PROVA FINALE

PROGETTO DI RETI LOGICHE

Anno 2019/2020

Prof. Salice Fabio

Mangano Davide 10627074

Nespoli Matteo 10570485

INTRODUZIONE

Scopo del progetto

Implementare attraverso VHDL una versione molto semplificata del Working Zone Encoding (WZE), descritto e presumibilmente ideato per la prima volta nell'agosto 1997 per benchmark di applicazioni specifiche. Esso è utilizzato in quanto permette un risparmio di potenza (che sarebbe stata spesa per l'interazione con pin esterni e che è all'incirca tra le 100 e le 1000 volte superiore all'energia spesa per interazioni con nodi interni) oltre che lo sfruttamento di una proprietà molto importante della memoria: la **località**, che garantisce un utilizzo ricorrente di stesse aree. L'unico limite è quello del maggior peso computazionale dovuto all'encoder ed al decoder, unito alle possibili sovrapposizioni dell'indirizzo modificato a veri indirizzi presenti in memoria. Aspetti più evoluti quali le Potential Working Zone e i loro meccanismi di sostituzione non saranno qui presi in considerazione. [1]

Descrizione componenti

Il progetto consta di un unico componente da implementare, che riceve in ingresso quattro parametri:

- **i_clk**: il clock della macchina;
- **i_start**: segnale che se alto fa iniziare il processo;
- **i_rst**: segnale asincrono che interrompe il processo in corso e resetta a valori prestabiliti tutti i registri e le variabili;
- **i_data**: input restituito dalla memoria all'indirizzo o_address.

E con in uscita cinque parametri:

- **o_address**: indirizzo della memoria di cui richiederne il contenuto;
- **o_done**: segnale che se alto quando i_start = 1 lo abbassa, riabbassandosi a sua volta quando i_start = 0 (segnale di fine);
- **o_en**: segnale che se alto permette l'interazione (in ogni forma) con la memoria;
- **o_we**: segnale che specifica il tipo di interazione con la memoria: se alto si intende *scrittura*, altrimenti *lettura*;
- **o_data**: indirizzo codificato.

La memoria invece ha indirizzi da 15 bit ma ne vengono usati, da specifica, solo i primi 10, come segue:

- Indirizzo base della Working Zone **0**;
- Indirizzo base della Working Zone **1**;
- Indirizzo base della Working Zone **2**;
- Indirizzo base della Working Zone **3**;
- Indirizzo base della Working Zone **4**;
- Indirizzo base della Working Zone **5**;
- Indirizzo base della Working Zone **6**;
- Indirizzo base della Working Zone **7**;
- **Indirizzo da codificare**;
- **Indirizzo codificato**, ovvero lo spazio di memoria in cui scrivere una volta terminato il processo.

Come si può facilmente notare, il numero di Working Zone (NWZ) è **otto**.

La dimensione delle Working Zone (DWZ) invece, a partire dall'indirizzo base, è di **quattro**.

Funzionamento

Quando **i_start viene portato a uno**, il componente progettato chiede alla RAM il suo ottavo indirizzo (partendo da zero): la memoria, se **o_we = 0** e **o_en = 1**, al clock successivo restituisce nell'i_data del nostro componente l'indirizzo (**NB**: nonostante siano da *7 bit* e possano quindi variare tra 0 a 127, sono sempre memorizzati su *8 bit*) contenuto in RAM [8]. Al clock successivo comincia l'iterazione: richiesto l'indirizzo RAM [0], il nostro componente riceve l'indirizzo base della Working Zone 0, esegue una sottrazione e verifica se il risultato è zero; se no, aggiunge uno ed effettua nuovamente la sottrazione; se ancora diverso da zero, aggiunge uno alla Working Zone e questo ciclo avviene per ognuno dei quattro offset, e in generale per ognuna delle otto Working Zone.

- Nel caso in cui l'indirizzo dovesse essere trovato, il componente procede a scrivere in memoria (**o_we = 1**, **o_en = 1**) alla posizione RAM [9] l'indirizzo codificato come segue:
  - **1** concatenato a
  - **NWZ in binario** concatenato a
  - **Offset codificato One-Hot**.
- Se invece l'indirizzo non dovesse essere trovato, si procede a scrivere in RAM [9] l'indirizzo così com'era in precedenza.

ARCHITETTURA

NELLA PAGINA SUCCESSIVA È RIPORTATO IL DATAPATH INIZIALMENTE DISEGNATO E UTILIZZATO COME RIFERIMENTO PER LA NOSTRA IMPLEMENTAZIONE. DI SEGUITO RIPORTATO IL FUNZIONAMENTO: UNA VOLTA RICEVUTO I_DATA, È POSSIBILE CAPIRE DOVE MEMORIZZARLO IN BASE AI DUE ENABLE DEI REGISTRI, **REGWZEN** E **REGINDEN**, CHE RISPETTIVAMENTE PERMETTONO L'ACCESSO A REGWZ E REGIND SE ALTI. GRAZIE A QUESTE DUE VARIABILI È POSSIBILE DISCERNERE CONVENIENTEMENTE DOVE MEMORIZZARE I DATI NELLE PRIME FASI, IN CUI È NECESSARIO MEMORIZZARE L'INDIRIZZO; DOPODICHÉ SARÀ POSSIBILE SEMPLICEMENTE MANTENERE ALTO REGWZEN PER PERMETTERE I CONTINUI AGGIORNAMENTI DI WZ REPERITI DALLA RAM. COM'È POSSIBILE VEDERE DAL GRAFICO, AD OGNI CICLO SI EFFETTUA LA SOTTRAZIONE TRA INDIRIZZO E INDIRIZZO BASE WZ, SE UGUALE A ZERO O_DONE VIENE ALZATO INTERROMPENDO IL TUTTO, ALTRIMENTI VIENE OGNI VOLTA AGGIUNTO 1 DI OFFSET PER VERIFICARE TUTTE GLI INDIRIZZI ADIACENTI.

Datapath iniziale di riferimento per lo sviluppo del progetto

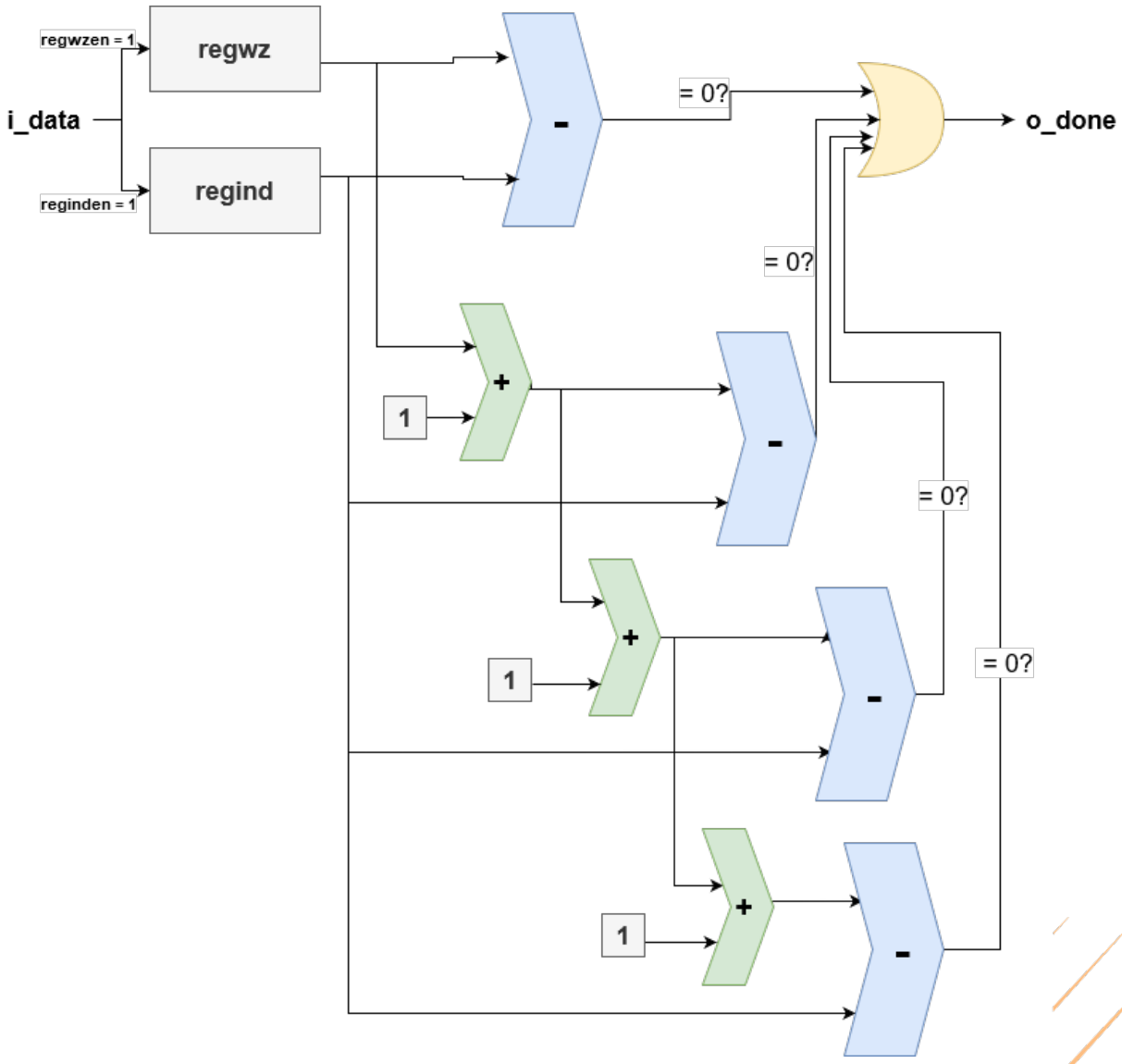


Figura 1 Datapath realizzato con draw.io

Macchina a stati

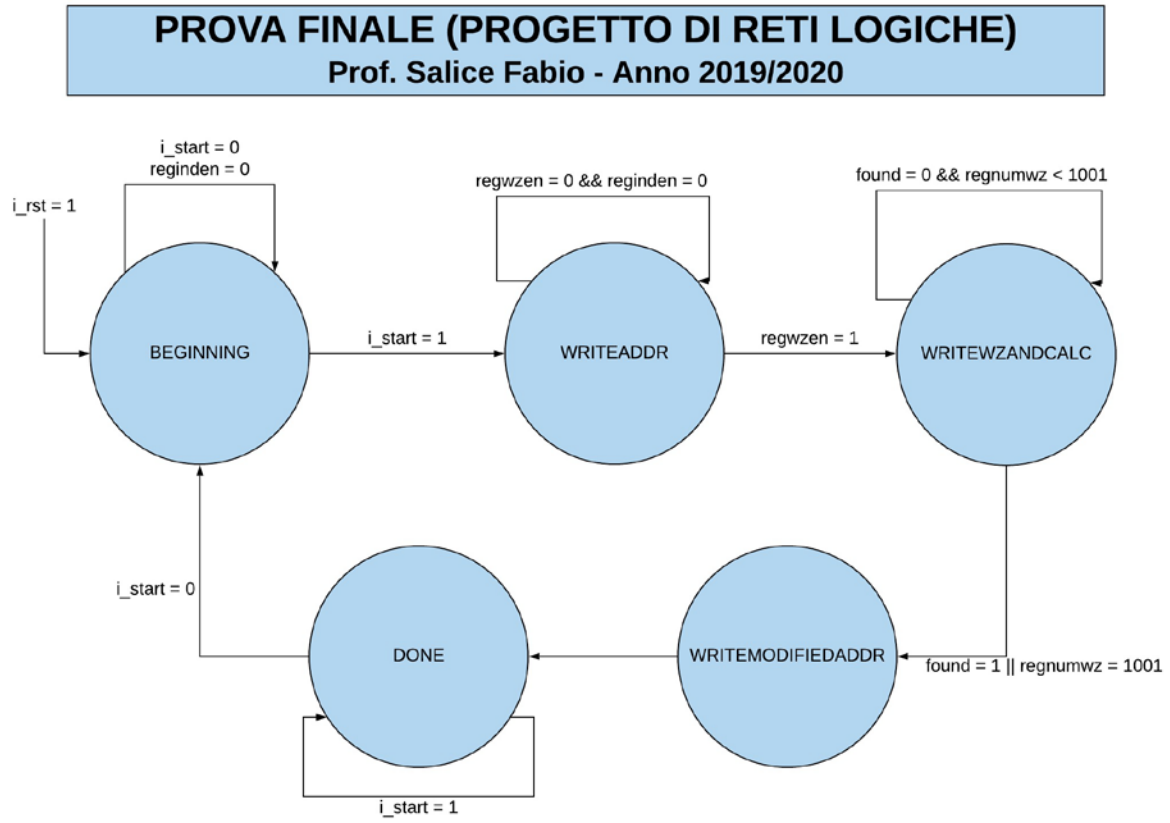


Figura 2 Macchina a stati realizzata su Lucidcharts.com

DESCRIZIONE DELL'IMPLEMENTAZIONE

Inizialmente abbiamo optato per una versione che implementasse non i finali cinque, ma ben **tredici** stati, in quanto ad ogni Working Zone veniva assegnato uno stato. A seguito di miglioramenti e perfezionamenti continui, siamo giunti alla versione finale con i quattro stati (all'inizio non si prevedeva l'esistenza del DONE), poi ci siamo resi conto dell'esigenza di uno stato per così dire "finale" in cui attendere l'abbassamento del segnale i_start. Lo stato iniziale, chiamato Beginning, svolge funzioni per cui sarebbe stato più giusto chiamarlo Idle, tuttavia Beginning fu il nome che gli demmo inizialmente e l'abbiamo mantenuto come portafortuna. Nella pagina seguente è possibile leggere la descrizione dei vari stati.

DESCRIZIONE DEGLI STATI

- **BEGINNING:** stato iniziale da cui si parte a inizio processo, ed a cui si ritorna in soli due casi: se i_rst viene alzato a uno, o se i_start, a seguito di o_done = 1, viene riportato a 0, causando a sua volta l'abbassamento di o_done. Si passa allo stato successivo se i_start viene alzato a 1.
- **WRITEADDR:** In questo stato si giunge con regwzen = 0 e reginden = 0. Successivamente, al primo clock di esecuzione, viene impostato reginden a 1, dopodiché viene rieseguito lo stesso stato scrivendo in regind il valore di i_data (ovvero la cella RAM [8]). A questo punto, reginden viene posto a 0 e regwzen viene alzato a 1.
- **WRITEWZANDCALC:** in questo stato si giunge con reginden a 0 e regwzen a 1. Quindi viene fatto partire subito il processo: si chiama tramite o_address la prima Working Zone che viene scritta in regwz, dopodiché si procede a fare i confronti con il contenuto di regind, aggiungendo l'offset fino a tre (partendo da zero). Se il risultato della sottrazione regwz + offset – regind = 0, found viene posto a 1e lo stato successivo è WRITEMODIFIEDADDR. Altrimenti si procede a incrementare una volta per clock o_address, fino a RAM [7], dopodiché regnumz raggiunge l'upperbound e pone found a zero e lo stato successivo a WRITEMODIFIEDADDR.
- **WRITEMODIFIEDADDR:** qui si giunge se l'indirizzo appartiene a una Working Zone (found uguale a 1) oppure se non vi appartiene (found uguale a 0). Ovviamente a seconda del valore di found l'indirizzo verrà codificato come da specifica.
- **DONE:** stato finale, inizialmente non previsto. Si è rivelato necessario per alcune esigenze quali il dover aspettare che i_start scendesse a 0 per poter riabbassare i_done, e di conseguenza il bisogno di avere uno stato in cui ritornare ricorsivamente finché ciò non fosse vero (solitamente e da specifica, un ciclo di clock).

SCELTE PROGETTUALI

Le principali scelte progettuali, come in parte riferito precedentemente, sono state le seguenti:

1. Inizialmente abbiamo previsto l'uso di uno stato per ogni Working Zone: questa tuttavia si è rivelata una scelta dettata più dalla volontà di racchiudere in ogni stato la propria logica che da un'effettiva necessità di programmazione, in quanto è bastato inserire un registro (**regwz**) con il proprio enable (**regwzen**) per potersi gestire le varie Working Zone chiamate tramite o_address e quindi presenti in i_data. Per gestire il numero della Working Zone a cui si è, è bastato anche qui aggiungere un registro chiamato **regnumwz** che viene incrementato ad ogni chiamata di o_address.
2. Lo stato DONE, inizialmente non previsto, è stato aggiunto per permettere un loop fino a che i_start non fosse sceso, da specifica, dopo un ciclo di clock.
3. Altra particolare scelta progettuale da noi fatta è stata di usare un registro per gestire le Working Zones invece di utilizzare direttamente i_data; mentre per l'indirizzo iniziale presente in RAM [8] è necessario un registro per memorizzarlo e poterci fare i confronti, per le Working Zone, per come abbiamo strutturato il programma, sarebbe bastato utilizzare i_data e confrontarlo con il registro contenente l'indirizzo. Tuttavia, abbiamo preferito l'utilizzo di un registro per evitare problemi con gli input e definire una migliore logica implementativa.
4. Altra scelta è stata quella di utilizzare solo tre processi: uno per il caso i_rst uguale a, uno per il caso in cui passi un ciclo di clock, e un processo "blob", contenente nei fatti tutta la logica di implementazione. Questa può sembrare una scelta avventata dato la particolare strutturazione di VHDL (che non assegna le variabili signal all'interno del processo finché non passa un clock) e il nostro deliberato inutilizzo delle variable; tuttavia, con un attento uso delle variabili "_next" e le giuste assegnazioni è stato possibile fare funzionare tutto.
5. È stato scelto di aggiungere uno stato apparentemente superfluo, quale WRITEADDR (in quanto la memorizzazione di i_data in regwz o regind poteva essere tranquillamente gestita attraverso gli enable) in quanto questo ci ha permesso di gestire meglio il passaggio da uno stato all'altro tramite, anche, un cambiamento di segnali (quali gli enable dei registri in cui memorizzare indirizzi/Working Zones) che ci potesse garantire di delineare più nitidamente le differenze di ogni stato.

RISULTATI DEI TEST

Qui sotto sono riportati i risultati forniti dal componente realizzato a seguito di una simulazione utilizzando i test bench forniti dal docente. Nella waveform è stato inserito solo la wave del test bench per chiarezza espositiva, in quanto la conoscenza dell'implementazione del nostro componente è stata già esposta prima e qui è necessario mostrare il corretto funzionamento. Importante ricordare che il Test Bench 1 va a testare il corretto funzionamento se l'indirizzo appartiene ad una Working Zone; il Test Bench 2, al contrario, testa il caso in cui l'indirizzo non appartiene a nessuna Working Zone e dev'essere di conseguenza scritto così com'è all'indirizzo RAM [9].

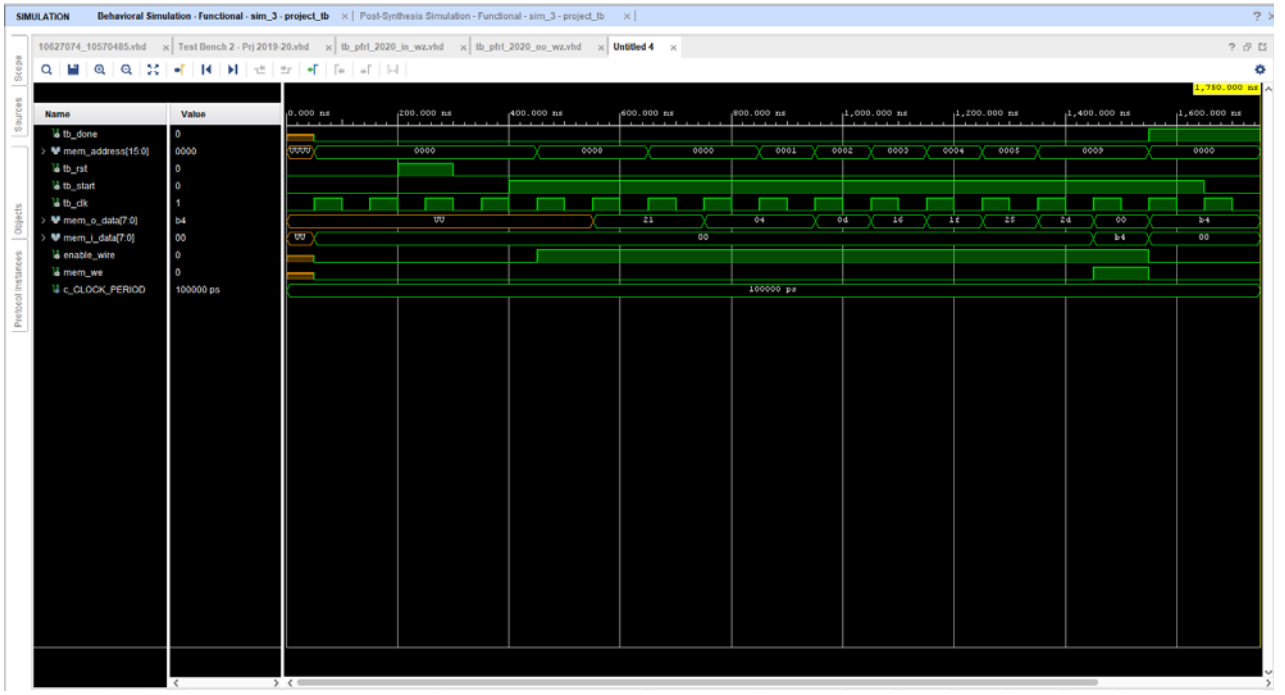


Figura 3 TEST BENCH 1 BEHAVIOURAL

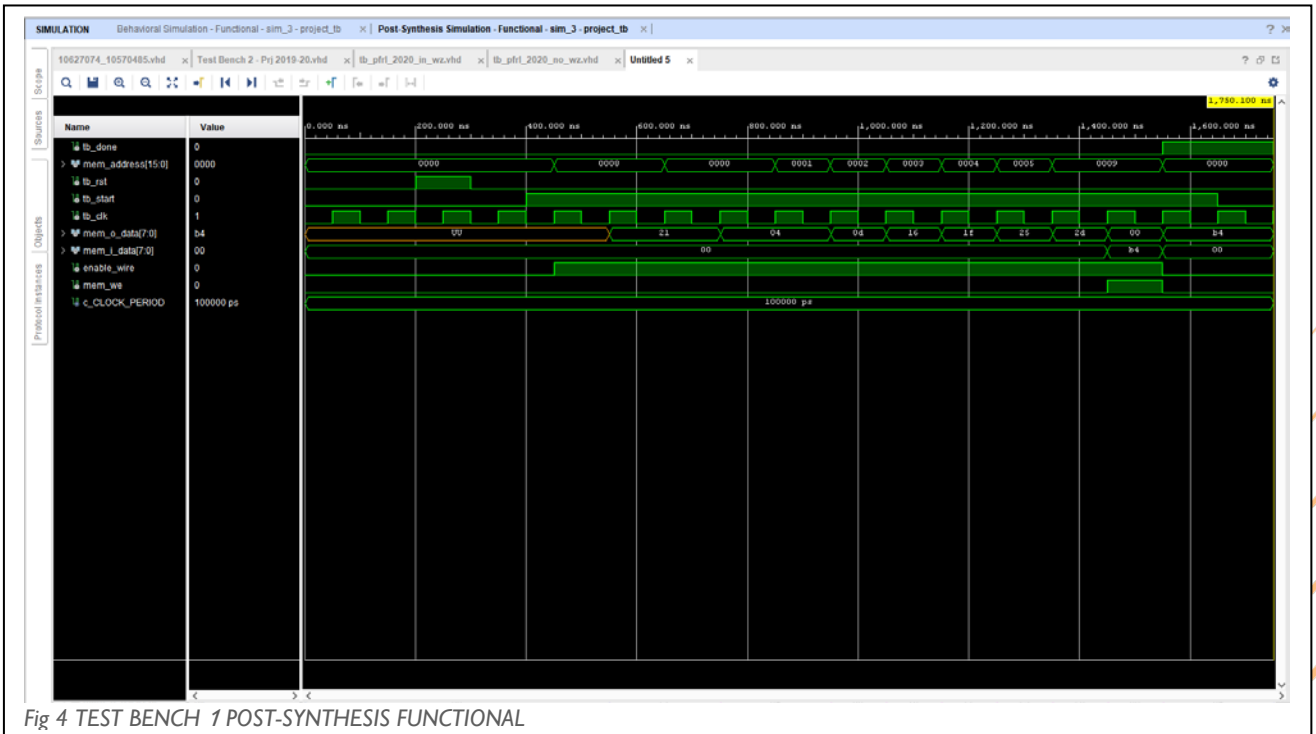


Fig 4 TEST BENCH 1 POST-SYNTHESIS FUNCTIONAL

Figura 6 TEST BENCH 2 BEHAVIOURAL

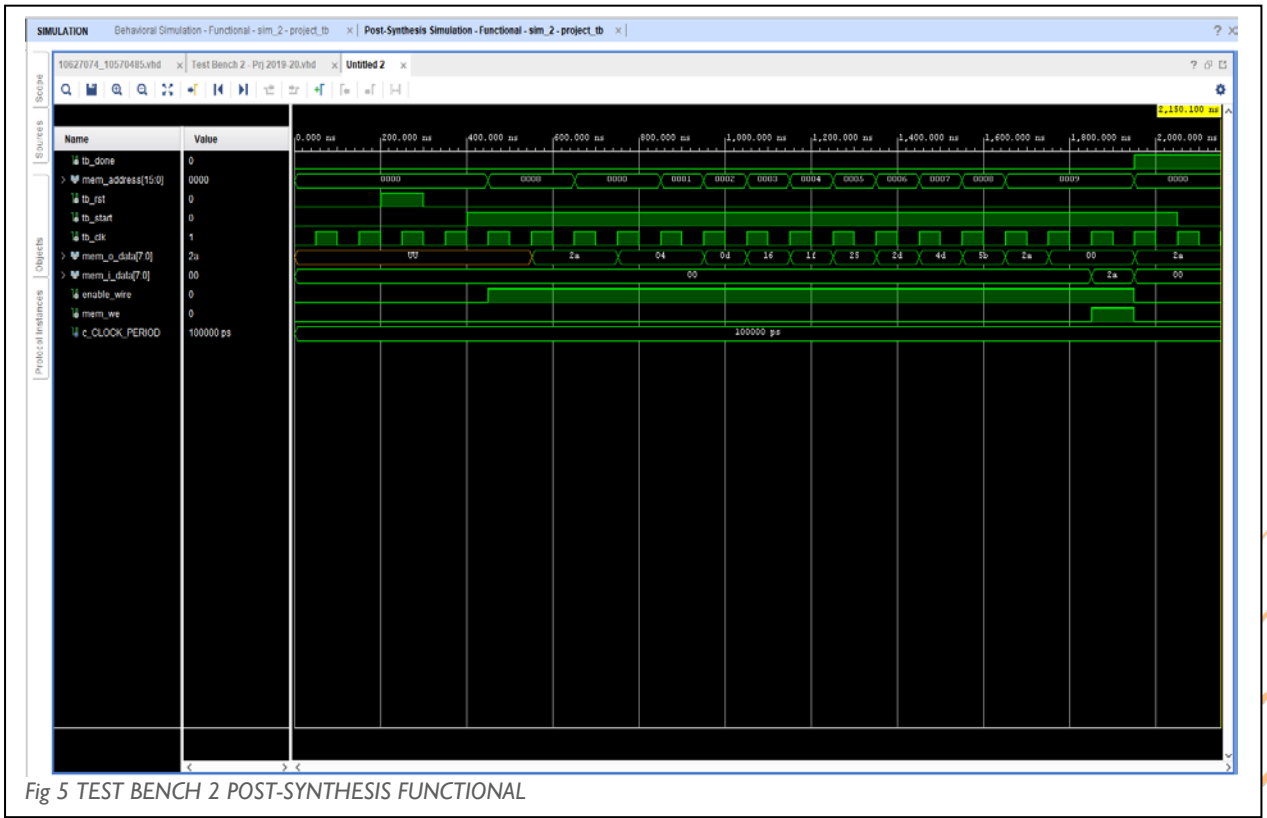
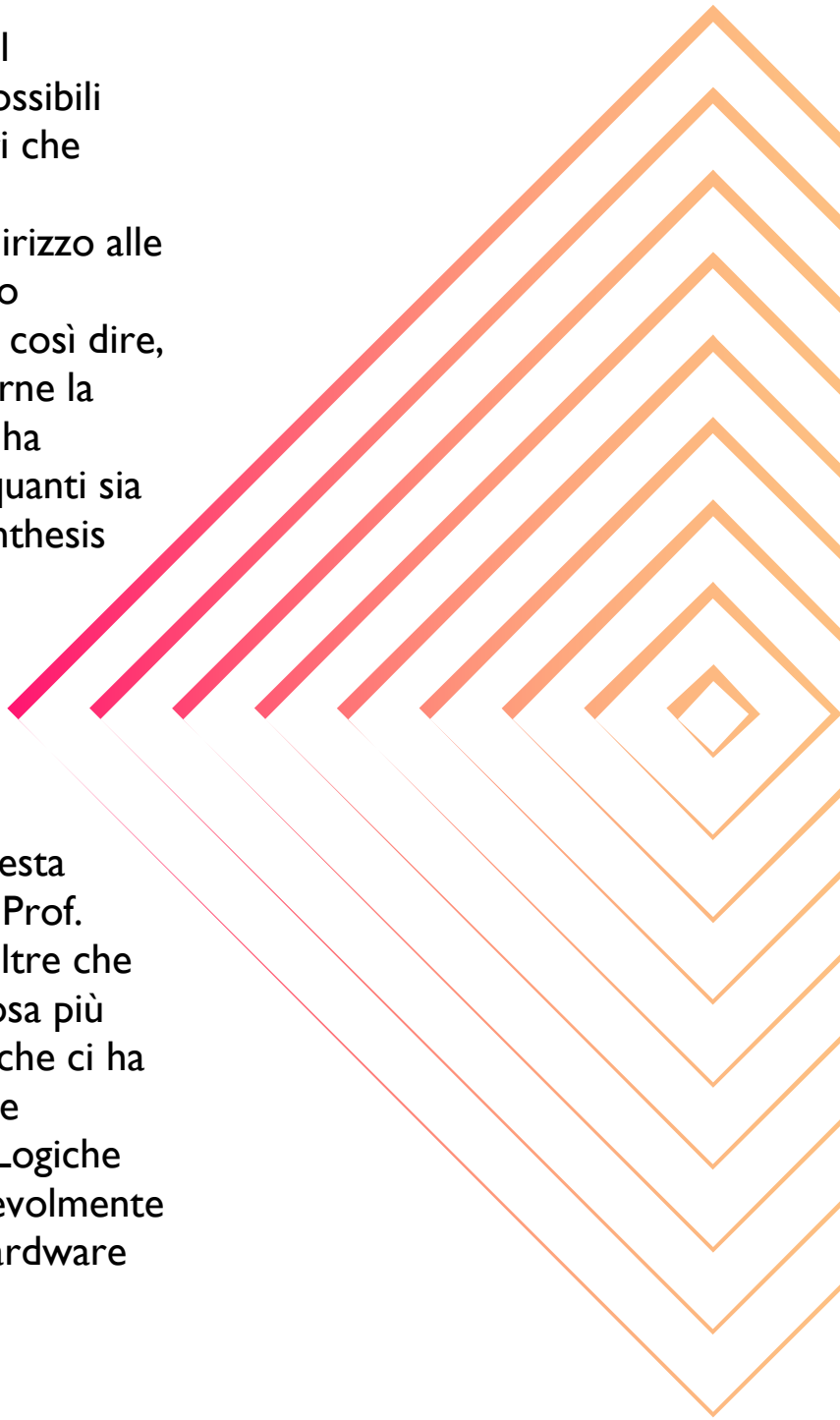


Fig 5 TEST BENCH 2 POST-SYNTHESIS FUNCTIONAL

Conclusioni

Oltre ai Test Bench forniti dal docente, per coprire tutti i possibili casi ne abbiamo utilizzato altri che testassero sia casualmente le appartenenze o meno dell'indirizzo alle Working Zones, sia un utilizzo appositamente pesante e, per così dire, "cattivo" dei segnali per testarne la correttezza, e il componente ha dimostrato di superarli tutti quanti sia in Behavioural che in Post-Synthesis Functional.



Vorremmo approfittare di questa conclusione per ringraziare il Prof. Salice per il suo lato umano oltre che prettamente professionale, cosa più unica che rara al Politecnico, che ci ha permesso non solo di svolgere serenamente il corso di Reti Logiche ma anche di apprendere piacevolmente non poca programmazione hardware grazie a questo progetto.

[1] 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.