Progetto di Reti Logiche

Anno Accademico 2022-2023

Prof. Fabio Salice – Prof. Federico Terraneo

Giammusso Samuele

POLITECNICO
MILANO 1863

Sommario

1. Introduzione

1.1 Obiettivo del progetto

L'obiettivo del progetto è implementare in linguaggio VHDL un modulo in grado di interpretare i dati forniti in input, da ingressi seriali, per poter interrogare una memoria da cui ricavare i dati da mostrare in output su uscite parallele.

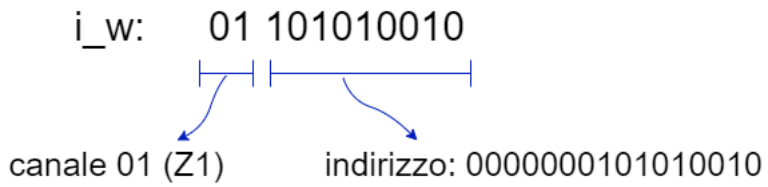
Per ottenere tale risultato, inizialmente è necessario calcolare su quale, tra le 4 porte di uscita disponibili, bisognerà mostrare il dato nuovamente ricevuto dalla memoria.

Secondariamente è necessario calcolare l'indirizzo con cui interrogare la memoria.

Infine, è richiesto di mostrare sulle 4 uscite i dati ricevuti da memoria, tenendo conto anche delle precedenti letture da memoria.

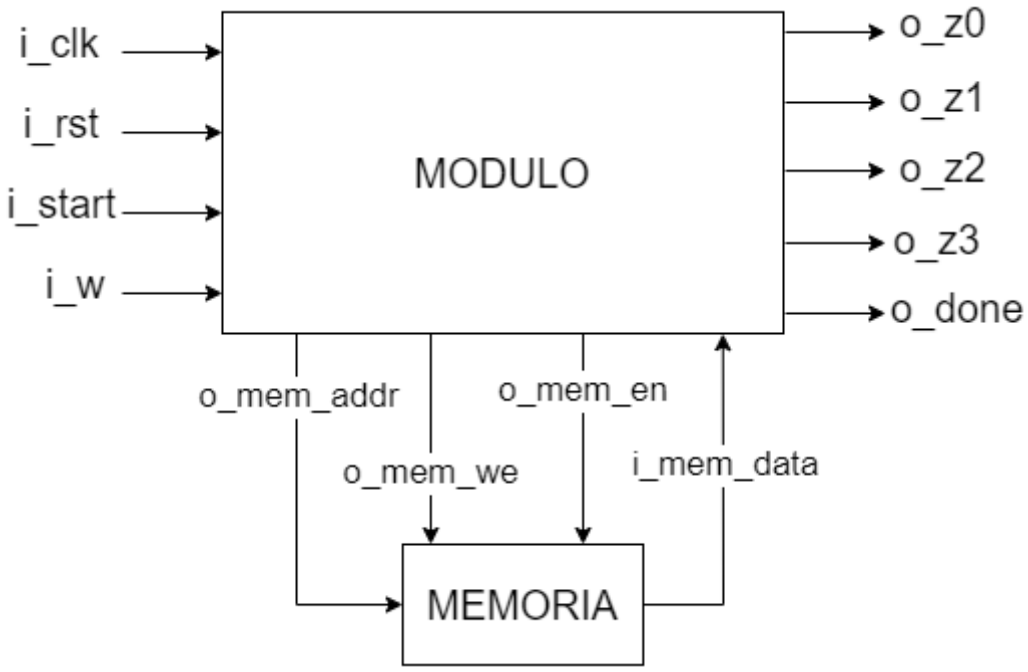
Questo procedimento viene ripetuto ogniqualvolta si ricevono in ingresso dei bit utili per il calcolo della porta d'uscita e dell'indirizzo.

Segue un esempio di come vengono interpretati i bit forniti in ingresso:



I primi due bit indicano su quale canale d'uscita bisognerà mostrare il dato letto da memoria, i bit successivi costituiscono l'indirizzo da 16 bit usato per interrogare la memoria.

1.2 Interfaccia del componente



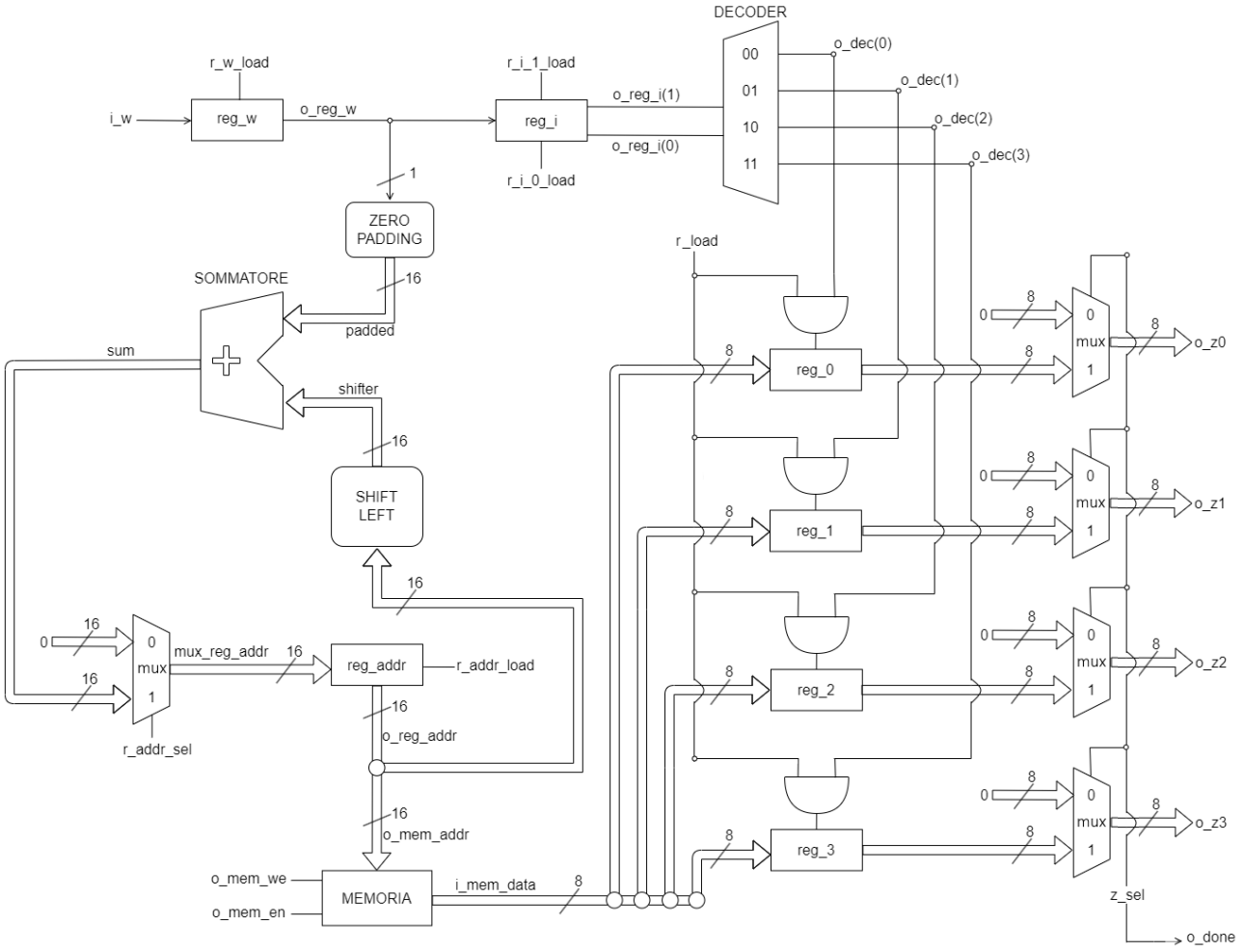
In particolare:

- 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 da 8 bit ciascuno;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale (vettore da 16 bit) di uscita che manda l'indirizzo alla memoria;
- i_mem_data è il segnale (vettore da 8 bit) 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 poter scriverci. Per leggere da memoria esso deve essere 0.

2. Architettura

2.1 Datapath

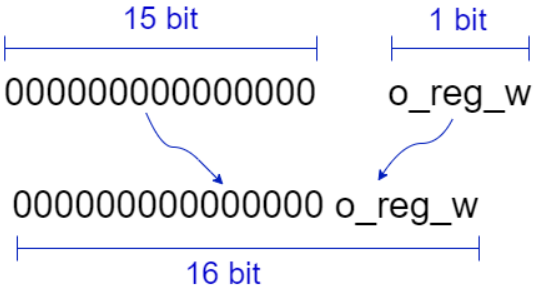
- Rappresentazione dello schema funzionale del Modulo top-level:



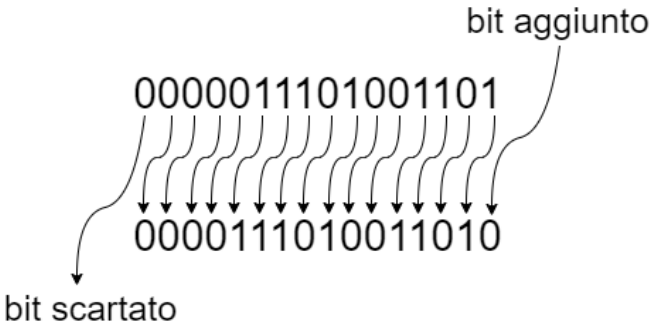
- Scelte implementative:

In ingresso viene posto un registro **reg_w** in quanto permette di ritardare il segnale i_w di 1 ciclo di clock. Infatti, utilizzando una macchina di Moore (vedere capitolo 2.3), l'uscita dipende dallo stato, ma lo stato dipende dal valore di i_start, quindi quando per esempio i_start passa da '0' a '1' si effettua un cambio di stato, ma così facendo si perde il primo bit di i_w, e da qui nasce la necessità di dover ritardare il segnale di 1 ciclo di clock.

Lo **Zero Padding** effettua la seguente operazione prima che venga eseguita la somma: il bit in ingresso viene esteso con 15 bit a '0' nella parte più significativa.



Lo **Shift Left** effettua la seguente operazione prima che venga eseguita la somma: ai 15 bit meno significativi di o_reg_addr viene aggiunto uno '0' nella parte meno significativa, ottendendo così uno shift dei bit di una posizione. Il MSB di o_reg_addr può essere scartato perché da specifica l'indirizzo di memoria può essere al massimo da 16 bit, e al ricevimento del 16-esimo bit si interrompe il calcolo.



Il registro **reg_addr** viene utilizzato per salvare progessivamente l'esito del calcolo della somma, che equivale al corrente valore dell'indirizzo di memoria. Nel momento in cui il segnale i_start passa da '1' a '0', il segnale r_addr_load si abbassa e quindi in o_reg_addr resta salvato l'indirizzo di memoria esatto. Il registro è preceduto da un mutex che permette di riportare il dato salvato a '0' all'inizio di ogni ciclo di calcolo dell'indirizzo.

Il registro **reg_i** salva nel vettore da 2 bit il primo e il secondo bit ricevuti in ingresso quando i_start = '1'. o_reg_i viene utilizzato come ingresso di selezione del decoder.

Per quanto riguarda la gestione del load nei registri reg_0, reg_1, reg_2, reg_3, la scelta è ricaduta sull'utilizzo di un **decoder**, il quale permette di avere un segnale d'uscita o_dec da 4 bit con codifica one-hot, ogniuno dei quali viene posto in AND con il segnale di load dei registri r_load. I primi due bit ricevuti in ingresso permettono, mediante il decoder, di portare a '1' uno dei quattro bit di o_dec. In questo modo, solamente quando entrambi i segnali sono alti (r_load e o_done(X) con X compresa tra 0 e 3), cioè nel momento in cui si intende leggere il dato da memoria (i_mem_data), il dato viene scritto in uno solo dei registri.

Le quattro uscite o_z0, o_z1, o_z2, o_z3 sono precedute da altrettanti **multiplexer** che permettono di selezionare se in uscita viene mostrato '0000 0000' oppure il contenuto salvato nei registri. Il segnale di selezione dei multiplexer z_sel viene inoltre utilizzato per comandare il segnale di o_done.

2.2 Segnali

- Sono stati definiti i seguenti segnali utilizzati dalla macchina a stati:

```
--SEGNALI DELLA MACCHINA A STATI
    --registri
    signal r_load : STD_LOGIC;
    signal r_addr_load : STD_LOGIC;
    signal r_i_0_load : STD_LOGIC;
    signal r_i_1_load : STD_LOGIC;
    signal r_w_load : STD_LOGIC;
    --multiplexer
    signal r_addr_sel : STD_LOGIC;
    signal z_sel : STD_LOGIC;
```

- Il segnale **r_load** viene utilizzato insieme ad o_dec per indicare quando il dato fornito da memoria deve essere salvato nei registri reg_0, reg_1, reg_2, reg_3.
- Il segnale **r_addr_load** viene utilizzato per indicare quando il dato fornito dal multiplexer (mux_reg_addr) deve essere salvato nel registro reg_addr.
- I segnali **r_i_0_load** e **r_i_1_load** vengono utilizzati rispettivamente per indicare quando il bit fornito in ingresso da o_reg_w deve essere salvato in o_reg_i(0) e in o_reg_i(1).
- Il segnale **r_w_load** viene utilizzato per indicare quando il bit fornito in input da i_w deve essere salvato nel registro. r_w_load resta sempre al valore '1' in quanto desidero che o_reg_w sia sempre uguale a i_w ma ritardato di 1 ciclo di clock.
- Il segnale **r_addr_sel** viene utilizzato per indicare quale tra i due ingressi del multiplexer portare in uscita. L'ingresso '0' contiene 16 bit a '0' e viene utilizzato per inizializzare reg_addr. L'ingresso '1' contiene i 16 bit uscenti dal sommatore.
- Il segnale **z_sel** viene utilizzato per indicare quale tra i due ingressi dei multiplexer portare in uscita. L'ingresso '0' contiene 8 bit a '0'. L'ingresso '1' contiene gli 8 bit salvati nei registri. Il segnale viene inoltre utilizzato per alzare il segnale di o_done.

- Sono stati definiti i seguenti segnali interni al datapath:

```
--SEGNALI DEL DATAPATH
    --registri
    signal o_reg0 : STD_LOGIC_VECTOR (7 downto 0);
    signal o_reg1 : STD_LOGIC_VECTOR (7 downto 0);
    signal o_reg2 : STD_LOGIC_VECTOR (7 downto 0);
    signal o_reg3 : STD_LOGIC_VECTOR (7 downto 0);
    signal o_reg_addr : STD_LOGIC_VECTOR (15 downto 0);
    signal o_reg_i : STD_LOGIC_VECTOR (1 downto 0);
    signal o_reg_w : STD_LOGIC;
    --somma
    signal sum : STD_LOGIC_VECTOR(15 downto 0);
    --zero padding
    signal padded : STD_LOGIC_VECTOR(15 downto 0);
    --left shifter
    signal shifter : STD_LOGIC_VECTOR(15 downto 0);
    --multiplexer
    signal mux_reg_addr : STD_LOGIC_VECTOR (15 downto 0);
    --decoder
    signal o_dec : STD_LOGIC_VECTOR (3 downto 0);
```

- I segnali **o_reg0, o_reg1, o_reg2, o_reg3** indicano rispettivamente i dati salvati nei registri reg_0, reg_1, reg_2, reg_3.
- Il segnale **o_reg_addr, o_reg_i, o_reg_w** indicano rispettivamente i dati salvati nei registri reg_addr, reg_i, reg_w.
- Il segnale **sum** è il risultato della somma tra padded e shifter.
- Il segnale **padded** è il risultato dello zero padding dei leftmost bits del segnale o_reg_w.
- Il segnale **shifter** è il risultato dello shift left di 1 posizione del segnale o_reg_addr.
- Il segnale **mux_reg_addr** indica il segnale in uscita dal multiplexer e dipende dal valore di r_addr_sel.
- Il segnale **o_dec** indica il segnale in uscita dal decoder, è caratterizzato da una codifica one-hot e dipende dal segnale di selezione o_reg_i.

- I segnali assumono i seguenti valori all'avvio:

```
--memoria
o_mem_we <= '0';
o_mem_en <= '0';
--registri
r_load <= '0';
r_addr_load <= '0';
r_i_0_load <= '0';
r_i_1_load <= '0';
r_w_load <= '1';
--multiplexer
r_addr_sel <= '0';
z_sel <= '0';
```

- I segnali di controllo della memoria sono entrambi a '0' perché non si intende leggere o scrivere da memoria.
- I segnali di load dei registri sono a '0' perché non si intende scrivere nei registri, ad eccezione di r_w_load, il quale resta ad '1' per tutto il periodo di funzionamento del modulo.
- I segnali di controllo dei multiplexer sono a '0', in particolare z_sel è '0' perché, ad eccezione di quando o_done = '1', per tutto il tempo restante si vuole mostrare sulle 4 uscite solamente '0000 0000'.

2.3 Macchina a stati

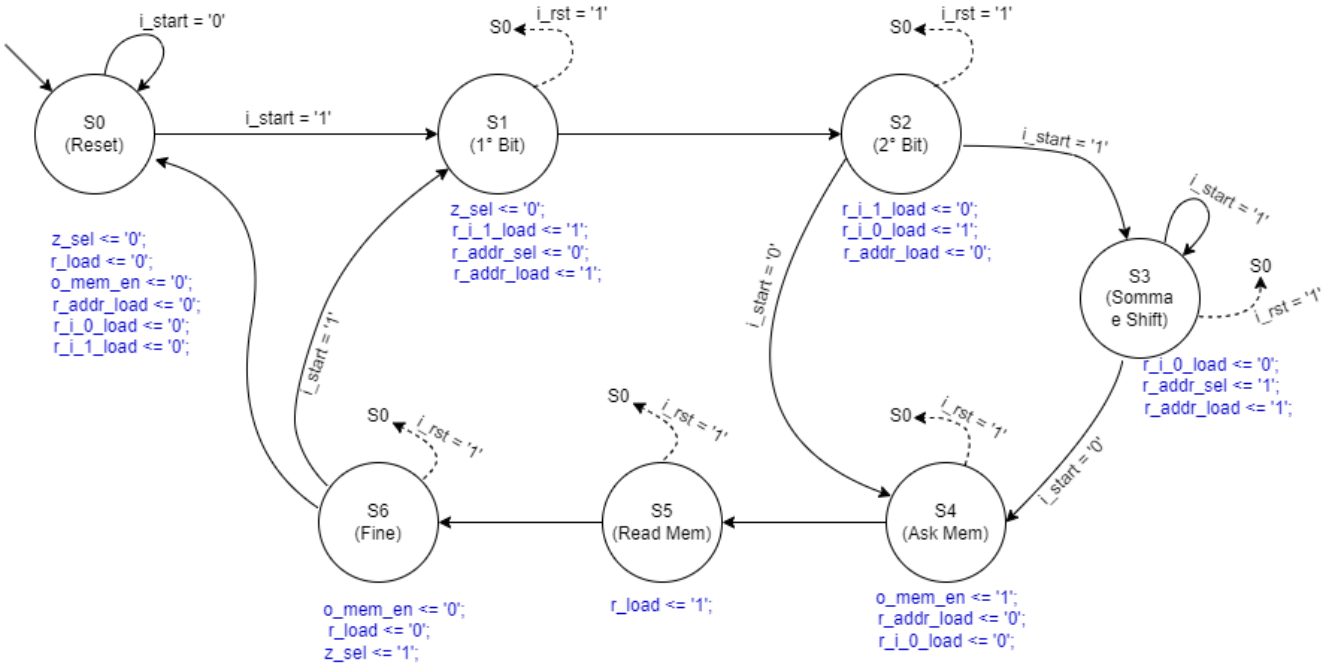
- Per la realizzazione della macchina a stati sono stati definiti 3 process:

- Il primo viene utilizzato per decidere se il prossimo stato sarà quello scelto dalla funzione di stato prossimo oppure S0 in caso di i_rst = '1'.

- Il secondo definisce la funzione di stato prossimo.

- Il terzo definisce la funzione di uscita.

- Il componente è stato progettato con una macchina di Moore, composta da 7 stati:



Ogni stato, ad eccezione di S0, ha un arco tratteggiato uscente con la condizione i_rst = ‘1’ che porta lo stato corrente allo stato S0. Questa condizione viene utilizzata nell’eventualità in cui si riceva in ingresso un reset asincrono, cioè mentre l’automa non si trova nello stato S0, e quindi il modulo deve essere re-inizializzato.

Nell’automa, in corrispondenza di ogni stato, sono stati specificati i segnali della funzione di uscita che cambiano. In particolare, nello stato S0 vengono azzerati tutti i segnali che potrebbero rimanere attivi a seguito di un reset asincrono, provenendo da uno qualsiasi degli altri stati.

- Descrizione degli stati

- Stato **S0**: è lo stato di reset, e pertanto la macchina a stati lo raggiunge quando riceve in ingresso i_rst =’1’, quindi è anche lo stato di partenza. Fintantoché i_start = ‘0’ si resta in questo stato in attesa di ricevere i_start = ‘1’ per poi spostarsi nello stato S1.

- Stato **S1**: durante questo stato si legge il MSB dei 2 bit che indicano su quale uscita scrivere i dati. Il bit viene scritto in o_reg_i(1). Inoltre, il registro reg_addr viene inizializzato a 0. Successivamente la funzione di stato prossimo si sposta incondizionatamente in S2.

- Stato **S2**: durante questo stato si legge il LSB dei 2 bit che indicano su quale uscita scrivere i dati. Il bit viene scritto in o_reg_i(0). Se a questo punto si riceve i_start = ‘0’, significa che l’indirizzo sarà composto da tutti zeri perché non verrà fornito nessun altro bit in ingresso, e la funzione di stato prossimo si sposta in S4. Se invece i_start = ‘1’, la funzione di stato prossimo si sposta in S3.

- Stato **S3**: fintantoché i_start = ‘1’ la macchina a stati resta in S3. In questo stato si leggono i bit che compongono l’indirizzo di memoria, dal MSB al LSB. Il bit ricevuto in ingresso viene sommato con l’indirizzo calcolato fino a quel momento shiftato di 1 posizione. Il calcolo continua finché non si riceve in ingresso i_start = ‘0’, il quale indica la fine del ricevimento dei bit e la funzione di stato prossimo si sposta in S4.

- Stato **S4**: durante questo stato viene interrogata la memoria con l’indirizzo precedentemente calcolato. Successivamente la funzione di stato prossimo si sposta incondizionatamente in S5 per poter leggere il dato (viene fornito nello stato successivo perché la memoria è sincrona: infatti la memoria commuta stato dopo un ciclo di clock).

- Stato **S5**: durante questo stato viene letto il dato fornito dalla memoria. Il segnale r_load viene portato ad ‘1’, ma la scrittura avviene solamente in uno dei 4 registri in quanto dipende anche dal segnale one-hot o_dec. Successivamente la funzione di stato prossimo si sposta incondizionatamente in S6.

- Stato **S6**: in questo stato si pone z_sel=’1’, quindi si alza il segnale di o_done e contemporaneamente vengono mostrate sulle 4 uscite o_z i dati salvati nei registri. Se a questo punto si riceve i_start = ‘1’ (questo caso da specifica non si verifica perché vengono garantiti 20 cicli di clock con i_start = ‘0’ dopo l’ultimo i_start = ‘1’, però viene inserito per completezza) allora la macchina a stati si sposta nello stato S1 per leggere i nuovi bit in ingresso, altrimenti si sposta incondizionatamente in S0.

3. Risultati Sperimentali

3.1 Sintesi

3.1.1 Utilization report

Il componente viene sintetizzato correttamente, utilizza 37 Look Up Table, 54 Flip Flop e 0 Latch, quindi non viene inferito nessun latch.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	37	0	134600	0.03
LUT as Logic	37	0	134600	0.03
LUT as Memory	0	0	46200	0.00
Slice Registers	54	0	269200	0.02
Register as Flip Flop	54	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.1.2 Timing report

Utilizzando un ciclo di clock di 100 ns, si ottiene uno slack di 97.463 ns, quindi rientra ampiamente nel margine di clock richiesto.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	97,463 ns	Worst Hold Slack (WHS):	0,139 ns	Worst Pulse Width Slack (WPWS):	4,500 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:	72	Total Number of Endpoints:	72	Total Number of Endpoints:	55

All user specified timing constraints are met.

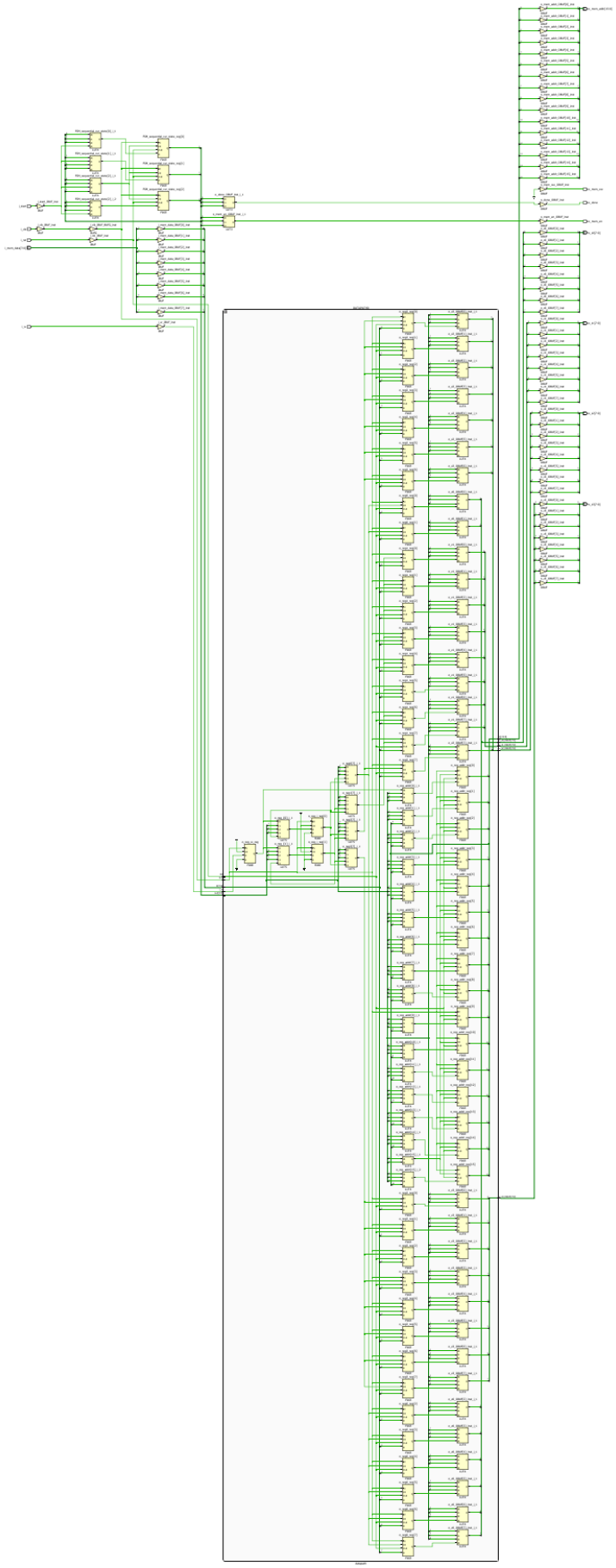
È possibile calcolare il periodo di funzionamento:

T = Tclk – WNS = 2.537 ns

Da cui si ottiene la frequenza di funzionamento:

$f = \frac{1}{Tclk}$ = 394.17 MHz

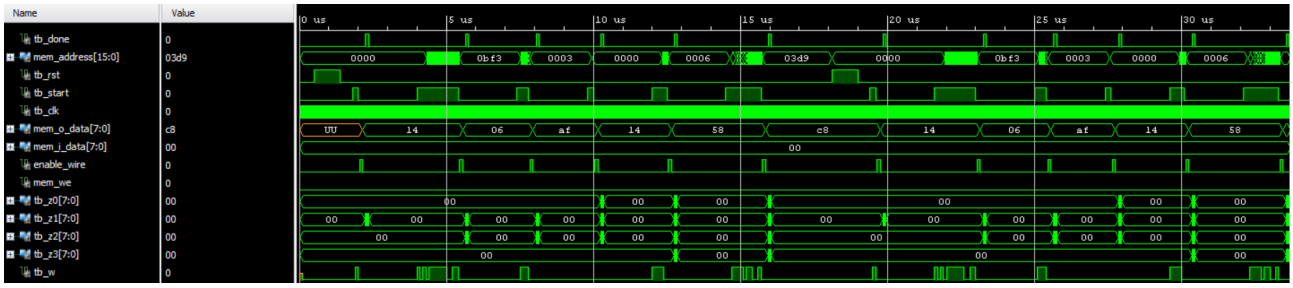
3.1.3 Schematico post-sintesi



3.2 Simulazioni

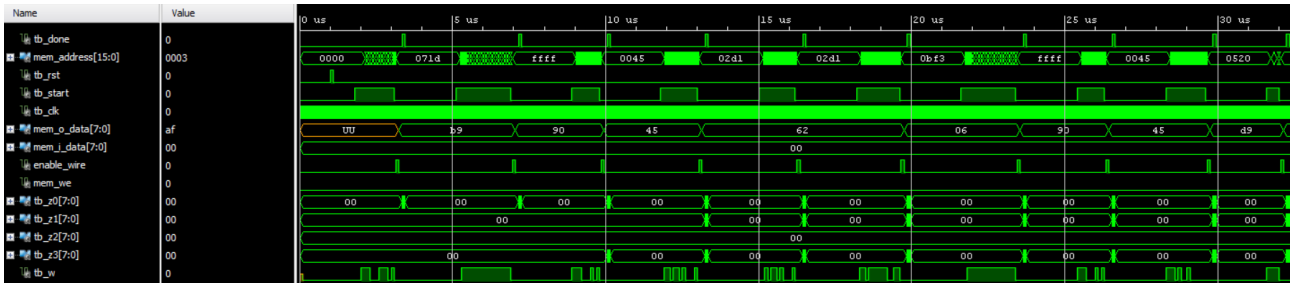
3.2.1 Test Bench 1

Il primo test bench inizia verificando che sulle uscite vengano mostrati determinati valori e verifica riscritture di valori diversi sulle stesse uscite, successivamente a metà del test avviene un reset, ed infine vengono verificate di nuovo gli stessi valori verificati nella prima metà del test. Quindi il primo test verifica anche che in seguito ad un reset, il modulo venga re-inizializzato correttamente.



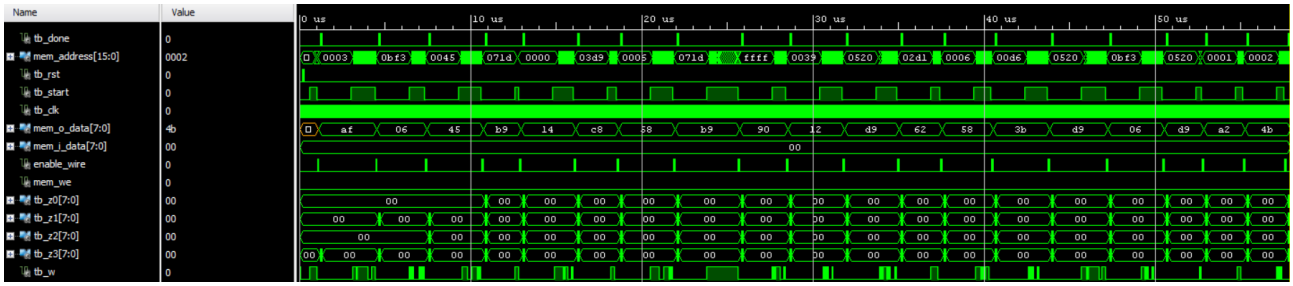
3.2.2 Test Bench 2

Il secondo test bench verifica che sulle uscite vengano mostrati determinati valori e verifica riscritture di valori diversi sulle stesse uscite, senza scrivere mai su Z2; inoltre viene verificato il caso della riscrittura dello stesso valore dove era già presente.



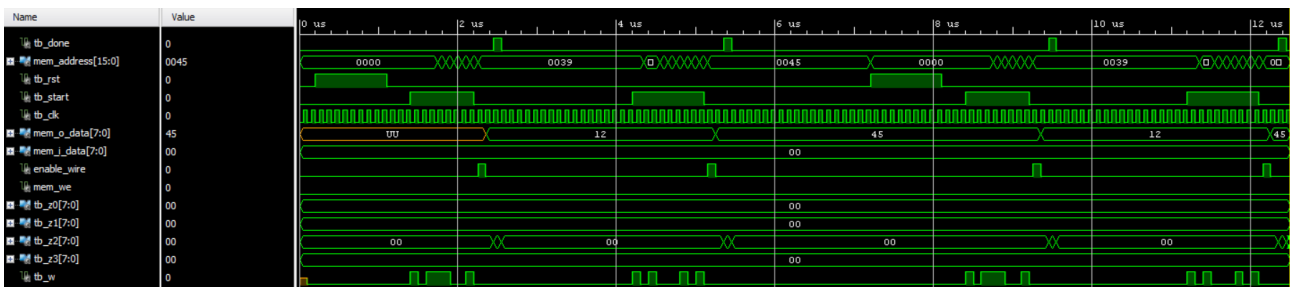
3.2.3 Test Bench 3

Il terzo test bench verifica prima che sulle uscite vengano mostrati determinati valori e successivamente verifica le riscritture di valori diversi sulle stesse uscite.



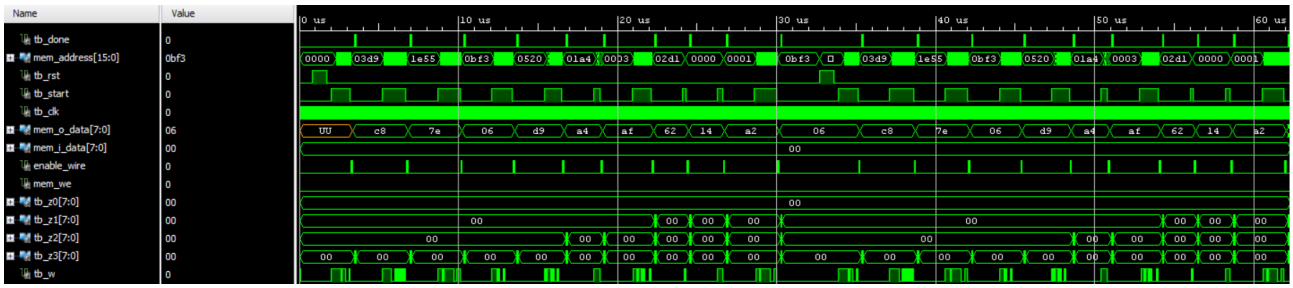
3.2.4 Test Bench 4

Il quarto test bench verifica che vengano scritti in ordine i valori 18, 69, 18, 69 sulla uscita Z2.



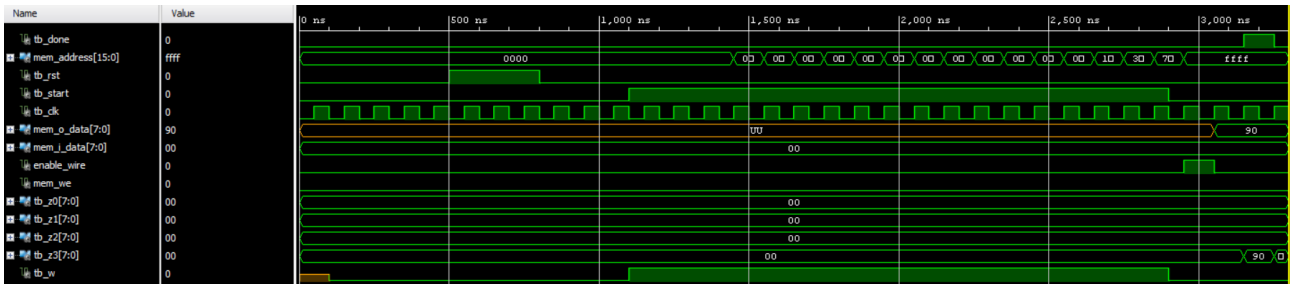
3.2.5 Test Bench 5

Il quinto test bench inizia verificando che sulle uscite vengano mostrati determinati valori e verifica riscritture di valori diversi sulle stesse uscite, successivamente a metà del test avviene un reset, ed infine vengono verificate di nuovo gli stessi valori verificati nella prima metà del test. Quindi il quinto test verifica anche che in seguito ad un reset, il modulo venga re-inizializzato correttamente.



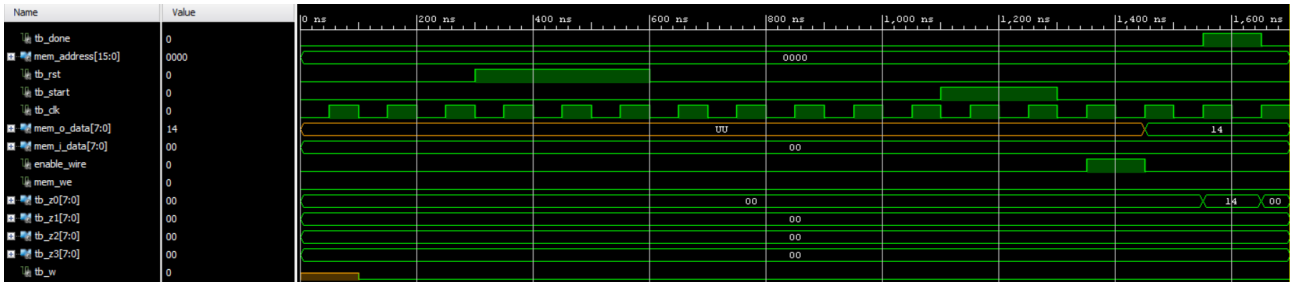
3.2.6 Test Bench 6

Il sesto test bench verifica il caso particolare in cui vengano forniti in ingresso 18 bit ad ‘1’ sia sull’ingresso i_start che sull’ingresso i_w. Quindi verifica che sull’uscita = “11” = Z3, venga mostrato ciò che è salvato all’indirizzo di memoria = “1111 1111 1111 1111”, cioè il valore 400.



3.2.7 Test Bench 7

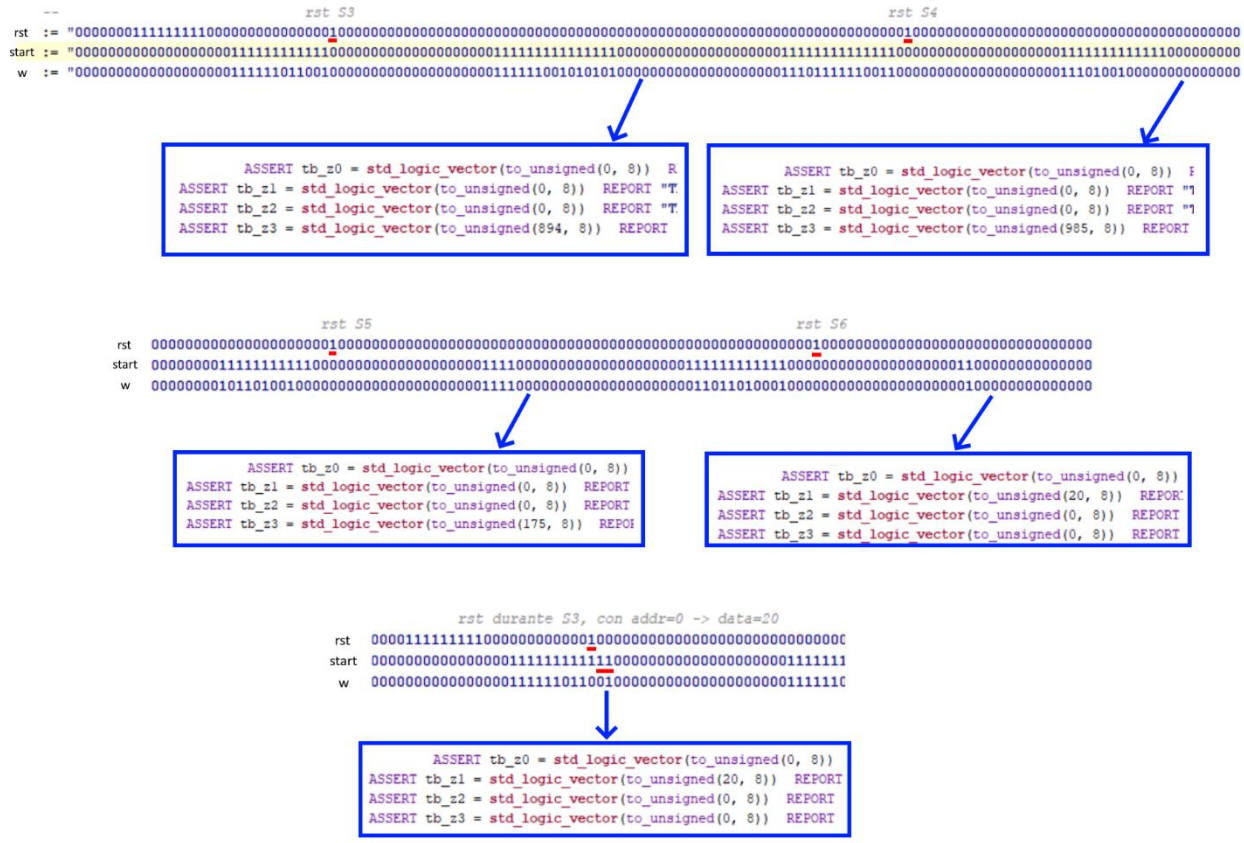
Il settimo test bench verifica il caso particolare in cui vengano forniti in ingresso 2 bit ad ‘1’ sull’ingresso i_start e in contemporanea 2 bit a ‘0’ sull’ingresso i_w. Quindi verifica che sull’uscita = “00” = Z0, venga mostrato ciò che è salvato all’indirizzo di memoria = “0000 0000 0000 0000”, cioè il valore 20. Quindi questo test bench verifica che nel caso limite in cui non venga fornito in ingresso un indirizzo di memoria, venga comunque interrogata la memoria all’indirizzo con 16 bit a ‘0’.



3.2.8 Test Bench Reset Asincrono

In aggiunta ho creato il seguente test bench per verificare il corretto funzionamento del modulo in caso vengano forniti in ingresso dei reset asincroni.

Il test bench è stato creato modificando il test bench 5 ed aggiungendo un reset ogni 2 indirizzi forniti in ingresso, con lo scopo di verificare che effettivamente dopo un reset il modulo sia re-inizializzato e che venga mostrato in uscita solo il nuovo valore letto all’indirizzo di memoria fornito.



Il test verifica che se il modulo riceve un reset asincrono mentre la macchina a stati si trova nello stato S3, S4, S5 o S6 allora il modulo venga re-inizializzato correttamente.

Inoltre, viene verificato il caso in cui si riceva il reset asincrono durante il calcolo dell'indirizzo di memoria nello stato S3. Suppongo che dopo il reset ci siano almeno due cicli con i_start='1': in questo caso i bit in ingresso successivi al reset vengono considerati come i bit che indicano su quale uscita mostrare il dato salvato in memoria. I bit che seguono sull'ingresso i_start sono tutti '0' e pertanto l'indirizzo di memoria è "0000 0000 0000 0000", nel quale è salvato il valore 20.

4. Conclusioni

Il codice VHDL viene sintetizzato correttamente da VIVADO in un componente hardware senza inferire nessun latch, utilizzando come FPGA target xc7a200tfbg484-1.

Il progetto ha superato con esito positivo tutti i Test Bench sia in simulazioni comportamentali (Behavioral Simulation) che in simulazioni funzionali (Post Synthesis Functional Simulation).

Il test bench del reset asincrono mi ha permesso di verificare che il modulo sviluppato sia robusto anche in questa eventualità.