Relazione Progetto Reti Logiche

Francesco Di Giore

Matricola : 959440

Codice Persona : 10776567

Docente : Fornaciari William

A.A. 2022/2023

Contents

1 Introduzione

1.1 Descrizione generale

La specifica della "Prova Finale (Progetto di Reti Logiche)" per l'Anno Accademico 2022/2023 richiede di implementare un modulo HW, descritto mediante VHDL, che si interfacci con una memoria e che rispetti le indicazioni riportate successivamente.

Ad elevato livello di astrazione, il sistema riceve indicazioni circa una locazione di memoria, il cui contenuto deve essere indirizzato verso un canale di uscita fra i quattro disponibili. Le indicazioni riguardo il canale da utilizzare e l'indirizzo di memoria a cui accedere sono forniti mediante un ingresso seriale, mentre le uscite del sistema, ovvero i canali, forniscono tutti i bit della parola di memoria in parallelo.

1.2 Interfacce

Il modulo da implementare ha due ingressi primari da 1 bit (W e START) e 5 uscite primarie. Le uscite sono le seguenti: quattro da 8 bit (Z0, Z1, Z2, Z3) e una da 1 bit (DONE). Inoltre, il modulo ha un segnale di clock (CLK), unico per tutto il sistema e un segnale di reset (RESET), anch'esso unico.

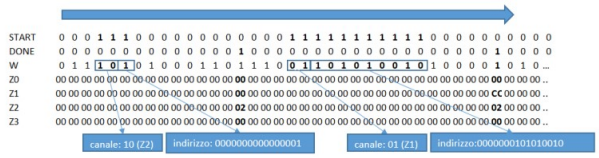
1.3 Funzionamento

All'istante iniziale, quello relativo al reset del sistema, le uscite hanno i seguenti valori: Z0, Z1, Z2 e Z3 sono tutte 0000 0000, DONE è 0.
I dati in ingresso sono letti sul fronte di salita del clock e sono ottenuti come sequenze sull'ingresso primario W. Inoltre sono organizzati come segue:

- i primi 2 bit sono di intestazione
- N bit che indicano un indirizzo di memoria

I due bit di intestazione indicano il canale di uscita (Z0, Z1, Z2, Z3) sul quale deve essere mostrato il dato ottenuto dalla memoria, in particolare: 00 identifica Z0, 01 identifica Z1, 10 identifica Z2 e, infine, 11 identifica Z3.

Mentre, gli N bit successivi, con N che varia da 0 fino ad un massimo di 16, indicano un indirizzo di memoria. Se N è minore di 16, l'indirizzo è esteso con 0 per i bit piu' significativi. Tutti i bit di W devono essere letti sul fronte di salita del clock.
Affinchè la sequenza in ingresso sia valida, è necessario che il segnale di START sia alto; esso rimane alto per almeno 2 cicli di clock e non piu' di 18. Le uscite Z0, Z1, Z2, Z3 sono inizialmente 0. Il loro valore cambia solo se sovrascritto ed è mostrato quando il segnale DONE è a 1. Quando il segnale DONE è a 0, il valore mostrato da tutte le uscite deve essere 0.



		reset	00	00	00	00	[LSB … MBS | LSB MSB]
CANALE	INDIRIZZO	DATO	Z0	Z1	Z2	Z3	W utile
02	0001	**02**	00	00	02	00	**101**
01	0152	**CC**	00	CC	02	00	**01001010110**
02	045B	A1	00	CC	A1	00	**1101101000101**
00	0751	7E	7E	CC	A1	00	**1000101011100**
00	07B2	69	69	CC	A1	00	**0100110111100**
02	0A58	7B	69	CC	7B	00	**00011010010101**
00	0B1C	AC	AC	CC	7B	00	**00111000110100**
03	0DFC	EA	AC	CC	7B	EA	**00111111101111**
00	1025	03	03	CC	7B	EA	**101001000000100**
01	1335	D8	03	D8	7B	EA	**101011001100110**
02	1605	4F	03	D8	4F	EA	**101000000110101**
…	…	…	…	…	…	…	…

Figure 1: Descrive i segnali ed il funzionamento del componente

1.4 Interfaccia del componente

Il componente da descrivere ha la seguente interfaccia (Figure 2):

- i_clk è il segnale di CLOCK in ingresso generato dal Test Bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;

- i_start è il segnale di START generato dal Test Bench;
- i_w è il segnale W precedentemente descritto e generato dal Test Bench;
- o_z0, o_z1, o_z2, o_z3 sono i quattro canali di uscita;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- i_mem_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poterci scrivere. Per leggere da memoria esso deve essere 0.

```
entity project_reti_logiche is
    port (
        i_clk   : in std_logic;
        i_rst   : in std_logic;
        i_start : in std_logic;
        i_w     : in std_logic;

        o_z0    : out std_logic_vector(7 downto 0);
        o_z1    : out std_logic_vector(7 downto 0);
        o_z2    : out std_logic_vector(7 downto 0);
        o_z3    : out std_logic_vector(7 downto 0);
        o_done  : out std_logic;

        o_mem_addr : out std_logic_vector(15 downto 0);
        i_mem_data : in std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

Figure 2: Interfaccia del componente

2 Architettura

L'architettura del componente progettato è costituita da 6 registri, dalla memoria, da un demultiplexer, da 4 multiplexer e da una macchina a stati finiti (FSM), che controlla il componente.

2.1 Struttura generale

La struttura generale del componente è riportata nella seguente immagine:

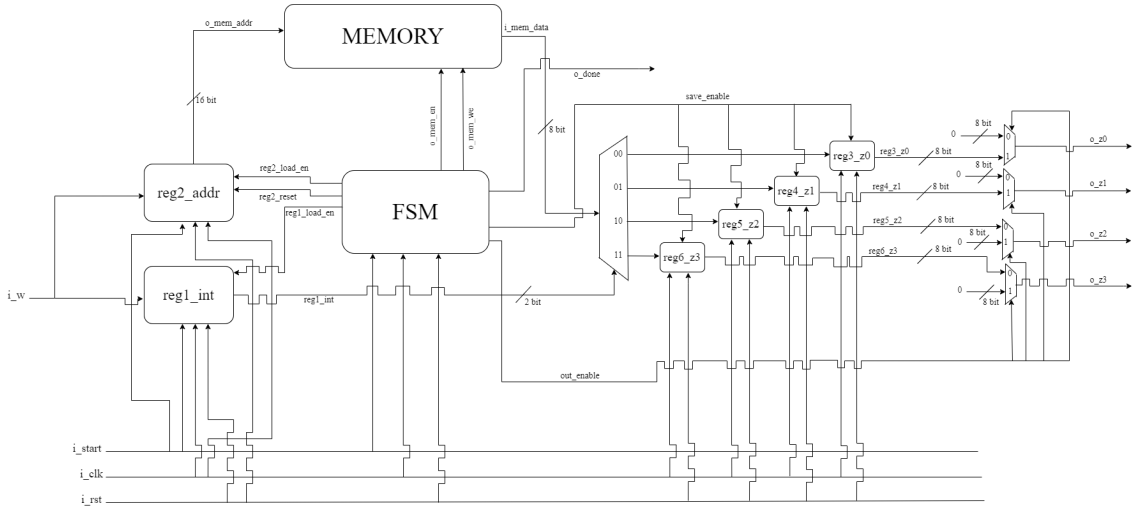


Figure 3: Struttura generale del componente

L'FSM si occupa della generazione dei segnali che influenzano il funzionamento dei registri e della memoria.
Nelle sezioni successive, si descrivono in modo piu' dettagliato i vari componenti ed i segnali generati dall'FSM.

2.2 Memoria

L'architettura del componente si interfaccia con una memoria la cui struttura è stata fornita da specifica ed è la seguente:

```
entity rams_sp_wf is
port(
  clk  : in  std_logic;
  we   : in  std_logic;
  en   : in  std_logic;
  addr : in  std_logic_vector(15 downto 0);
  di   : in  std_logic_vector(7 downto 0);
  do   : out std_logic_vector(7 downto 0)
);
end rams_sp_wf;

architecture syn of rams_sp_wf is
type ram_type is array (65535 downto 0) of std_logic_vector(7 downto 0);
signal RAM : ram_type;
begin
  process(clk)
    begin
    if clk'event and clk = '1' then
      if en = '1' then
        if we = '1' then
          RAM(conv_integer(addr)) <= di;
          do                      <= di after 2 ns;
        else
          do <= RAM(conv_integer(addr)) after 2 ns;
        end if;
      end if;
    end if;
  end process;
end syn;
```

Figure 4: Struttura della memoria

2.3 Segnali interni

I segnali gestiti dall'FSM sono:

```
signal reg1_load_en : std_logic;
signal reg2_load_en : std_logic;
signal reg2_reset : std_logic;
signal save_enable : std_logic;
signal out_enable : std_logic;
```

Figure 5: Segnali dell'FSM

I segnali che rappresentano le uscite dei registri sono:

```
signal reg1_int : std_logic_vector(1 downto 0);
signal reg2_addr : std_logic_vector(15 downto 0);
signal reg3_z0 : std_logic_vector(7 downto 0);
signal reg4_z1 : std_logic_vector(7 downto 0);
signal reg5_z2 : std_logic_vector(7 downto 0);
signal reg6_z3 : std_logic_vector(7 downto 0);
```

Figure 6: Uscite dei registri

- *reg1_load_en* : abilita la scrittura nel registro reg1;
- *reg2_load_en* : abilita la scrittura nel registro reg2;
- *reg2_reset* : serve per resettare il registro reg2 che contiene l'indirizzo di memoria, per evitare che il nuovo indirizzo di memoria si sovrapponga al vecchio;
- *save_enable* : abilita la scrittura nei registri che contengono i valori di output;
- *out_enable* : serve per scegliere quale valore mandare in uscita (0 oppure il valore contenuto in ciascun registro d'uscita);

2.4 Registro reg1_int

Il registro reg1_int è utilizzato per memorizzare i primi due bit di ciascuna sequenza, che costituiscono l'intestazione. È un registro di tipo SIPO, poichè l'input arriva in modo seriale (dato dal segnale i_w) ed è caricato shiftando in avanti i valori precedentemente caricati nel registro. La lettura avviene in parallelo, poichè tale registro è rappresentato mediante un vettore, perciò è possibile accedere ai valori contemporaneamente. Di seguito il blocco di codice che rappresenta il funzionamento di questo registro:

```
reg1_int_proc : process(i_clk, i_rst)
    begin
        if i_rst = '1' then
            reg1_int <= "00";

        elsif i_clk'event and i_clk = '1' and i_start = '1' then
            if reg1_load_en = '1' then
                reg1_int (1 downto 0) <= reg1_int(0) & i_w;
            end if;
        end if;
    end process;
```

Figure 7: Comportamento del registro reg1_int

Si noti che il registro è resettato solo quando il segnale i_rst è alto, non è necessario resettarlo prima di ogni scrittura poichè shiftando i valori in avanti, i valori precedenti sono man mano eliminati e sostituiti da quelli nuovi. Inoltre, la scrittura avviene solo se il segnale reg1_load_en è alto, purchè il segnale i_start sia alto, altrimenti i bit in input non sono validi. Infine, la scrittura avviene sul fronte di salita del clock, quindi sfalsata di mezzo periodo rispetto a i_start.

2.5 Registro reg2_addr

Il registro reg2_addr è utilizzato per memorizzare l'indirizzo di memoria. Esso contiene 16 valori. Se l'input è inferiore a 16 bit, allora l'indirizzo è esteso con 0 bit dal bit piu' significativo. L'estensione è fatta resettando il registro prima di ogni scrittura, inserendo 16 bit a 0. In fase di scrittura, il valore è scritto shiftando i bit in avanti, così alla fine l'indirizzo avrà sempre 16 bit, eventualmente con degli 0 in testa.

Tale registro è di tipo SIPO, con lettura in parallelo grazie alla sua rappresentazione mediante un vettore.Di seguito il blocco di codice che rappresenta il funzionamento di questo registro:

```
reg2_addr_proc : process(i_clk, reg2_reset, i_rst)
    begin
        if reg2_reset = '1' or i_rst = '1' then
            reg2_addr <= "0000000000000000";

        elsif  i_clk'event and i_clk = '1' and i_start = '1' then
            if reg2_load_en = '1' then
                reg2_addr(15 downto 0) <= reg2_addr(14 downto 0) & i_w;
            end if;
        end if;
    end process;
```

Figure 8: Comportamento del registro reg2_addr

Si noti che il reset del registro è fatto quando il segnale in input i_rst è alto, oppure quando il segnale reg2_reset, gestito dall'FSM, è alto. La scrittura del valore avviene sul fronte di salita del clock quando, dato il segnale i_start alto, il segnale reg2_load_en è alto.

2.6 Registri reg3_z0, reg4_z1, reg5_z2, reg6_z3

Ciascuno dei 4 registri contiene il valore che è mostrato, quando è opportuno, sulla corrispondente uscita del componente. Quando il valore è ottenuto dalla memoria, tramite un demultiplexer (descritto in seguito), viene selezionato il corretto registro in base ai due bit di intestazione salvati nel registro reg1_int. Il valore è salvato quando il segnale save_enable è alto.
Il valore da mostrare per ogni uscita è scelto mediante un multiplexer per ogni registro. Se il segnale out_enable è basso, in uscita è trasmesso 0, altrimenti sono trasmessi i valori salvati nei registri.
Questi registri sono di tipo PIPO, cioè sono scritti e letti in parallelo, rendendo tali operazioni rapide.

2.7 Demultiplexer

Il demultiplexer è utilizzato per scegliere in quale registro d'uscita salvare il valore proveniente dalla memoria, in base ai due bit di intestazione precedentemente salvati nel registro reg1_int. Di seguito si riporta il blocco di codice che ne descrive il comportamento:

```
case reg1_int(1 downto 0) is
    when "00" =>
        reg3_z0(7 downto 0) <= i_mem_data(7 downto 0);
    when "01" =>
        reg4_z1(7 downto 0) <= i_mem_data(7 downto 0);
    when "10" =>
        reg5_z2(7 downto 0) <= i_mem_data(7 downto 0);
    when "11" =>
        reg6_z3(7 downto 0) <= i_mem_data(7 downto 0);
    when others =>
        reg3_z0(7 downto 0) <= "00000000";
        reg4_z1(7 downto 0) <= "00000000";
        reg5_z2(7 downto 0) <= "00000000";
        reg6_z3(7 downto 0) <= "00000000";
end case;
```

Figure 9: Demultiplexer

2.8 Multiplexer

I 4 multiplexer sono usati per scegliere se trasmettere in output 8 bit a 0 oppure i valori contenuti all'interno dei registri d'uscita, in base al segnale out_enable. Di seguito si riporta il blocco di codice che ne descrive il comportamento:

```
case out_enable is
    when '0' =>
        o_z0 <= "00000000";
        o_z1 <= "00000000";
        o_z2 <= "00000000";
        o_z3 <= "00000000";
    when '1' =>
        o_z0 <= reg3_z0(7 downto 0);
        o_z1 <= reg4_z1(7 downto 0);
        o_z2 <= reg5_z2(7 downto 0);
        o_z3 <= reg6_z3(7 downto 0);
    when others =>
        o_z0 <= "00000000";
        o_z1 <= "00000000";
        o_z2 <= "00000000";
        o_z3 <= "00000000";
end case;
```

Figure 10: Multiplexer

2.9 Macchina a stati finiti (FSM)

La macchina a stati finiti, implementata per il controllo del componente, è una macchina di Moore costituita da 6 stati. Di seguito si riporta la sua struttura e la funzione lambda:

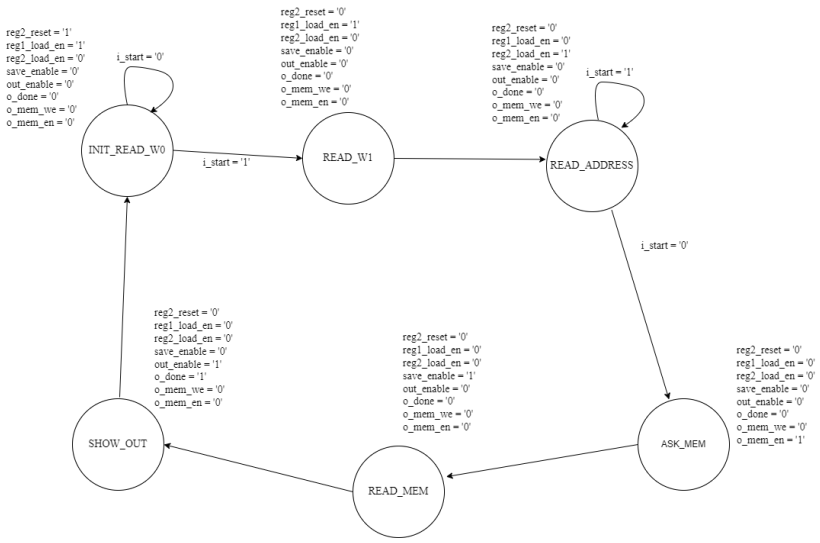


Figure 11: Macchina a stati finiti

Si riportano le operazioni svolte da ciascuno stato:

- INIT_READ_W0 : stato iniziale della macchina, si occupa di resettare il registro che contiene l'indirizzo di memoria mediante il segnale reg2_reset e di memorizzare il primo bit di intestazione nel registro reg1_int, abilitando la scrittura con il segnale reg1_load_en; finchè i_start è 0, l'FSM rimane in questo stato, la transizione avviene quando i_start passa a 1; è possibile salvare il primo bit già in questo stato perchè i_start passa a 1 nel fronte di discesa del clock, mentre il salvataggio avviene sul fronte di salita del clock;

- READ_W1 : stato in cui si legge il secondo bit dell'intestazione e lo si scrive nel registro reg1_int ponendo a 1 il segnale reg1_load_en;

- READ_ADDRESS : stato in cui vengono salvati i bit, che costituiscono l'indirizzo di memoria, nel registro reg2_addr, abilitando la scrit-

tura con il segnale reg2_load_en; finchè i_start è 1, la macchina rimane in questo stato, la transizione avviene quando i_start passa a 0;

- ASK_MEM : stato in cui si abilita il segnale o_mem_en per leggere dalla memoria;
- READ_MEM : stato in cui si abilita il segnale save_enable per salvare il valore ottenuto dalla memoria nel registro d'uscita corretto;
- SHOW_OUT : stato in cui, abilitando il segnale out_enable, si trasmette sulle uscite il valore corretto, abilitando anche il segnale o_done;

Il segnale che mantine lo stato corrente della FSM è:

```
type S is (INIT_READ_W0, READ_W1, READ_ADDRESS, ASK_MEM, READ_MEM, SAVE_SHOW_OUT);
signal curr_state : S;
```

Figure 12: Stato corrente e stati della FSM

3 Risultati sperimentali

Il linguaggio utilizzato per realizzare il progetto è *VHDL behavioral*. Vi è un'unica *entity* e una collezione di *process* che implementano l'architettura sopra descritta. Il progetto sintetizza ed implementa correttamente, inoltre passa tutti i test ai quali è stato sottoposto.

3.1 Sintesi

Di seguito si riporta la tabella ottenuta mediante comando *report_utilization* in post sintesi:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	27	0	0	134600	0.02
LUT as Logic	27	0	0	134600	0.02
LUT as Memory	0	0	0	46200	0.00
Slice Registers	53	0	0	269200	0.02
Register as Flip Flop	53	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figure 13: Tabella Flip Flop post sintesi

Come si può notare dalla tabella soprastante, non sono presenti Latch, ed il numero di Flip Flop ottenuto (53) è coerente con la progettazione effettuata, infatti:

- per le uscite, vi sono 4 registri da 8 bit ciascuno, quindi 32 Flip Flop;
- per l'intestazione, vi è un registro da 2 bit, quindi 2 Flip Flop;
- per l'indirizzo, vi è un registro da 16 bit, quindi 16 Flip Flop;
- 3 Flip Flop per i 6 possibili stati dell'FSM;

I 53 Flip Flop sono tutti di tipo D (in particolare FDCE), con reset asincrono. Lo schematico risultante è il seguente:

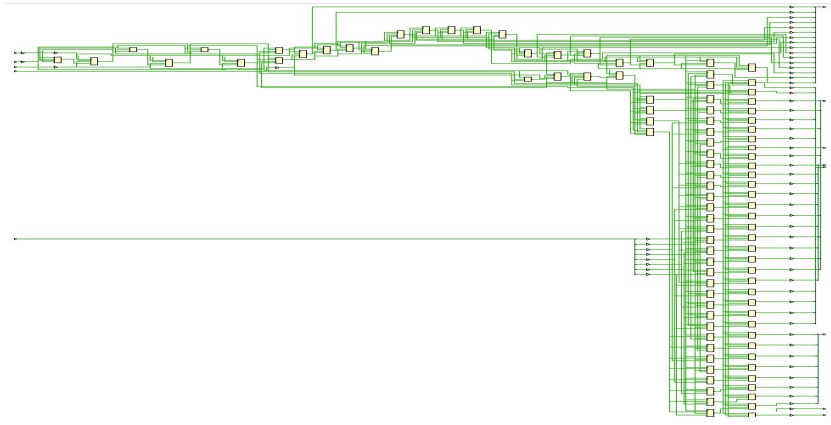


Figure 14: Schematico post sintesi

Da tale schematico, si possono identificare: in alto a sinistra i 3 Flip Flop per lo stato della FSM con i relativi LUT; a seguire, 2 LUT per la gestione del registro reg2_addr ed i 16 Flip Flop di tale registro; 2 Flip Flop ed i relativi LUT per il registro reg1_int che contiene l'intestazione; 4 LUT per la gestione dei registri d'uscita; dall'alto verso il basso, i 32 Flip Flop dei registri delle uscite ed i LUT relativi ai segnali di uscita (o_z0, o_z1, o_z2, o_z3). Il *report_timing* mostra uno Slack (differenza tra tempo richiesto e tempo di arrivo) di 97,010 ns.

```
Max Delay Paths
--------------------------------------------------------------------------
Slack (MET) :             97.010ns  (required time - arrival time)
```

Figure 15: Slack post sintesi

Infine, il *Device* generato in post sintesi è il seguente:

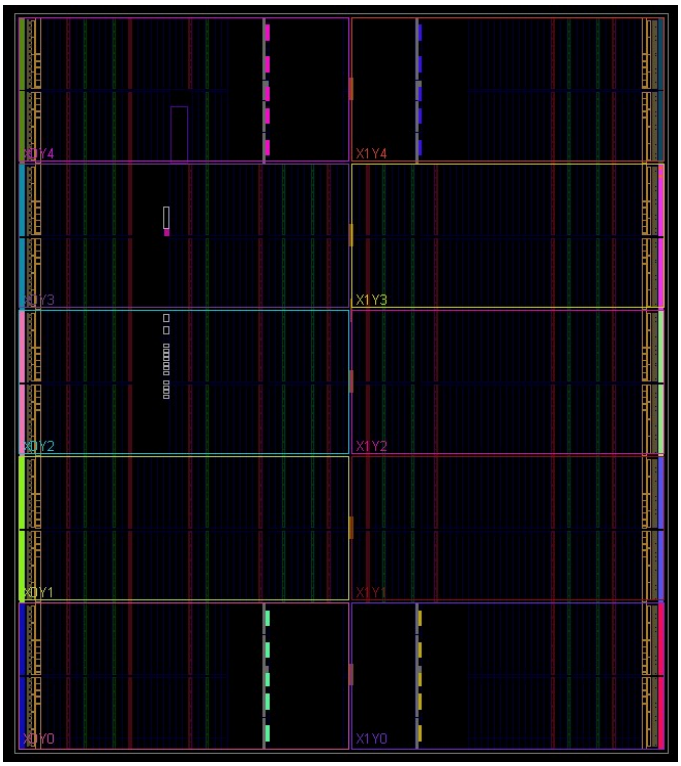


Figure 16: Device post sintesi

3.2 Implementazione

Mediante comando *report_utilization*, in post implementazione si è ottenuta la seguente tabella:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	23	0	0	134600	0.02
LUT as Logic	23	0	0	134600	0.02
LUT as Memory	0	0	0	46200	0.00
Slice Registers	67	0	0	269200	0.02
Register as Flip Flop	67	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Figure 17: Tabella Flip Flop post implementazione

Si noti come, anche in questo caso, non ci sono Latch. Il numero di Flip Flop, rispetto alla sintesi, è aumentato di 14 poichè l'uscita dei Flip Flop del registro reg2_addr è mappata con o_mem_addr. Questi nuovi Flip Flop si trovano nella parte alta del seguente schematico:

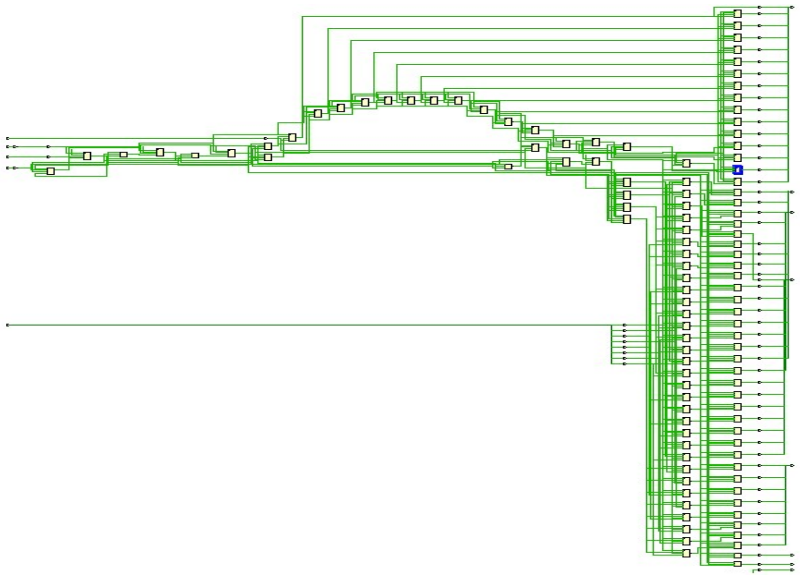


Figure 18: Schematico post implementazione

Il *report_timing* è il seguente:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	95,766 ns	Worst Hold Slack (WHS):	0,191 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	126	Total Number of Endpoints:	126	Total Number of Endpoints:	68

All user specified timing constraints are met.

Figure 19: Slack post implementazione

Infine, il *Device* generato dall'implementazione è il seguente:

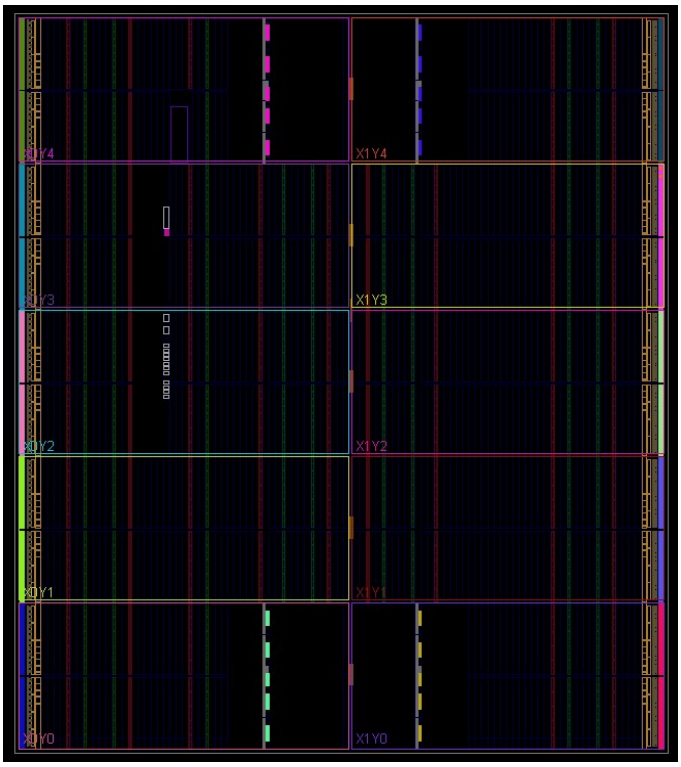


Figure 20: Device post implementazione

4 Simulazioni

Per testare il componente progettato, sono stati effettuati molteplici test per verificarne l'integrità ed il corretto funzionamento. In particolare, tali

test si sono concentrati sulla verifica dei casi limite individuati, come per esempio in condizioni di indirizzo pieno o vuoto in input. È stato testato anche in condizioni ritenute standard per verificare se il funzionamento fosse conforme alle specifice. Di seguito, si riportano i test piu' significativi.

4.1 Simulazione 1: Empty Address

L'obiettivo di questo test è verificare il comportamento del componente quando in input l'indirizzo di memoria è vuoto, cioè il segnale i_start dura solo per due cicli di clock ed i due bit in input costituiscono l'intestazione. L'indirizzo di memoria risultante deve essere 0, dato dall'inizializzazione a 0 di tutti i Flip Flop del registro contenente l'indirizzo. Come si può notare dalla figura sottostante, l'andamento ottenuto rispecchia le aspettative, in quanto l'indirizzo al quale si preleva il valore è l'indirizzo 0 (mem_address).

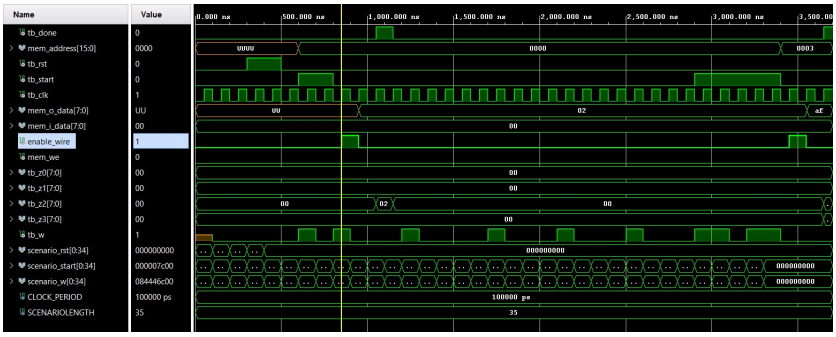


Figure 21: Test Empty Address

4.2 Simulazione 2: Full Address

L'obiettivo di questo test è verificare il comportamento del componente quando l'indirizzo di memoria in input è quello massimo (ffff). Infatti, si noti come il segnale mem_address, ad ogni ciclo di clock, sia incrementato fino a raggiungere il valore massimo.

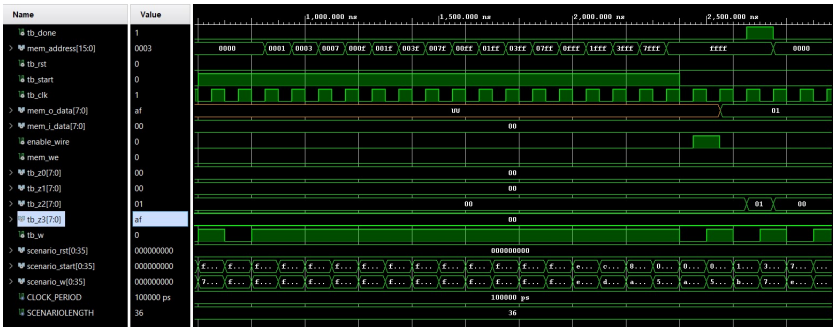


Figure 22: Test Full Address

4.3 Simulazione 3: All Outs and 1 bit Address

L'obiettivo di questo test è verificare che il componente riesca a scrivere su tutte le uscite. Infatti, come si nota dalla simulazione, alla quarta scrittura, le uscite risultano tutte con un valore diverso da 0. Inoltre, la quarta scrittura produce sull'uscita z1 il valore preso da memoria corrispondente all'indirizzo 1, verificando così anche il comportamento del componente quando in input si presenta un solo bit per l'indirizzo.

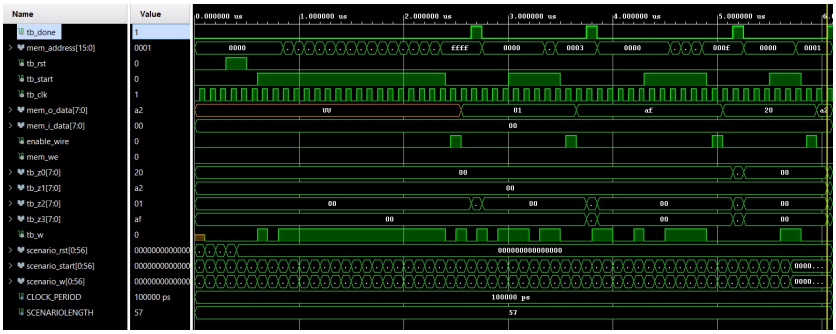


Figure 23: Test All Outs and 1 bit Address

4.4 Simulazione 4: All Outs with Overwrite

L'obiettivo di questo test è verificare la scrittura di tutte le quattro uscite, con successiva riscrittura per tutte. Dalla simulazione, si noti come il segnale o_done sia posto a 1 per 8 volte, quindi vi sono 8 scritture sulle uscite: prima si scrive un valore in ciascuna delle uscite, poi si sovrascrivono.

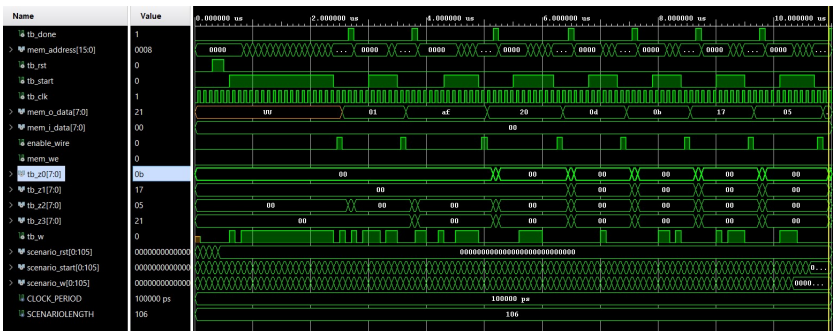


Figure 24: Test All Outs with Overwrite

4.5 Simulazione 5: Reset in any state of FSM

L'obiettivo dei test seguenti è verificare il corretto reset del componente in ogni stato della FSM. Infatti, il segnale di reset viene mandato a 1 per ogni stato della FSM. Di seguito le immagini di ogni simulazione di reset effettuata.

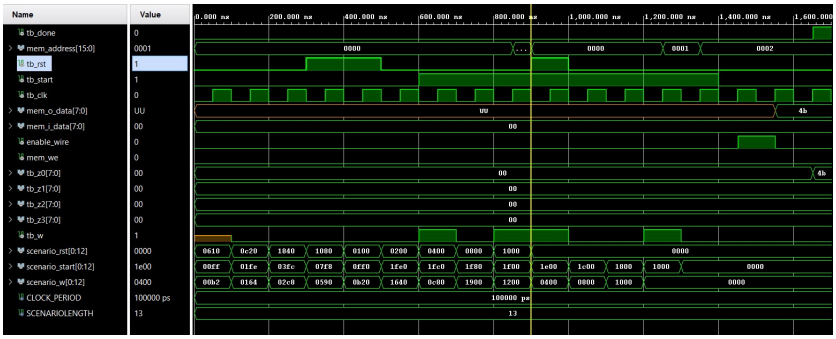


Figure 25: Test Reset in Start states (INIT_READ_W0, READ_W1, READ_ADDRESS)

Nella Figure 25, si può notare il segnale di reset alto quando anche il segnale di start è alto, quindi si è in fase di lettura da input. Dopo il segnale di reset, le uscite ed i registri risultano resettati e si riprende a leggere dall'input. Il valore letto dopo il reset è 00 per l'intestazione (quindi uscita z0) e 2 per l'indirizzo (mem_address assume valore 0002), quindi il comportamento è corretto perchè mostra il valore ottenuto da memoria all'indirizzo 0002 sull'uscita z0.

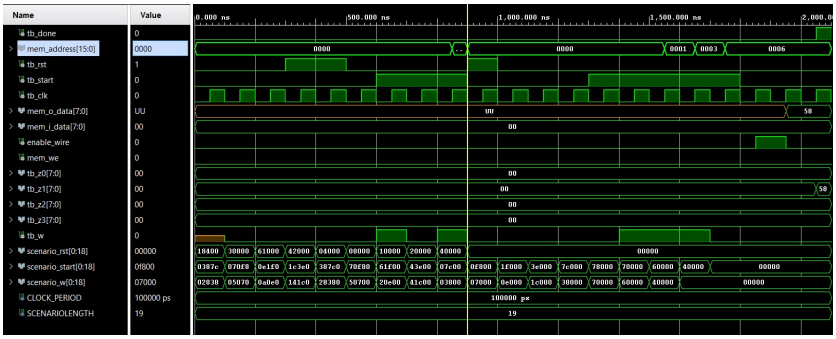


Figure 26: Test Reset in ASK_MEM

Nella Figure 26, si nota che il segnale di reset è alto subito dopo la lettura completa dell'input, quindi nello stato della FSM che gestisce la lettura da memoria. A causa del reset, i registri sono resettati, incluso l'indirizzo di memoria precedentemente salvato. Infatti, l'indirizzo mem_address è diventato 0 a causa del reset, proprio come ci si aspettava. Segue la lettura di un nuovo input e una scrittura dell'uscita corrispondente in base a tale input.

Figure 27: Test Reset in READ_MEM

Nella Figure 27, il reset avviene nello stato della FSM in cui viene salvato il valore ottenuto da memoria nel registro d'uscita corretto. Si noti come l'indirizzo di memoria (mem_address) sia resettato, mentre i registri di uscita non variano il loro valore perchè il salvataggio di tale valore avviene sul fronte di salita del clock, mentre il reset avviene sul fronte di discesa, dunque il valore non è stato ancora salvato. Segue la lettura di un nuovo

input e una scrittura dell'uscita corrispondente in base a tale input.

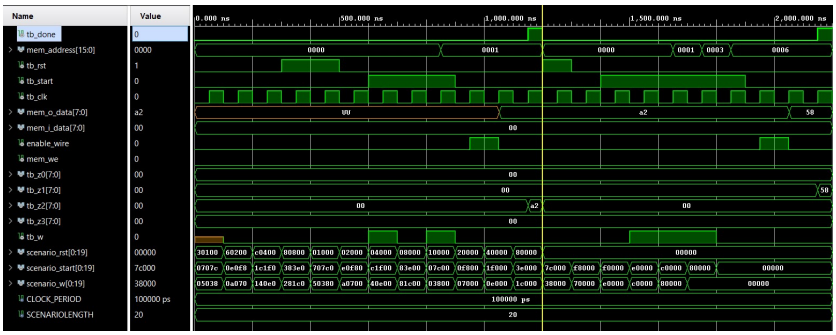


Figure 28: Test Reset in SHOW_OUT

Nella Figure 28, il reset è alto quando il segnale o_done è sul fronte di discesa, quindi il componente deve aver mostrato il valore preso da memoria sull'uscita corretta e subito dopo resettare tutto. Dall'immagine, si noti come il valore sia effettivamente mostrato sull'uscita e subito dopo avviene il reset, infatti, nell'iterazione successiva, l'uscita che prima aveva tale valore ora mostra 0. Si può concludere che il componente progettato si comporta correttamente.

4.6 Simulazione 6: Long Scenario

Questo test è stato fatto semplicemente per testare il componente quando lo scenario di input è lungo, per verificare se in un tempo piu' lungo rispetto ai test precedenti, il componente si comporta correttamente.

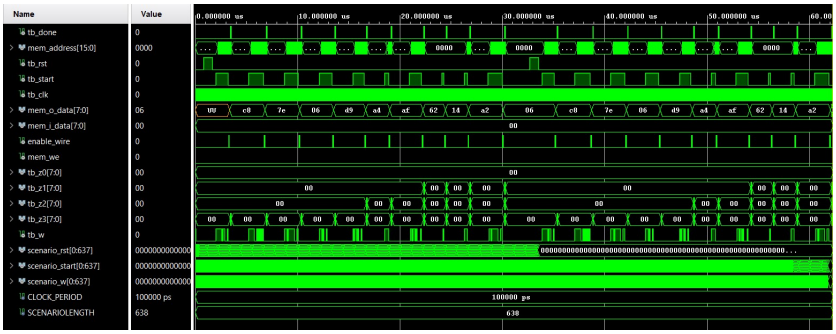


Figure 29: Test Long Scenario

5 Conclusioni

Per realizzare questo progetto, si è reso necessario effettuare una meticolosa ricerca della soluzione, caratterizzata da una fase di studio circa la specifica e le varie alternative disponibili. Dopo aver individuato la soluzione migliore, essa è stata scritta in VHDL. Successivamente, ulteriori modifiche sono state effettuate tenendo conto dei risultati ottenuti dai vari test e simulazioni, giungendo così alla versione definitiva ed efficace.
È stato un percorso che ha contribuito ad arricchire il personale know-how in materia poichè ha permesso di mettere in pratica ciò che era stato studiato precedentemente, permettendo di osservare come la teoria della progettazione digitale si leghi alla realizzazione concreta dei sistemi digitali.
Inoltre, la fase di progettazione della soluzione è stata utile per migliorare l'approccio generale per affrontare problemi di varia natura, soprattutto informatica.
Infine, considerando il lavoro svolto, si ritiene che il componente progettato, con relativa architettura ed FSM, rispetti le specifiche assegnate e si comporti correttamente alla luce dei test effettuati sia in *Behavioral Simulation* che in *Post-Synthesis Functional Simulation*.