POLITECNICO
MILANO 1863

Progetto di Reti Logiche 2019-20

Francesco Fulco Gonzales

Alberto Latino

Introduzione

Il metodo di codifica Working-Zone, utilizzato nella trasmissione degli indirizzi di memoria attraverso il Bus, si prefigge l'obiettivo di ridurre la potenza dissipata dal circuito. Questo tipo di encoding, definisce delle aree di memoria dette appunto Working-Zone, ognuna delle quali è identificata da un indirizzo base. Ogni indirizzo appartenente ad una working zone viene inoltre distinto dall'offset rispetto all'indirizzo base della WZ di appartenenza.
In particolare, la macchina analizza l'appartenenza o meno di un indirizzo dato in input (ADDR) ad un' eventuale WZ, e restituisce la sua codifica.
Si può quindi verificare uno di due casi:

- ADDR non appartiene a nessuna WZ
  viene trasmesso così com'è, lasciando il bit iniziale (WZ_BIT) posto a 0, che segnala la codifica di non appartenenza ad alcuna WZ.

- ADDR appartiene ad una WZ
  viene codificato concatenando un bit iniziale (WZ_BIT) posto ad 1 con altri due campi:
  1. Il numero della working-zone a cui appartiene(WZ_NUM), codificato in binario naturale.
  2. L'offset rispetto all'indirizzo base della working-zone (WZ_OFFSET codificato in one-hot.

  Dato quindi in input un ADDR appartenente a una WZ, verrà trasmesso WZ_BIT=1 concatenato con WZ_NUM e WZ_OFFSET (WZ_BIT & WZ_NUM & WZ_OFFSET).

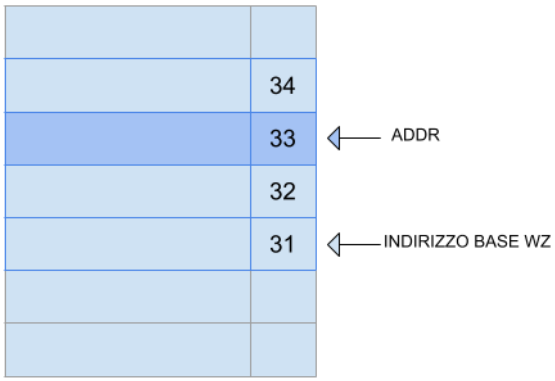
In questa codifica, il numero di bit dell'indirizzo da codificare è 7 e vi sono 8 diverse WZ, ognuna delle quali contiene 4 indirizzi. Quindi, l'indirizzo codificato, risulterà avere una lunghezza di 8 bit così divisi:

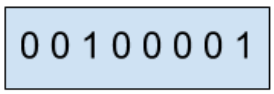
- 1 bit - WZ_BIT, per indicare l'appartenenza o meno di ADDR ad una WZ

- 3 bit - WZ_NUM, per indicare eventualmente a quale delle 8 WZ appartiene ADDR

- 4 bit - WZ_OFFSET, per indicare eventualmente lo spiazzamento rispetto alla WZ

Riportiamo un esempio in cui ADDR, l'indirizzo da codificare, indirizza la cella 33 della memoria e la seconda delle 8 WZ (che supponiamo già definite), ha come indirizzo base la cella 31.
Possiamo notare che ADDR si trova all'interno della WZ numero 2, dal momento che essa comprende sé stessa e le tre celle di memoria successive, quindi l'indirizzo 33 di ADDR è la terza cella della suddetta WZ.

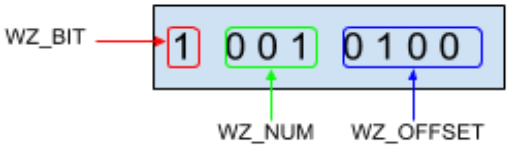




Memoria **ADDR da codificare**

La codifica pertanto risulta:

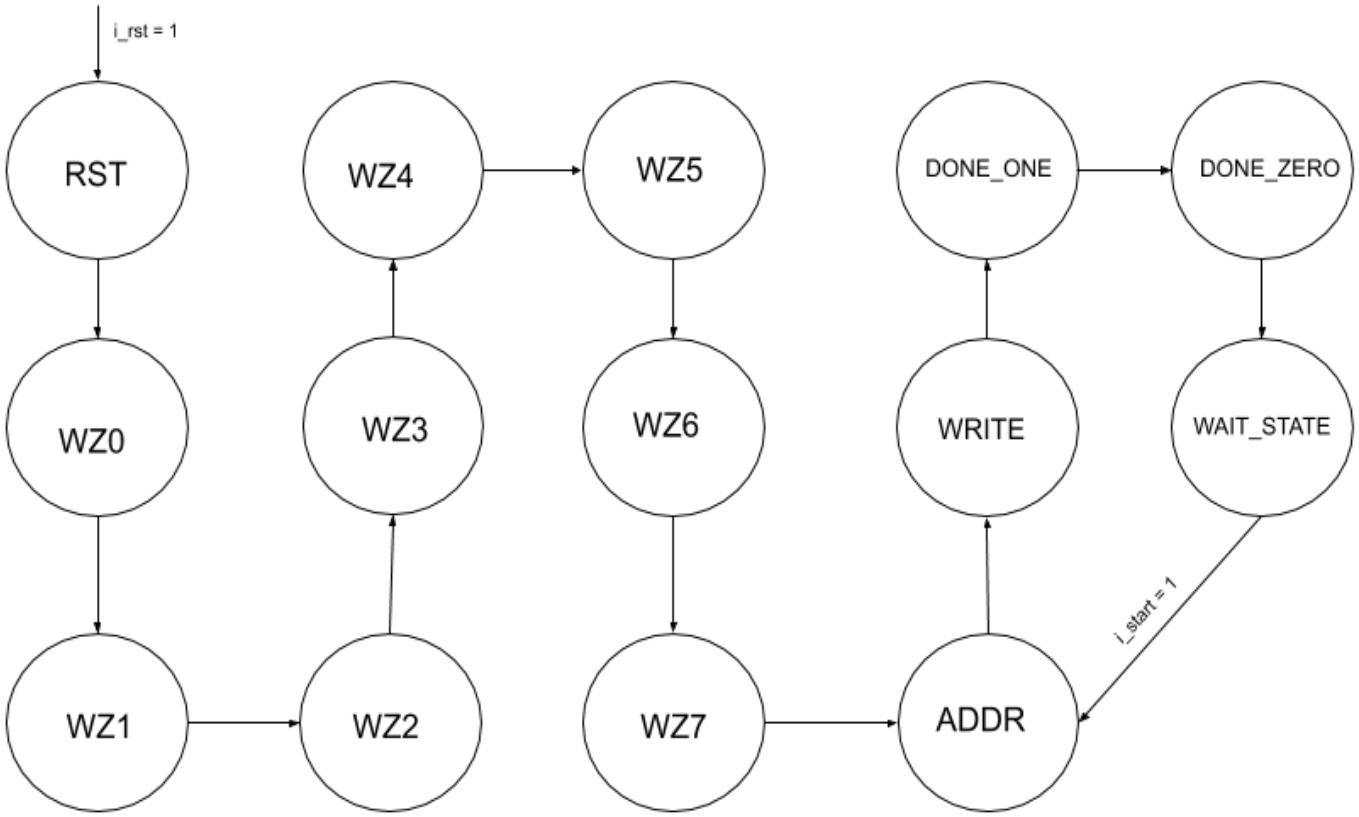
- WZ_BIT = 1, in quanto ADDR appartiene a una WZ.
- WZ_NUM = 001, in quanto la WZ a cui ADDR appartiene è la numero 2.
- WZ_OFFSET = 0100, in quanto ADDR è la terza parola di memoria della WZ.



Architettura

La nostra macchina si propone di eseguire la codifica prediligendo l'efficienza temporale, in quanto ci si aspetta che nel normale funzionamento di una memoria, le WZ non cambino spesso, e quindi più indirizzi vengano codificati su una stessa mappatura. A tale scopo, abbiamo deciso di salvare le WZ solo alla prima lettura di un ADDR, e salvarle nuovamente solo in caso di una nuova mappatura della memoria, segnalata dal segnale di reset.

Prima di presentare dettagliatamente la macchina da noi progettata, riportiamo qui un suo grafo, al fine di illustrare in maniera chiara i diversi stadi di funzionamento.



Nella FSM avviene una transizione di stato ad ogni ciclo di clock, quindi per eseguire un'intera codifica, da RST a DONE_ONE si impiegano 11 cicli di clock. Da WAIT_STATE si torna in ADDR nel caso di nuova computazione con diverso indirizzo da codificare in input, ma WZ invariate rispetto alla computazione precedente. In questo caso la codifica impiega solo 4 cicli di clock da ADDR a WAIT_STATE.

Segnaliamo inoltre, che ogni qual volta la macchina riceve il segnale di reset, questa si riporta nello stato RST. Nella figura non sono state riportate le transizioni di reset al fine di ottenere una maggiore chiarezza illustrativa.
Riportiamo di seguito una tabella descrittiva degli stati e la loro funzione, riportando anche i valori assunti dai segnali principali.

Nome Stato	Descrizione
RST	Stato di Reset, inizializzati i segnali di output o_done = '0', o_en = '1', o_we = '0' e o_address viene inizializzato con '00000000' in modo che la memoria restituisca l'indirizzo della WZ0 al ciclo di clock successivo.
WZ0 - WZ6	In ognuno di questi stati, otteniamo gli indirizzi delle varie WZ. Inoltre ogni stato assegna ad o_address l'indirizzo di memoria di cui si vuole ottenere il contenuto, che si otterrà allo stato successivo dopo un ciclo di clock. Una volta ottenuto l'indirizzo di una WZ, la macchina lo salva in una memoria interna.
WZ7	Questo stato, come i precedenti, ottiene e successivamente salva l'indirizzo dell'ultima WZ. A differenza degli altri però, assegna ad o_address l'indirizzo in cui si trova il dato ADDR da codificare, in modo da ottenerlo e salvarlo nell'omonimo stato successivo.
ADDR	Ottiene da memoria l'ADDR da codificare e lo salva. Da notare che questo stato è utilizzato per salvare ADDR anche nella fase di funzionamento in cui viene ricevuto un segnale di start in WAIT_STATE senza che la macchina sia stata resettata.
WRITE	Questo stato ha il effettua l'operazione di riconoscimento dell'appartenenza o meno di ADDR ad una delle WZ salvate nella memoria interna e ne calcola la codifica opportuna, come riportato nell'introduzione. Viene quindi caricato l'indirizzo codificato nella cella 9 della memoria esterna, e viene attivato il segnale di write (o_we = '1').
DONE_ONE	In questo stato, poiché vi giungiamo dopo esattamente 1 ciclo di clock, siamo sicuri che la memoria adesso contenga il risultato della computazione. Di conseguenza, possiamo porre o_done a '1' per segnalare che la codifica è effettivamente terminata e il risultato è stato scritto in memoria.
DONE_ZERO	In questo stato, la macchina è in attesa che il segnale di i_start sia portato a '0'.Se ciò avviene, anche o_done viene portato a '0', altrimenti o_done rimane a '1' in attesa di tale evento.
WAIT_STATE	In questo stato, la macchina ha completato la computazione ed è in attesa di un nuovo segnale di start. Se invece riceve un reset, torna in RST. Se riceve i_start a '1' la macchina si porta in ADDR, per eseguire la computazione con un nuovo ADDR, e le stesse WZ. Da notare che qui viene caricato in o_address, l'indirizzo 8 in modo da salvare il nuovo indirizzo da codificare una volta giunti in ADDR.

Risultati sperimentali

Elementi sintetizzati

Di seguito si riporta una tabella dei componenti RTL sintetizzati, in prevalenza adoperati nello stato WRITE per la codifica di ADDR.

	Numero Input	Bit	Numero elementi
Adder	2	8	24
Mux	14	14	1
	2	14	14
	14	8	1
	14	4	1
	14	1	2

Area occupata

Dalla “report utilization” possiamo estrarre le informazioni riguardo l’area occupata dal design sintetizzato.

Risorsa	Utilizzo	Disponibilità	Utilizzo in %
Look Up Table	284	134600	0.21%
Flip Flop	14	269200	<0.01%
Latch	72	269200	0.03%

Come si può notare, le risorse utilizzate risultano inferiori alle risorse disponibili di diversi ordini di grandezza. Anche per questa ragione non abbiamo ritenuto proficuo ottimizzare l’area occupata, concentrandoci piuttosto sull’ottimizzazione temporale.

Warning di sintesi

La sintetizzazione ha generato alcuni tipi di warning, che abbiamo consapevolmente ignorato. Di seguito sono riportati i tre tipi principali:

1. Segnali non inseriti nella sensitivity list
   Alcuni segnali non sono stati inclusi nella sensitivity list in quanto ridondanti.
2. Porta fissa a 0 (port driven by constant 0)
   Questo warning è causato dall'inutilizzo dei bit 15 - 4 dell'o_address, che restano fissi a 0, come da specifica.
3. Latch inferiti
   Abbiamo consapevolmente lasciato che vengano creati dei latch per salvare i valori delle WZ nella memoria interna.

Note aggiuntive

Per questo progetto abbiamo utilizzato "Vivado 2019.2 WebPACK Edition", e la FPGA xc7a200tfbg484-1.

Simulazioni

Abbiamo sottoposto il componente sia ai testbench ufficiali sia ad ulteriori testbench da noi creati al fine di verificare il corretto funzionamento del componente in condizioni critiche. Inoltre sono stati effettuati dei test con input e indirizzi di memoria casuali (ordine dei milioni di test generati automaticamente).

Di seguito i casi critici testati:

1. ADDR

1.1. ADDR appartente a WZ (testbench ufficiale)
Verifica che il componente riconosca un indirizzo che appartiene a una delle WZ date, ne calcola la codifica e la scrive in memoria.

1.2. ADDR non appartenente a WZ (testbench ufficiale).
Verifica che il componente riconosca un indirizzo che non appartiene a nessuna delle WZ date, riscrivendolo quindi in memoria immutato.

1.3. ADDR in WZ minima e massima
L'indirizzo da codificare appartiene alla WZ con indirizzo minimo/massimo.

2. Working Zone

2.1. WZ minima e massima (WZ = 0, 124)

L’indirizzo di una WZ è 0, l’indirizzo di un’altra WZ è 124, ovvero il massimo ammesso.

2.2. WZ adiacenti

Tutte le WZ sono adiacenti, ovvero l’indirizzo successivo all’ultimo indirizzo di ciascuna WZ è l’indirizzo base della prossima WZ.

3. Input multipli

3.1. Multi start

Dopo una prima codifica viene dato un nuovo segnale di start con un nuovo indirizzo ADDR da codificare e senza cambiare le WZ.

3.2. Multi reset

Dopo una prima codifica viene dato un nuovo segnale di reset con un nuovo indirizzo ADDR da codificare e delle nuove WZ.

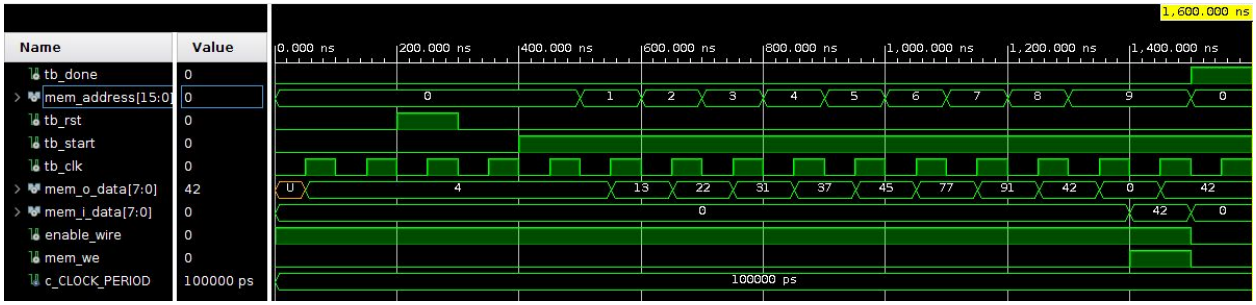
3.3. Reset durante esecuzione

Durante l’esecuzione, prima della fine della codifica, viene dato un reset, asincrono rispetto al clock.

Di seguito sono riportati i risultati grafici dell’esecuzione dei testbench ufficiali.

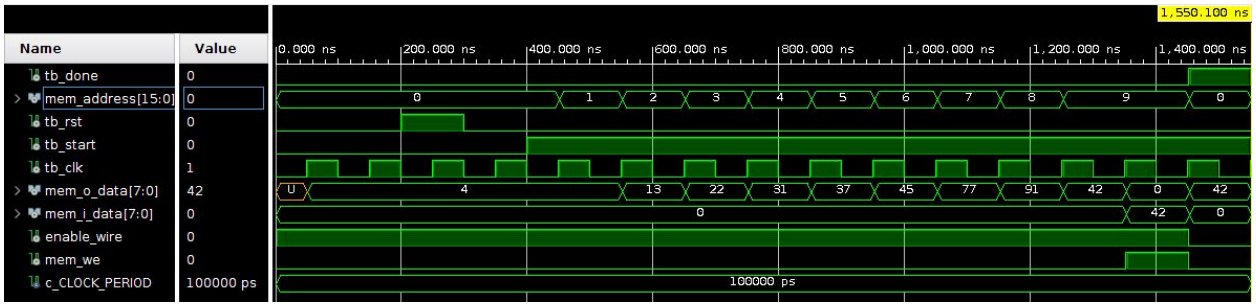
No Working Zone Testbench

Behavioral Simulation



- **Tempo di Esecuzione : 1600 ns**

Post-Synthesis Functional Simulation

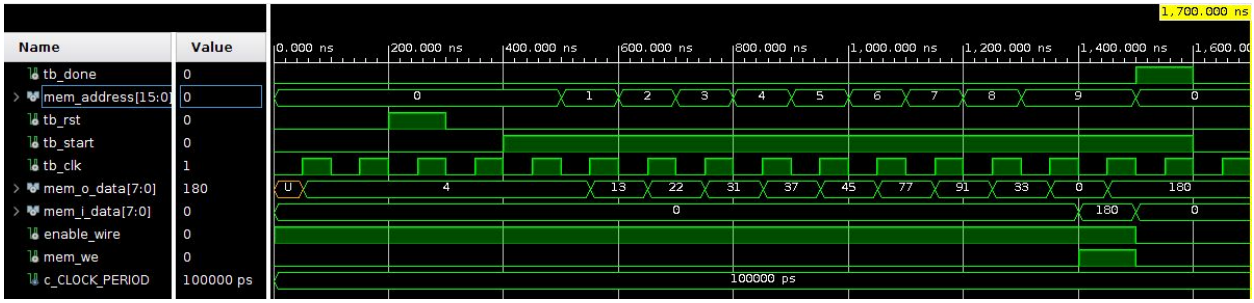


- **Tempo di Esecuzione : 1500 ns**

In questo test, dopo aver letto e salvato tutte le WZ, viene letto l’indirizzo da codificare ADDR che viene lasciato immutato in quanto non appartenente ad alcuna working zone e scritto direttamente in memoria. Il bit più significativo WZ_BIT è già posto a 0, in quanto da specifica gli indirizzi in input sono di 7 bit.

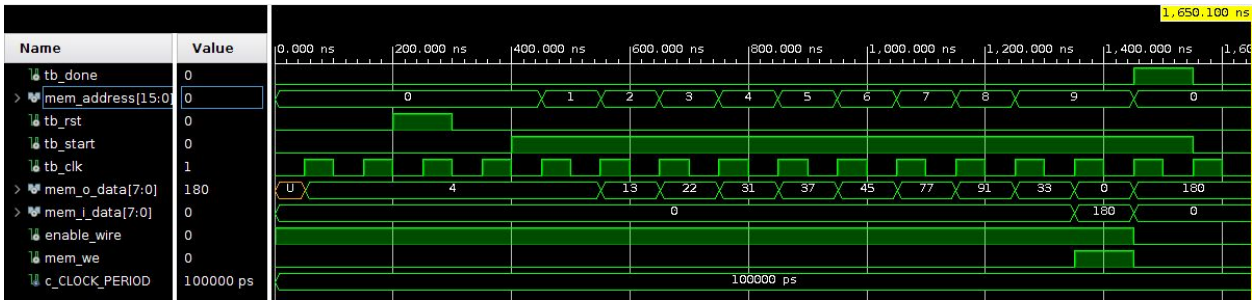
In Working Zone Testbench

Behavioral Simulation



- **Tempo di Esecuzione : 1700 ns**

Post-Synthesis Functional Simulation



- **Tempo di Esecuzione : 1650 ns**

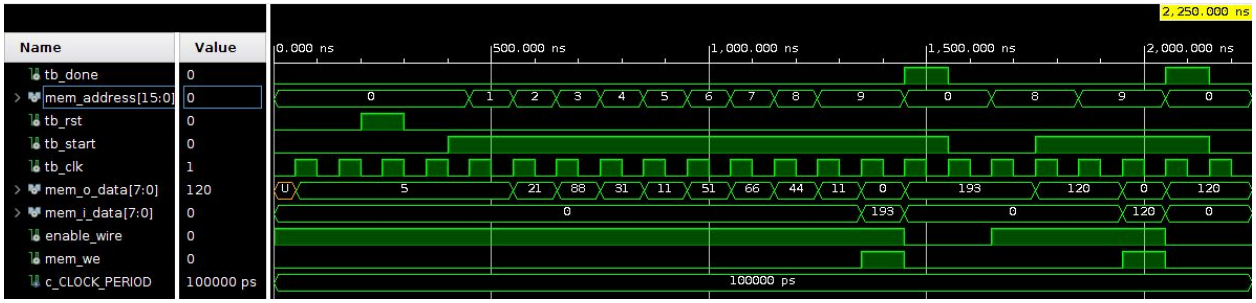
In questo caso di test invece l'indirizzo ADDR in input è pari a 33 e appartiene quindi alla WZ3 che include gli indirizzi da 31 a 34.
Viene quindi codificato nello stato WRITE per poi essere scritto in memoria.

Come si osserva dal Waveform Viewer il componente impiega al più 12 cicli di clock dalla ricezione del segnale di start fino alla segnalazione di fine con o_done = 1, con differenze minimali tra l'esecuzione in comportamentale e post sintesi.

Tuttavia, come discusso precedentemente, il componente è stato ottimizzato per avere prestazioni più efficienti nel caso in cui venga dato un nuovo indirizzo di codifica ADDR mantenendo le WZ precedenti.
Appare dunque opportuna l'inclusione di un test che verifichi tale caso.

Multistart Testbench

`Post-Synthesis Simulation`



- **Tempo di Esecuzione : 2250 ns**

In questo testbench vengono effettuate le codifiche di due ADDR diversi.
Dopo una prima computazione con conseguente caricamento delle WZ, viene posto start a 1 dopo la ricezione dell'output o_done basso come da specifica. Viene quindi salvato il nuovo ADDR e effettuata la sua codifica e conseguente scrittura in memoria.

Questo test mette in luce il notevole guadagno temporale della scelta progettuale di ottimizzazione temporale: si può infatti notare che tra il secondo start e l'o_done successivo intercorrono solo 4 cicli di clock, rispetto agli 11 necessari per la prima codifica, impiegando così un tempo di codifica totale di soli 2250 ns invece di 3000 ns. Chiaramente il guadagno aumenterà con l'aumentare di codifiche effettuate con le stesse WZ.
Questo test dimostra dunque che l'obiettivo delle scelte progettuali a cui ambivamo è stato raggiunto.

Conclusioni

Il componente realizzato rispetta tutte le caratteristiche della specifica ed è stato sottoposto a un numero elevato di test per verificarne il corretto funzionamento.
Inoltre è stato ottimizzato al fine di riutilizzare le WZ e in modo da evitarne il caricamento in memoria ad ogni lettura di ADDR, permettendo così di effettuare la codifica di un nuovo ADDR che sfrutta le WZ salvate precedentemente.
Questa ottimizzazione velocizza il componente di un fattore 2.75 rispetto all'alternativa di caricare le WZ ad ogni lettura di un nuovo ADDR, infatti la codifica viene fatta in soli 4 cicli di clock invece degli 11 necessari alla codifica con caricamento delle WZ.