** e, nel caso di ADDR appartenente alla Working Zone, codifica la differenza (offset) in one-hot.
La codifica one-hot è stata realizzata mediante l'uso di un registro inizializzato ad ogni avvio della FSM a "0001" e l'operazione di **shift** (a sinistra) di un numero pari alla differenza tra i due indirizzi.
In aggiunta, si è preferito tenere memorizzate nel componente meno informazioni possibili. Ciò ha favorito un'occupazione d'area del modulo (in termini di flip-flop e look-up tables) molto piccola rispetto alla FPGA scelta inizialmente.
Queste soluzioni offrono il **vantaggio di essere scalabili**, poiché se aumentasse il numero di working-zone da controllare, non sarebbe necessario aumentare di molto l'area del modulo solo per tenere memorizzati tutti gli indirizzi delle working-zone, nè reimplementare interamente parti del codice per supportare le ulteriori codifiche one-hot necessarie.

Inoltre si è adottato un approccio che cercasse di:

- ridurre il numero di transizioni effettuate per portare a termine la codifica dell'indirizzo;
- chiedere un dato alla RAM solo quando strettamente necessario, in modo da minimizzare gli accessi in memoria. Per esempio, una volta trovata la working-zone di appartenenza dell'indirizzo da codificare, risulta inutile un controllo delle eventuali working-zone rimanenti.

Il tutto si è tradotto in una progettazione della FSM con un numero di stati ridotto e in un attento utilizzo dei bus per comunicare con la RAM.

2.2 Macchina a stati

La macchina implementata (Figura 5) è composta dai seguenti sei stati:

- IDLE
  Stato iniziale di reset in cui si attende che venga alzato il segnale i_start. In caso di segnale i_rst = '1' si torna in questo stato.

- FETCH_ADDR
  Stato in cui viene richiesto alla memoria l'indirizzo da codificare.

- WAIT_RAM
  Stato in cui si attende la risposta della memoria dopo una richiesta di lettura.

- GET_ADDR
  La prima volta (dopo un reset) che la macchina entra in questo stato viene memorizzato, in un apposito registro, l'indirizzo da codificare. Per risparmiare un periodo di clock, viene inoltre richiesto alla memoria l'indirizzo base della prima Working Zone (WZ 0).
  Le successive volte viene richiesto l'indirizzo base della prossima Working Zone e viene letto dalla memoria l'indirizzo base della Working Zone richiesta precedentemente. Questo verrà confrontato direttamente con l'indirizzo da codificare per stabilirne l'eventuale appartenenza. In caso positivo, viene salvato l'offset in un registro e il sistema passa allo stato WRITE_BACK.
  Si torna in questo stato finché non si trova la Working Zone di appartenenza dell'indirizzo da codificare o fino a quando non si sono esaurite le Working Zone da controllare.

- WRITE_BACK
  Stato in cui l'indirizzo codificato viene scritto nell'indirizzo 9 della memoria e si passa allo stato DONE.

- DONE
  Stato finale in cui si pone il segnale o_done = '1'.
  Si resta in questo stato fino a quando non si riceve i_start = '0' per poter poi abbassare il segnale o_done e tornare nello stato IDLE.

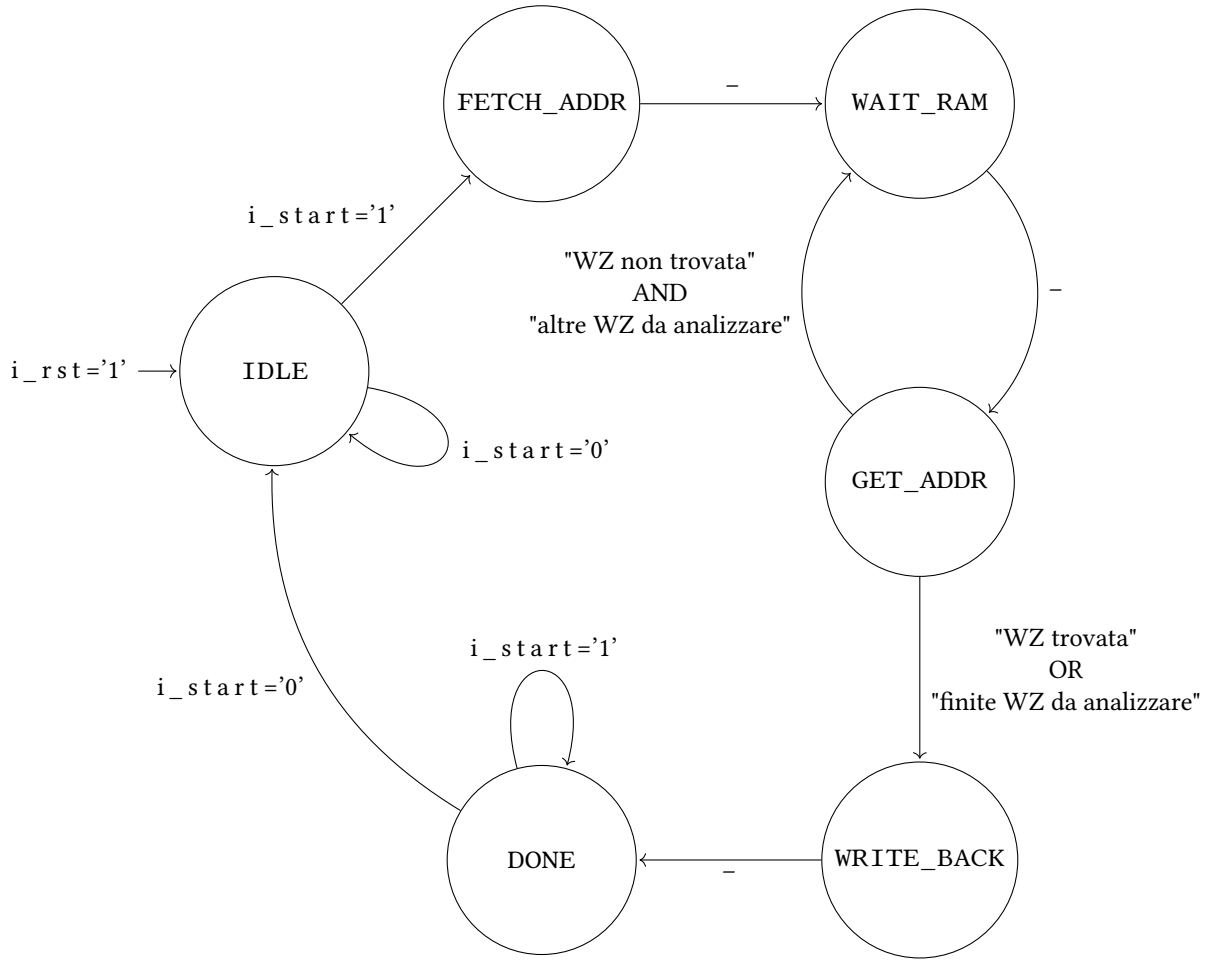


Figura 5: rappresentazione della macchina a stati.

3 Testing del componente

Il corretto funzionamento del componente è stato verificato tramite l'utilizzo dei test bench. Tre sono le modalità di simulazione eseguite: *Behavioural*, *Post-Synthesis Functional* e *Post-Synthesis Timing*.
A partire dai test bench di esempio, sono stati costruiti altri test bench che cercassero di simulare il maggior numero di possibili scenari d'esecuzione del modulo.
Di seguito sono riportati i casi di test più significativi. I primi due forzano il controllo di tutte le working-zone presenti in RAM.

1. **WZ MISS**: l'indirizzo da codificare non appartiene a nessuna working-zone.

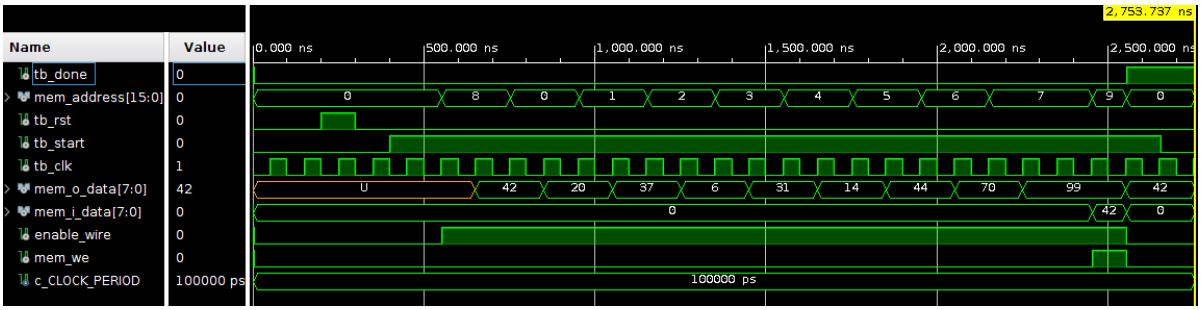


Figura 6: waveform della simulazione con ADDR non appartenente a nessuna WZ.

2. **Ultima WZ HIT**: il test verifica la codifica dell'indirizzo nel caso in cui esso appartenga all'ultima working-zone disponibile in RAM (WZ 7).
3. **Prima WZ HIT**: il test verifica la codifica dell'indirizzo nel caso in cui esso appartenga alla prima working-zone disponibile in RAM (WZ 0).

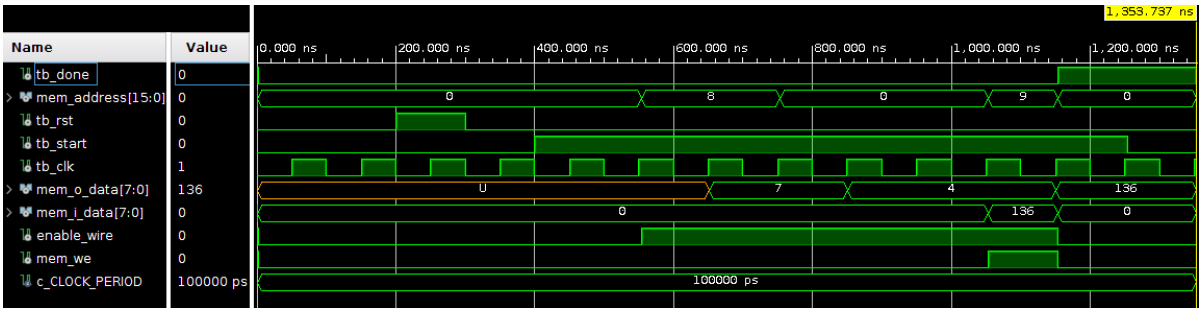


Figura 7: waveform della simulazione con *ADDR* = 7 appartenente alla prima WZ.

Ciascuno dei successivi test è stato eseguito sia in occorrenza ***WZ MISS*** che ***WZ HIT***.

4. **WZ adiacenti**: le working-zone caricate nella RAM sono una adiacente all'altra.
5. **ADDR/WZ minimi**: l'indirizzo da codificare e/o una WZ corrispondono alla codifica in binario composta da soli zero "00000000" (0 in decimale).

6. **ADDR massimo**: l'indirizzo da codificare corrisponde alla codifica in binario "01111111" (127 in decimale).
7. **WZ massima**: l'indirizzo di una working-zone corrisponde alla codifica in binario "01111100" (124 in decimale). Questa rappresenta la massima codifica possibile su 8 bit (con il bit più significativo a 0) affinché la working-zone risulti completa, cioè con 4 indirizzi di offset (124, 125, 126, 127).
8. **Reset asincrono**: il test verifica il comportamento della macchina quando il segnale di reset viene alzato in modo asincrono. La macchina torna nello stato di reset IDLE dove rimane in attesa di un nuovo segnale di inizio.

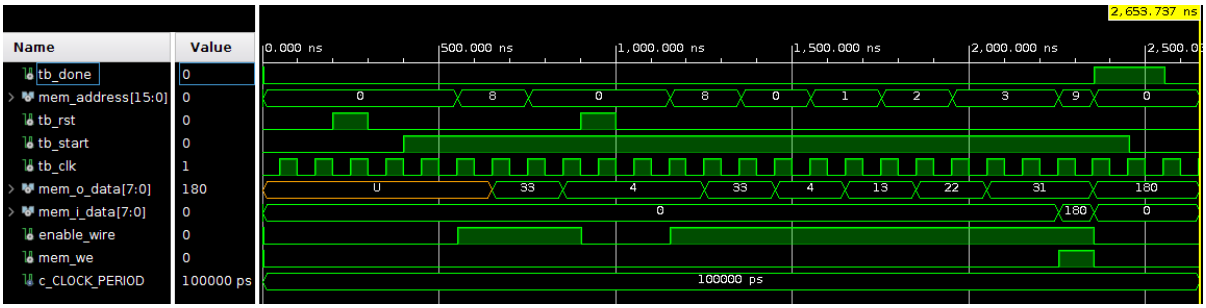


Figura 8: waveform della simulazione con reset asincrono.

9. **Multi-start, stesse WZ**: il test effettua la codifica (in sequenza) di almeno due indirizzi diversi mantenendo invariate le working-zone durante ogni computazione.
10. **Multi-start con reset (diverse WZ)**: il test effettua la codifica (in sequenza) di almeno due indirizzi. Le working-zone tra una computazione e l'altra vengono cambiate tramite l'asserzione del segnale `i_rst`.
11. **Frequenza elevata**: un caso di test è stato dedicato alla variazione della frequenza di elaborazione del modulo. Dal periodo di clock di default pari a 100ns si è testato fino ad un minimo di circa 5ns.

Oltre ai test mirati proposti sopra, sfruttando la modalità batch di Vivado, sono stati simulati numerosi test bench generati casualmente da un apposito script scritto in Python.

4 Risultati della sintesi

Il componente è stato sintetizzato con successo. Di seguito vengono riportati alcuni dei risultati ottenuti durante questa fase.

4.1 Registri sintetizzati

In Tabella 1 sono elencati i registri sintetizzati.

# bit	Quantità	Descrizione
16	1	Indirizzo della RAM. Si utilizzano 16 bit per semplificarne la gestione.
8	2	Indirizzo da codificare e indirizzo codificato in uscita. Vengono salvati per assicurare la corretta propagazione dei segnali.
4	1	Offset dell'indirizzo (WZ_{OFFSET}).
3	1	Stato presente della macchina. Sono necessari 3 bit per la codifica dei sei stati.
1	5	WZ_{BIT}, segnale di controllo e tre segnali di uscita del componente.

Tabella 1

4.2 Area utilizzata sulla FPGA

Come da scelte progettuali, si è cercato di ridurre al minimo l'area occupata dal componente sulla FPGA target. Nell'ottica di realizzare componenti più complessi, si è ritenuto che questo encoder non dovesse richiedere risorse che andrebbero impiegate per altri scopi.
Dal report fornito dallo strumento di sviluppo Vivado otteniamo le seguenti percentuali di area occupata rispetto alla FPGA target utilizzata.

Tipo	Utilizzati	Disponibili	Percentuale
FF	32	269200	0.01%
LUT	39	134600	0.03%

Tabella 2

4.3 Frequenza di funzionamento

Con un calcolo approssimativo è possibile ottenere la massima frequenza a cui può operare il componente. Si consideri il periodo di clock T_{clk} di $100ns$ (da specifica). Dal report si ottiene un *Worst Negative Slack* WNS pari a $94.701ns$. Il minimo periodo utilizzabile risulta:

$$T_{min} = T_{clk} - WNS = 100ns - 94.701ns = 5.299ns$$

e quindi si ottiene la massima frequenza di funzionamento:

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{5.299ns} \approx 188.72MHz.$$

5 Conclusione

Il componente sviluppato simula correttamente un encoder di indirizzi secondo il metodo *Working Zone*.
La trasformazione della specifica in una macchina a stati finiti ha semplificato notevolmente la fase di architettura del modulo.
Successivamente la macchina è stata trascritta in codice VHDL che è stato testato e sintetizzato correttamente.
Infine, si è cercato di ottimizzare il codice (e quindi la macchina stessa) in modo da ridurre l'area occupata e abbattere i tempi d'esecuzione del modulo.