Progetto Reti Logiche 2019

Redattore: Marco Montorsi

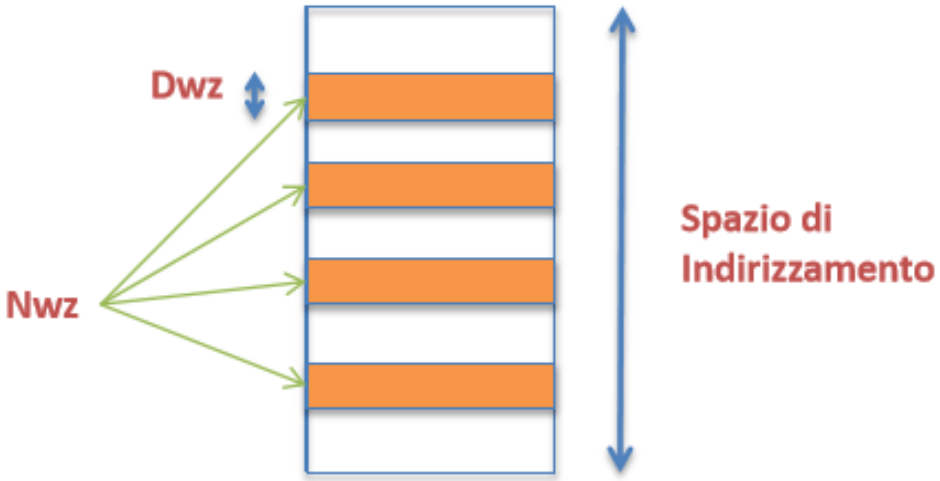
Sommario

1 Introduzione

1.1 Specifiche Di Progetto

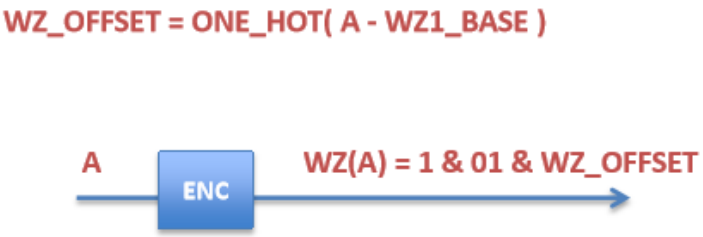
La specifica della Prova finale (Progetto di Reti Logiche) 2019 è ispirata al metodo di codifica a bassa dissipazione di potenza denominato “Working Zone”.
Il metodo di codifica Working Zone è un metodo pensato per il Bus Indirizzi che si usa per trasformare il valore di un indirizzo quando questo viene trasmesso, se appartiene a certi intervalli (detti appunto working-zone).

La working-zone è un intervallo di indirizzi di dimensione fissa (Dwz) che parte da un indirizzo base. All’interno dello schema di codifica possono esistere multiple working-zone (Nwz). Lo schema modificato di codifica da implementare è il seguente:



Quando l’indirizzo da trasmettere (ADDR) non appartiene a nessuna Working Zone, viene trasmesso così come è, e un bit addizionale rispetto ai bit di indirizzamento (WZ_BIT) viene messo a 0. In pratica dato ADDR, verrà trasmesso WZ_BIT=0 concatenato ad ADDR (WZ_BIT & ADDR, dove & è il simbolo di concatenazione

Quando l'indirizzo da trasmettere (ADDR) appartiene ad una Working Zone, il bit addizionale WZ_BIT assume il valore pari a 1, mentre i bit di indirizzo vengono divisi in 2 sotto campi rappresentanti: Il numero binario della working-zone al quale l'indirizzo appartiene (WZ_NUM), e L'offset rispetto all'indirizzo di base della working zone (WZ_OFFSET), codificato come one-hot .



Nella versione da implementare il numero di bit per la codifica è pari a 7. Gli indirizzi validi vanno da 0 a 127. Il numero di working-zone è 8 (Nwz=8) mentre la dimensione della working-zone è 4 indirizzi incluso quello base (Dwz=4), vedi **Tab.1**.

WZ_OFFSET	0	0001
WZ_OFFSET	1	0010
WZ_OFFSET	2	0100
WZ_OFFSET	3	1000

Tab.1 Codifica One-Hot

Ne deriva 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.

1.2 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 0

• o_data è il segnale (vettore) di uscita dal componente verso la memoria.

L'interfaccia del componente dovrà comunicare con la RAM per chiedere in lettura i valori delle WZ e infine per chiedere la scrittura del risultato, per questo progetto vi sono a disposizione 10 indirizzi, così suddivisi nella **Tab.2**:

Indirizzo 0	**Valore 1° WZ**
Indirizzo 1	**Valore 2° WZ**
Indirizzo 2	**Valore 3° WZ**
Indirizzo 3	**Valore 4° WZ**
Indirizzo 4	**Valore 5° WZ**
Indirizzo 5	**Valore 6° WZ**
Indirizzo 6	**Valore 7° WZ**
Indirizzo 7	**Valore 8° WZ**
Indirizzo 8	**Valore Indirizzo Da Codificare**
Indirizzo 9	**Valore Indirizzo Codificato**

Tab.2 Suddivisione indirizzi RAM

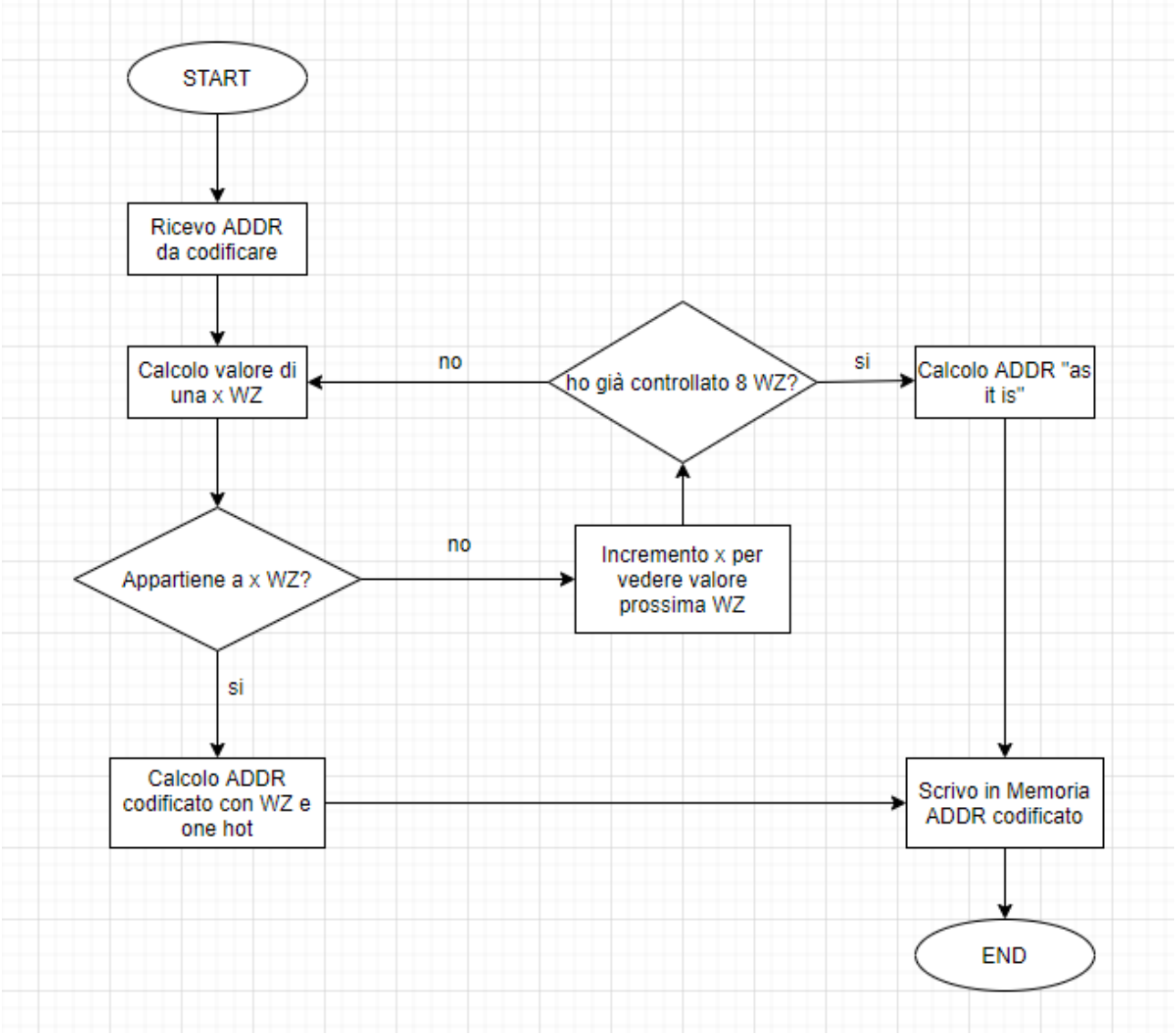
Note Finali Su Specifica:

- Si consideri che gli indirizzi base delle working-zone non cambieranno mai all'interno della stessa esecuzione, inoltre le working-zone non possono sovrapporsi
- Il modulo partirà nella elaborazione quando un segnale START in ingresso verrà portato a 1. Il segnale di START rimarrà alto fino a che il segnale di DONE non verrà portato alto; Al termine della computazione (e una volta scritto il risultato in memoria), il modulo da progettare deve alzare (portare a 1) il segnale DONE che notifica la fine dell'elaborazione. Il segnale DONE deve rimanere alto fino a che il segnale di START non è riportato a 0. Un nuovo segnale start non può essere dato fin tanto che DONE non è stato riportato a zero. Se a questo punto viene rialzato il segnale di START, il modulo dovrà ripartire con la fase di codifica.
- Il software di sviluppo utilizzato è Vivado e il linguaggio del codice è il VHDL.

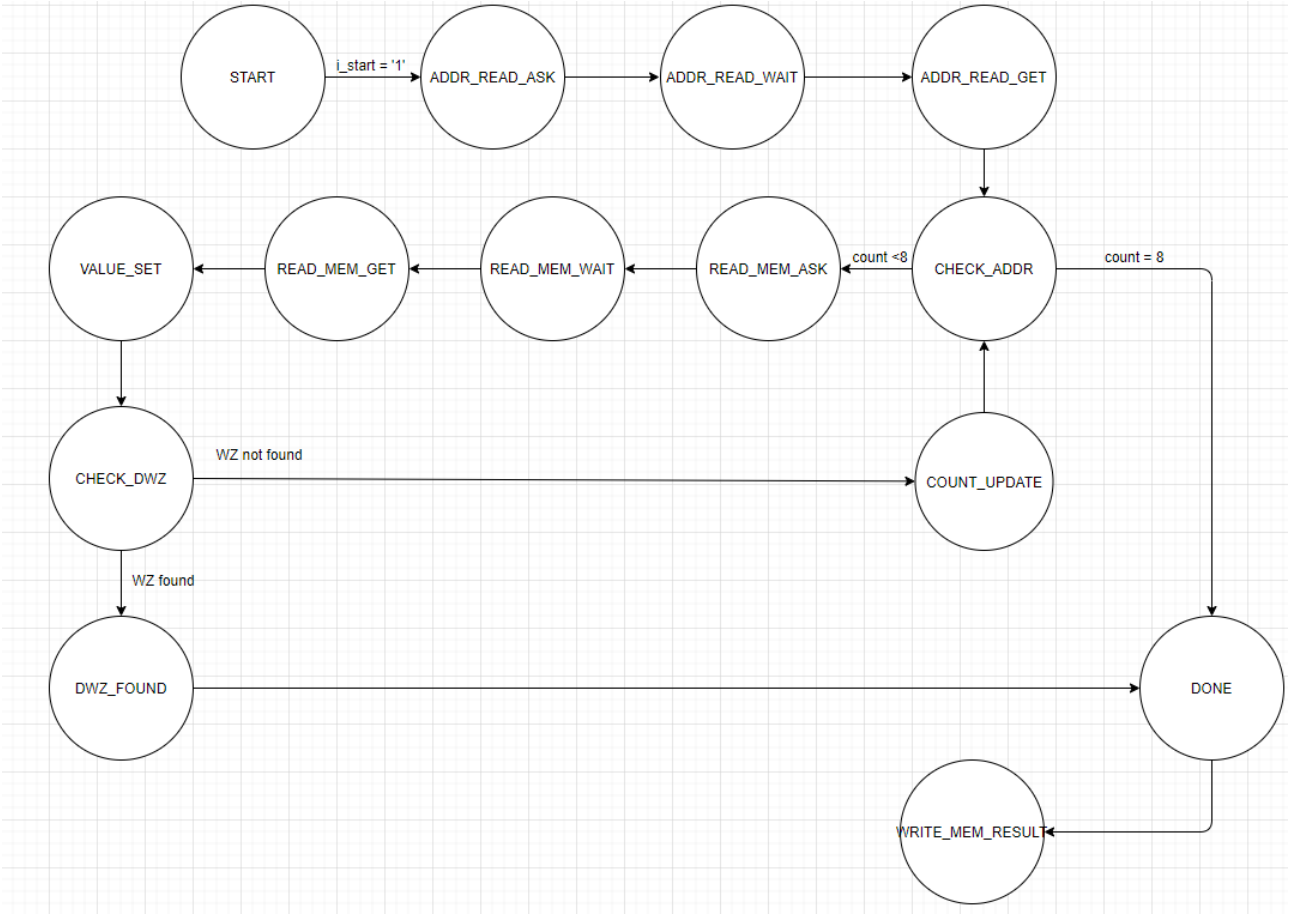
2 Scelte Progettuali

2.1 Scelta Design

- Inizialmente è stato elaborato un primo semplice algoritmo, con lo scopo di visualizzare le fasi del componente:



- Per implementare questo algoritmo si utilizza una Macchina a Stati Finiti (FSM), con la quale si costruiscono gli stati della macchina, aggiungendone diversi per un corretto funzionamento e per una corretta comunicazione con la RAM.



Per una lettura più comprensibile è stato omesso da ogni stato la scelta di i_rst, perché nel caso in cui durante l'esecuzione, i_rst venga portato a 1, la macchina tornerà in START

2.2 Descrizione Stati

- START : Stato iniziale della macchina dove inizializza le variabili utilizzate e aspetta fino a quando il segnale i_start diventi 1, una volta letto, procede verso ADDR_READ_ASK
- ADDR_READ_ASK : In questo stato la FSM chiede alla RAM il valore da codificare all'indirizzo 8, e poi procede verso ADDR_READ_WAIT
- ADDR_READ_WAIT : In questo stato la FSM non fa altro che aspettare per un ciclo di clock la risposta della RAM e procede verso ADDR_READ_GET
- ADDR_READ_GET : In questo stato la FSM salva il valore dell'indirizzo da codificare e procede verso CHECK_ADDR
- CHECK_ADDR : In questo stato la FSM controllerà quante WZ ha controllato grazie a una variabile intera count, se dovrà controllare ancora andrà in READ_MEM_ASK, altrimenti si porterà in DONE settando il parametro booleano "found_dwz"
- READ_MEM_ASK : In questo stato la FSM richiede l'indirizzo della prossima WZ da controllare alla RAM basandosi sul valore di count, per poi procedere in READ_MEM_WAIT
- READ_MEM_WAIT : In questo stato la FSM non fa altro che aspettare per un ciclo di clock la risposta della RAM e procede verso READ_MEM_GET

- READ_MEM_GET : In questo stato la FSM salva il valore della WZ e procede verso VALUE_SET
- VALUE_SET : In questo stato la FSM calcola la differenza, salvandola nella variabile difference_variables, la quale rappresenta la differenza tra l'indirizzo da codificare e della WZ controllata, infine procede verso CHECK_DWZ
- CHECK_DWZ : In questo stato la FSM fa un controllo su difference_variables, nel caso sia negativa sicuramente il valore dell'indirizzo della WZ è maggiore quindi non appartiene alla WZ, e quindi la FSM procede in COUNT_UPDATE, in caso contrario, quindi se la differenza è positiva, controllo se è minore o uguale di 3, se così allora l'indirizzo appartiene alla WZ e quindi la FSM procede verso DWZ_FOUND per la codifica dell'indirizzo, se invece la differenza è maggiore di 3 allora l'indirizzo non appartiene alla WZ, quindi si muoverà verso COUNT_UPDATE
- COUNT_UPDATE : In questo stato la FSM semplicemente incrementerà count di 1 per poi procedere verso CHECK_ADDR
- DWZ_FOUND : In questo stato la FSM codificherà l'indirizzo one hot , appartenente alla WZ settando la variabile booleana found_dwz a "true", e infine procederà verso DONE
- DONE : In questo stato la FSM, in base alla variabile found_dwz, invierà alla RAM l'indirizzo codificato corretto, per poi procedere verso WRITE_MEM_RESULT
- WRITE_MEM_RESULT : In questo stato la FSM setta o_done a 1, e aspetta fino a un segnale di i_start uguale a 0, una volta ricevuto torna a START

2.3 Analisi Punti Chiave Codice VHDL

1. Per il codice ho utilizzato un unico processo, process(i_clk,i_rst), con sensitivity list solo il parametro i_clk e i_rst , cosi da attivarsi ogni volta che i_clk o i_rst commuta.

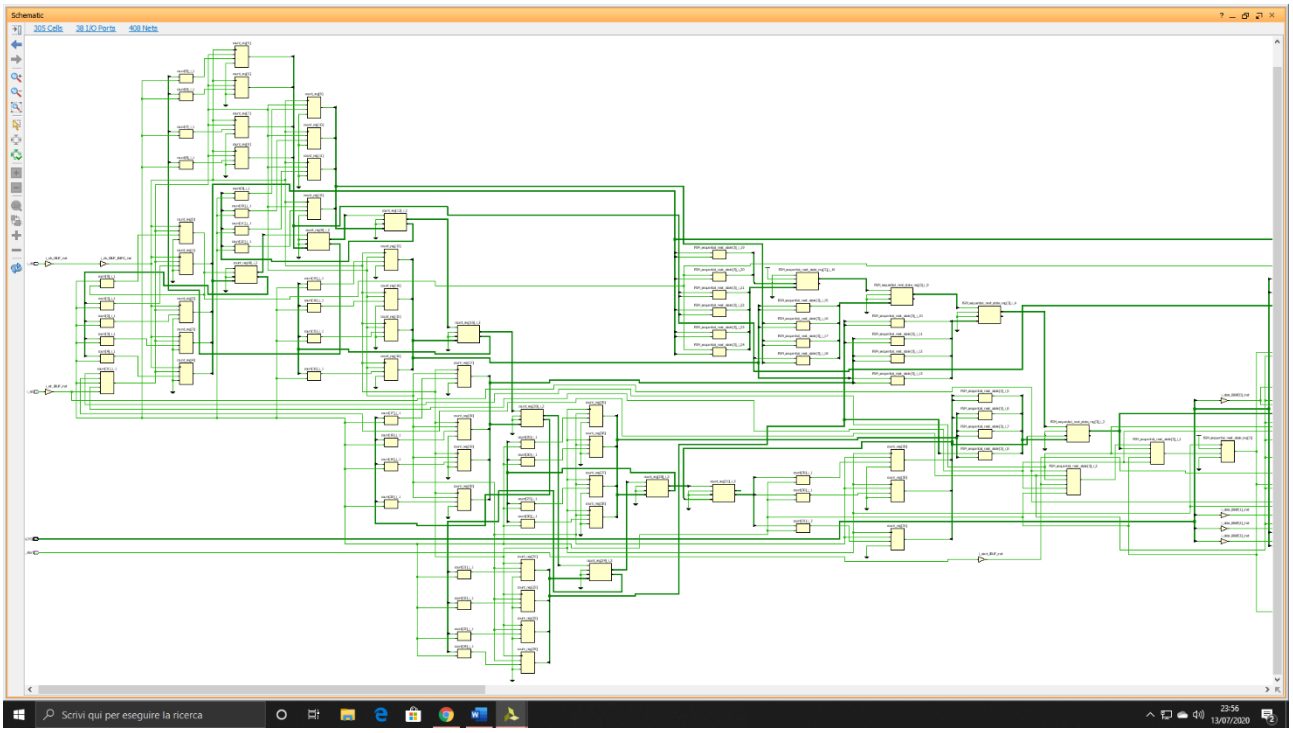
2. Per gli stati ho utilizzato una singola variabile di tipo state, chiamata next_state, alla fine di ogni stato, la FSM, setterà next_state al prossimo stato, quindi al prossimo clock la macchina entrerà nello stato next_state correttamente, può non essere immediata la comprensione di questa scelta, ma deriva da una meno recente versione del codice nella quale utilizzavo due variabili per lo stato, una per l'attuale e una per il prossimo, settandole in modo corretto quando il clok era a 0, operazione che poi è risultata intuile perché non comportava alcun beneficio rispetto all'utilizzo di un'unica variabile state.

3. I vari stati verranno gestiti attraverso un case sulla variabile next_state, dove ogni stato_x avrà il suo when "STATO_X" seguito dalle operazione svolte dalla FSM in quello stato.

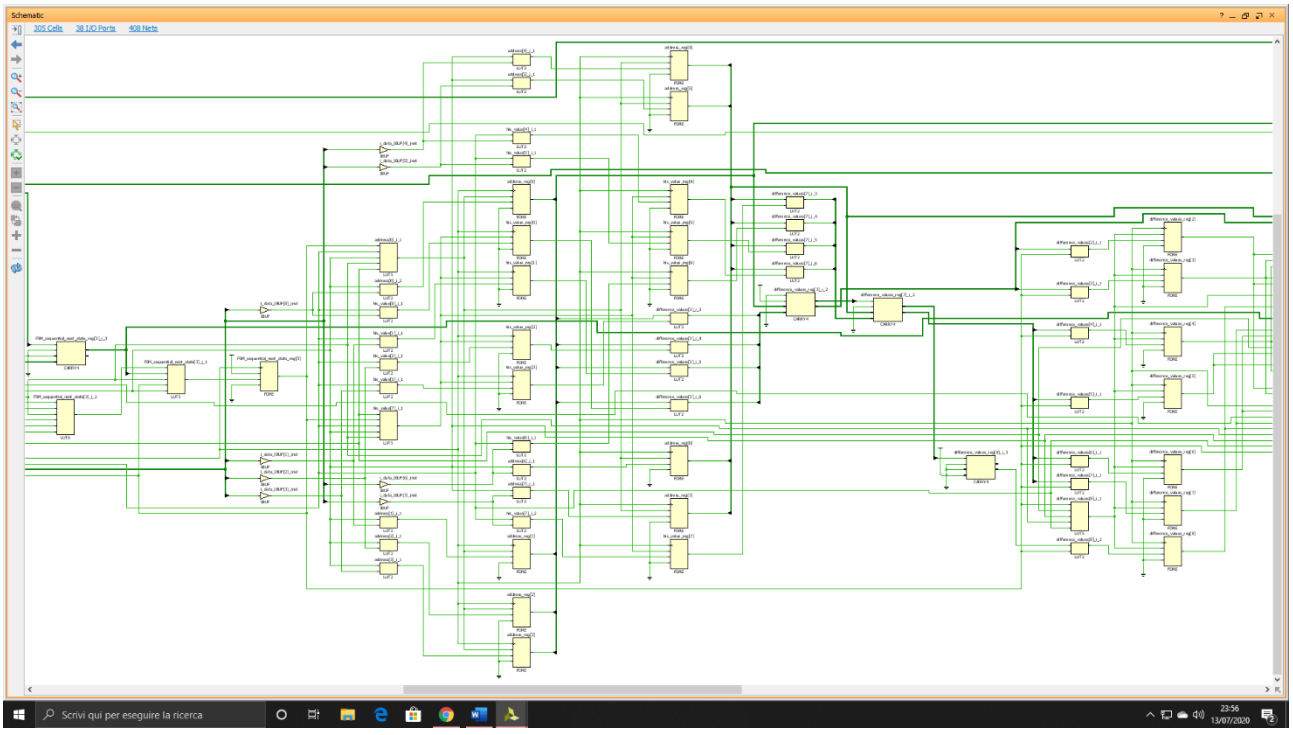
4. Una parte fondamentale di questo processo è il ruolo che gioca la variabile count, facendola partire da 0 la FSM riesce sempre ad indirizzarsi al corretto indirizzo di una WZ contenuto nella RAM, e quindi poi nel caso riesce a codificare la parte corretta della WZ nel caso l'indirizzo appartenga a tale WZ.

2.4 Register Transfer Level Schematic

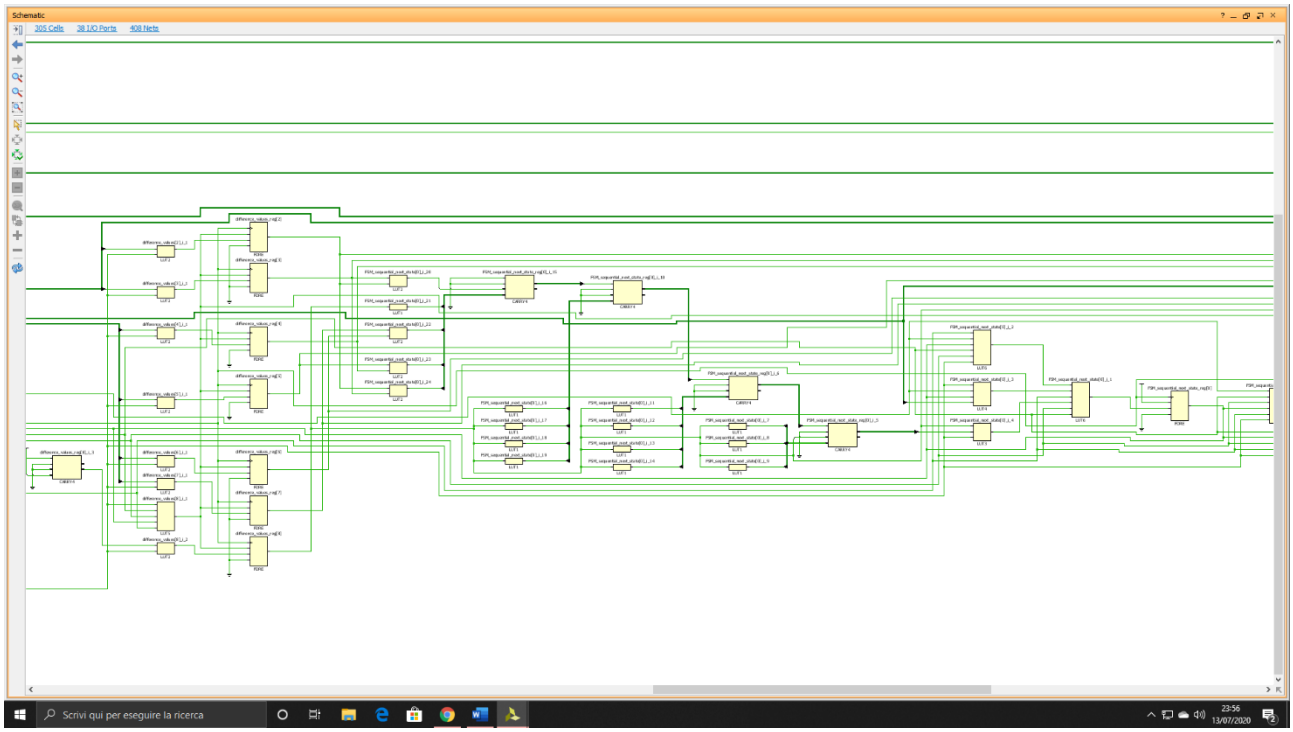
Per una più approfondita lettura del componente appena descritto, allego la RTL schematic, ovviamente l'immagine può risultare difficile da comprendere, ma può aiutare a capire alcuni punti chiave del componente.



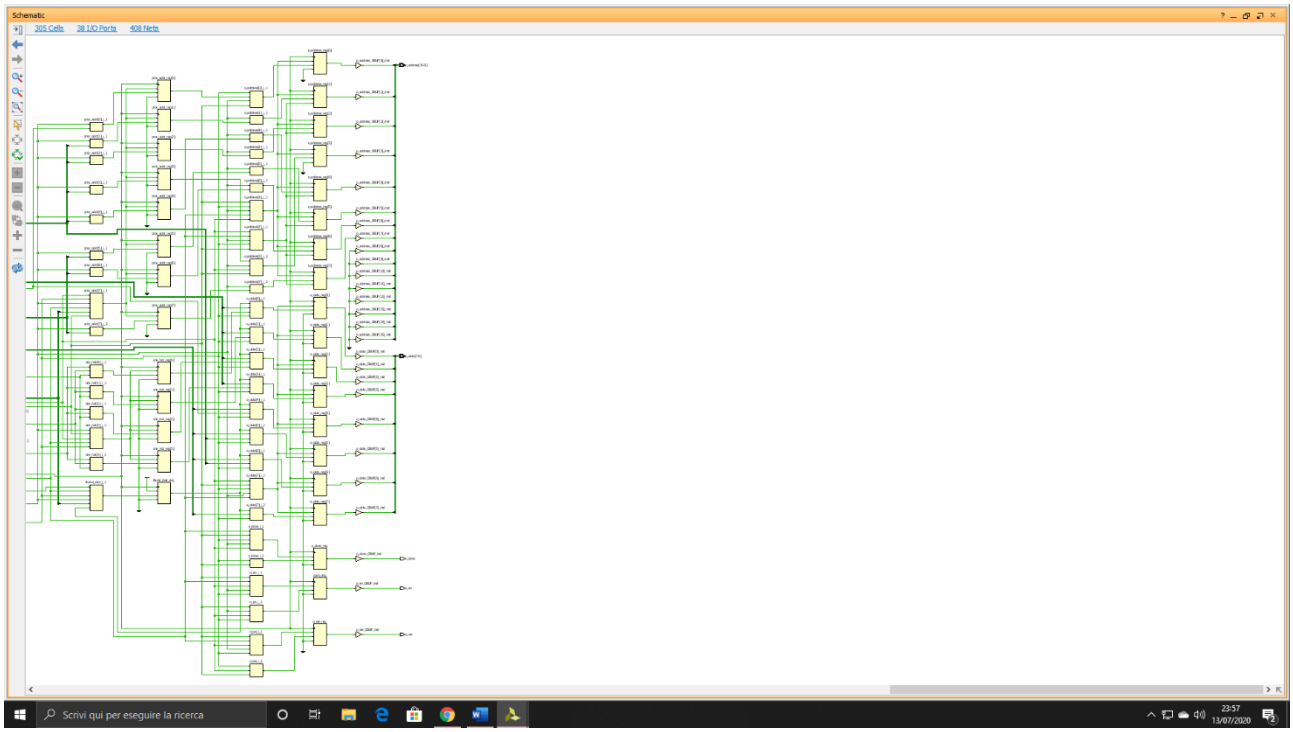
Parte (1/4)



Parte (2/4)



Parte (3/4)



Parte (4/4)

3 Risultati Test

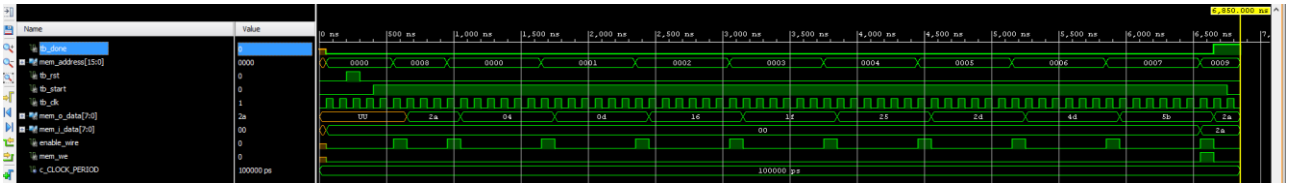
3.1 Test Generici

Come primi test ho eseguito quelli proposti da specifica per verificare un primo corretto funzionamento della macchina, dove tutti i test hanno un periodo di clock di 100 ns, riporto i valori della memoria con relativa wave window:

CASO 1 CON VALORE NON PRESENTE IN NESSUNA WORKING-ZONE

Indirizzo Memoria	Valore	Commento
0	4	Indirizzo Base WZ 0
1	13	Indirizzo Base WZ 1
2	22	Indirizzo Base WZ 2
3	31	Indirizzo Base WZ 3
4	37	Indirizzo Base WZ 4
5	45	Indirizzo Base WZ 5
6	77	Indirizzo Base WZ 6
7	91	Indirizzo Base WZ 7
8	42	ADDR da codificare
9	42	Valore codificato in OUTPUT

WAVE WINDOW



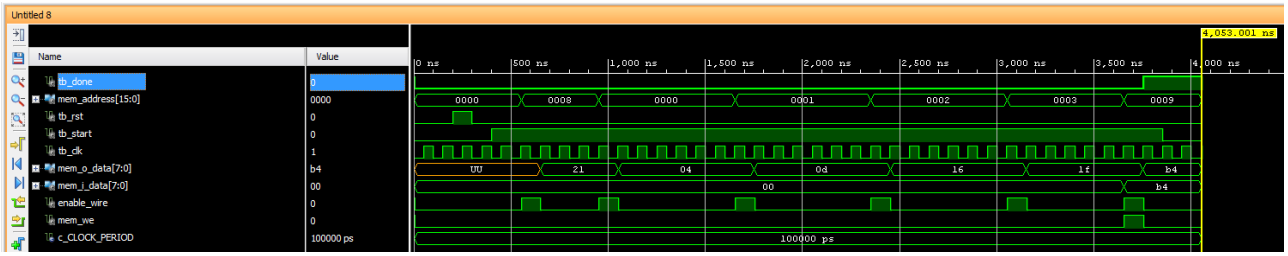
Come si può notare la macchina prima chiede l'indirizzo 8 alla RAM, poi controlla i valori delle WZ e infine non trovando nessuna corrispondenza, scrive nell'indirizzo 9 il valore 2a, che in decimale corrisponde a 42 come da consegna, il tutto in circa 7 us.

CASO 2 CON VALORE PRESENTE IN UNA WORKING-ZONE

Indirizzo Memoria	Valore	Commento
0	4	Indirizzo Base WZ 0
1	13	Indirizzo Base WZ 1
2	22	Indirizzo Base WZ 2
3	31	Indirizzo Base WZ 3
4	37	Indirizzo Base WZ 4
5	45	Indirizzo Base WZ 5
6	77	Indirizzo Base WZ 6

7	91	Indirizzo Base WZ 7
8	33	ADDR da codificare
9	180	Valore codificato in OUTPUT

WAVE WINDOW

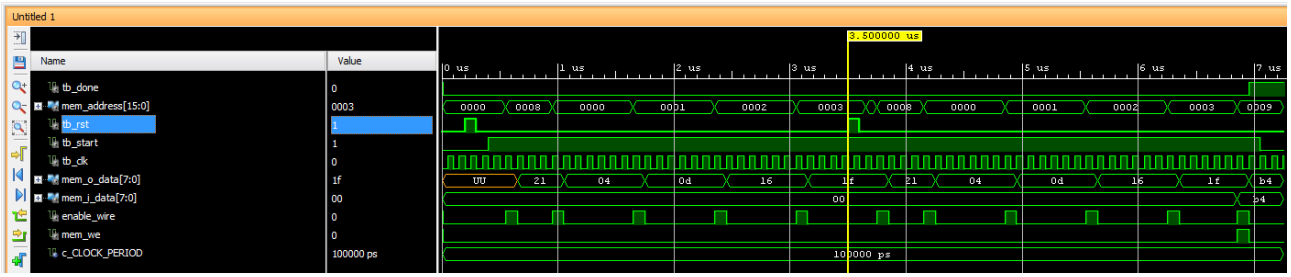


Come si può notare la macchina trova la WZ all'indirizzo 3 e scrive in memoria all'indirizzo 9 b4, ovvero 180, poiché l'indirizzo codificato risulta 1(WZ found) – 011(terzo indirizzo WZ) – 0100(one-hot corrispondente), il processo impiega circa 4 us.

3.2 Test casi critici

Un primo caso particolare è l'attivazione del segnale di reset durante la computazione, come esempio prendiamo quello relativo all'esempio precedente, ovvero il caso 2, riporto sempre la wave window di tale test:

WAVE WINDOW



La FSM si comporta come l'esempio precedente, quando a 3,5 us, la macchina riceve un segnale di reset, e come da specifica ritorna allo stato iniziale per poi riprendere il processo.

Per testare le tempistiche di processo della macchina, ho deciso di verificare i casi peggiore e migliore per capire l'intervallo temporale di funzionamento, ovviamente senza considerare un possibile segnale di reset, evento che aumenterebbe le tempistiche.

Caso Migliore : l'indirizzo da codificare appartiene alla prima WZ, cosi la macchina codificherà al primo ciclo l'indirizzo e terminerà la computazione, tutto questo con tempistiche di circa 1,8 us, sempre con periodo di clock di 100 ns.

Caso Peggiore : l'indirizzo da codificare non appartiene a nessuna WZ, in questo caso la macchina dovrà leggere tutti gli otto valori in memoria prima di poter codificare l'indirizzo,

questo con tempo di esecuzione di circa 6,8 us, sempre con periodo di clock di 100 ns, risultato uguale al test del caso 1 trattato infatti.

Dati i due esempi si evince che l'intervallo di esecuzione, con periodo di clock di 100 ns, si posiziona tra 2 us e 7 us, in cifra arrotondata; tutti i test vengono superati correttamente in *Behavorial*, *Post-Synthesis functional* e *Post Synthesis timing* simulation.

3.3 Ottimizzazioni

Una possibile modifica per maggiore efficienza sarebbe quella di terminare il controllo delle WZ una volta che i valori letti dalla RAM diventano maggiori dell'indirizzo da codificare, ma questa modifica si potrebbe attuare solo nel caso che nella memoria i valori delle WZ siano inserite in maniera crescente, per rendere la FSM più universale ho deciso di non inserire questa modifica, anche se una volta stabilito un possibile protocollo potrebbe rendere molto più efficiente la macchina.