Progetto di Reti Logiche

Stefano Maxenti
Codice Persona 10526141
stefano.maxenti@mail.polimi.it

Ivan Motasov
Codice Persona 10563149
ivan.motasov@mail.polimi.it

Sommario

La specifica della Prova è ispirata al metodo di codifica denominato "Working Zone", il quale permette di verificare se uno specifico indirizzo, trasmesso sul bus degli indirizzi, appartiene a certi intervalli, detti appunto "Working Zone".
Realizzato da Stefano Maxenti e Ivan Motasov come Prova Finale di Reti Logiche A.A 2019-2020, prof Fabio Salice. Linguaggio utilizzato: VHDL

Indice

Specifica

Scopo della Prova Finale di Reti Logiche è implementare un modulo hardware, descritto in VHDL, che permetta una codifica di indirizzi a bassa dissipazione di potenza, come descritto nell'articolo di E. Musoll, T. Lang and J. Cortadella, *"Working-zone encoding for reducing the energy in microprocessor address buses"*, in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 6, no. 4, pp. 568-572, Dec. 1998.
Questa tecnica, applicata al bus degli indirizzi, permette di trasformare un indirizzo quando lo stesso viene trasmesso nel caso appartenga a determinati intervalli (chiamati working zone, in seguito abbreviati in WZ) di dimensioni fissate. Nella versione da implementare, il valore degli indirizzi può variare fra 0 e 127 e la dimensione di ogni WZ è fissata a 4. L'ultima WZ valida può avere indirizzo base uguale a 124. Pur essendo sufficienti 7 bit per memorizzare tali valori, essi vengono memorizzati in celle da 8 bit (il MSB vale 0) con indirizzamento al byte. La memoria sarà organizzata nel seguente modo:

- Cella 0: Indirizzo base WZ0
- Cella 1: Indirizzo base WZ1
- Cella 2: Indirizzo base WZ2
- Cella 3: Indirizzo base WZ3
- Cella 4: Indirizzo base WZ4
- Cella 5: Indirizzo base WZ5
- Cella 6: Indirizzo base WZ6
- Cella 7: Indirizzo base WZ7
- Cella 8: Indirizzo da codificare (abbreviato in addr)
- Cella 9: Risultato

In posizione 8 è presente l'indirizzo che deve essere codificato, mentre in posizione 9 avverrà la scrittura del risultato finale. Quest'ultimo sarà lungo 8 bit, organizzati nel seguente modo:

- **bit n.7 WZ_BIT**, che verrà posto a 1 qualora addr appartenga a una delle WZ;
- **bit n.6-4 WZ_NUM**, ossia la codifica binaria del numero della WZ che contiene addr;
- **bit n.3-0 WZ_OFFSET**, ossia la codifica one-hot della distanza dall'indirizzo base della WZ che contiene addr.

Qualora addr non appartenga a nessuna WZ, il risultato sarà uguale ad addr, ossia il WZ_BIT sarà posto a 0 e i rimanenti 7 bit saranno gli stessi di addr.
Si riporta ora l'interfaccia del dispositivo:

```
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;
```

Per quanto riguarda l'input:

- **i_clk** identifica il clock col quale il componente opera;
- **i_start** identifica il segnale che dà inizio a ogni operazione sul componente e vale 1 fino a quando il dispositivo non termina la computazione;
- **i_rst** identifica il segnale di reset asincrono dopo il quale il componente dovrà tornare allo stato di partenza e rimanere in attesa del segnale di start;
- **i_data** contiene il valore letto da memoria.

Per quanto riguarda gli output:

- **o_en** è il segnale che deve essere posto a 1 in fase sia di lettura sia di scrittura da memoria;
- **o_we** è il segnale che attiva la scrittura su memoria;
- **o_address** è l'indirizzo richiesto in lettura o scrittura in memoria;
- **o_data** indica il dato che deve essere scritto in memoria;
- **o_done** indica la fine dell'esecuzione di ogni operazione sul modulo: vale 1 finchè il segnale di **i_start** non viene posto a 0.

Architettura e scelte progettuali

Considerando il fatto che la logica stessa del progetto non richiede un elevato grado di parallelismo e al fine di evitare problemi di "inferred latches", si è deciso di implementare il dispositivo tramite l'utilizzo di un solo process.
Per garantire estendibilità, è stato usato un approccio parametrico (si veda l'omonima sezione riportata più avanti).
Come da specifiche, è stata considerata la possibilità di un reset asincrono che permetta di ritornare allo stato iniziale, nel quale tutti i valori vengono inizializzati.
Seguono ora una rappresentazione del dispositivo tramite un ibrido FSM-Algoritmo e la descrizione dei suoi stati.

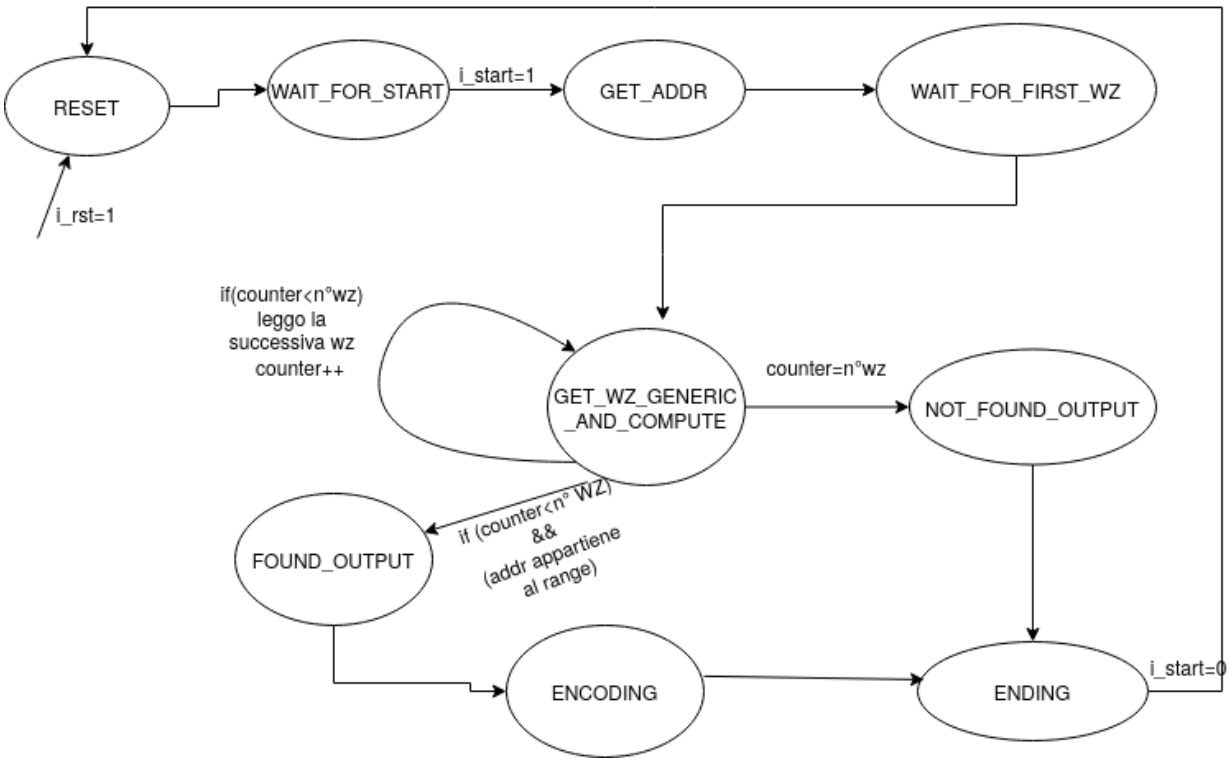

Figura 1: Rappresentazione della macchina tramite ibrido FSM-Algoritmo

Descrizione degli stati della macchina

RESET

Lo stato di **RESET** è la condizione di default da cui il dispositivo parte per effettuare la computazione. Tale stato è anche il punto al quale si ritorna in caso di reset asincrono. I segnali utilizzati nel process vengono sempre inizializzati. In particolar modo, o_address viene impostato a 8, in maniera tale che la prima lettura sarà quella dell'indirizzo da codificare.
Conclusa l'inizializzazione, si passa direttamente allo stato **WAIT_FOR_START**.

WAIT_FOR_START

Il dispositivo rimane in idle fino a quando il segnale i_start proveniente dal test bench non assume valore 1. In questo caso, si passa allo stato **GET_ADDR**.

GET_ADDR

Si effettua la lettura dell'indirizzo da codificare che si trova in posizione 8 nel segnale addr. Si è deciso di richiedere l'indirizzo prima delle WZ con il fine di ottenere maggiore velocità nella computazione. Al segnale o_address viene impostato il valore 0, che verrà assunto nel ciclo di clock successivo con lo stato **WAIT_FOR_FIRST_WZ**.

WAIT_FOR_FIRST_WZ

Si tratta di uno stato cuscinetto che garantisce la completa memorizzazione di addr per evitare problemi di timing. Al segnale o_address viene impostato il valore 1.
È importante notare che il valore della WZ0 non è ancora presente in mem_i_data: lo sarà alla fine del ciclo di clock corrente
Si passa ora allo stato **GET_WZ_GENERIC_AND_COMPUTE**.

GET_WZ_GENERIC_AND_COMPUTE

Questo stato è il cuore dell'elaborazione del dispositivo.
Per garantire la massima velocità nella lettura dei dati della RAM, evitando il ritardo intrinseco che intercorre fra la richiesta di o_address e l'effettiva disponibilità del valore in i_data, abbiamo deciso di anticipare la richiesta di o_address di un ciclo di clock. In questo modo, è possibile leggere un valore diverso in i_data ad ogni ciclo di clock e al contempo preparare o_address al valore corrispondente a due posizioni successive.
Una volta entrato in questo stato, il dispositivo si comporta come segue:

- primo ciclo di clock: il dispositivo legge WZ0, la RAM richiede il valore WZ1 puntato da o_address (settato a 1 nel ciclo di clock precedente, stato **WAIT_FOR_FIRST_WZ**), si incrementa il valore o_address (che diventa 2), counter diventa 1 se addr non appartiene alla WZ0;
- secondo ciclo di clock: il dispositivo legge WZ1, la RAM richiede il valore WZ2 puntato da o_address (settato a 2 nel ciclo di clock precedente), si incrementa il valore o_address (che diventa 3), counter diventa 2 se addr non appartiene alla WZ1;
- i successivi cicli fino al settimo si ripetono con lo stesso meccanismo;
- ottavo ciclo di clock: il dispositivo legge WZ7, la RAM richiede il valore puntato da o_address (settato a 8 nel ciclo di clock precedente), che lo stesso addr, si incrementa il valore o_address (che diventa 9), counter diventa 8 se addr non appartiene alla WZ7.

Questa soluzione permette di leggere le 8 caselle della RAM in 8 cicli di clock. Si legge (senza memorizzarlo) l'indirizzo base della WZ e lo si confronta con addr. Se esso risulta appartenente alla WZ, ci si sposta nello stato **FOUND_OUTPUT**. Inoltre, o_address viene forzato al valore attuale della cella contenente la WZ individuata, in modo da bloccare la finestra di scorrimento della lettura delle WZ.
Se l'indirizzo non appartiene alla WZ considerata, si incrementa di 1 il valore in o_address (attraverso un segnale di supporto next_o_address) e il valore del contatore (counter) delle WZ. Lo stato rimane uguale.
Quando tutte le WZ sono state analizzate e nessuna corrispondenza è stata trovata (ossia nel caso counter assuma valore 8), si può terminare la computazione spostandosi nello stato **NOT_FOUND_OUTPUT**.

FOUND_OUTPUT

Come avviene per lo stato **WAIT_FOR_FIRST_WZ**, si tratta di uno stato cuscinetto che permette di non avere ritardi e/o errori nell'assegnazione dei valori. Ci si sposta ora nello stato ENCODING, dove i_data coinciderà con il valore della WZ a cui addr appartiene.

ENCODING

La codifica viene ottenuta mediante la somma aritmetica dei seguenti valori:

- 128, ovvero la rappresentazione binaria di un numero avente solo un bit pari a 1 in posizione 7 e i rimanenti posti a 0 (da specifica è **1**-000-0000);
- 16 · counter, ossia un numero avente obbligatoriamente tutti i bit posti a 0 tranne il 6, il 5 e il 4 (ad esempio, se la WZ corrispondente è la seconda - ossia WZ1, il valore di questo vettore è 0-**001**-0000);
- lo shift a sinistra di n posizioni della rappresentazione binaria di 1, dove n coincide con la differenza fra l'indirizzo da codificare e l'indirizzo base della WZ (ad esempio, se l'offset è uguale a 2, si shifta il valore 0001 di 2 posizioni verso sinistra, ottenendo 0-000-**0100**).

Questo meccanismo di codifica può sembrare complesso; esso permette tuttavia una facile estendibilità, aspetto di cui si discuterà più avanti. L'indirizzo così codificato viene scritto direttamente in o_data, che da specifiche è uno *std_logic_vector(7 downto 0)*; o_we viene portato a 1 per attivare il processo di scrittura; o_address viene forzato al valore della cella in cui effettuare la scrittura (nel caso specifico, la numero 9).
Si passa automaticamente allo stato **ENDING**.

NOT_FOUND_OUTPUT

Come da specifiche, nel caso addr non sia contenuto in alcuna WZ, l'indirizzo codificato non sarà altro che lo stesso indirizzo letto senza alcuna variazione. o_we viene portato a 1 per attivare il processo di scrittura; o_address viene forzato al valore della cella in cui effettuare la scrittura (nel caso specifico, la numero 9).
Si passa automaticamente allo stato **ENDING**.

ENDING

La scrittura viene portata a termine; o_done viene alzato a 1. Si rimane in attesa che i_start torni a 0 come da specifiche per poi riportarsi nello stato di **RESET**, in attesa di iniziare una nuova computazione.

Performance e Test comportamentali

Il nostro dispositivo è stato studiato per essere adeguatamente veloce nella computazione. Si consideri come possibile soluzione un'implementazione naive che memorizzi le 8 WZ e l'indirizzo da codificare, iteri fra i valori memorizzati e successivamente calcoli la codifica. Considerando il ritardo che intercorre fra la richiesta di un indirizzo alla RAM e la sua effettiva disponibilità, il caso pessimo necessiterebbe di 25 cicli di clock solo per ottenere l'appartenenza a una WZ. Inoltre sarebbe utilizzata una notevole quantità di memoria per la memorizzazione delle WZ e non sarebbe immediatamente estendibile.
L'approccio da noi adottato permette invece di memorizzare solamente l'indirizzo da codificare (la WZ corrente non viene infatti salvata); inoltre, il meccanismo di lettura che abbiamo descritto nello stato **GET_WZ_GENERIC_AND_COMPUTE** riduce il numero di cicli di clock; per finire, la richiesta di lettura dell'indirizzo da codificare come primo valore (e non come ottavo) permette di eseguire i confronti in tempo reale.
Per semplicità, consideriamo la salita e la discesa di i_start come inizio e fine della computazione, approssimando per eccesso il mezzo-ciclo di clock finale. Si possono presentare 9 casi: addr che appartiene alla WZ0, alla WZ1, ..., alla WZ7 o a nessuna. Mediamente sono necessari 12 cicli di clock per terminare una computazione.
Le seguenti waveforms sono state ottenute attraverso simulazioni in *post-synthesis functional* (risultata correttamente funzionante anche con periodo di clock di 2 ns).

Caso migliore

Il caso migliore possibile per l'implementazione proposta si ha quando addr appartiene alla prima WZ: tale casistica viene eseguita in 8 cicli di clock. Inoltre abbiamo testato un possibile *corner case*: WZ0 impostata a 0 e addr impostato a 0 (codifica finale: 1-000-0001). Abbiamo evidenziato con una linea verticale la lettura della WZ0 (Vivado non mostra un nuovo valore 0 in quanto il valore di i_data non è variato, essendo WZ0 e addr uguali).

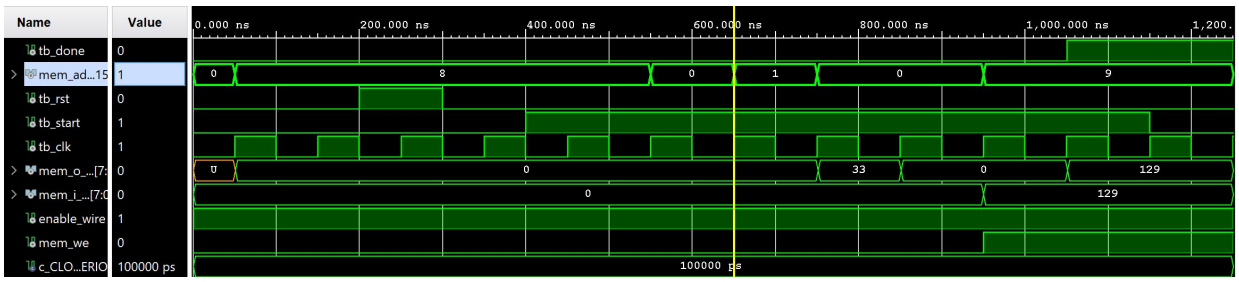

Figura 2: Best case scenario

Caso peggiore

Il caso peggiore possibile per l'implementazione proposta si ha quando addr appartiene all'ultima WZ: in tal caso sono necessari 15 cicli di clock per poter giungere alla fine della computazione. Inoltre abbiamo testato un possibile *corner case*: WZ7 impostata a 124 e addr impostato a 127 (codifica finale: 1-111-1000).

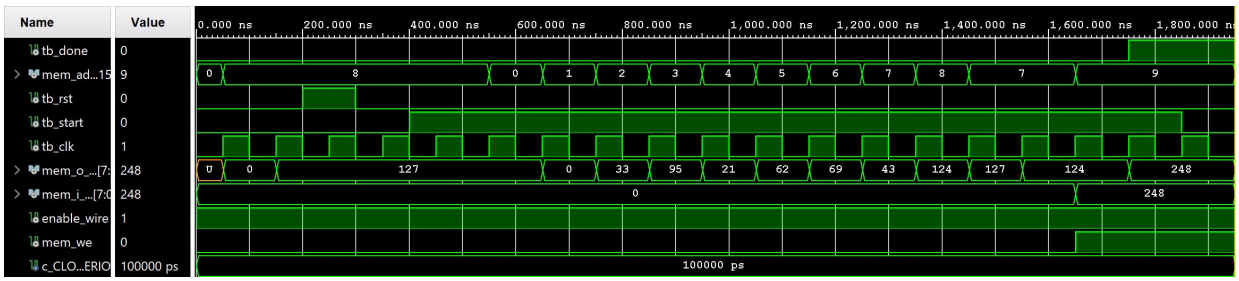

Figura 3: Worst case scenario

Not found

Tale caso è uguale al caso peggiore, poiché è necessario leggere tutte le WZ possibili. Sono necessari 15 cicli di clock per potere giungere alla fine della computazione.

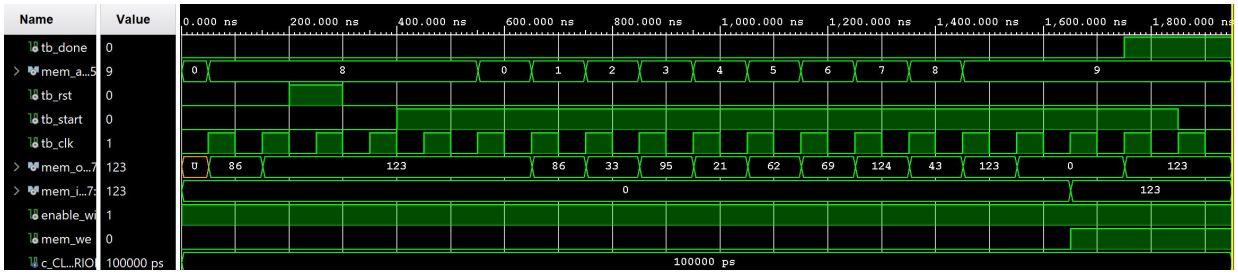

Figura 4: Not found case

Doppio start

Una volta terminata la computazione, è possibile che il segnale i_start torni alto: in questo caso, l'indirizzo da codificare è cambiato. Non avendo nessun dato in memoria, si riparte dallo stato **GET_ADDR**.

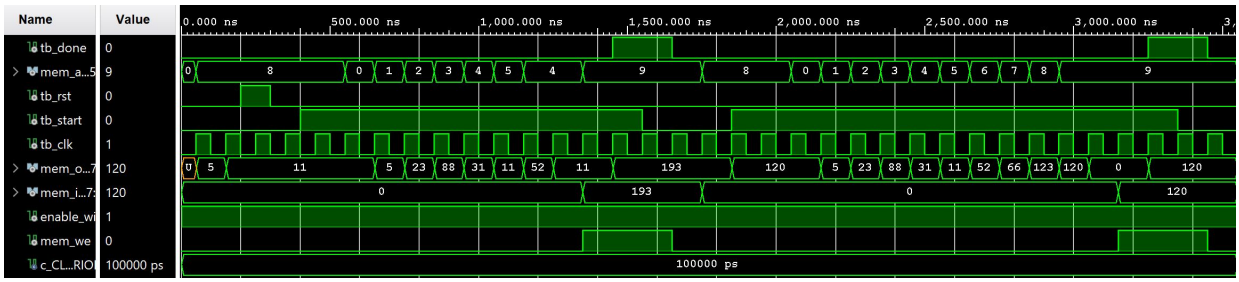

Figura 5: Doppio start

Reset-stress test

Si è deciso di inserire più reset all'interno di un caso di test per verificare la corretta inizializzazione di tutti i valori indipendentemente dallo stato in cui il dispositivo si trova; in particolare, in questo test bench, è stato inviato inizialmente un segnale di reset sincrono, seguito da un altro (asincrono) di breve durata in un istante casuale all'interno della computazione e infine un altro segnale (sempre asincrono) di reset di breve durata quando o_done viene portato a 1. In ognuno di questi casi il dispositivo ritorna nello stato iniziale **RESET**.

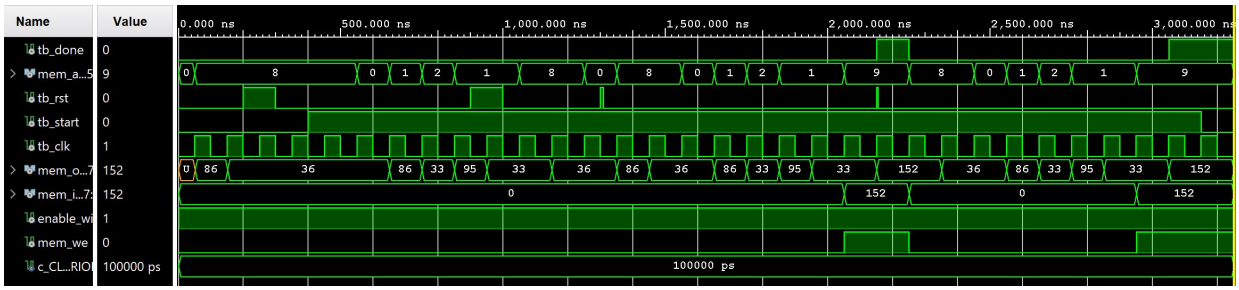

Figura 6: Reset-stress test

Test casuali

Oltre a quelli appena citati, il dispositivo è stato inoltre sottoposto a centinaia di migliaia di altri test casuali (comprendenti casi a singolo start, a doppio start e con la presenza di reset vari), al fine di valutare le performance e la correttezza dei risultati, che si sono dimostrati corretti senza produrre alcun warning.

Risultati della sintesi

Il software Vivado (versione 2019.2, installato su Microsoft Windows 10 Pro) ha correttamente sintetizzato il nostro dispositivo usando 59 LUT e 58 FF. La nostra FSM è stata correttamente riconosciuta e i suoi nove stati sono stati memorizzati tramite codifica binaria naturale. Dalle opzioni del sintetizzatore sarebbe stato possibile impiegare codifiche differenti (ad esempio, one-hot o Gray).
La sezione *Schematic* informa che sono state utilizzate 180 celle, 38 porte I/O e 252 nets.
L'unico warning segnalato è collegato alla mancanza di un file di constraint per la scrittura. Abbiamo tuttavia inserito un constraint file per il clock (fissato come da specifiche a 100 ns), ottenendo i seguenti valori di timing:

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	94,714 ns	Worst Hold Slack (WHS):	0,145 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	144	Total Number of Endpoints:	144	Total Number of Endpoints:	59

All user specified timing constraints are met.

Figura 7: Risultati della timing

Sebbene non richiesto nella consegna iniziale, il dispositivo è risultato correttamente implementabile e, oltre alle simulazioni richieste, supera con profitto anche le seguenti:

- *post-synthesis timing* (fino a 6 ns)
- *post-implementation functional* (fino a 2 ns)
- *post-implementation timing* (fino a 10 ns)

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								59	58	0.0	0	0
✓ impl_1	constrs_1	route_design Complete!	93.724	0.000	0.193	0.000	0.000	0.132	0	57	58	0.0	0	0

Synthesis		**Implementation**	**Summary** | Route Status
Status:	✓ Complete	Status:	✓ Complete
Messages:	! 1 warning	Messages:	No errors or warnings
Part:	xc7a200tfbg484-1	Part:	xc7a200tfbg484-1
Strategy:	Vivado Synthesis Defaults	Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Synthesis Default Reports	Report Strategy:	Vivado Implementation Default Reports
Incremental synthesis:	None	Incremental implementation:	None

Figura 8: Risultati di sintesi e implementazione

Estendibilità

All'interno del codice si è scelto di utilizzare costanti (definite usando il tipo *constant*) al posto di valori hard-codati, in maniera da rendere il dispositivo facilmente modificabile ed estendibile nel caso di indirizzi con lunghezza diversa da 8 bit, con più WZ e con offset maggiori di 3. Si riporta in seguito una spiegazione dettagliata del significato delle costanti presenti nel codice:

- **NUMBER_WZ**: numero totale delle WZ (da specifica: 8);
- **SIZE_WZ**: numero di indirizzi che possono appartenere a una WZ (da specifica: 4);
- **ADDR_IN_RAM_POSITION**: indirizzo all'interno della RAM dove viene memorizzato il valore da codificare (da specifica: 8);
- **WZ_FACTOR_MULTIPLY**: fattore moltiplicativo che permette di calcolare il numero della WZ a cui l'indirizzo appartiene (da specifica: 16);
- **SIZE_ADDRESSES**: dimensione degli indirizzi (da specifica: 8 - includiamo anche il bit più significativo che è impostato a 0);
- **WZ_BIT_INT**: codifica decimale della rappresentazione binaria lunga SIZE_ADDRESSES e avente il bit più significativo posto a 1 (da specifica: 128, ossia **1**000 0000).

Per testare l'effettiva estendibilità del componente, si è realizzato un test bench apposito, avente indirizzi lunghi 10 bit, con 16 working zone e con offset fino a 5. La codifica del risultato ha la seguente struttura:

- **WZ_BIT**: 1 bit che indica l'appartenenza;
- **WZ_NUM**: 4 bit che indicano il numero di WZ (da 0000 a 1111);
- **WZ_OFFSET**: 5 bit di offset (da 0 a 5, codificati in one-hot).

Con opportune modifiche al valore numerico delle costanti (e modificando i campi i_data e o_data all'interno dell'interfaccia dell'entity, operazione automatizzabile dichiarando le costanti in un package prima di definire l'entity stessa), si è ottenuto un valore coerente con il meccanismo dell'encoding e delle working zone.
A titolo d'esempio, ipotizziamo le seguenti WZ: 68, 123, 99, 270, 278, 255, 141, 318, 350, 490, 26, 13, 0, 306, 397, 50 con indirizzo da codificare pari a 493 (appartenente alla WZ9 con offset pari a 4). Il dispositivo restituisce correttamente il valore 808 (1-1001-01000, dove 1 indica l'appartenenza, 1001 indica la WZ9, 01000 è la codifica di 4 in one-hot), che viene memorizzato in posizione NUMBER_WZ+1 (17).

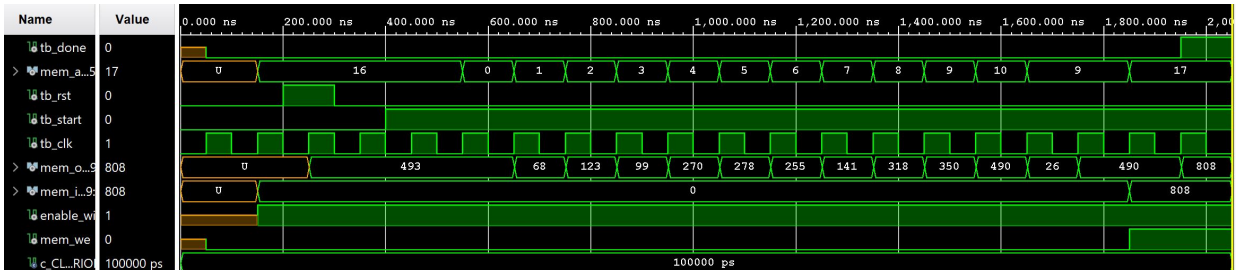

Figura 9: Esempio di estendibilità con addr appartenente ad una wz

Se invece l'indirizzo da codificare fosse 500 (non appartenente ad alcuna WZ), il valore codificato rimarrebbe uguale a quello dato in ingresso.

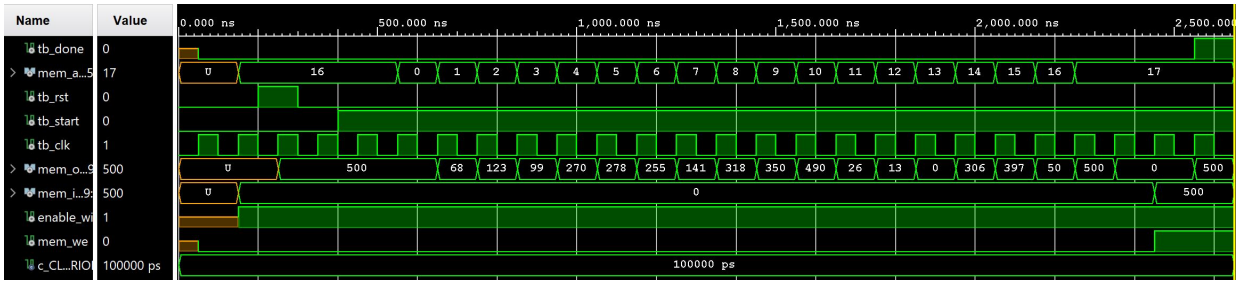

Figura 10: Esempio di estendibilità con addr non appartenente ad una wz

Conclusioni

Nel caso di specifiche più stringenti o di implementazioni reali, sono possibili i seguenti miglioramenti:

- Si è assunto che le WZ cambino ad ogni computazione, in maniera tale che al termine di ognuna di esse si effettui una nuova lettura completa. In una vera implementazione hardware, è possibile che le WZ tendano a rimanere stabili (come nel caso di un doppio start). In questo caso salvare le WZ risulterebbe essere un ottimo compromesso per ridurre il tempo complessivo di lettura, a scapito però di un maggior consumo di memoria;
- il numero di stati potrebbe essere ridotto accorpando funzioni all'interno di macrostati; non si è ritenuto di seguire questo approccio in quanto si è preferito spezzare il funzionamento dell'algoritmo in diversi cicli di clock in modo da permettere il corretto funzionamento del dispositivo anche in presenza di un periodo di clock notevolmente inferiore a 100 ns.