Progetto di Reti Logiche

Francesco Galbiati (codice persona: 10728028)
Ermelinda Giulivo (codice persona: 10790499)

March 2023

Contents

1 Descrizione della composizione del gruppo

Il progetto è stato svolto in gruppo.
Composizione del gruppo:
Francesco Galbiati CODICE PERSONA: 10728028 MATRICOLA: 956301
Ermelinda Giulivo CODICE PERSONA: 10790499 MATRICOLA: 959581

2 Introduzione

La "Prova Finale (Progetto di Reti Logiche)" per l'anno accademico 2022-2023 richiede di implementare un modulo hardware, interfacciato con una RAM, a cui vengono passate indicazioni riguardanti un indirizzo di memoria. Il contenuto di quest'ultimo deve essere poi inoltrato verso uno dei canali di uscita del componente. La selezione del canale di uscita è anch'essa specificata in ingresso al modulo hardware.

2.1 Interfacce

Il componente dispone di 2 ingressi e 5 uscite. Gli ingressi sono seriali da 1 bit (W e START), le 5 uscite invece si dividono in 4 da 8 bit (Z0, Z1, Z2, Z3) e una da 1 bit (DONE). Il modulo ha inoltre un segnale unico di clock (CLK) e un segnale unico di reset (RESET).

2.2 Funzionamento

All'istante relativo al reset del sistema le uscite sono poste a zero. I dati di ingresso arrivano tramite W ma vengono considerati validi solo se in ingresso START è uguale a '1', se questo accade le sequenze sono lette sul fronte di salita del clock e vengono interpretate nel seguente modo:

- I primi 2 bit identificano il canale di uscita dove verrà indirizzato il messaggio
- Gli N bit che seguono specificano l'indirizzo della memoria da cui prendere il messaggio da indirizzare sull'uscita specificata in precedenza

La sequenza è valida fintanto che START è uguale a '1', nel momento in cui START passa a '0' la sequenza in ingresso è conclusa, il componente può dunque, avendo salvato l'uscita dove mostrare il messaggio, recuperare dalla memoria il dato da mandare in uscita sul canale specificato grazie all'indirizzo passato in ingresso, anch'esso salvato in precedenza. I valori dei canali di uscita sono visibili solo quando DONE è '1', inoltre DONE passa da '0' a '1', restando attivo per un solo ciclo di clock, contemporaneamente alla scrittura del messaggio sul canale.

2.3 Ulteriori dettagli

- Gli N bit di indirizzo possono variare da '0' a un massimo di 16 bit
- Gli indirizzi di memoria sono da 16 bit, dunque se N è inferiore a 16 l'indirizzo è esteso con '0' sui bit più significativi

- START rimane alto per almeno 2 cicli di clock e per non più di 18 cicli di clock (minimo i 2 bit del canale più massimo 16 bit per indirizzare la memoria)
- Z0, Z1, Z2 e Z3 sono inizialmente a '0' e i valori restano inalterati eccetto per il canale sul quale viene mandato il messaggio letto in memoria
- Quando DONE è uguale a '1' i valori sui canali sono visibili
- Quando DONE è uguale a '0' tutti le uscite sono a '0'
- Quando DONE è '1' solo il canale associato al messaggio cambierà il suo valore, gli altri canali, invece, mostreranno l'ultimo valore trasmesso
- Finché il segnale DONE non torna a '0', il segnale START rimarrà a '0'
- Il tempo massimo per produrre il risultato (ovvero il tempo trascorso tra START='0' e DONE='1') deve essere inferiore a 20 cicli di clock
- Prima di richiedere il corretto funzionamento del modulo e dunque prima che START sia posto a '1' per la prima volta, verrà sempre dato il seganale di RESET (RESET='1'). Per quanto riguarda invece le eventuali successive elaborazioni con START posto a '1', non si dovrà attendere per il reset del modulo
- Il modulo viene re-inizializzato ogni volta che RESET è posto a '1'

3 Architettura

L'architettura del componente hardware richiesto è stata divisa in due moduli che operano a diversi livelli di astrazione: a più basso livello si trova il datapath, che gestisce gli ingressi, le uscite e i segnali intermedi del modulo a livello di bus, a più alto livello invece il componente è comandato da una macchina a stati finiti, che controlla i segnali del datapath in base agli ingressi dati.

3.1 Datapath

Il modulo datapath si divide in due parti, la prima ha il compito di gestire la lettura dei dati in input e di passarli alla memoria, mentre la seconda gestisce la visualizzazione dei dati che arrivano dalla RAM (Immagine riportata nella pagina successiva).

3.1.1 Scrittura dei dati in input

I segnali in ingresso sono **i_w**, che indica i bit da prendere in ingresso, e **i_start** che segnala al modulo quando leggere il dato da **i_w**, oltre ai consueti clock e reset (in questo caso chiamati **i_clk** e **i_rst**). Per memorizzare i dati del canale e dell'indirizzo di memoria il componente hardware fa uso dei due registri **reg1** (a 2 bit) e **reg2** (a 16 bit), viene inoltre usato un demultiplexer a due uscite per selezionare il registro su cui mandare il dato (visto che l'ingresso **i_w** comunica sia il canale che l'indirizzo) chiamato **demux1**, il quale ha un segnale di controllo che gli indica il canale su cui

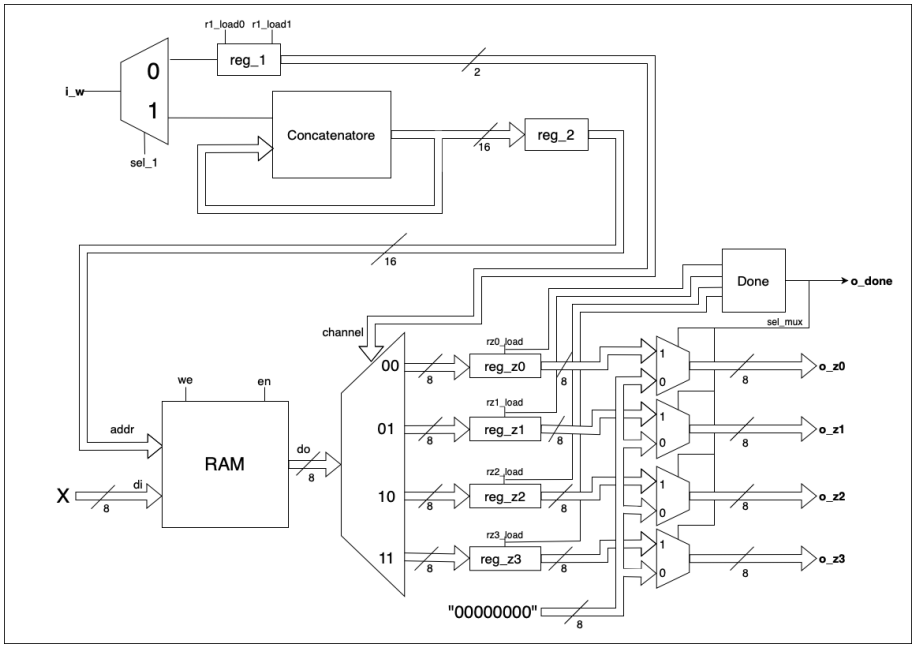

Figure 1: Datapath

trasmettere il dato preso da **i_w**. La scrittura sul registro **reg1** avviene per singolo bit, tramite i segnali di controllo **r1_load0** e **r1_load1**, mentre la scrittura sul registro **reg2** avviene per tutti i bit contemporaneamente tramite il segnale di controllo **r2_load**.
La lettura del canale di uscita avviene quindi tramite la scrittura dei singoli bit sul registro apposito, tuttavia lo stesso metodo risulta scomodo per la composizione dell'indirizzo di memoria, dal momento che quest'ultimo è un bus formato da 16 bit. Per far fronte a questo problema il datapath si avvale di un modulo di concatenazione, che partendo da un bus di 16 bit a '0' fa uno shift graduale dei bit al suo interno, aggiungendo a destra i bit che arrivano dal demultiplexer. Nel dettaglio il concatenatore funziona in questo modo: per ogni ciclo di clock esso concatena i 15 bit più a destra dell'uscita al ciclo di clock precedente con il bit in arrivo da **demux1**, posizionandolo a destra. Si noti, nella descrizione riportata sotto, la presenza del segnale **i_cln**, la funzione di questo segnale consiste nel riportare tutti i bit del concatenatore a '0' in preparazione di un nuovo indirizzo in ingresso, che altrimenti andrebbe a mescolarsi con l'indirizzo precedente causando un malfunzionamento del modulo.

Algorithm 1: Descrizione del modulo concatenatore

```
1 process(i_clk, i_rst)
2     begin
3         if(i_rst = '1' or i_cln = '1') then
4             o_conc <= "0000000000000000";
5         elsif(i_clk'event and i_clk = '1') then
6             o_conc <= o_conc(14 downto 0)&o_sel1_1;
7         end if;
8     end process;
```

L'output del concatenatore viene salvato nel registro **reg2** ad ogni ciclo di clock

in maniera sincrona, di conseguenza **reg2** sarà sempre un ciclo di clock "indietro" rispetto al concatenatore e sarà possibile per la memoria prelevare il dato al momento corretto. I dati contenuti in **reg1** e **reg2** verranno poi letti dalla memoria quando il segnale **i_start** passerà da '1' a '0', terminando così la lettura dei dati in input.

3.1.2 Lettura e visualizzazione dei dati

La seconda parte del modulo datapath si occupa della lettura dei dati dalla RAM e della loro visualizzazione. Per mandare la parola in ingresso al canale corretto il datapath si avvale di un secondo demultiplexer **demux2** a quattro uscite, una per ciascun canale. **Demux2** è comandato dal segnale in uscita da **reg1** (corrispondente al canale letto dall'ingresso). I quattro canali sono gestiti con altrettanti registri sincroni, chiamati **regz1**, **regz2**, **regz3** e **regz4** (generalizzati come **regzX** in questa relazione). La scrittura di questi ultimi è comandata da quattro segnali di controllo, uno per registro (**rz1_load**, **rz2_load**, **rz3_load** e **rz4_load**, da qui in poi **rzX_load**). La visualizzazione dei dati sui canali è gestita da 4 multiplexer (**mux0**, **mux1**, **mux2** e **mux3**), che mandano sulle uscite il dato preso dai registri **regzX** o un bus composto da 8 bit a '0'. I multiplexer appena descritti hanno il segnale di controllo in comune (chiamato **sel_mux**), ciò è necessario per sincronizzare la visualizzazione sull'uscita. **Sel_mux** viene generato da un modulo sincrono (chiamato **"Done"**) in risposta alla scrittura su uno dei registri **regzX**. Viene riportata sotto la specifica VHDL del modulo appena menzionato:

Algorithm 2: Descrizione del modulo Done

```
1  process(i_clk, i_rst)
2      begin
3          if(i_clk'event and i_clk = '1') then
4              sel_mux <= '0';
5              if(rz0_load = '1' or rz1_load = '1' or
6                  rz2_load = '1' or rz3_load = '1') then
7                      sel_mux <= '1';
8              end if;
9          end if;
10     end process;
```

Si noti come il modulo trasmetta in maniera sincrona (quindi al ciclo di clock successivo) il segnale **sel_mux** appena uno dei quattro segnali di scrittura dei registri **regzX** viene messo a '1'. Questo segnale viene inoltre trasmesso in uscita come **o_done**, in modo che la salita di quest'ultimo risulti contemporanea alla visualizzazione dei dati sulle uscite, come da specifica.

3.2 Macchina a stati finiti

La macchina a stati finiti opera a un livello di astrazione più alto del datapath e ne gestisce i segnali di controllo.

La macchina a stati di questo modulo è una macchina di Moore e si compone di 10 stati (da S0 a S9), sviluppandosi come nell'immagine. In particolare controlla i seguenti segnali:

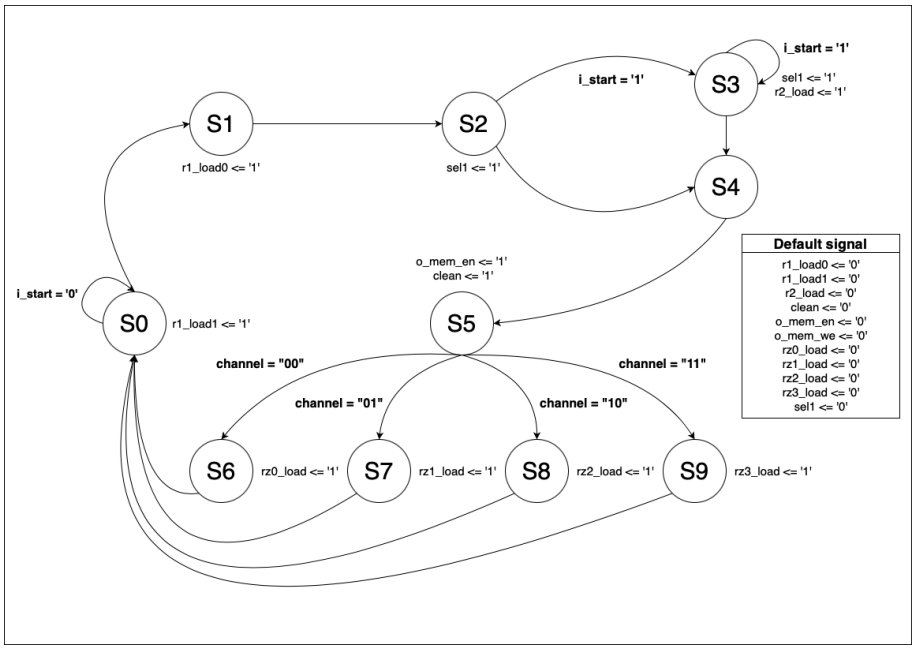

Figure 2: Macchina a stati finiti

- **r1_load0** e **r1_load1**, che controllano il registro **reg1**
- **r2_load** che controlla la scrittura sul registro **reg2**
- **clean** che controlla il segnale **i_cln** del concatenatore e del registro **reg2**
- **o_mem_en** e **o_mem_we** che controllano lettura e scrittura sulla memoria
- I segnali **rzX_load** che controllano la scrittura sui registri **regzX**
- **sel1** che controlla il demultiplexer **demux1**

In ogni stato in cui non è specificato altrimenti la funzione di uscita pone i segnali di controllo al valore di default '0'.
Segue l'analisi dei singoli stati:

Stato S0: Lo stato S0 gestisce l'inizio della lettura dei dati da **i_w**, in particolare la funzione di uscita mette il segnale **r1_load1** a '1', in questo modo il datapath caricherà il bit proveniente dall'ingresso nel bit 1 del registro **reg1**. La funzione stato prossimo invece fa rimanere la macchina a stati in S0 se il segnale **i_start** è '0' (poichè ciò indica che la lettura del canale non è ancora iniziata) e la manda in S1 se **i_start** è '1', che continuerà la lettura del canale.

Stato S1: Lo stato S1 è il secondo stato di gestione della lettura del canale (dopo S0), di conseguenza in S1 il segnale **r1_load0** verrà messo a '1', in modo da caricare sul bit 0 del registro **reg1** il bit preso dall'ingresso, completando così la lettura del canale di uscita. La funzione stato prossimo invece porta in qualsiasi caso l'esecuzione su S2, dove inizierà la lettura dell'indirizzo di memoria.

Stato S2: Con lo stato S2 inizia la lettura dell'indirizzo, dal momento che il modulo concatenatore non necessita di segnali di controllo, lo stato S2 si limita a cambiare il canale di **demux1**, portando il segnale **sel1** a '1'. Da S2 l'esecuzione può essere portata in S3 se **i_start** è '1' (ovvero se vanno letti dei bit di indirizzo) o in S4 se **i_start** è '0' (se non vanno letti bit di indirizzo).

Stato S3: Lo stato S3 è visitato solo nel caso in cui **i_start** sia a '1' per più di due cicli di clock. Lo stato S3 mantiene **sel1** a '1' e porta **r2load** a '1', iniziando il salvataggio degli indirizzi "parziali" su **reg2**. Da S3 la funzione stato prossimo fa rimanere in S3 in caso **i_start** sia ancora a '1', mentre porta in S4 se **i_start** scende a '0'.

Stato S4: Lo stato S4 completa la scrittura ed è necessario per fare in modo che **reg2** salvi correttamente l'indirizzo di memoria da passare successivamente alla RAM, inoltre porta l'esecuzione in S5.

Stato S5: La funzione di uscita dello stato S5 si occupa di controllare la lettura dalla RAM, ponendo **o_mem_en** a '1'. Inoltre, mettendo **clean** uguale a '1', azzera l'uscita del concatenatore, così preparandolo a ricevere il prossimo indirizzo. Dallo stato S5 si può andare negli stati S6, S7, S8, o S9. Questa decisione è comandata dal canale letto nei primi stati, in particolare "00" porta in S6, "01" in S7, "10" in S8 e "11" in S9.

Stati S6, S7, S8, S9: Questi stati si occupano di comandare i registri **regzX**, in particolare controllano i relativi segnali **rzX_load**, che permettono la scrittura dei dati in ingresso provenienti dalla memoria RAM. Dopo aver visitato uno di questi stati l'esecuzione si riporta in S0 e aspetta un nuovo input.

4 Risultati Sperimentali

Di seguito vengono elencati i risultati sperimentali ottenuti, in particolare i report di sintesi e le simulazioni dei casi di test considerati.

4.1 Sintesi

La sintesi del componente viene portata a termine da Vivado senza errori, in questa sezione vengono quindi esaminati alcuni punti notevoli dei report di sintesi.

4.1.1 Utilization Report

L'utilization report mette in evidenza le risorse dell'FPGA target (come registri e look-up tables) istanziate da Vivado per garantire il corretto funzionamento del componente hardware. Di particolare interesse è la prima tabella (denominata "Slice Logic") in cui vengono indicati il numero delle look-up tables e dei registri istanziati. Da questa tabella si nota come il numero di latch usati per i registri sia pari a 0, ciò è indice di un corretto sviluppo del modulo.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	51	0	134600	0.04
LUT as Logic	51	0	134600	0.04
LUT as Memory	0	0	46200	0.00
Slice Registers	77	0	269200	0.03
Register as Flip Flop	77	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Figure 3: Utilization report

4.1.2 Timing Report

Questa parte del report fa riferimento al constraint sul tempo dato dalla specifica (100 nanosecondi) e indica il tempo avanzato dalla sintesi rispetto al limite imposto tramite il constraint. Nel caso della sintesi di questo modulo il parametro di slack è pari a $97.484ns$, di conseguenza il vincolo imposto è rispettato.

required time	101.834
arrival time	-4.351
slack	97.484

Figure 4: Timing report

4.2 Simulazioni

Questa sezione si occupa di riportare le simulazioni dei test bench utilizzati nella fase di testing del modulo, soffermandosi in particolare su alcuni casi limite degni di nota, talvolta esaminando anche i segnali interni al modulo per illustrarne meglio il funzionamento. La fonte principale dei test bench utilizzati è stata il set condiviso su WeBeep dai docenti, i test bench dati sono stati analizzati per individuare i casi limite presenti in essi e, in alcuni casi, modificati per coprire dei casi limite altrimenti non considerati. Le immagini presenti in questa sezione provengono tutte da simulazioni funzionali post-sintesi.

4.2.1 Test Bench 1: Funzionamento Generale

Il primo test bench analizzato non presenta particolari casi limite, tuttavia è comunque necessario per mostrare come il modulo funzioni correttamente per esecuzioni "standard". Il test bench analizzato proviene dal file *tb_example23.vhd*, condiviso sulla piattaforma WeBeep.

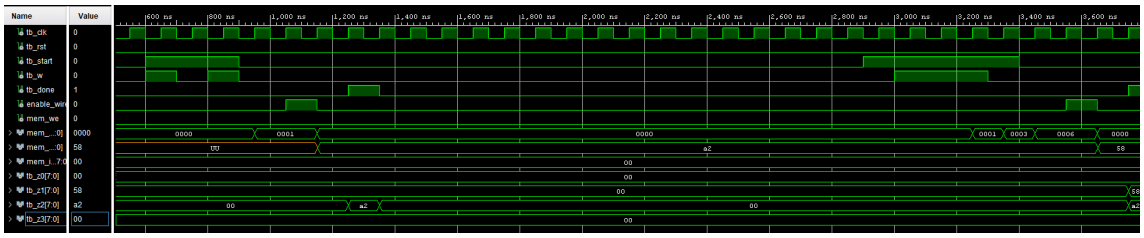

Figure 5: Simulazione del test bench 1

Nell'immagine sono visibili le forme d'onda e i bus principali del modulo, si noti inoltre come l'indirizzo di memoria venga composto gradualmente durante l'esecuzione e prelevato al momento giusto (**enable_wire** a '1'), per poi essere azzerato in preparazione dell'esecuzione successiva.

4.2.2 Test Bench 2: Short start

Il secondo test bench è volto a mostrare il funzionamento del modulo nel caso di un segnale **tb_start** che rimane alto per soli 2 cicli di clock. Il corretto funzionamento del modulo in questo caso è quello di porre l'indirizzo di memoria a "0000000000000000" e riportare il dato sul canale di uscita indicato dai due bit di **tb_w**. Questo test bench è un estratto dal file *tb_1.vhd*.

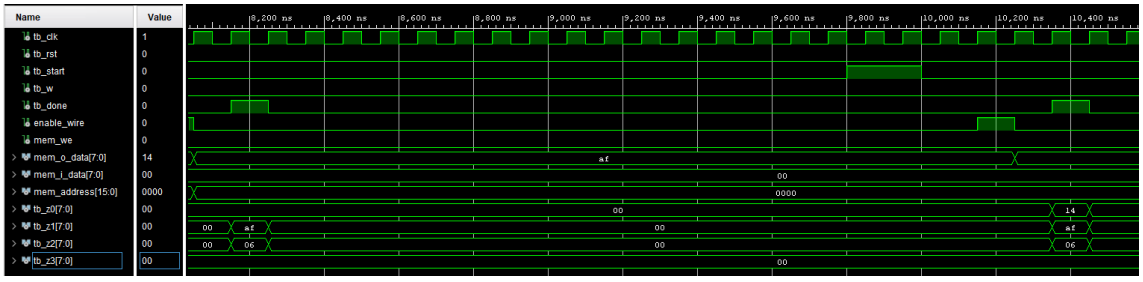

Figure 6: Simulazione del test bench 2

Il caso considerato si ritrova oltre l'indice dei 9800ns, dall'immagine si può verificare il corretto funzionamento del modulo in questo caso, anche rispetto ai dati già presenti nei canali di uscita, che non sono impattati dal caso limite.

4.2.3 Test Bench 3: Long start

Il terzo test bench proposto si occupa di mostrare il funzionamento del modulo nel caso opposto di quello presentato in 4.2.2, ovvero nel caso in cui il segnale **tb_start** resti a '1' per 18 cicli di clock, il massimo possibile richiesto dalla specifica. In questo caso il modulo dovrebbe utilizzare tutti i 16 bit disponibili per l'indirizzo. L'immagine che segue è presa dalla simulazione di *tb_2.vhd*.

Dalla simulazione si può constatare l'utilizzo di tutti i bit dell'indirizzo (in questo caso tutti a '1', "ffff" in notazione esadecimale), il corretto funzionamento della lettura del dato e la corretta visualizzazione dei dati precedentemente letti.

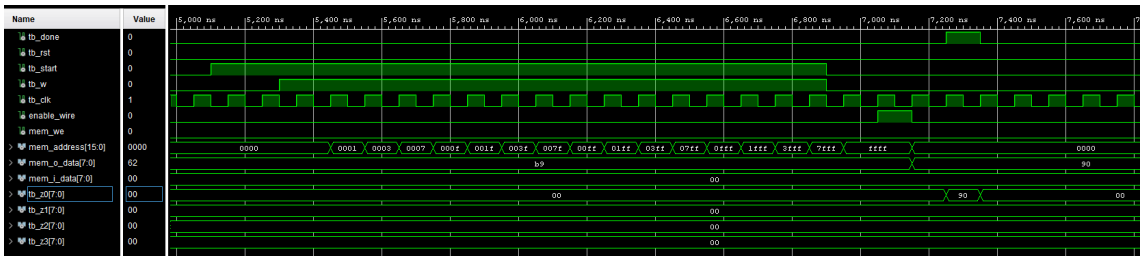

Figure 7: Simulazione del funzionamento generale

4.2.4 Test Bench 4: Long w

Il quarto caso analizzato è molto simile al caso 4.2.3, infatti si tratta sempre di un caso in cui **tb_start** rimane alto per 18 cicli di clock, la differenza tra i due test bench si ritrova nel segnale **tb_w**, in questo caso anch'esso rimarrà alto per tutti i 18 cicli di clock. Il test bench è estratto dalla simulazione di *tb_6.vhd*.

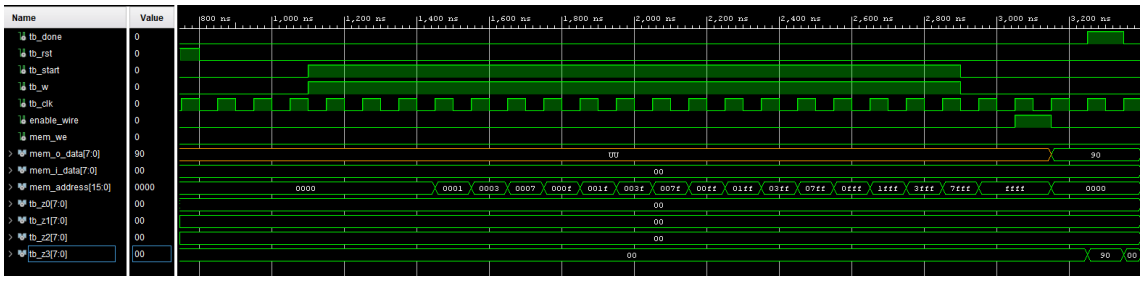

Figure 8: Simulazione del test bench 4

Il corretto funzionamento del modulo nel caso considerato è verificabile dalle forme d'onda riportate nell'immagine.

4.2.5 Test Bench 5: Reset before read

Il caso numero 5 si concentra sul segnale **tb_rst** e ne verifica il funzionamento nel caso in cui sia posto tra una visualizzazione e la successiva lettura. Il comportamento corretto del modulo prevede che tutti i registri che fanno riferimento ai canali d'uscita vengano riportati a "00000000". L'immagine proposta è presa dalla simulazione di *tb_1.vhd*.

A differenza delle immagini precedenti, in quella proposta sono evidenziati solo

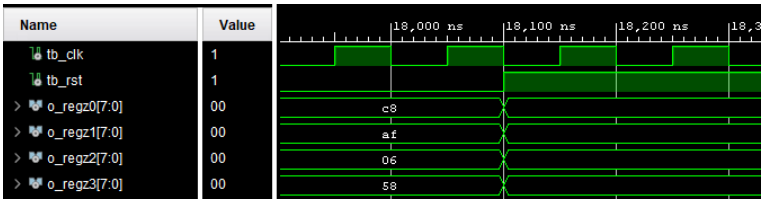

Figure 9: Simulazione del test bench 5

il segnale di reset e il contenuto dei registri collegati ai canali di uscita. Si può verificare il corretto funzionamento constatando come appena il reset viene portato a '1', i registri azzerino il proprio contenuto.

4.2.6 Test Bench 6: Resent during read

Come il caso 4.2.5, anche questo caso di test si concentra sul segnale di reset, la differenza tra i due test bench è il momento in cui viene dato il segnale, se nel caso precedente veniva dato all'infuori delle fasi critiche dell'esecuzione, in questo caso viene dato durante la lettura di un input. La simulazione ha il compito di mostrare come il modulo reagisce a questa situazione ed è presa da una versione modificata di *tb_7.vhd*.

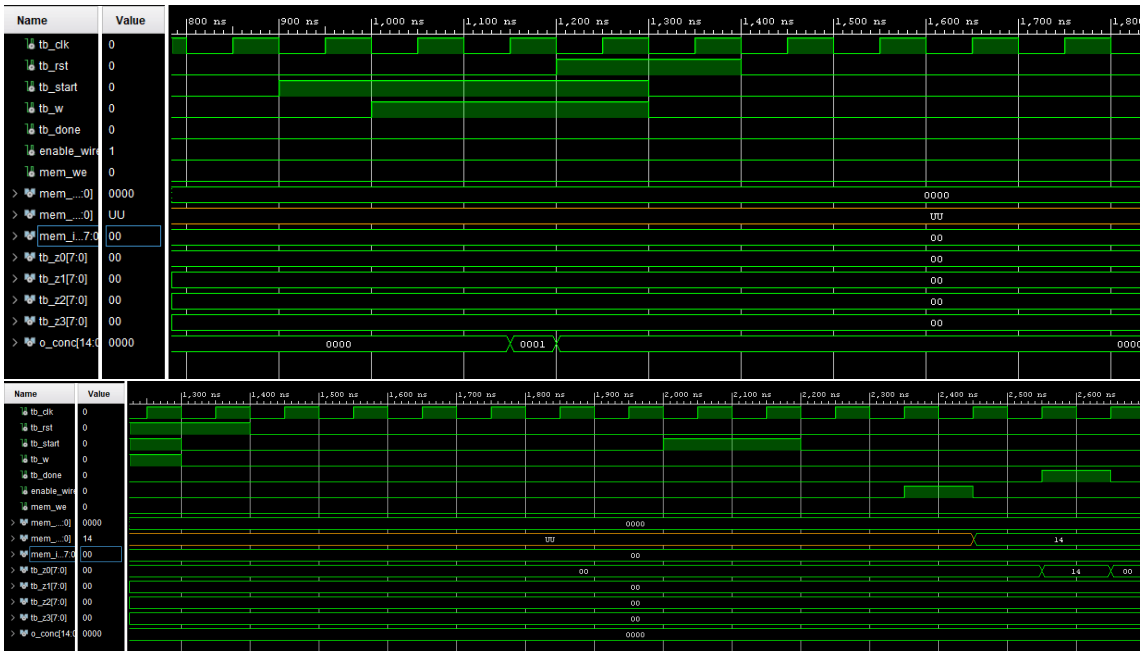

Figure 10: Simulazione del test bench 6

Nelle immagini date, che coprono l'intera esecuzione, viene evidenziato anche il funzionamento del modulo di concatenazione, che viene resettato insieme agli altri componenti.

5 Conclusioni

Come illustrato nelle sezioni precedenti, il modulo hardware implementato soddisfa nella loro totalità le richieste della specifica, assumendo un corretto comportamento, oltre che in situazioni standard, anche in casi limite, che se non gestiti potrebbero compromettere il funzionamento del circuito.
In conclusione, la realizzazione del progetto assegnato ci ha permesso di approfondire le nostre capacità di sviluppo hardware e la nostra conoscenza del linguaggio VHDL, introducendoci inoltre ad un ambiente di sviluppo specializzato e di ampio utilizzo come Vivado, dandoci così l'opportunità di confrontarci con l'aspetto pratico degli argomenti appresi durante il corso di Reti Logiche.