Prova Finale Progetto Reti Logiche

**Anno Accademico 2019-2020**

Giusti Leonardo (Codice Persona: 10633778)

La Manna Giuseppe (Codice Persona: 10608466)

**INDICE**

1. **Introduzione . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2**

1. **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. **Risultati di test funzionali . . . . . . . . . . . . . . . . . . . . . . . . . . 6**
2. **Risultati test non funzionali . . . . . . . . . . . . . . . . . . . . . . . . . 7**

1. **Risultati della sintesi . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6**

**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 messo a 0:**

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

Se l’indirizzo da trasmettere appartiene ad una Working Zone, verrà trasmesso WZ\_BIT posto a 1 concatenato a WZ\_NUM, che rappresenta 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**

Nel seguente esempio viene esposto approfonditamente il metodo di codifica:

Immagine che contiene disegnando

Descrizione generata automaticamente

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).**

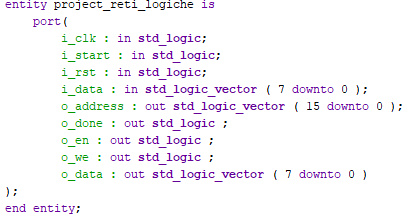
**Il valore codificato avrà come output (1 – 001 – 0010), che corrisponde al valore 146.**

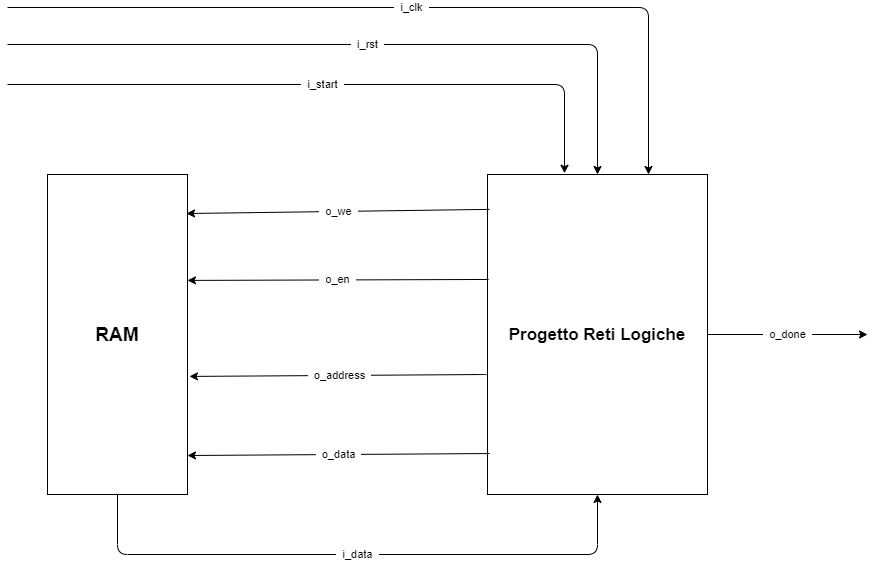
**2. Architettura**

**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

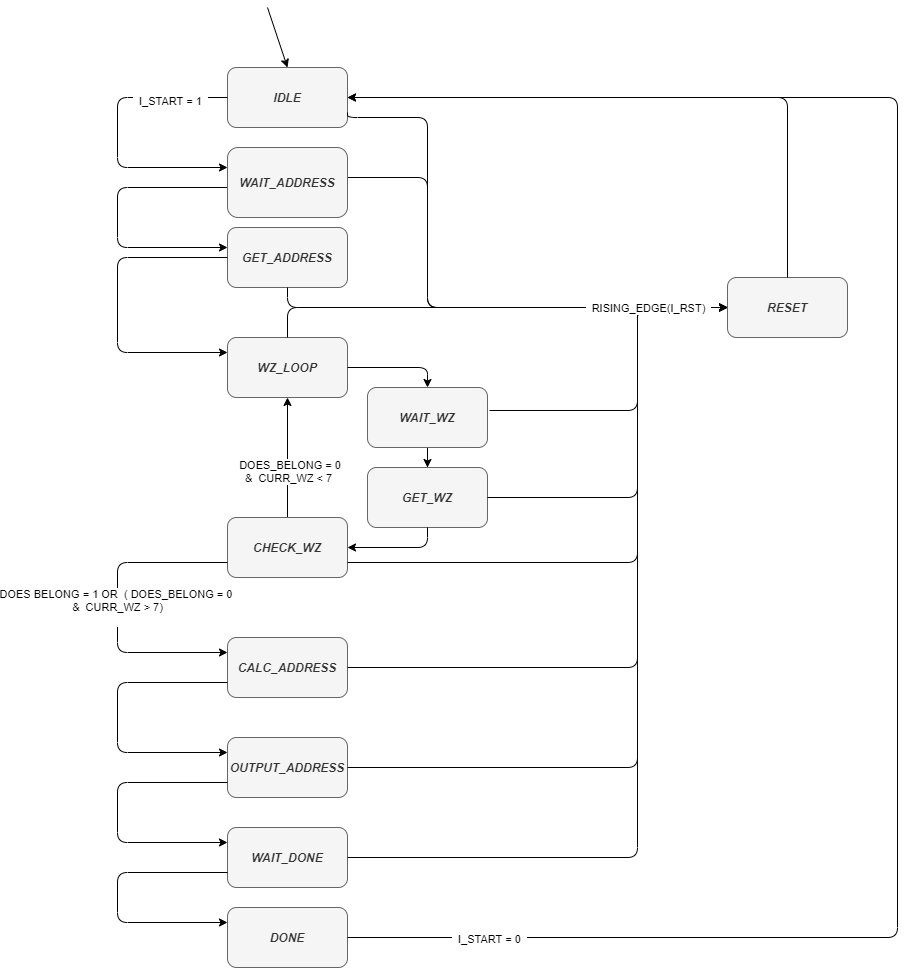
**2.1.2. Segnali utilizzati**

**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* codificato 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*;

**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 stata una gestione ottimale 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 non funzionali.**

**4. Risultati di test funzionali**

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

* **reset multipli in sequenza**
* **start multipli in sequenza**
* **alternanza tra reset e start**
* **rilevamento del reset fuori dal clock**
* **rilevamento del reset durante un rising edge**

**Reset multipli**

**--copre reset multipli durante un'esecuzione**

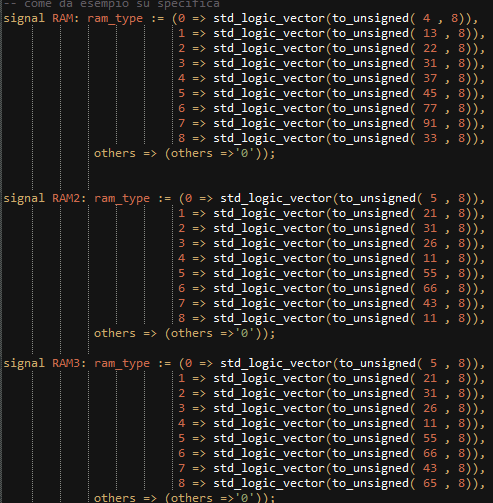
**--copre anche il rising\_edge(i\_rst) in falling\_edge(i\_clk)**

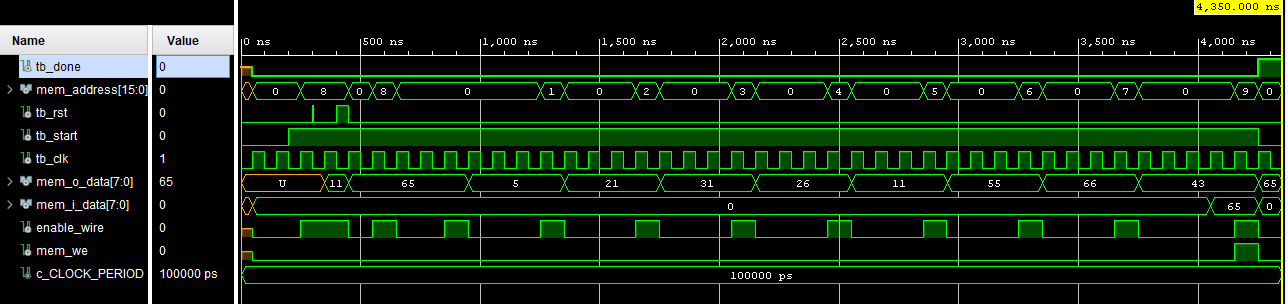
**--copre anche un pulse(i\_rst) su un falling\_edge(i\_clk)**

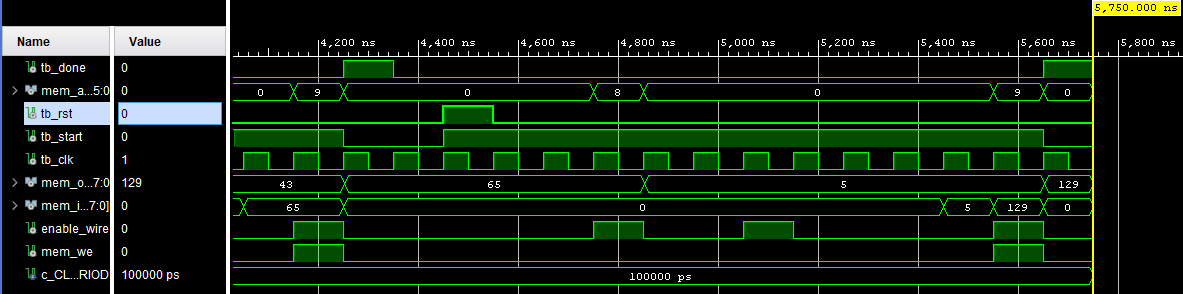
**--copre un rising\_edge(i\_rst) su un rising\_edge(i\_start)**

**--TODO copre un rising\_edge(i\_rst) su un falling\_edge( i\_start)**

**--TODO copre un pulse(i\_rst) su un rising\_edge(i\_clk)**

****

****

****

**Multi start**

**--start sul rising\_edge(i\_clk); < RAM 2**

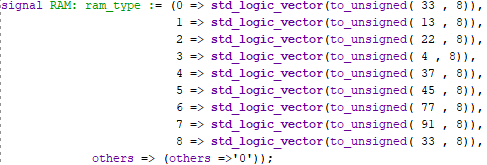
**--start sul falling\_edge(i\_clk); < RAM 1**

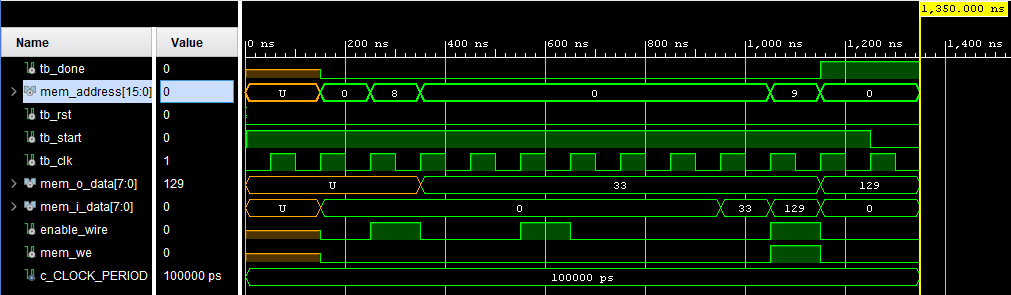
**--start su i\_clk = 0; < RAM 3**

**5. Risultati di test non funzionali**

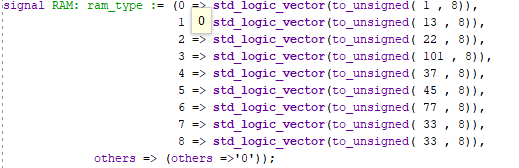
**Come precedentemente accennato, in questo paragrafo sono presi in esame tre casi particolari per la nostra scelta implementativa. Il primo test si evidenzia la computazione di un *address* appartenente alla prima WZ,nel secondo si evidenzia la computazione di un address appartenente all’ultima WZ,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

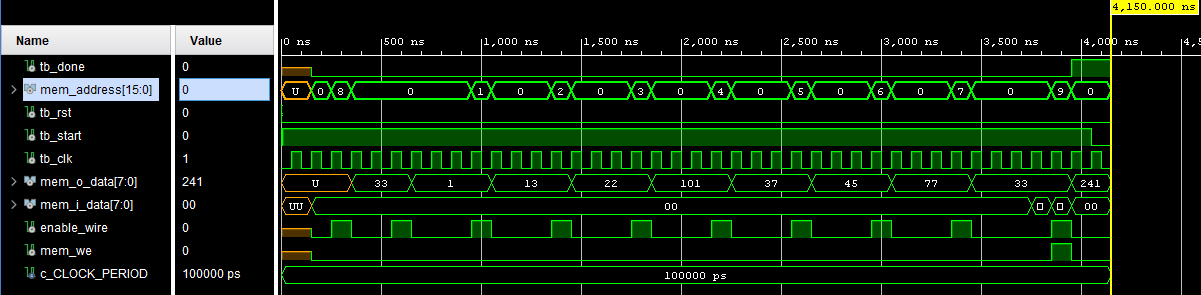
**--1° wz**

****

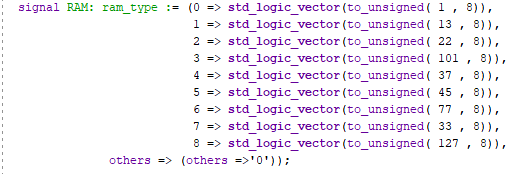
****

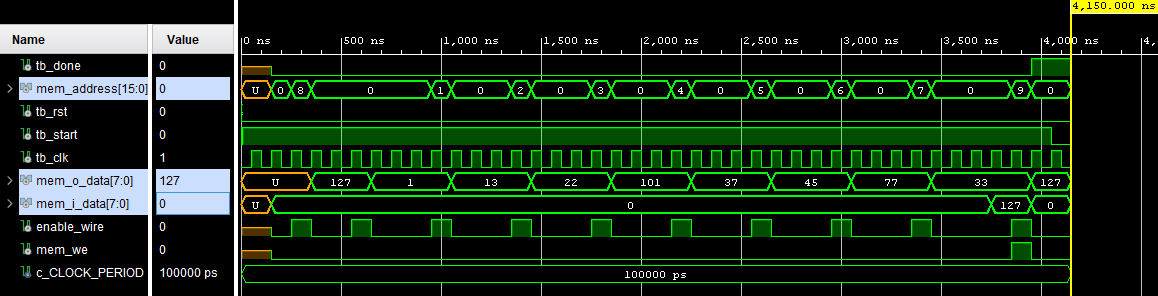
**--ultima wz**

****

****

**--no wz**

****

****

**6. Risultati della sintesi**