POLITECNICO
MILANO 1863

Progetto di Reti logiche

Prof William Fornaciari - Anno 2022/2023

Chiara Auriemma:
(Codice Persona: 10722613 - Matricola: 956170)
Giacomo Ballabio:
(Codice Persona: 10769576 - Matricola: 959913)

Indice

1 Introduzione

1.1 Scopo del progetto

Lo scopo è quello di implementare un modulo HW descritto in VHDL capace di interfacciarsi con una memoria. Più nel dettaglio il sistema deve ricevere indicazioni sulla locazione di memoria il cui contenuto deve essere indirizzato verso un canale di uscita fra quelli disponibili. L'ingresso W (ingresso seriale da un bit) ci fornirà le informazioni circa il canale scelto e l'indirizzo di memoria. Le uscite del sistema forniscono tutti i bit della parola di memoria in parallelo.

1.2 Interfaccia del componente

Il componente da descrivere deve avere la seguente interfaccia:

```
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;
```

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;
- 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 poter scriverci. Per leggere da memoria esso deve essere 0.

1.3 Funzionamento

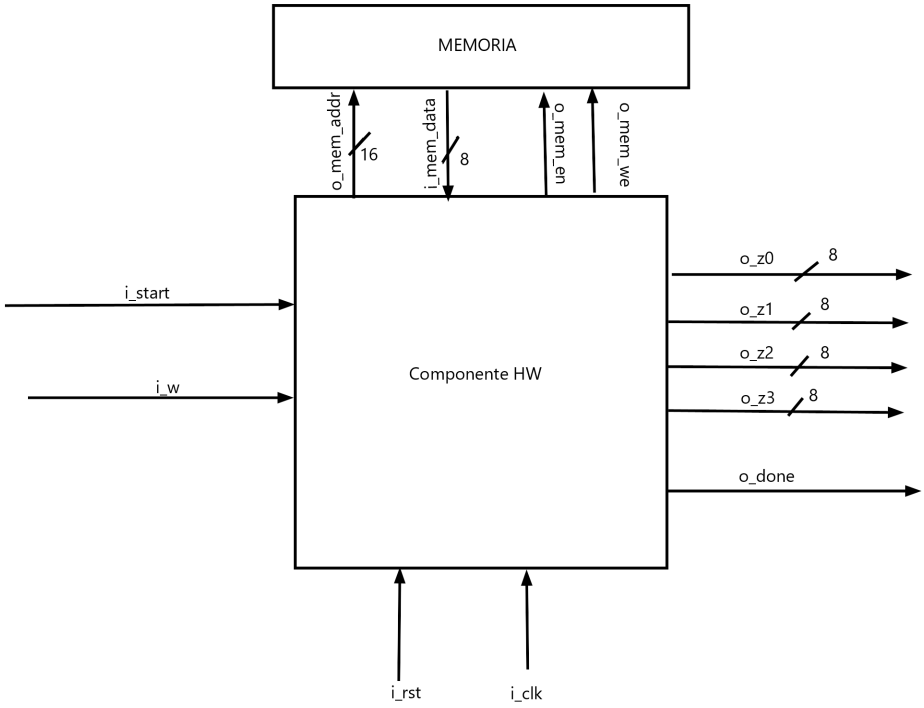

Figure 1: Schema di funzionamento

1.3.1 Spiegazione dello schema

All'istante iniziale o_z0, o_z1, o_z2, o_z3 sono a 0000 0000 e o_done è a 0. L'ingresso seriale i_w ci fornisce i dati nel seguente modo: i primi 2 bit indicano il canale di uscita scelto (00 identifica Z0, 01 identifica Z1, 10 identifica Z2 e, infine, 11 identifica Z3). Gli altri N bit indicano l'indirizzo di memoria.Gli N bit di indirizzo posso variare da 0 fino a un massimo di 16. Se N è inferiore a 16 l'indirizzo viene esteso con 0 sui bit più significativi. Tutti i bit su i_w devono essere letti sul fronte di salita del clock. La sequenza di ingresso è valida solo se i_start è uguale a 1, termina quando start diventa 0. Lo start rimane alto tra i 2 e i 18 clicli di clock. Una volta trovato il nostro indirizzo di memoria, ci accediamo e la memoria ci restituisce il contenuto che dovrà essere mandato sul

canale di uscita prescelto. Sugli altri canali vengo memorizzati i valori precedentemente inviati. Questi vengono visualizzati sono se o_done è alto. Il tempo massimo per produrre il risultato è 20 cicli di clock.

2 Architettura

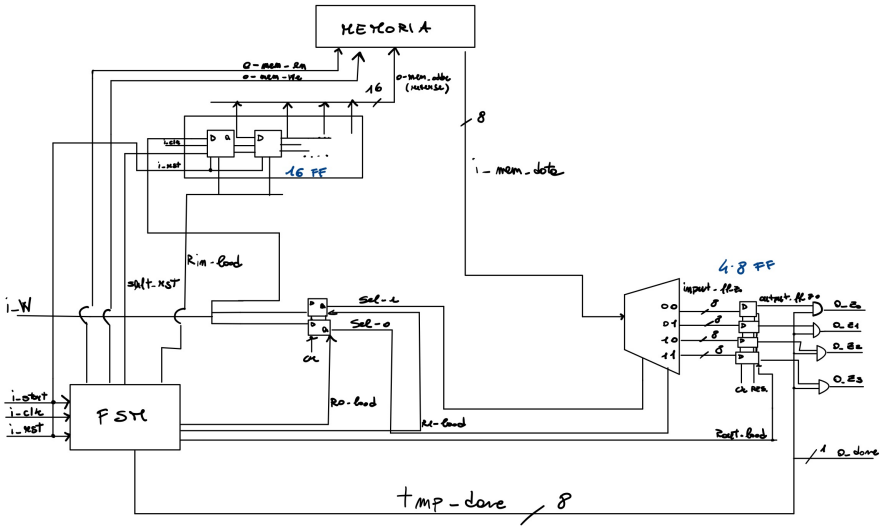

Figure 2: Architettura

2.1 Descrizione Architettura

Il componente è composto da una parte sequenziale e da una macchina a stati finiti di Moore che si occupa di abilitare i registri e definire le varie parti della computazione. Gli stati di quest'ultima vengono analizzati piú nel dettaglio nel prossimo paragrafo. Per gestire la scelta del canale in uscita i primi 2 bit in ingresso da i_w vengono indirizzati in due flip flop D abilitati da i segnali R1_load e R0_load. I segnali in uscita dai 2 flip flop si chiamano sel_1 e sel_0 ed entrano entrambi in un decoder , che grazie ai 2 bit in arrivo saprà dove direzionare i dati in arrivo da i_mem_data. Dopo i primi 2 bit , i_w inizia a comunicare l'indirizzo di memoria. Per salvarlo utilizziamo un registro composto da 16 flip flop D, che viene abilitato da un segnale chiamato Rin_load. Il registro è inzialmente pieno di 0 e man mano che i bit arrivano da i_w vengono posizionati uno alla volta sul bit meno significativo (quindi nell'ultimo flip flop

del registro) e i precedenti scorrono in avanti. Ciò è possibile prendendo i 15 bit meno singificativi e concatenandoci il bit in arrivo da i_w. Così facendo si evita di dover inserire ulteriori zeri per il padding. Arrivati a questo punto i_start sarà diventato basso, mettiamo a 1 il segnale o_mem_en così da avere la lettura della memoria abilitata. Il segnale o_mem_addr arriva in ingresso alla memoria e comunica l'indirizzo nel quale guardare. Ora con il segnale shift_reset riempiano di 0 il registro di prima, dal momento che non necessitiamo più dell'indirizzo. Nel frattempo il decoder riceve il dato recuperato all'indirizzo di memora grazie al segnale i_mem_data in uscita dalla memoria. Ora il nostro decoder conosce sia l'uscita sulla quale direzionare il dato , che il dato stesso. Dopo il decoder abbiamo 32 flip flop D, 8 per ogni uscita. Questi vengono abilitati dal segnale Rout_load per essere sovrascritti. Il decoder comunica con loro attraverso i segnali input_ff_z0, input_ff_z1, input_ff_z2 e input_ff_z3 (da 8 bit). Il decoder invia il dato alla serie di flip flop D giusti, mentre sugli altri rimane salvato un eventuale dato precedente. I segnali in uscita dai flip flop dell' uscita: output_ff_z0, output_ff_z1, output_ff_z2, output_z3 (da 8 bit) sono messi in and con un segnale tmp_done(da 8 bit) che diventa alto nel momento in cui tutte le operazioni sono terminate. o_done equivale al bit più significativo di tmp_done (quindi se tmp_done = 1111111 o_done = 1). Sull'uscita indicata da i_w vediamo il nuovo dato, mentre sulle altre vediamo i vecchi valori. Inoltre, in caso il segnale i_rst diventasse alto, si resetterebbero a tutti zeri i flip flop di uscitta e i flip flop adibiti a memorizzare l'indirizzo di memoria.

2.2 Stati della FSM

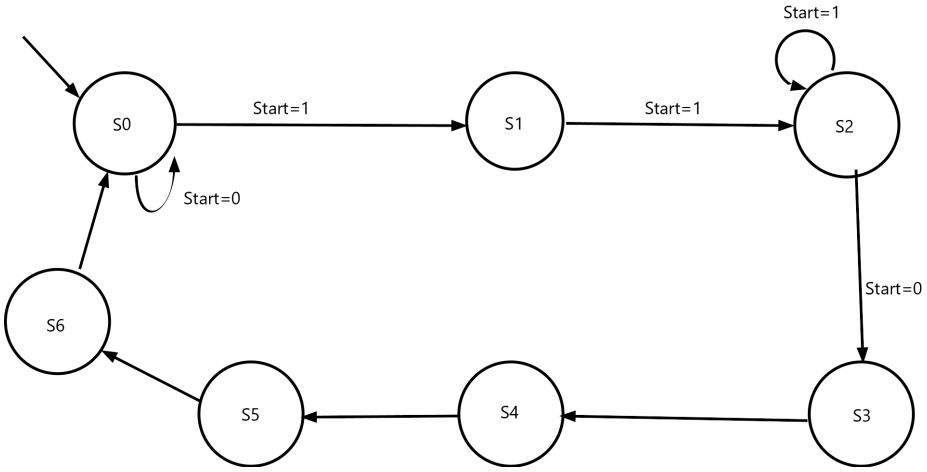

Figure 3: FSM

2.2.1 S0

Lo stato S0 è lo stato iniziale, si rimane in questo stato finchè lo Start rimane basso. In questo stato abbiamo il segnale R1_load alto che serve per rendere attivo il flip flop dove salviamo il primo bit in arrivo.

Table 1: Output stato S0

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	0	0
R1_load	Rout_load	shift_rst	tmp_done
1	0	0	000000000

2.2.2 S1

Arriviamo qui dopo che lo Start è passato a 1. R1_load torna a 0 e R0_load passa a 1. Viene abilitato il secondo flip flop che salverà il secondo bit in arrivo. Questi primi 2 bit definiranno l'uscita.

Table 2: Output stato S1

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	0	1
R1_load	Rout_load	shift_rst	tmp_done
0	0	0	000000000

2.2.3 S2

In S2 siamo ora interessati a salvare l'indirizzo di memoria che ci viene dato. Abilitiamo dunque il registro che se ne occupa. Restiamo in questo stato finchè Start è alto.

Table 3: Output stato S2

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	1	0
R1_load	Rout_load	shift_rst	tmp_done
0	0	0	00000000

2.2.4 S3

Ora che abbiamo l'indirizzo di memoria e Start è basso, mettiamo a 1 o_mem_en e leggiamo il dato corrispondente dalla memoria.

Table 4: Output stato S3

o_mem_en	o_mem_we	Rind_load	R0_load
1	0	0	0
R1_load	Rout_load	shift_rst	tmp_done
0	0	0	00000000

2.2.5 S4

Resettiamo i flip flop dove abbiamo salvato l'indirizzo di memoria.

Table 5: Output stato S4

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	0	0
R1_load	Rout_load	shift_rst	tmp_done
0	0	1	00000000

2.2.6 S5

Abilitiamo i flip flop dove vengono salvati i dati in uscita.

Table 6: Output stato S5

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	0	0
R1_load	Rout_load	shift_rst	tmp_done
0	1	1	00000000

2.2.7 S6

Done è alto e allora possiamo far uscire il dato sul canale selezionato.

Table 7: Output stato S6

o_mem_en	o_mem_we	Rind_load	R0_load
0	0	0	0
R1_load	Rout_load	shift_rst	tmp_done
0	0	0	11111111

3 Risultati Sperimentali

Per testare il corretto funzionamento del modulo, procediamo a utilizzare sia i test bench forniti ma anche a individuare casistiche particolari che provino la robustezza del progetto.

3.1 Report di sintesi

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	38	0	134600	0.03
LUT as Logic	38	0	134600	0.03
LUT as Memory	0	0	46200	0.00
Slice Registers	53	0	269200	0.02
Register as Flip Flop	53	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

Figure 4: Non sono presenti latch

```
Timing Report

Slack (MET) :             97.463ns  (required time - arrival time)
  Source:                 FSM_sequential_curr_state_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Destination:            output_ff_z1_reg[1]/CE
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        2.155ns  (logic 0.751ns (34.849%)  route 1.404ns (65.151%))
  Logic Levels:           1  (LUT5=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Figure 5: il progetto rientra senza problemi nelle tempistiche

3.2 Risultati test bench ufficiale

Nella figura si riportano i risultati del test bench fonitoci dai docenti.

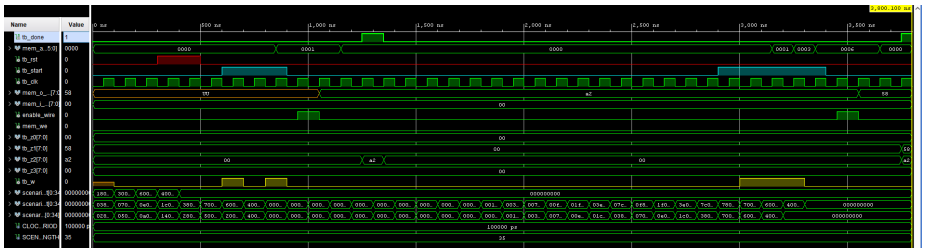

Figure 6: Risultati test ufficiale

3.3 Test reset improvvisi

In questo test controlliamo il comportamento in caso di reset improvvisi posizionati in diversi punti.

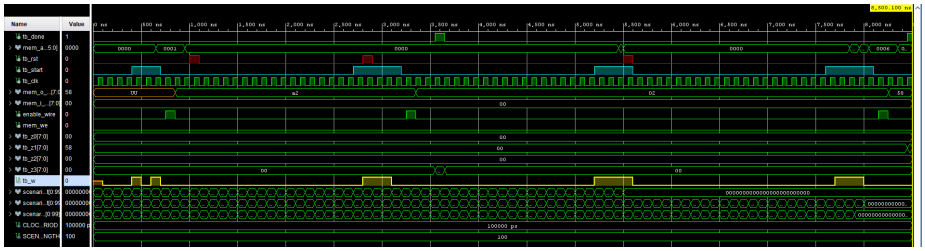

Figure 7: Risultati test reset

3.4 Test con start alto 2 cicli di clock

In questo test proviamo il caso particolare nel quale i_start rimane alto solo 2 cicli di clock e quindi i_w segnala solo l'uscita selezionata. L'indirizzo della memoria sarà quindi di tutti 0.

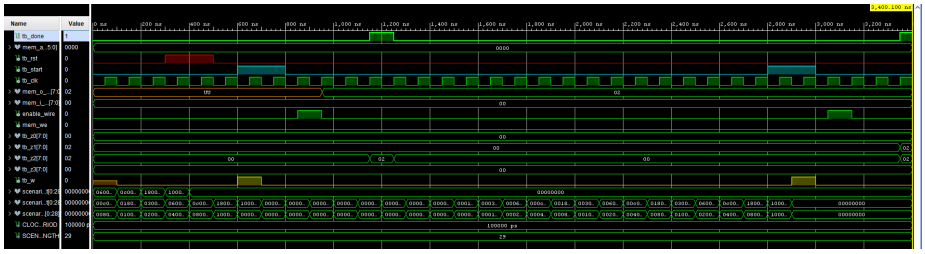

Figure 8: Risultati start alto 2 cicli di clock

3.5 Test con start alto 18 cicli di clock

In questo caso testiamo il caso in cui i_start è lungo 18 cicli di clock. Nel primo caso l'indirizzo di memoria inviato è 1111 1111 1111 1111 , nel secondo caso è 0000 0000 0000 0000

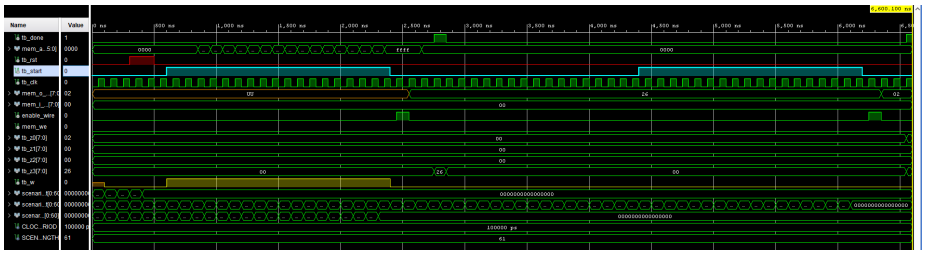

Figure 9: Risultati start alto 18 cicli di clock

4 Conclusioni

Il progetto supera correttamente tutti i test ai quali è stato sottoposto. Risulta essere anche molto veloce nell'elaborazione con un tempo rimanente di circa 97,5 ns e senza generare latch.