Prova Finale di Reti Logiche

Eleonora Cabai

Matricola 954966
Codice Persona 10696943

A.A. 2022/2023
Prof. Wiliam Fornaciari

Indice

1 Introduzione

La specifica della Prova Finale per l'A.A. 2022/2023 richiede di implementare, tramite il linguaggio VHDL, un modulo hardware in grado di communicare con una memoria.

Il sistema deve leggere dalla memoria la parola presente all'indirizzo identificato tramite il segnale in input e deve mostrare quest'ultima su un canale di uscita, anch'esso identificato mediante il segnale in input, fra i quattro disponibili.

1.1 Funzionamento

Il modulo legge il segnale dall'ingresso i_w sul fronte di salita di i_clk solo se i_start = 1 e termina la lettura nel momento in cui i_start = 0. È dotato dei seguenti segnali:

- due ingressi da 1 bit (i_start ed i_w)
- quattro uscite da 8 bit (o_z0, o_z1, o_z2 ed o_z3)
- un'uscita da 1 bit (o_done)

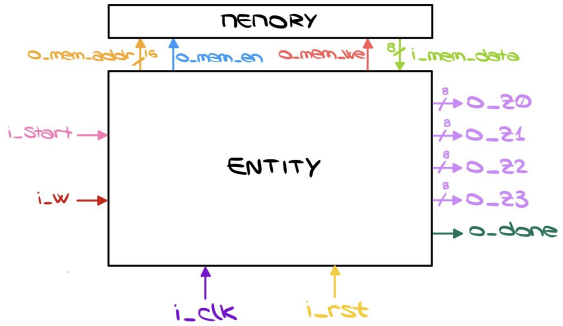

Figura 1: Segnali esterni all'entità

Prendendo in considerazione il segnale in ingresso, il modulo separa i primi 2 bit dai rimanenti in modo da identificare il canale di uscita, ed invia alla memoria la sequenza degli N bit successivi, con $0 \leq N \leq 16$ ed estesa a 16 bit.

Il modulo riceve infine dalla memoria un messaggio da 8 bit che deve mostrare in uscita sul canale identificato nella fase iniziale tramite la seguente mappatura:

- la parola 00 identifica il canale o_z0
- la parola 01 identifica il canale o_z1
- la parola 10 identifica il canale o_z2
- la parola 11 identifica il canale o_z3

1.2 Specifica

Il modulo da progettare possiede 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.

2 Architettura

L'entità da me progettata è composta da tre moduli: uno che elabora e salva i due bit che identificano il canale di uscita (modulo ChooseReg), uno nel quale viene salvato l'indirizzo da inviare alla memoria (modulo SIPO) ed uno che permette di mostrare i dati in uscita (modulo Demux).

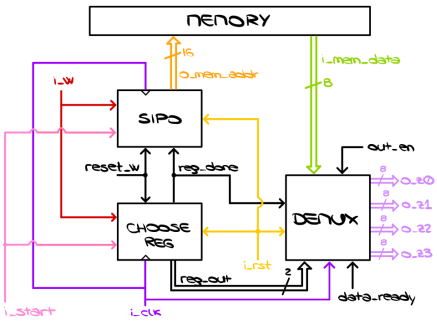

Figura 2: segnali interni ed esterni

L'entità esegue un processo sequenziale (fsm) che implementa la macchina a stati la quale regola le operazioni di lettura, elaborazione e produzione dei dati, ed un processo combinatorio (fsm_signals) che, in base allo stato in cui si trova il processo fsm, permette di comandare i segnali interni dell'entità che vengono poi inviati ai tre moduli.

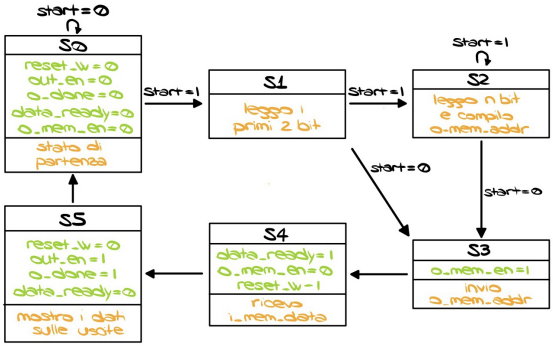

Figura 3: macchina a stati

La macchina a stati, che è stata schematizzata nella figura qui sopra riportata, è composta da 6 stati:

- stato S0: avviene il reset di tutti i segnali dell'entità. È lo stato di partenza e al quale si ritorna ogni volta in cui o_done = 1, cioè dopo lo stato S5
- stato S1: vengono letti i primi 2 bit del segnale i_w, tramite i quali si identifica il canale di uscita. Raggiungibile da S0 se i_start = 1

- stato S2: vengono letti i successivi N bit del segnale i_w, con $1 \leq N \leq 16$, tramite i quali si produce il vettore o_mem_addr. Raggiungibile da S1 se i_start = 1
- stato S3: il vettore o_mem_addr viene inviato alla memoria, infatti o_mem_en = 1. Raggiungibile da S1 o da S2 se i_start = 0
- stato S4: l'entità riceve il vettore i_mem_data dalla memoria e si prepara a mostrare i dati in uscita. Raggiungibile direttamente da S3
- stato S5: i valori presenti nei registri o_z0, o_z1, o_z2 ed o_z3 vengono mostrati in uscita e o_done = 1. Raggiungibile direttamente da S4,

2.1 Modulo ChooseReg

Il modulo ChooseReg è un modulo sincrono comandato dal segnale di clock (i_clk) dell'entità che permette di separare i primi due bit ricevuti in ingresso, in modo da poter identificare il canale sul quale il dato deve essere trasmesso.

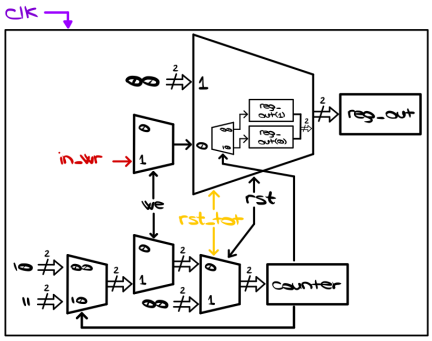

Figura 4: modulo ChooseReg

Questo modulo riceve in input il segnale we ed il segnale in_wr, che coincidono rispettivamente con il segnale i_start ed i_w dell'entità, ed è dotato di due segnali di reset:

- il segnale rst, che resetta i dati salvati nel modulo ogni volta che termina un ciclo di lavoro dell'entità, cioè il periodo tra il momento in cui i_start = 1 e i_done = 1
- il segnale rst_tot, che coincide con il segnale i_rst dell'entità

In output invece produce il segnale done_c, che comunica agli altri moduli il termine delle operazioni di lettura, ed il vettore reg_out, che contiene il registro identificato.

2.2 Modulo SIPO

Il modulo SIPO è un modulo sincrono, comandato dal segnale i_clk dell'entità, che rappresenta un registro Serial Input Parallel Output e nel quale i segnali rst, rst_tot ed in_wr hanno le medesime funzioni che hanno nel modulo ChooseReg.

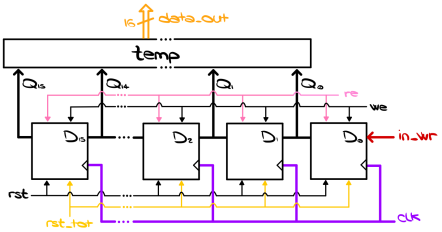

Figura 5: modulo SIPO

Questo modulo, inoltre, riceve in input due segnali che gli permettono di iniziare le operazioni di lettura dei dati, cioè re e we, i quali coincidono rispettivamente con il segnale i_start dell'entità ed il segnale done_c del modulo ChooseReg. Produce invece in output il vettore data_out che verrà inviato alla memoria tramite il vettore o_mem_addr.

2.3 Modulo Demux

Il modulo Demux è un modulo sincrono, comandato dal segnale i_clk dell'entità, che permette di controllare l'output prodotto dell'entità. I suoi quattro segnali in output reg_z0, reg_z1, reg_z2 e reg_z3 ed il suo unico segnale di reset rst coincidono rispettivamente con i segnali o_z0, o_z1, o_z2, o_z3 ed i_rst dell'entità.

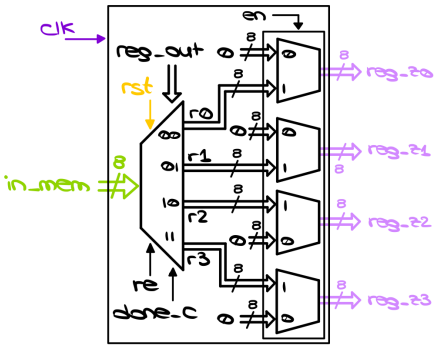

Figura 6: modulo Demux

Questo modulo, oltre a ricevere in input il vettore in_mem, che coincide con il vettore i_mem_data proveniente dalla memoria, ed il vettore reg_out proveniente dal modulo ChooseReg, riceve da quest'ultimo il segnale done_c e dall'entità i segnali data_redy e out_en, che comunicano rispettivamente che è stato ricevuto un dato dalla memoria e che le uscite sono abilitate.

3 Risultati Sperimentali

Il corretto funzionamento del componente è stato verificato sia tramite il testbench fornito, sia tramite i testbench che rappresentano i corner case.

Innanzitutto è stata eseguita la sua sintesi, il cui report è riportato di seguito

```
Synth Design complete, checksum: b557ca1d
INFO: [Common 17-83] Releasing license: Synthesis
synth_design completed successfully
synth_design:
    Time (s): cpu = 00:00:25 ; elapsed = 00:00:29 .
    Memory (MB): peak = 1452.855 ; gain = 1045.766
```

In seguito è stato eseguito e superato il testbench fornito come esempio dai docenti. Come si evince dai report che seguono, è stato eseguito in modalità Behavioural, in modalità Post-Synthesis Functional ed in modalità Post-Synthesis Timing

Report Simulazione Behavioural

```
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 3700 ns
Iteration: 0
Process: /project_tb/testRoutine
File: C:.../tb_example23_agg.vhd
INFO: [USF-XSim-96] XSim completed.
    Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 100000ns
launch_simulation:
    Time (s): cpu = 00:00:02 ; elapsed = 00:00:05 .
    Memory (MB): peak = 2132.531 ; gain = 11.402
```

Report Simulazione Post-Synthesis Functional

```
Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 3700200 ps
Iteration: 0
Process: /project_tb/testRoutine
File: C:.../tb_example23_agg.vhd
INFO: [USF-XSim-96] XSim completed.
    Design snapshot 'project_tb_func_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 100000ns
launch_simulation:
    Time (s): cpu = 00:00:5 ; elapsed = 00:00:8 .
    Memory (MB): peak = 2419.824 ; gain = 79.199
```

Report Simulazione Post-Synthesis Timing

Failure: Simulation Ended! TEST PASSATO (EXAMPLE)
Time: 3705071 ps
Iteration: 0
Process: /project_tb/testRoutine
File: C: .../tb_example23_agg.vhd
INFO: [USF-XSim-96] XSim completed.
Design snapshot 'project_tb_time_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 100000ns
launch_simulation:
Time (s): cpu = 00:00:01 ; elapsed = 00:00:10 .
Memory (MB): peak = 2419.824 ; gain = 0.000

Infine il componente è stato testato con diversi testbench che rappresentano i corner case. Questi, che vengono spiegati nel dettagio nei paragrafi successivi, verificano che

- non venga memorizzato alcun dato quando i_start = 0 (test di scrittura)
- non vengano mostrati dati in uscita se o_done = 0 (test sulle uscite)
- il componente venga resettato quando i_reset = 1 (test di reset)
- il componente funzioni correttamente quando N = 2 (test con N = 2)

3.1 Test di Scrittura

Il test verifica il funzionamento dei componenti ChooseReg e SIPO, ovvero verifica che i dati inviati attraverso i_w vengano memorizzati solo se i_start = 1.

Scegliendo il valore "11111111" (ff) per il segnale i_mem_data, la sequenza "1011101111" per i_w e ipotizzando che i_start sia attivo per 7 cicli di clock si deve verificare che venga scelta l'uscita o_z2 (10) e che il valore salvato in o_mem_data sia "11101" (001d). Dalla forma d'onda riportata nell'immagine seguente è verificato il corretto funzionamento dei componenti testati.

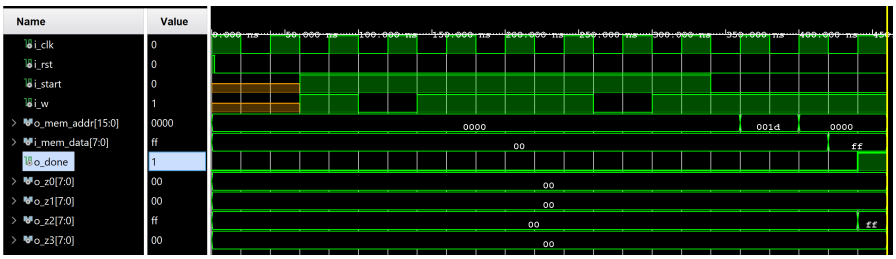

Figura 7: Test Scrittura

3.2 Test sulle Uscite

Il test verifica il funzionamento del componente Demux, dimostrando che i dati ricevuti tramite reg_out, che identifica l'uscita, ed i_mem_data, che trasmettte il dato da mostrare, vengano mostrati sull'uscita corretta solo se o_done = 1.

Scegliendo i valori "11111111" (ff), "00000000" (00), ed "100000000" (80) per il segnale i_mem_data ed utilizzando solamente le uscite o_z0 ed o_z1, in base al valore di reg_out, si deve verificare che sul registro corrispondente venga mostrato il valore in i_mem_data solo se o_done = 1. Dalla forma d'onda riportata nell'immagine seguente viene verificato il corretto funzionamento del componente testato. Lo stesso test è stato eseguito sulle uscite o_z2 ed o_z3 ottenendo risultati analoghi.

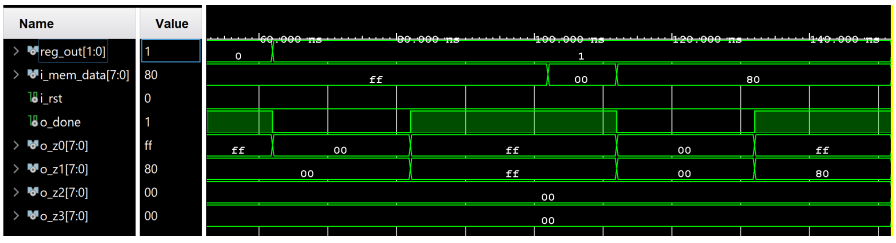

Figura 8: Test Uscite

3.3 Test di Reset

Il test verifica che, ogni volta che i_reset = 0, i segnali o_mem_addr, o_done, o_z0, o_z1, o_z2 ed o_z3 vengano posti a 0.

Scegliendo il valore "11111111" (ff) per il segnale i_mem_data ed utilizzando solamente le uscite o_z2 ed o_z3, si deve verificare che il valore in esse e su o_mem_addr venga posto a 0 quando i_reset = 1. Dalla forma d'onda riportata nell'immagine seguente è possibile notare che ogni volta che i_reset = 1 tutti i segnali di output vengono azzerati. Lo stesso test è stato eseguito sulle uscite o_z0 ed o_z1 ottenendo risultati analoghi.

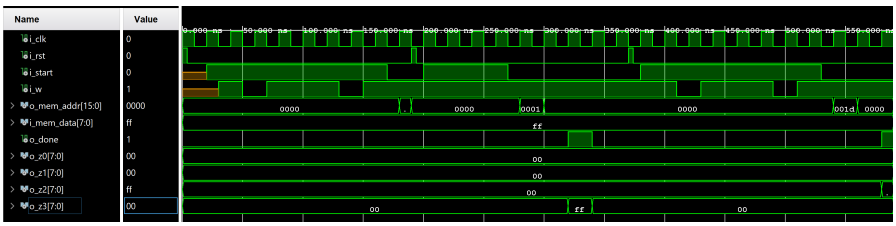

Figura 9: Test Reset

3.4 Test con N = 2

Il test verifica il corretto funzionamento del processo fsm, cioè che se N = 2 il dato salvato o_mem_addr sia 0 e che il dato in uscita venga mostrato nel registro corretto.

Scegliendo il valore "11111111" (ff) per il segnale i_mem_data, la sequenza "10" per i_w e ipotizzando che i_start sia attivo per 2 cicli di clock, si deve verificare che venga scelta l'uscita o_z2 (10) e che il valore salvato in o_mem_data sia 0. Dalla forma d'onda riportata nell'immagine seguente è verificato il corretto funzionamento della macchina a stati.

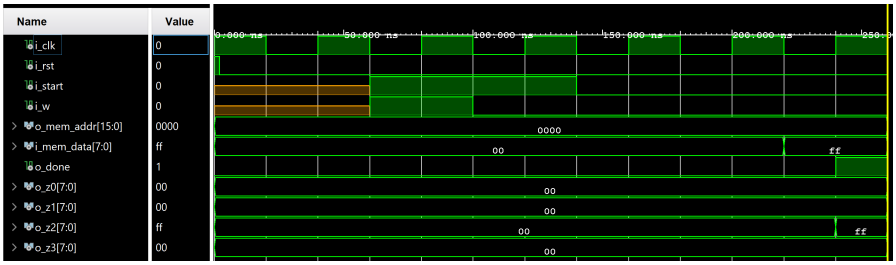

Figura 10: Test con N = 2

4 Conclusioni

Durante le fasi di progettazione del componente mi sono concentrata in particolar modo su due aspetti:

- il caso in cui N = 2
- la gestione dei dati ricevuti in input

Nel caso in cui N = 2 ho ottimizzato la macchina a stati in modo che non esegua lo stato S2, che è superfluo, infatti passa direttamente allo stato S3, lasciando o_mem_addr invariato, cioè inizializzato a 0.

Per consentire il corretto salvataggio del dato il ingresso, ho introdotto un segnale di reset interno che coinvolge solamente i componenti ChooseReg e SIPO, resettandoli al termine della computazione.

Inizialmente il modulo Demux era stato progettato come modulo asincrono ma è stato reso sincrono in quanto la sua implementazione precedente generava dei latch.

L'entità così progettata supera correttamente tutti i test eseguiti sia in modalità Behavoiural che in modalità Post-Synthesis e quindi risulta idonea alle specifiche date.