POLITECNICO DI MILANO

INGEGNERIA DELL'INFORMAZIONE

Anno Accademico 2019 / 2020

Prova Finale
Progetto di Reti Logiche

Etis Peza

Docente: Gianluca Palermo

MATRICOLA: 917608 CODICE PERSONA: 10404126

Indice

1. Introduzione

1.1 Scopo del Progetto

Il rapido incremento della complessità dei chip e la popolarità dei dispositivi portatili, manifestatosi negli ultimi decenni, hanno conseguito che la richiesta di prestazioni non fosse più l'unico vincolo importante nel sistema incorporato. Al contrario, il consumo di energia e' diventato uno dei principali problemi di progettazione per i sistemi embedded contemporanei. A tale proposito il consumo di energia di un microprocessore puo' essere sostanzialmente ridotto, ottimizzando il numero di transizioni sui pin I/O. La maggior parte di questa attività si trova negli indirizzi e nei data lines. Lo scopo di tale progetto è ridurre l'attività in uscita sull'indirizzo attraverso l'implementazione del cosidetto metodo Working Zone Encoder (WZE).

1.2 Specifiche generali

Se l'indirizzo che dovrà essere trasmesso non appartiene agli intervalli delle Working Zone, esso verrà trasmesso cosi com'e' e in aggiunta,gli viene concatenato un bit detto, WZ_BIT, d'avanti e messo a valore 0. Questa codifica è importante perchè permette l'imposizione di limitazioni alla WZO (Working Zone Offset) in modo da poter, così, ridurre il numero delle combinazioni possibili.

WZ_BIT – 1 bit (Costante 0)	Address – 7 bit

Figura 1 – Codifica originaria dell'indirizzo

Se invece l'indirizzo "cade" nell'intervallo di una delle Working Zone, allora l'indirizzo codificato sarà diviso in tre sezioni: un singolo bit (detto WZ_BIT) a cui gli verrà assegnato il valore 1, tre bit per l'indirizzo della Working Zone (WZ_NUM) e quattro bit per la Working Zone Offset (WZO).

WZ_BIT – 1 bit (Costante 1)	WZ_NUM – 3 bit	WZ_OFFSET – 4 bit

Figura 2 – Indirizzo WZE

A questo punto l'indirizzo di output e' stato cambiato. Minimizzando il numero delle transizioni in WZO, l'attività dell'indirizzo in uscita verrà ridotto. Quindi viene applicata una codifica one-hot attraverso cui non ci possano essere più di due transizioni mentre viene trasmesso il Working Zone Offset (WZO).

Offset	One Hot Encoding
0	0001
1	0010
2	0100
3	1000

Figura 3 – Working Zone Offset Codifica

Per fare in modo che un indirizzo venga codificato, bisogna che vengano create, in modo dinamico oppure statico, una lista di indirizzi accessibili. Quest'ultima viene salvata sulla memoria RAM, la quale viene letta in modo sequenziale fino al punto in cui non si troverà un indirizzo in ingresso appartenente ad un intervallo di una delle tante Working Zone. Ovviamente tali indirizzi non cambieranno mai all'interno della stessa esecuzione.

Il modulo inizierà a elaborare nel momento in cui un segnale START in ingresso verrà portato a 1 e rimarrà fisso fin quando un secondo segnale, DONE, non verrà anch'esso portato a 1. Quando ciò accade, START verra' settato a 0 e successivamente anche DONE verra' settato a 0, altrimento il ciclo non può ricominciare.

1.3 Interfaccia del componente

Il componente da descrivere deve avere 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 TestBench;
- *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 O;
- *o_data* è il segnale (vettore) di uscita dal componente verso la memoria.

1.4 Dati e descrizione

I dati ciascuno di dimensione 8 bit sono memorizzati in una memoria con indirizzamento al Byte partendo dalla posizione 0. Anche l'indirizzo che è da specifica di 7 bit viene memorizzato su 8 bit. Il valore dell'ottavo bit sarà sempre zero.

-Le posizioni in memoria da 0 a 7 sono usati per memorizzare gli otto indirizzi base delle working-zone:

- 0 - Indirizzo Base WZ 0
- 1 - Indirizzo Base WZ 1
- ...
- 7 - Indirizzo Base WZ 7

-La posizione in memoria 8 avrà al suo interno il valore (indirizzo) da codificare (ADDR);

-La posizione in memoria 9 è quella che deve essere usata per scrivere, alla fine, il valore codificato secondo le regole precedenti.

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	NumeroNaturale	Indirizzo Base WZ 0
1	NumeroNaturale	Indirizzo Base WZ 1
2	NumeroNaturale	Indirizzo Base WZ 2
3	NumeroNaturale	Indirizzo Base WZ 3
4	NumeroNaturale	Indirizzo Base WZ 4
5	NumeroNaturale	Indirizzo Base WZ 5
6	NumeroNaturale	Indirizzo Base WZ 6
7	NumeroNaturale	Indirizzo Base WZ 7
8	NumeroADDR	ADDR da codificare
9	ADDRCodificato	Valore codificato

Figura 4 – Tabella RAM esempio

Nella versione da implementare il numero di bit da considerare per l'indirizzo da codificare è 7, ciò definisce come indirizzi validi quelli da 0 a 127. Il numero di working-zone è 8 (Nwz = 8) mentre la sua dimensione è 4 indirizzi incluso quello base (Dwz = 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 in codifica one-hot il valore dell'offset di ADDR rispetto all'indirizzo base.

Il modulo da implementare leggerà l'indirizzo da codificare e gli 8 indirizzi base delle working-zone e dovrà produrre l'indirizzo opportunamente codificato.

2. Architettura

2.1 Tabella dei signal interni

NOME	TIPO	VALORE INIZIALE	DESCRIZIONE
current_state	state_type	U	Memorizza stato corrente
next_state	state_type	U	Memorizza stato successivo
Input_addr	std_logic_vector (6 downto 0)	0000000	Input da codificare
wz_match	std_logic	0	Usato come il primo bit che va a concatenarsi con ADDR
wz_base	std_logic_vector (2 downto 0)	000	Indirizzo di WZBase
wz_offset	std_logic_vector (3 downto 0)	0000	Indirizzo di WZO
Address	std_logic_vector (15 downto 0)	0000000000000000	Indirizzo attuale mandato alla RAM
previous_address	std_logic_vector (15 downto 0)	0000000000000000	Indirizzo precedente
done	std_logic	0	Attivato quando finisce la codifica
enable	std_logic	0	Attiva RAM
write	std_logic	0	Attiva il diritto di scrivere sulla RAM
dout	std_logic_vector (7 downto 0)	00000000	Segnale per l'usita

Figura 5 – Tabella dei segnali e I loro valori

2.2 Macchina a stati

Il primo modello della macchina a stati (FSM) e' stato basato su una FSM a 12 stati seguendo il seguente schema:

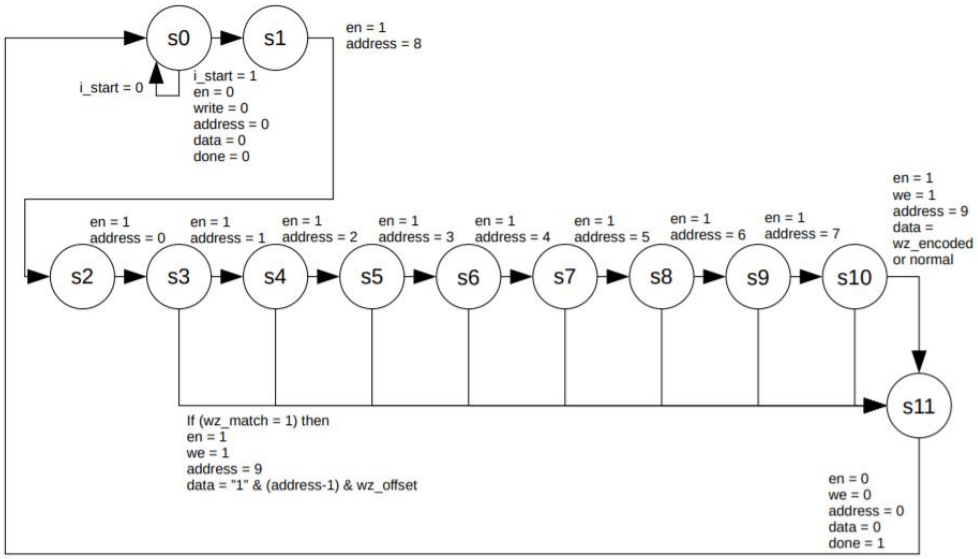

Figura 6 – Diagramma macchina a stati usata all'inizio

La qualità non è sempre correlata alla quantità, perciò ho operato in termini di efficienza unendo tutti i processi da S2 a S9 in un singolo stato, quest'ultimo capace di gestire anche l'incremento +1 dell'indirizzo della memoria RAM(da 0 fino a 7,valro che corrispondono alle WZ). L'utilizzo di questa procedura è supportata dalla letteratura (J. J. Jensen, 2018. *"One-process vs two-process vs three process state machine")*[1]. In relazione alla scelta tra Mealy o Moore, ho preferito il primo approccio dato che lo scopo primario è quello di ridurre il numero degli stati, anche se ciò ha comportato una perdita sul lato safety. Perciò tra safety e speed ho optato per la seconda.

Quindi la WZE e' stata disegnata usando una Macchina (a stati finiti) di Mealy in grado di leggere e scrivere su RAM.

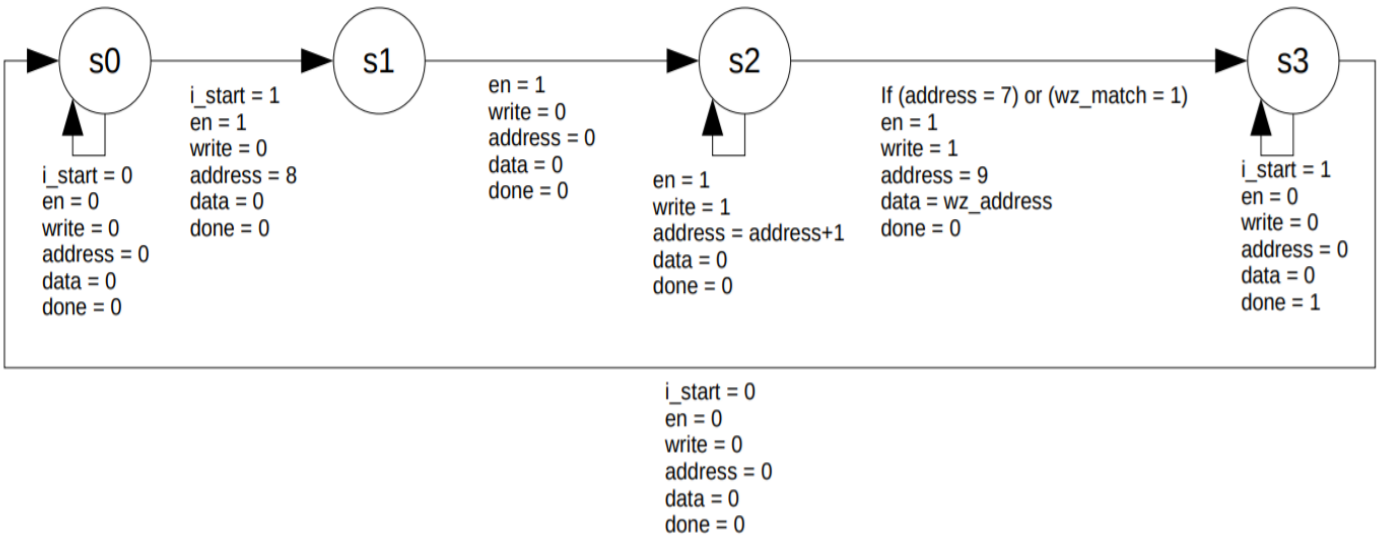

Figura 7 – La macchina a stati finale usata nel progetto

[1] https://vhdlwhiz.com/n-process-state-machine/

2.2 Stati

Come si puo' notare, ho optato nel ridurre il numero degli stati in soli quattro. E' doveroso sottolineare che lo stato piu' importante (a livello logico e non funzionale o di importanza) è l'S2.

Inoltre, tengo a precisare che ho preferito non nominare gli stati nè all'interno del codice nè sul diagramma ma per avere un idea piu' chiara riporto i nomi di essi nei seguenti quattro paragrafi.

2.2.1 IDLE State (S0)

IDLE State (S0) è il primo stato da dove si parte. Gestisce la lettura dell'indirizzo in ingresso. Il segnale input_addr rimane lo stesso (vettore di 7 bit settati a 0). Si rimane in questo stato finche' RESET e' uguale a 1 (ho preferito usare un reset sincrono, quindi ovviamente stiamo ipotizzando che ci sia un rising edge del CLOCK) altrimenti si passa ad un altro stato.

Infatti lo scopo di S0 è quello di fare lettura, rimanere in IDLE se non vi è una transizione (i_start='1' quando ovviamente i_rst=0') e arrivare a S1.

2.2.2 SIRR State (Store Input address/Read RAM address 0) S1

Come si capisce dal nome, questo stato ha la sua importanza nel copiare e salvare l'indirizzo che ci servira poi da codificare. Una volta in questo stato, il valore di input_data che allo stato S0 era settato a x"0000", prende il valore dell'ingresso i_data (6 downto 0). Il motivo è che ci servirà tale indirizzo per poter fare successivamente la codifica.

Da questo stato si procede verso lo stato S2.

2.2.3 Compare State (S2)

Compare State è lo stato piu importante e allo stesso tempo piu' intelligente. Come dichiarato prima, la logica implementata nel primo modello a 12 stati era stata quella di dover usare 8 stati solo per incrementare gli indirizzi di memoria della RAM da 0 a 7, leggerla e controllare in ogni stato se c'era un intervallo di Working Zone dove il nostro indirizzo iniziale ne faceva parte o meno. Nel caso affermativo, si andava subito allo stato S11.

Nel modello a 4 stati, invece, il funzionamento è il seguente. Ho implementato una logica fuori dai processi, dove viene prima ottenuto WZO (WZ_OFFSET) di lunghezza 4 bit, poi usando il segnale previous_address "taglio" i 3 bit (2 downto 0) e tale valore glielo assegno a wz_base (in questo modo ricavo anche I tre bit di wz_base). Infine con un semplice controllo tra wz_offset e "0000' vedo se wz_match (che per noi poi sarebbe il WZ_BIT) vale 0 oppure 1. Come? Se wz_offset="0000' allora vuol dire che l'indirizzo ADDR non appartiene a nessun working zone e quindi WZ_BIT=0 , altrimenti=1.

Riporto il codice utilizzato:

```
wz_offset(0) <= '1' when (unsigned(i_data)+0 = unsigned(input_addr)) else '0'; --WZE Offset 0
wz_offset(1) <= '1' when (unsigned(i_data)+1 = unsigned(input_addr)) else '0'; --WZE Offset 1
wz_offset(2) <= '1' when (unsigned(i_data)+2 = unsigned(input_addr)) else '0'; --WZE Offset 2
wz_offset(3) <= '1' when (unsigned(i_data)+3 = unsigned(input_addr)) else '0'; --WZE Offset 3

wz_base      <= previous_address(2 downto 0);                                  --WZ Base! Qua
wz_match     <= '1' when (wz_offset /= "0000") else '0';                       --WZ trovato! 1
```

Figura 8 – Ottenimento di WZO,WZ_BASE e WZ_MATCH

Una volta che mi trovo sullo stato S2, se wz_match='1' vuol dire che il mio indirizzo si trova in un working zone e inoltre conosco i valori WZO,WZ_NUM e WZ_BIT. Quindi cosa faccio? Concateno "1"&wz_base&wz_ofsset e il valore glielo assegno al segnale dout.

Se wz_match vale 0 ma l'indirizzo precedente letto sulla RAM e' stato il settimo, allora significa che non appartiene a nessun intervallo delle working zone e quindi dout assume il valore "0"&input_addr. Se, invece, l'indirizzo precedente non è 7,ma un numero inferiore, verrà incrementato di 1.
Poi ovviamente, se wz_match diventa '1' allora si procede verso lo stato S3, altrimenti si ritorna nello stesso stato (S2).

2.2.4 Done State (S3)

Un volta raggiunto questo stato (attraverso S2) controllo il valore di i_start:

- se i_start = '1' ,allora il segnale done diventa '1' e si ritorna in S0,
- se i_start = '0' si rimane nello stato S3.

2.3 Scelte progettuali

La scelta progettuale effettuata è stata quella di descrivere con la macchina a stati non piu di 5 processi. Infatti si usano 3 processi:

1. ***registers_ps***, processo che ha nella propria sensibility list soltanto il clock. Serve per controllare il fronte di salita del clock e i valori assunti dal reset;
2. ***outputs_ps***, processo che fa capire l'uso di una macchina di Mealy. Ha un sensibility list abbastanza vasta, infatti si usa per la parte sequenziale della macchina e serve per gestire il modo come vengono manipolati i registri.
3. ***next_state_ps***, come è possibile intuire dal nome, esso gestisce il passaggio da uno stato all'altro.

3.Risultati sperimentali

3.1 Sintesi (report di sintesi)

Synthesis finished with 0 errors, 0 critical warnings and 0 warnings.

Resource	Estimation	Available	Utilization %
LUT	66	134600	0.05
FF	25	269200	0.01
IO	38	285	13.33
BUFG	1	32	3.13

Figura 9 – Report di sintesi

Come rilevato dal report di sintesi, il componente e' correttamente sintetizzabile e implementabile dal tool con un totale di 66 LUT e 25 FF.

In un modo un po piu' dettagliato, abbiamo:

ADDERS

2 Input	16 Bit	Adders:=1
2 Input	8 Bit	Adders:=3

REGISTRI

16 Bit	Registers:=1
7 Bit	Registers:=1

MUX

2 Input	16 Bit	Muxes:=1
4 Input	16 Bit	Muxes:=1
2 Input	8 Bit	Muxes:=1
4 Input	8 Bit	Muxes:=1
4 Input	2 Bit	Muxes:=1
2 Input	2 Bit	Muxes:=3
3 Input	1 Bit	Muxes:=1
4 Input	1 Bit	Muxes:=3
2 Input	1 Bit	Muxes:=1

Usando i due test bench offerti dal docente e altri 2 custom, il componente ha superato correttamente tutti i test sia in pre-sintesi sia in post-sintesi.

3.2 Simulazioni.

Il componente una volta sintetizzato,supera senza errori e in modo corretto tutti I test specificati nelle 3 simulazioni:

- **Behavioral:**
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:06 . Memory (MB): peak = 2667.352 ; gain = 96.156

-Post-Synthesis Functional
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:08 . Memory (MB): peak = 2480.313 ; gain = 10.508

-Post-Synthesis Timing
launch_simulation: Time (s): cpu = 00:00:04 ; elapsed = 00:00:09 . Memory (MB): peak = 2503.738 ; gain = 23.426

4. Testbenches

4.1. Custom testbenches

Si sono testati 4 testbench (2 offerti dal docente e altri 2 custom).
Il primo custom testbench genera combinazioni di WZ e address. Inizia a fare singolo start, doppio start e reset dopo 4 clock. In totale sono stati testati fino a 1.5 millioni di testcases. Il codice ha superato tutti i test cases.
Il secondo custom testbench invece e' piu dinamico. E' stato creato per testare tutti gli indirizzi di input usando come rifferimento la tabella d'esempio proposta nel file "*Regole del docente*". Tale testbench non cambia gli indirizzi WZ_base ma invece modifica l'input da 0 a 127. Infatti testa tutti gli input possibili per una tabella WZ_base gia fornita in hard-coding. Nel testbench custom, la macchina è riuscita a codificare tutti i 128 address possibili. Si nota che WZE ci impiega molto meno cicli.
A livello di codice ho aggiunto un altro ramo elsif dentro il processo MEM:

```
elsif (tb_rst = '1') then --cambiamo il valore che vogliamo testare
  RAM(8) <= std_logic_vector(unsigned(RAM(8))+1);
end if;
```

Figura 10 – Ramo elsif del custom testbench

Ho iterato il processo test 128 volte per poter testare infatti tutti gli indirizzi.

Il componente ha passato tutti i suoi test:

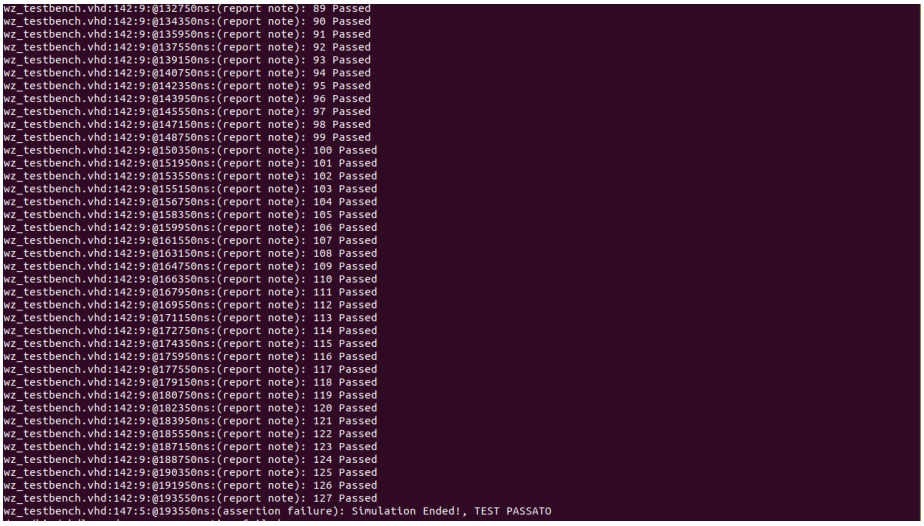

Figura 11 – Test passati tutti nell secondo custom testbench

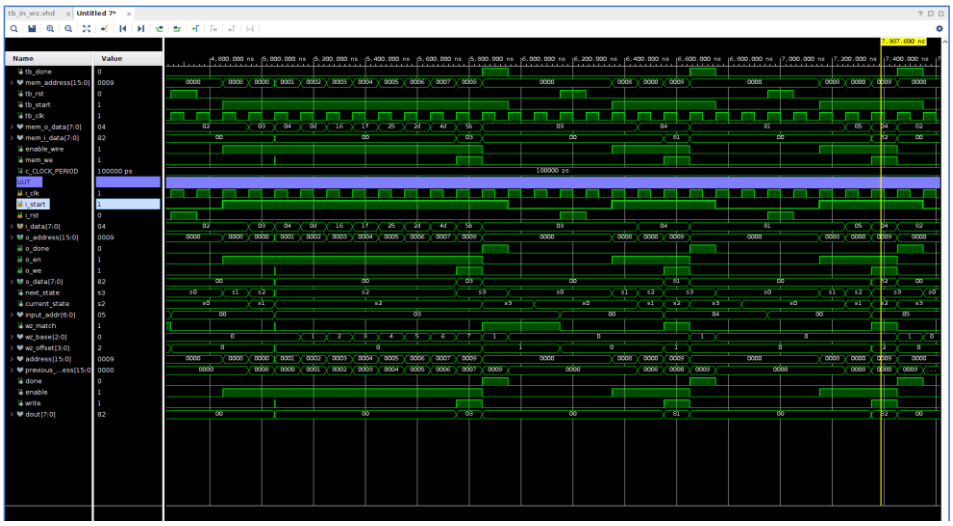

Figura 12 – Simulazione waveform del testbench custom

4.2 Default Testbenches

Testbench1:

Il testbench tb_in_wz inizia dichiarando un segnale di tipo ram, nominato RAM, il quale poi viene anche inizializzato facendo uso della tabella numero 2, trovata nel file *“PFRL_Specifica_1920.pdf*”.
Si è effettuata la mappatura delle porte del componente del nostro progetto.
Si procede con un processo il cui scopo e’ quello di lavorare con periodi di clock dimezzati. (ricordiamoci che o_CLOCK_PERIOD equivale a 100ns,quindi stiamo lavorando con cambio di fronte del clock ogni 50ns).
Il processo MEM invece si concentra a restituire il valore codificato all’indirizzo 9 della RAM. Si controlla prima se si ha un fronte di salita del clock, poi si verifica se ENABLE è pari a ‘1’ e infine se anche il valore di o_we è 1. Se ciò è vero allora l’indirizzo in ingresso viene salvato sulla RAM e dopo 1ns anche sulla variabile della memoria in uscita.
Tutto questo viene fatto attraverso il processo test, nel quale i segnali tb_rst e tb_start, cambiano dopo certi periodi di clock il loro valore,da 1 a 0. Nel testbench1 si aspetta che l’indirizzo di ingresso, cioe’ 33, venga controllato se appartiene in uno degli intervalli delle WZ. Nel momento in cui ci si arriva alla quarta working zone con intervallo [31,34] , il segnale (creato da me) wz_match diventa 1,si calcola il wz_offset e wz_num e si fa il concatenamento.100ns +1ns(dovuto al delay imposto nello statement) dopo il valore viene mandato a mem_o_data.

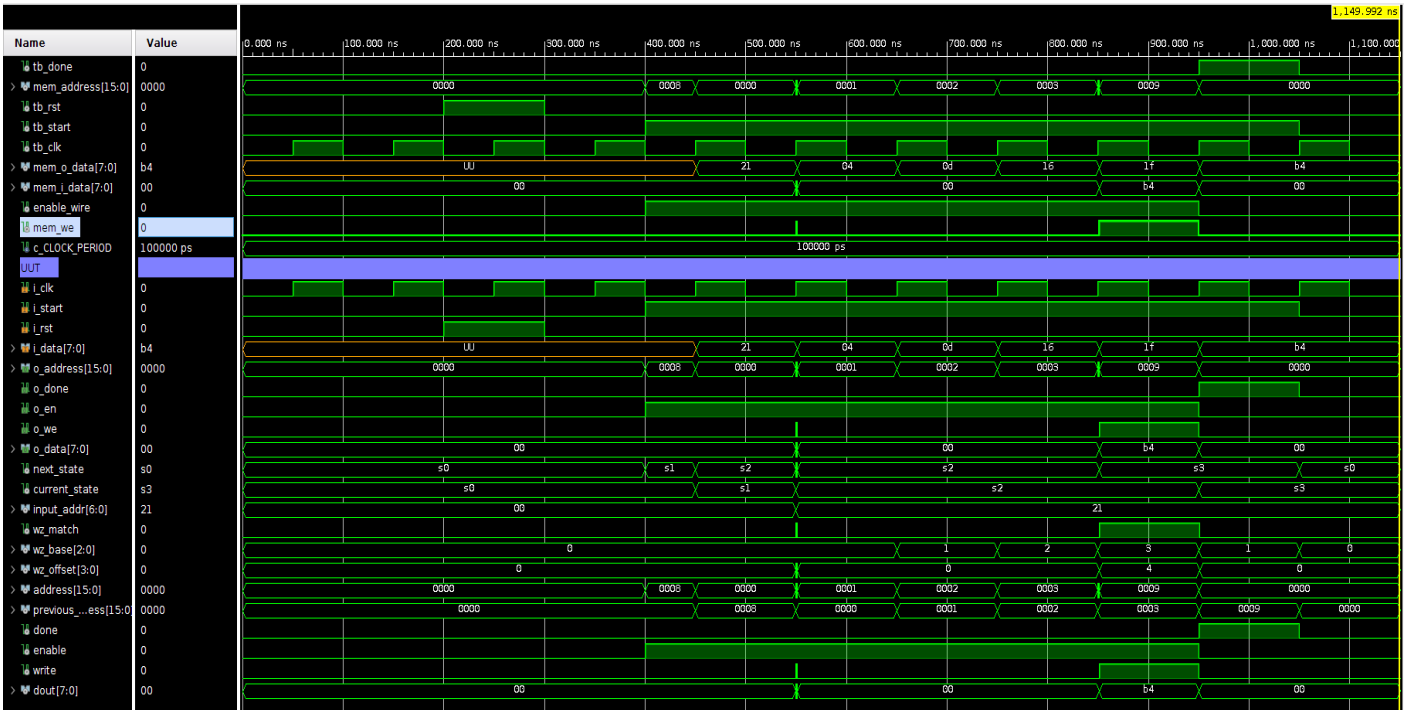

Figura 13 – Simulazione pre-synth waveform del testbench tb_in_wz

Figura 14 – Simulazione post-synth waveform del testbench tb_in_wz

Testbench2(tb_no_wz): Nel testbench 2 invece,avendo un indirizzo in ingresso diverso da prima (infatti vale 42),l'indirizzo non entra in nessuno degli intervalli delle WZ ed è per tale motivo che quello in uscita, che verra' poi salvato sulla RAM, sara' di 8 bit. Di essi, il primo bit sara impostato a 0 (wz_match nel mio caso) e poi concatenato con l'indirizzo in ingresso.
Al testbench uno (tb_in_wz) sono serviti meno cicli rispetto al secondo testbench perchè la macchina a stati non ha dovuto leggere tutti i possibili indirizzi dalla RAM.

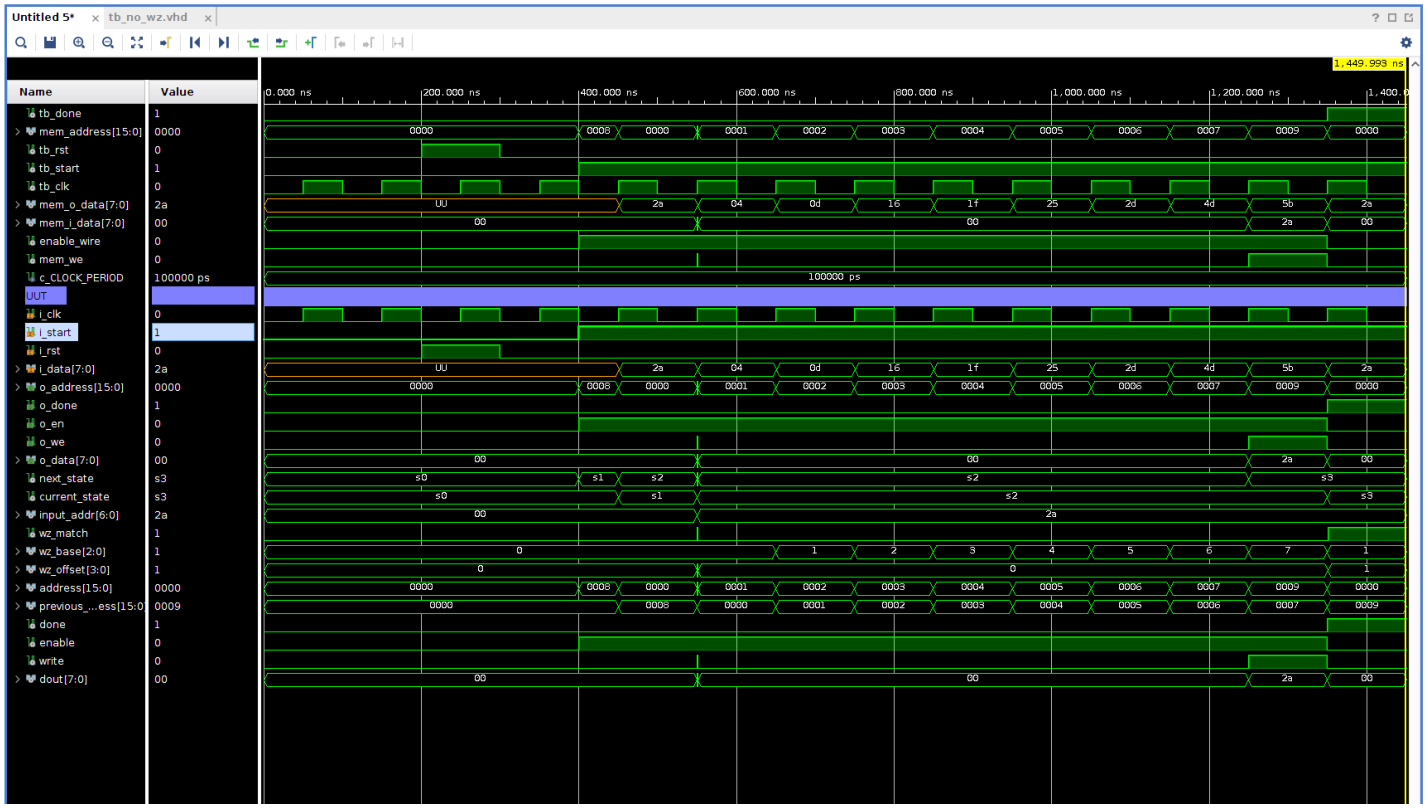

Figura 15 – Simulazione pre-synth waveform del testbench tb_no_wz

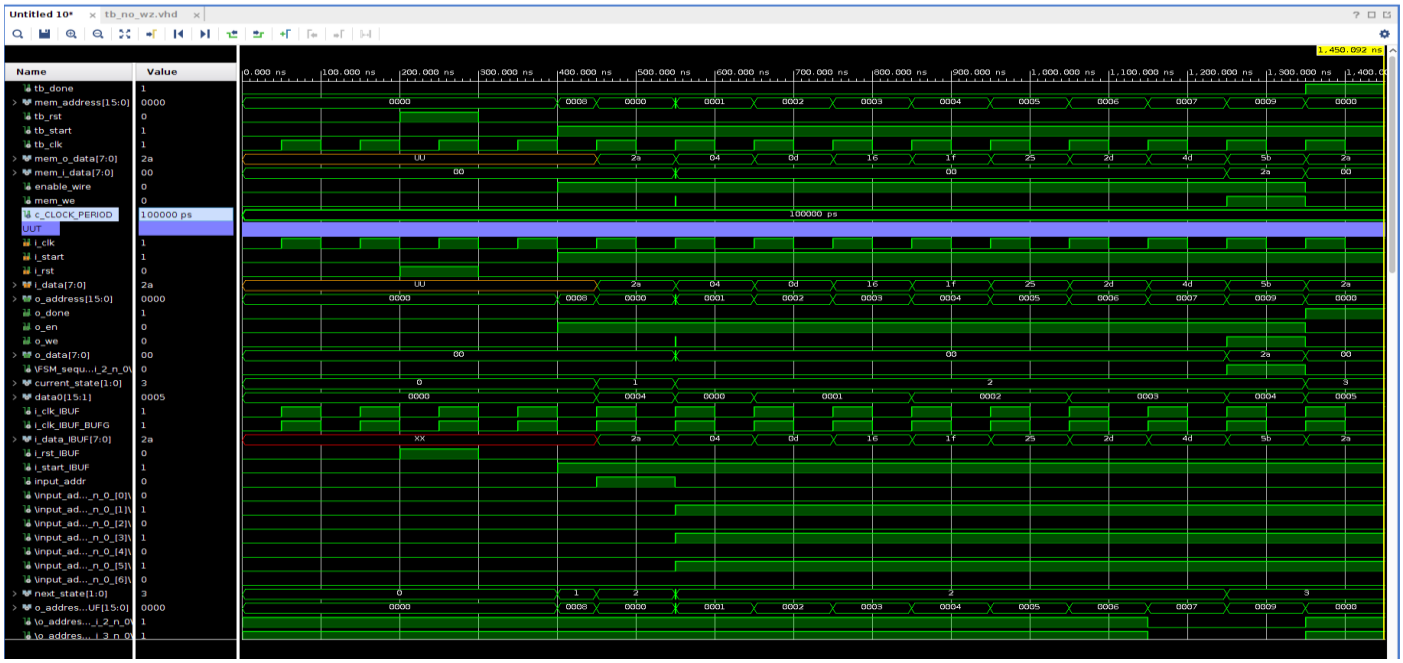

Figura 16 – Simulazione post-synth waveform del testbench tb_out_wz

5. Conclusioni

La macchina a Stati di Mealy, in grado di funzionare come Work Zone Encoder, e' in grado di leggere 8 indirizzi WZB dalla RAM e confrontare l'indirizzo di input alle 4 WZO in modo simultaneo. Se l'indirizzo di ingresso non appartiene agli intervalli delle working zone, allora viene codificato cosi com'e' (ovviamente concatenato con 0 prima), altrimenti vanno calcolati i bit per il WZ_BIT e WZ_NUM e poi concatenati con 1.

Dove sta il vantaggio? Facendo meno cicli consuma meno tempo e quindi meno energia. Si assume che l'energia consumata in un pin esterno presenta valori da 100 a 1000 volte piu grandi rispetto ad un nodo interno. Ad oggi tale dissipazione di energia si traduce in un rallentamento per il mezzo di comunicazione ad alta velocità. Nel corso degli ultimi anni si sono presentati tanti modelli di codifica, per alcuni di essi non serve sapere gli indirizzi in ingresso (come il modello Bus-Invert) mentre altri si affidano alla sequenza che viaggia nei bus e a quali algoritmi utilizzare per codificarla. Il problema che tutti questi modelli (incluso il WZE) si pongono è come minimizzare il numero delle transizioni che avviene nei bus. Questo modello poteva essere applicato alle tecnologie di inizio anni 2000, oggi invece bisogna accettare tali stati di transizione data la crescita esponenziale delle connessioni intra-wire o crosstalking. Infatti, si stima che una transizione ODD/EVEN (cioe' passare in modo simultaneo di due valori opposti) di due linee bus adiacenti dissipa quasi 4 volte più energia di quanto accada senza considerare gli effetti di coupling.[2]

Oggi si parla di permutazioni di alcune linee di bus che, forse, insieme agli altri algoritmi di codifica potranno aiutare a diminuire la quantita di energia dissipata.

[2] P.P. Sotiriadis, A.Chandrakasan. 2000. "Bus Energy Minimization by Transition Pattern Coding (TPC) in Deep SubMicron Technologies".