Prova Finale Progetto Reti Logiche

**Anno Accademico 2019-2020**

Giusti Leonardo (Codice Persona: 10633778)

La Manna Giuseppe (Codice Persona: 10608466)

**INDICE**

1. **Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2**
2. **Architettura . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3**

**2.1. *Schema progettuale***

***2.1.1. Interfaccia del componente***

***2.1.2 Segnali utilizzati***

***2.1.3 Macchina a Stati Finiti***

**2.2. *Scelte progettuali***

1. **Testing del componente. . . . . . . . . . . . . . . . . . . . . . . . . . . 7**

**3.1 *Risultati dei test non-funzionali***

***3.1.1. Test del reset***

***3.1.2 Test dello start***

***3.1.3 Test misti***

**3.2 *Risultati dei test funzionali***

***3.2.1. Best Case Scenario***

***3.2.2 Worst Case Scenario***

***3.2.3 Confronto***

**3.3. *Altri test effettuati***

1. **Conclusioni. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13**

***4.1 Risultati della sintesi***

***4.1.1 Warnings in post synthesis***

***4.2 Sommario***

**1. Introduzione**

**L’obbiettivo del progetto è creare il metodo di codifica Working Zone pensato per trasformare il valore di un indirizzo trasmesso se questo appartiene a certi intervalli denominati *working-zone.***

**Il codice si occuperà di leggere l’indirizzo da codificare (ADDR) e gli 8 indirizzi base delle Working Zone, aventi dimensione di 4 indirizzi incluso quello base, confrontarli e produrre l’indirizzo opportunamente codificato.**

**Se l’indirizzo da trasmettere non appartiene a nessuna Working Zone, ai bit di indirizzamento verrà concatenato un bit addizionale WZ\_BIT posto a 0:**

**WZ(ADDR) = WZ\_BIT & ADDR**

Se invece l’indirizzo da trasmettere appartiene ad una Working Zone, verrà trasmesso WZ\_BIT posto a 1, concatenato a WZ\_NUM, rappresentante la codifica in binario del numero della Working Zone al quale l’indirizzo appartiene, e WZ\_OFFSET, che rappresenta lo spiazzamento in codifica one-hot:

**WZ(ADDR) = WZ\_BIT & WZ\_NUM & WZ\_OFFSET**

Immagine che contiene disegnando

Descrizione generata automaticamenteNel seguente esempio viene esposto approfonditamente il metodo di codifica:

Notiamo che non vi è alcun ordine tra gli address di base delle Working Zones quindi ottimizzazioni del componente basate su questo non sono possibili.

Mostriamo di seguito la codifica di due address:

* *Address1* da codificare: 68 (01000100)
* *Address2* da codificare: 15 (00001111)

Immagine che contiene screenshot

Descrizione generata automaticamente

**L’*Address 1* non appartiene ad alcuna Working Zone quindi avrà WZ\_BIT = 0 ed i successivi bit invariati.**

**L’*Address 2* appartiene alla Working Zone 1 con WZ\_OFFSET = 1. WZ\_NUM corrisponde alla codifica binaria di 2 su 3 bit (001) invece il WZ\_OFFSET sarà la codifica one hot di 1 su 4 bit (0010).**

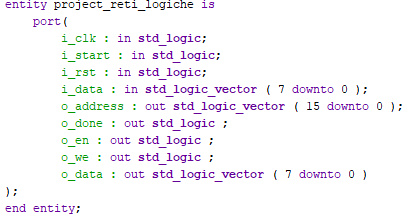
**Il valore codificato avrà come output (1 – 001 – 0010), che corrisponde al valore 146.**

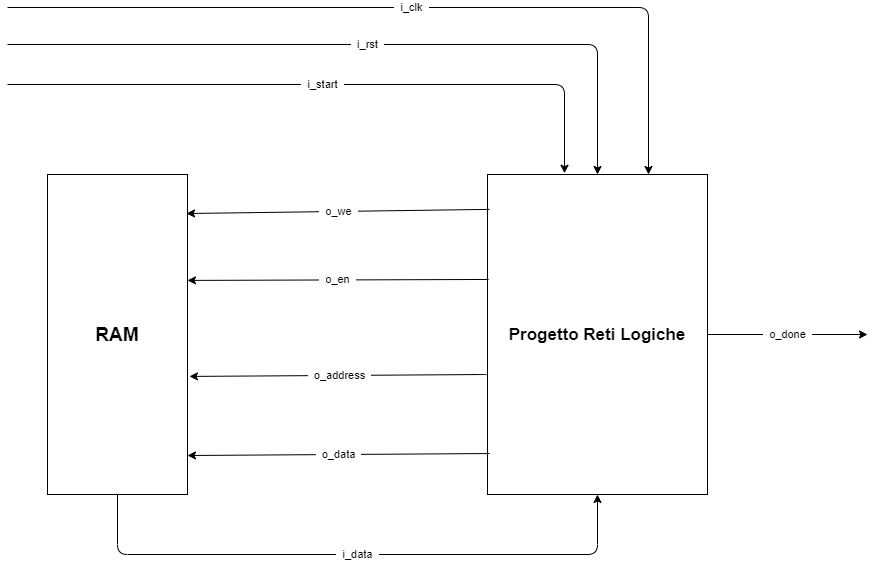
**2. Architettura**

**2.1 Schema progettuale**

**2.1.1 Interfaccia del componente**

**Il componente è collegato ad una memoria RAM tramite la seguente interfaccia:**

****



Schema dell'interfaccia del componente con la memoria RAM

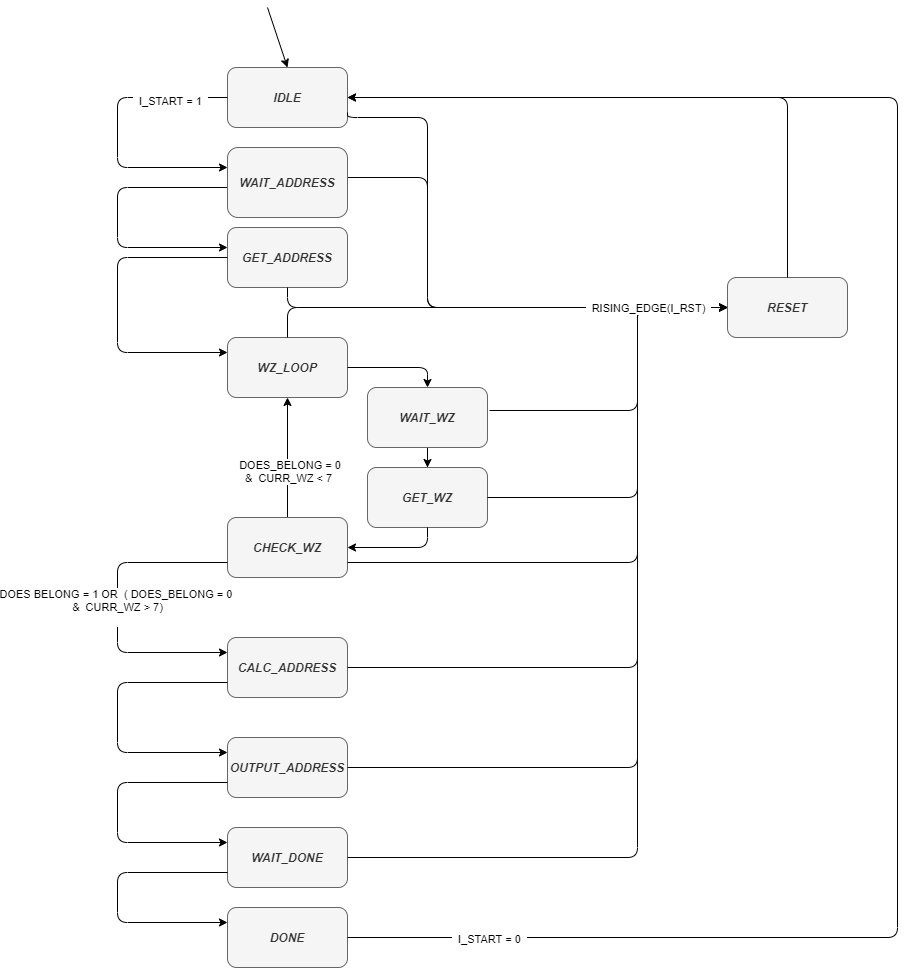
**2.1.2 Segnali utilizzati**

**La seguente tabella riassuntiva rappresenta i segnali introdotti per la realizzazione del modulo gestiti all’interno degli stati della FSM. Per ogni segnale presente in tabella ne è stato inserito uno aggiuntivo (*signal\_name\_next*) nel quale viene memorizzato il valore del segnale al ciclo di clock successivo.**

|  |  |
| --- | --- |
| **Nome Segnale** | **Contenuto** |
| ***current\_state*** | **Memorizza lo stato corrente della FSM** |
| ***current\_wz*** | **Memorizza l’indirizzo base della Working Zone corrente** |
| ***read\_address*** | Memorizza il valore dell’indirizzo da codificare ricevuto dalla RAM |
| ***does\_belong*** | **Specifica se l’indirizzo da codificare appartiene o meno alla Working Zone corrente** |
| ***wz\_num*** | **Numero intero che indica quale tra le 8 Working Zone sia caricata in quel momento** |
| ***coded\_address*** | Contiene l’indirizzo finale da restituire |
| ***need\_rst*** | Booleano che indica se il componente deve essere inizializzato |

**2.1.3 Macchina a Stati Finiti**

Il funzionamento alla base del componente è stato implementato attraverso una FSM che usa *i\_start* come segnale di avvio della codifica e *i\_rst* come segnale per l’inizializzazione della macchina. Il passaggio tra gli stati successivi sarà approfondito in seguito alla figura dello schema.



Schema rappresentante la macchina a stati finiti utilizzata per la codifica dell’indirizzo

* **idle: s**tato iniziale in cui la macchina attende un segnale di *i\_start* per iniziare la sua computazione: una volta ricevuto la macchina richiede l'*address* da codificare alla RAM;
* **wait\_address:** stato in cui attendo che la RAM presenti in *i\_data* l'*address* da codificare;
* **wz\_loop:** stato in cui richiedo alla RAM l'address base della Working Zone corrente, inizialmente carico WZ 0;
* **get\_wz:** stato in cui salvo l'address base della Working Zone corrente presentatomi dalla RAM su *i\_data;*
* **check\_wz:** stato in cui verifico se l'*address* da codificare appartiene alla Working Zone corrente o meno: se appartiene mi preparo a codificarlo, in caso contrario passo alla Working Zone successiva; se sono all'ultima Working Zone allora l'*address* non appartiene ad alcuna di queste e mi preparo a codificarlo di conseguenza;
* **calc\_address:** stato in cui calcolo l'*address* tenendo in considerazione il numero della Working Zone di appartenenza e l'offset dell'*address* da codificare;
* **output\_address:** stato in cui richiedo alla RAM di poter scrivere l'*address* codificato;
* **wait\_done:** stato in cui attendo che l’*address* codificato sia stato effettivamente scritto in memoria;
* **done:** stato di completamento in cui attendo che il segnale di *i\_start* scenda a 0 per tornare in Idle e poter continuare a codificare l'*address* successivo;
* **reset:** stato in cui azzero il valore di ogni segnale del componente in seguito ad una richiesta di *i\_rst*, per tornare poi allo stato di Idle e aspettare la codifica successiva;

**2.2 Scelte progettuali**

Per il design del componente si è deciso di usare un modello *behavioural* con tre processi, al fine di semplificare la gestione di ogni singola operazione :

* **reset\_change** : processo sensibile al segnale di *i\_rst;* è incaricato di rilevare ogni istanza di reset che arriva in qualsiasi momento al sistema;
* **state\_change** : processo sensibile al segnale di *i\_clk*; è incaricato di aggiornare i valori dei registri usati dal componente ad ogni ciclo di clock; in presenza di una richiesta di reset si occupa di portare la FSM allo stato di reset;
* **lambda** : processo sensibile ai segnali del componente; si occupa di gestire i 12 stati della FSM e contiene le operazioni che vengono eseguite al fine di codificare correttamente un *address*;

Come accennato nel reset\_change, la rilevazione del segnale di reset è asincrona mentre la gestione del reset avviene in modo sincrono attraverso lo stato reset.

**Per il design del componente, è stata intrapresa una decisione riguardante il caricamento e il confronto degli address base delle Working Zone con l*’address* da codificare. È stato scelto di caricare l’address base ed effettuare le operazioni di confronto una Working Zone alla volta riutilizzando precedenti segnali e costanti, invece di caricare tutte le Working Zones all’inizio della computazione e di doverle ricaricare in seguito ad ogni reset.**

**Nello sviluppo di questo progetto, il nostro intento è stato la gestione della memoria. Questo approccio permette di caricare soltanto le Working Zones necessarie e quindi un minor utilizzo di segnali e costanti per la memorizzazione di indirizzi a discapito di una prestazione temporale non ottimizzata.**

**Nella computazione di un address appartenente alla prima WZ (*best case*) sarà caricata solamente quest’ultima e la FSM arriverà direttamente allo stato destinato al calcolo dell’address da ritornare, favorendo così una conversione veloce. Al contrario, nel caso di un address che non appartiene ad alcuna WZ (*worst case*) sarà necessario caricare e confrontare l*’address* da codificare con l’address base di ogni WZ producendo una conversione più lenta.**

**Questi due casi sono stati approfonditi nel paragrafo dei test funzionali.**

**3. Testing del componente**

**I seguenti test sono stati effettuati con un ciclo di clock di 100 ns.**

**3.1 Risultati dei test funzionali**

**In questo paragrafo presentiamo i seguenti test che sono stati effettuati per verificare il corretto funzionamento del modulo in presenza di:**

* **reset**
* **start**
* **alternanza tra reset e start**

**I test si avvalgono della presenza di RAM multiple che permettono di cambiare gli address base delle Working Zone e degli address da codificare al fine di verificare il corretto funzionamento della memoria durante il run-time del componente :**

**3.1.1 Test del reset**

**Tra i test riguardanti i reset abbiamo scelto di mostrare in particolare quelli che comprendono multipli reset in sequenza durante una esecuzione della codifica coprendo più scenari di attuazione del reset possibili.**

Immagine che contiene elettronico, telefono, cellulare

Descrizione generata automaticamente

il contenuto delle RAM nel test di reset multipli.

**Nella prima immagine si può vedere una spike di *i\_rst* a 300ns che precede la lettura dell’indirizzo da codificare della RAM 1. A seguire, un segnale di reset durante il falling edge del segnale *i\_clk* durante la computazione relativa alla seconda RAM 2.**

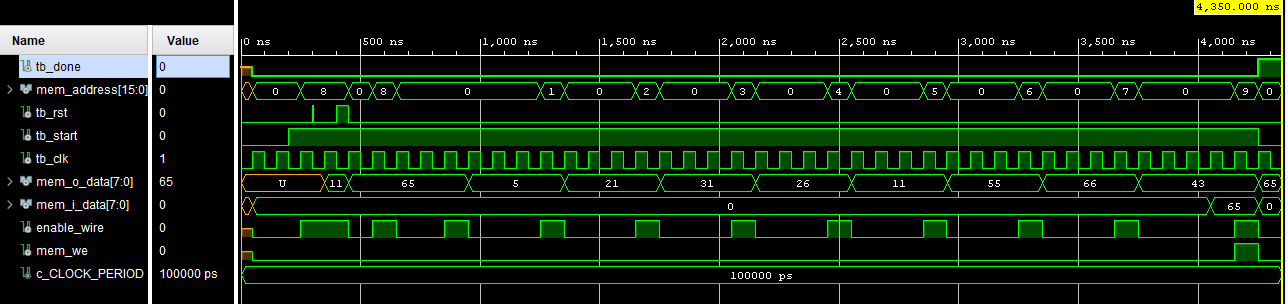
****

Figura 1: ***rappresenta il comportamento del modulo in seguito a spike e al segnale di reset sul falling edge del clock nel test relativo ai reset multipli in una simulazione behavioral.***

**Nello stesso test è stato verificato il corretto funzionamento del segnale di reset durante il rising edge del segnale *i\_start* relativo alla computazione dell’indirizzo in RAM 3 a 4,450 ns. Il test prosegue in seguito con l’esecuzione della codifica dell’indirizzo contenuto in RAM 2.**

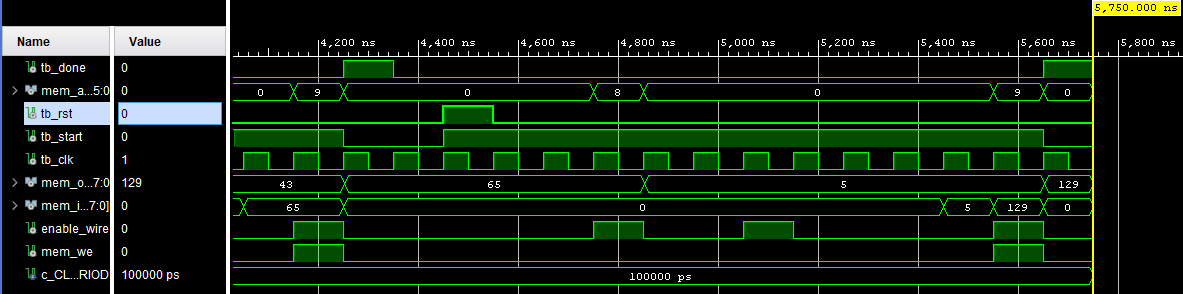
****

Figura 2: ***rappresenta il comportamento del modulo in seguito al segnale di reset sul rising edge del clock e del segnale start in behavioral del test relativo ai reset multipli.***

Immagine che contiene orologio

Descrizione generata automaticamente

Figura 3: risultato finale della codifica

**3.1.2 Test dello start**

**Tra i test sui segnali di start abbiamo scelto di mostrare in particolare quelli che comprendono start multipli durante il rising edge, il falling edge del clock e sul segnale di clock posto a 0.**

**Per ogni istanza di conversione è possibile notare come gli indirizzi base delle Working Zones delle 3 RAM restino invariati ed è effettuata una verifica della correttezza dell’elaborazione per ogni indirizzo.**

Immagine che contiene elettronico, telefono, cellulare

Descrizione generata automaticamente

il contenuto delle RAM nel test di start multipli.

**Nella prima immagine si può vedere a 300ns una spike di *i\_rst* ed un rising edge del segnale di start e a seguire la normale conversione dell’indirizzo presente in RAM[8].**

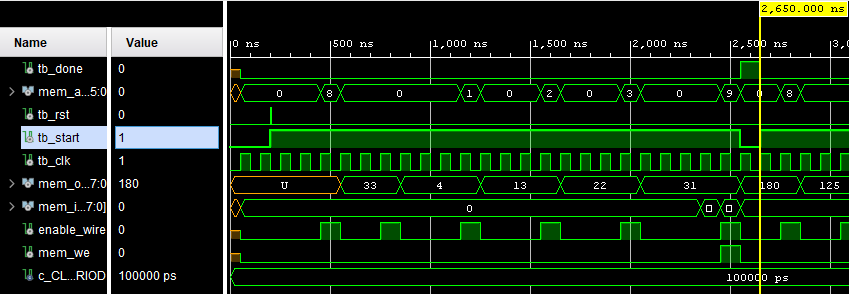
****

Figura 3: rappresenta il comportamento del modulo in seguito ad uno spike del segnale di reset e del rising edge di start sul falling edge del clock nel test relativo agli start multipli in una simulazione behavioral.

Nella seconda immagine si può vedere a 2,500ns simultaneamente il rising edge del segnale di start e di clock a cui segue la normale elaborazione dell’indirizzo contenuto in RAM 2[8].

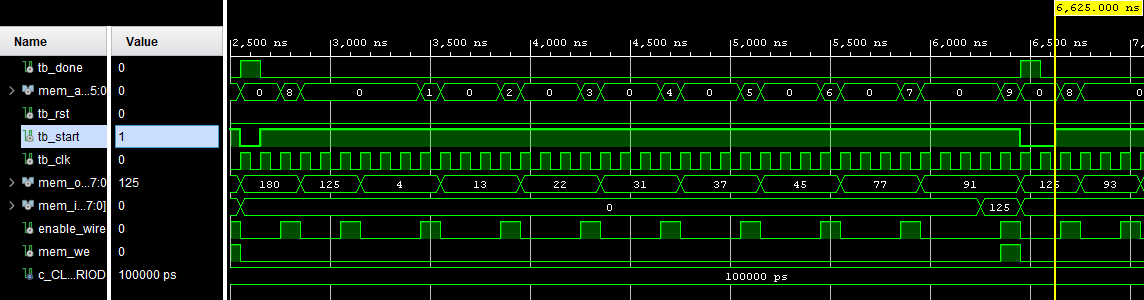
****

Figura 4: rappresenta il comportamento del modulo al rising edge di start su un rising edge del clock in seguito ad un’elaborazione completata nel test relativo agli start multipli in una simulazione behavioral.

**Infine, nella terza immagine è possibile osservare la rilevazione di un segnale di start durante un livello di clock basso a cui segue una normale elaborazione dell’indirizzo contenuto in RAM 3[8].**

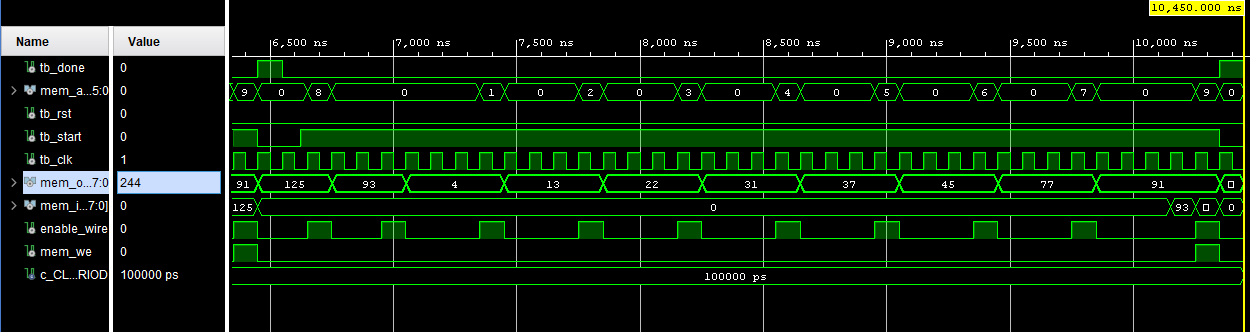
****

Figura 5: rappresenta il comportamento del modulo al rising edge di start durante clock=0 in seguito ad un’elaborazione completata nel test relativo agli start multipli in una simulazione behavioral.

**3.1.3 Test misti**

**Per questo caso di test abbiamo alternato esecuzioni portate a termine con interruzioni per reset al fine di testare meglio il funzionamento del componente.**

Immagine che contiene elettronico, telefono, cellulare

Descrizione generata automaticamente

il contenuto delle RAM nel test misto.

In questo test alterniamo l’uso di due memorie RAM 1 e RAM 2 e vediamo come in seguito al reset del componente dopo il falling edge di done si passi alla seconda elaborazione su una memoria diversa che prosegue poi cambiando l’address da codificare nell’ultima conversione ( RAM 3).

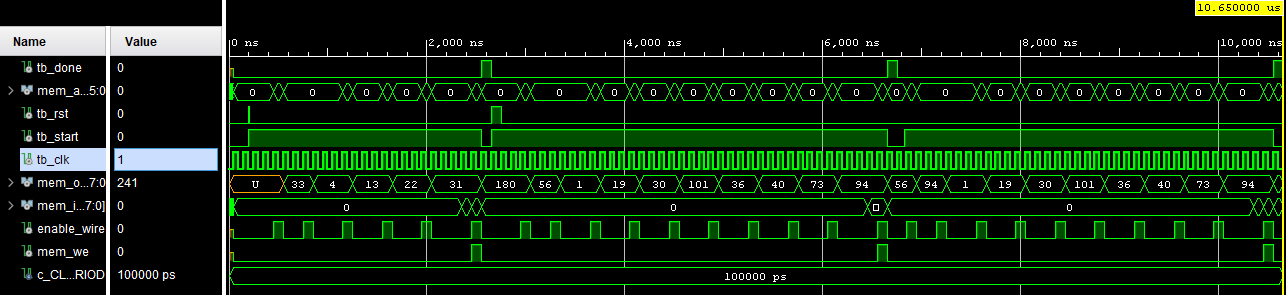
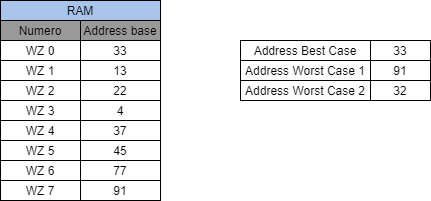
****

Figura 6: rappresenta il comportamento del modulo in una simulazione behavioral del test misto.

**3.2 Risultati dei test non funzionali**

**Come precedentemente accennato, in questo paragrafo sono presi in esame tre casi particolari per la nostra scelta implementativa. Il primo test si evidenzia la computazione di un *address* appartenente alla prima WZ, mentre nel secondo viene testato il caso di non appartenenza ad alcuna WZ.**

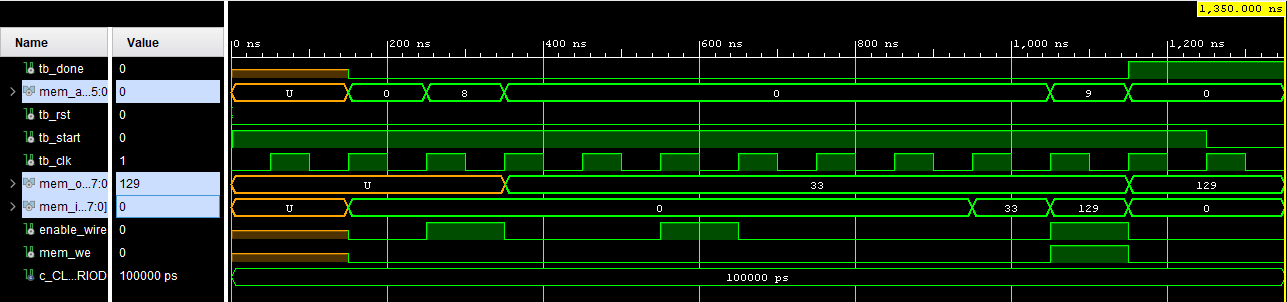
**In seguito, vengono mostrati i risultati dei test precedentemente citati in *Behavioral*.**



Contenuto della RAM e degli address da codificare

**3.2.1 Best Case Scenario**

**Nel test del *best case,* il modulo avrà la necessità di caricare soltanto il contenuto dell’address da codificare e confrontarlo direttamente con l’address base della prima Working Zone.**



***Figura 7: mostra il risultato del test in Behavioral del best case test.***

**3.2.2 Worst Case Scenario**

Nelle seguenti immagini viene rappresentata la computazione della codifica nel *worst case*, distinta in due casi :

* appartenenza all’ultima Working Zone
* non appartenenza ad alcuna Working Zone

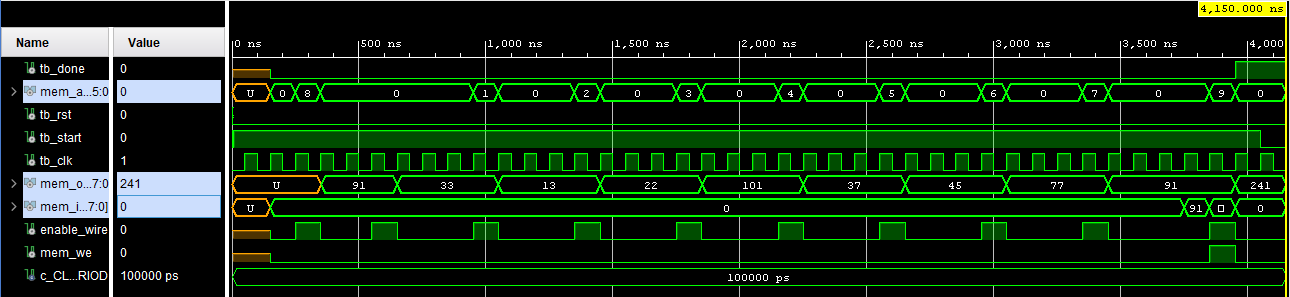
****

Figura 8: waveform dell'elaborazione di un address appartenente all'ultima Working Zone.

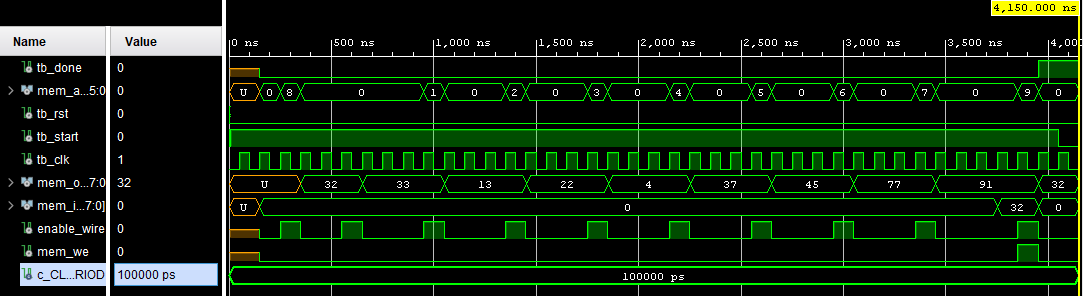
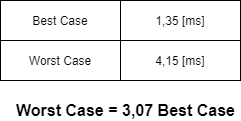


Figura 9: waveform dell’elaborazione di un indirizzo non appartenente ad alcuna Working Zone.

Dai test effettuati si verifica dunque che i due casi siano equivalenti, ed in entrambi si giunge all’ultima operazione di confronto allo stesso ciclo avendo come unica differenza da notare il valore del segnale *does\_belong\_next*. Si nota come tutti gli address base delle Working Zone vengono caricati in *mem\_o\_data* e confrontati volta con l’*address* da codificare.

**3.2.2 Confronto**

**La scelta progettuale quindi incide significativamente sulle prestazioni temporali del componente nei due casi.**



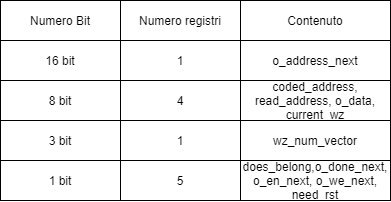
**3.3 Altri test effettuati**

**Sono stati effettuati entrambi i test allegati dal professore ed è stato usato un generatore di casi di test randomici di diversa lunghezza fino a diversi milioni di elaborazioni consecutive al fine di migliorare la robustezza del componente creato. I risultati dei seguenti test non sono stati riportarti per sinteticità.**

**4. Conclusioni**

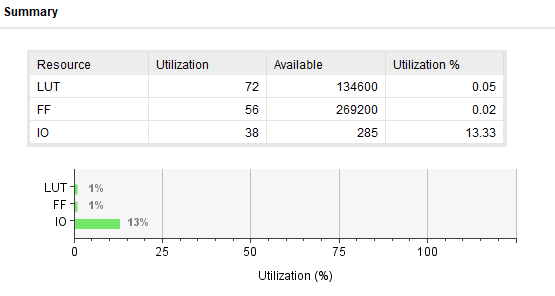
**4.1 Risultati della sintesi**

**Dal documento di report della sintesi fornito da Vivado troviamo che sono stati sintetizzati 11 registri in totale, così strutturati:**



**L’unico segnale non associato ad un registro è *wz\_offset* poiché si è deciso di calcolare sul momento il valore per la codifica.**

**Eseguendo un “Report Utilization” possiamo soffermarci sull’area occupata dal design sintetizzato. Notiamo valori di utilizzo per LUT e FF minori di diversi ordini di grandezza rispetto a quelli disponibili. Riteniamo che l’area utilizzata sia stata gestita in modo adeguato non avendo però riferimenti alle prestazioni di altri progetti.**

****

**4.1.1 Warnings in post synthesis**

**È presente un unico warning in sede di sintesi:**

***[Synth 8-3332] Sequential element (FSM\_onehot\_current\_state\_reg[12]) is unused and will be removed from module project\_reti\_logiche.***

**Questo è dovuto al mancato assegnamento dello stato di reset nel processo *lambda.* Il nostro design prevede che questo stato non sia normalmente raggiungibile ma possa essere raggiunto solo grazie al rilevamento di un segnale di reset da parte del processo *reset\_change.***

**4.2 Sommario**

**Il componente sviluppato supera tutti i test trattati in precedenza in tutte le simulazioni:**

***Behavioral, Post-Synthesis Functional e Post-Synthesis Timing*.**

Il componente rileva la variazione del segnale di reset asincronamente e la gestisce in modo sincrono attraverso lo stato reset.

**La progettazione e lo sviluppo ha prediletto la semplicità implementativa riducendo il numero di segnali impiegati e di conseguenza una minore area occupata, sacrificando però la velocità di codifica.**