POLITECNICO
MILANO 1863

Prova Finale di Reti Logiche

Prof. Gianluca Palermo- a.a. 2022-2023

Elaborato di: Xuwen Ye

Codice Persona- 10754160
Matricola – 961047

1. Introduzione

La ***Prova Finale di Reti Logiche 2022/2023*** ha come scopo quello di sviluppare (in VHDL) un componente hardware che si interfacci ed interagisca con la memoria a seguito della ricezione di una serie di input da un bit, che indicano l'allocazione nella memoria dei dati che si devono leggere e trascrivere in uscita.

Di seguito si riporta una rappresentazione ad alto livello del componente descritto:

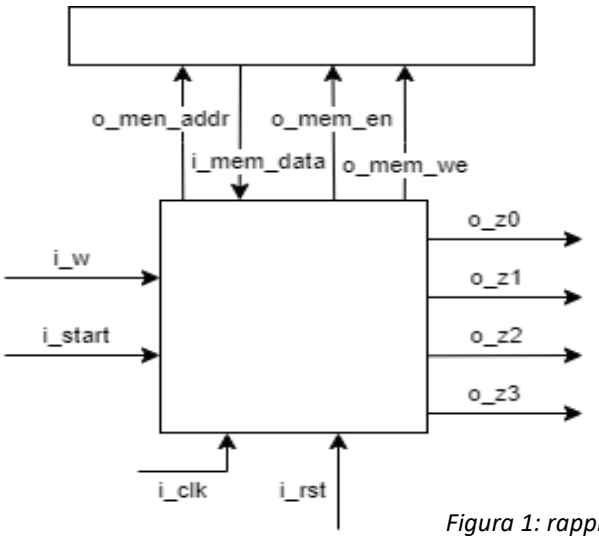

Figura 1: rappresentazione del componente HW che si interfaccia con la memoria ed i relativi segnali

I segnali:

- **i_w:** segnale in ingresso di 1 bit rappresentante l'input (generato dal test bench)
- **i_start:** segnale di START che indica l'inizio della lettura dell'input (generato dal test bench)
- **i_clk:** segnale di CLOCK (generato dal test bench)
- **i_rst:** segnale di RESET, asincrono rispetto ai cicli di clock (generato dal test bench)
- **o_mem_addr:** segnale in uscita contenente un l'indirizzo in memoria
- **i_mem_data:** segnale in ingresso contenente il dato letto all'indirizzo in memoria
- **o_mem_en:** segnale di ENABLE da inviare alla memoria per accedervi in lettura
- **o_mem_we:** segnale di WRITE ENABLE da inviare alla memoria per accedervi in scrittura
- **o_z0:** canale di uscita associato ai primi due bit di i_w "00"
- **o_z1:** canale di uscita associato ai primi due bit di i_w "01"
- **o_z2:** canale di uscita associato ai primi due bit di i_w "10"
- **o_z3:** canale di uscita associato ai primi due bit di i_w "11"

L'HW in questione inizia a leggere l'input tramite un ingresso **i_w** a 1 bit in modo seriale soltanto quando il segnale **i_start** vale 1, terminando la lettura quando esso torna ad essere 0.

I primi due bit rappresentano l'uscita (a 8 bit) su cui il dato letto in memoria deve essere trascritto, con il risultato visibile in output soltanto una volta che il segnale **o_done** diventa 1 (e rimane alto per un solo ciclo di clock) ed i risultati delle altre uscite inalterati rispetto alla loro configurazione precedente, mentre i restanti 16 bit corrispondono all'indirizzo di memoria in cui si deve accedere per leggere il dato stesso.

La selezione dell'uscita in base al valore dei primi due bit:

'00' => uscita o_z0;

'01' => uscita o_z1;

'10' => uscita o_z2;

'11' => uscita o_z3;

1.1 Esempio

In seguito, si riporta un semplice esempio (presente anche all'interno delle specifiche del progetto) che rappresenta il funzionamento nominale del componente hardware.

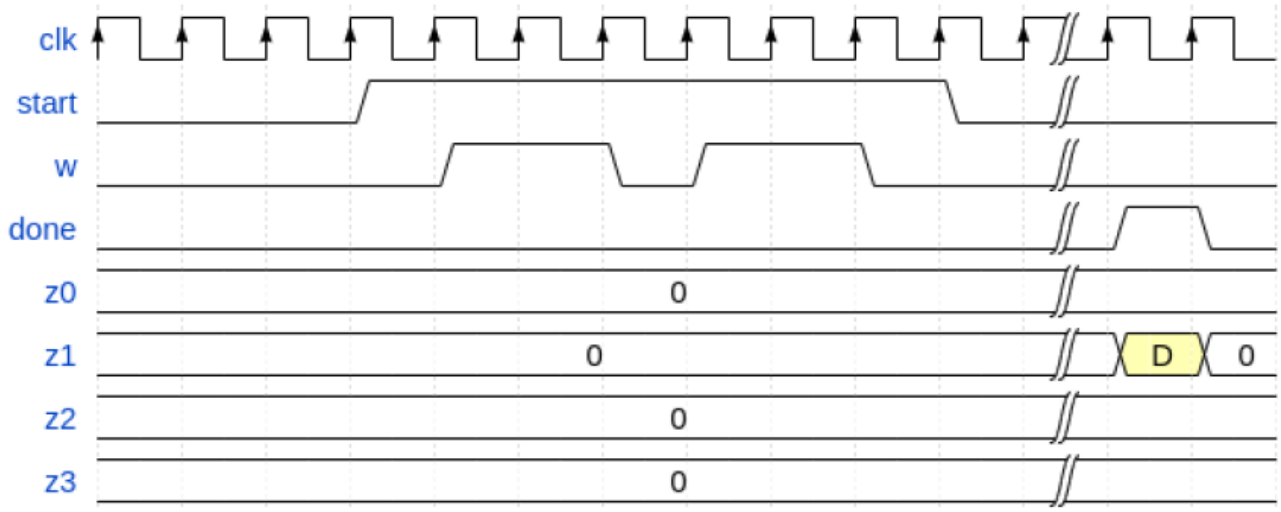

Figura 2: forme d'onda di un esempio di corretto funzionamento del componente hardware

Come si può ben notare dal diagramma delle forme d'onda, vengono letti i 2 bit '**01**' che ci indica che la porta in uscita che su cui verrà scritto il dato letto: **o_z1**. Fino a quando i_start rimane a 1, i bit i_w vengono letti in modo sequenziale per andare a costituire l'indirizzo di memoria su cui faremo in seguito l'accesso in memoria.

Il dato letto '**D**' viene trascritto su o_z1 quando **o_done** diventa 1, tornando al valore iniziale non appena quest'ultimo segnale torna a 0.

2. Architettura

Di seguito si riporta la macchina a stati finiti (FSM) che simula il funzionamento del componente hardware.

(*i segnali di i_rst = '1' che riportano ciascuno degli stati a quello di RESET sono stati omessi per semplicità, mentre si passa allo stato successivo solo al fronte di salita del clock*)

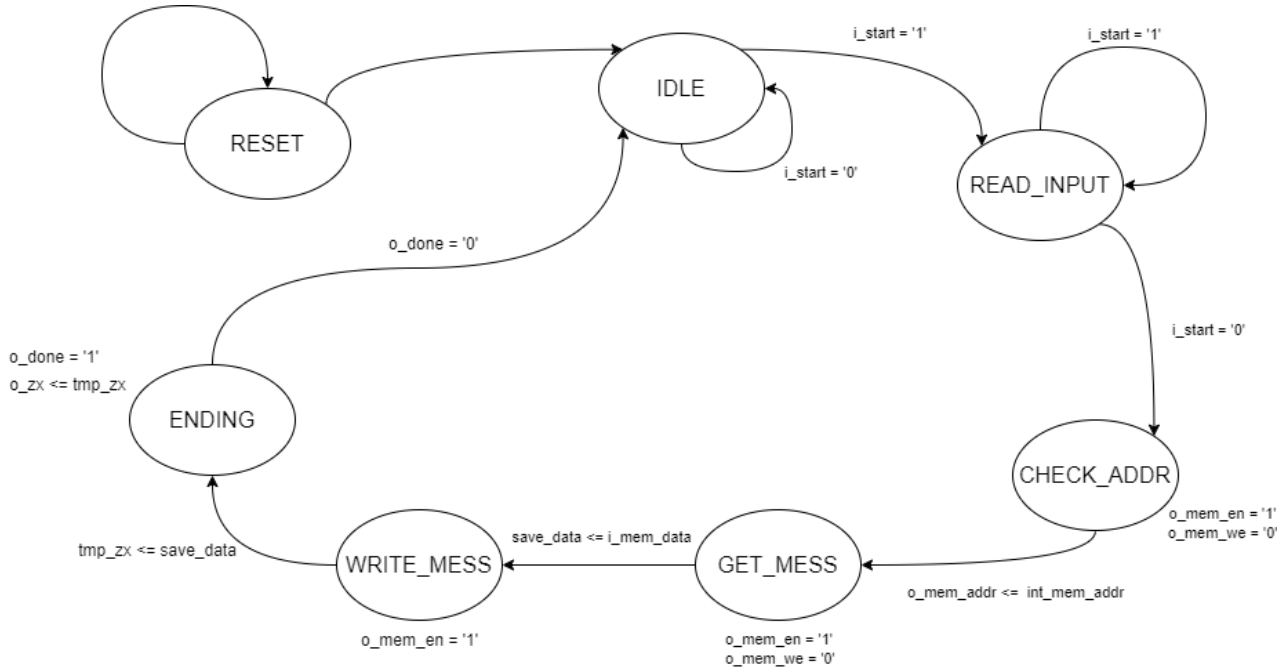

Figura 3: FSM ad 8 stati rappresentanti le principali transizioni di stato

RESET: stato in cui si vengono inizializzati tutti i segnali a 0: **o_zx**, **o_done** e i segnali di comunicazione con la memoria.

IDLE: stato di attesa in cui inizialmente si aspetta l'arrivo del segnale **i_start = '1'**. Alla prima lettura del segnale alto di i_start, si inizia a salvare i bit di **i_w** in **heading_bit**, che andranno a determinare il canale d'uscita su cui scriveremo il dato letto dalla memoria.

READ_INPUT: stato in cui si continua a leggere gli l'input i_w e memorizzare i suoi valori all'interno del vettore di **address** (inizializzato anch'esso in RESET) fintanto che **i_start** rimane alto. Una volta che i_start diventa 0, si memorizza address in una variabile temporanea int_mem_addr e si passa allo stato successivo.

CHECK_ADDR: stato in cui si l'HW si prepara ad inviare il segnale per l'accesso in memoria assegnando **int_mem_addr** a **o_mem_addr** e **o_mem_en** = '1' (per accedere alla memoria esclusivamente in lettura).

GET_MESS: stato in cui il dato letto dalla memoria (**i_mem_data**) viene salvato in una variabile temporanea **save_data**.

WRITE_MESS: stato in cui si scrive il dato letto in memoria (**save_data**) su una variabile temporanea **tmp_zx**, con x può che assumere i valori {0, 1, 2, 3}, a seconda dei bit di header memorizzati in READ_INPUT (cfr. Introduzione). L'utilizzo di questi registri temporanei tmp_zx è dovuto al fatto di soddisfare la specifica in cui le uscite o_zx devono essere a "0000..00" quando o_done = '0', per questo si è trovato questo modo per tenere traccia dei valori precedenti di ciascun canale d'uscita, visto che allo stato IDLE si assegnano a quest'ultimi nuovamente i valori di default.

ENDING: stato in cui si copia i valori di ciascun segnale **tmp_zx** sui rispettivi canali **o_zx**, assegnando 1 a **o_done** per visualizzare le uscite in output. Subito dopo un ciclo di clock, si riassegna 0 a o_done e si torna allo stato IDLE, in attesa del prossimo segnale di i_start = '1'.

3. Risultati sperimentali

Per verificare la correttezza del componente sviluppato, si sono effettuati test su alcuni test bench forniti dai docenti, andando anche a toccare spesso eventuali casi limite.

La presente implementazione passa tutti i test soltanto con la simulazione **Behavioral in pre-sintesi**. Si mostrano pertanto i risultati di tali simulazioni.

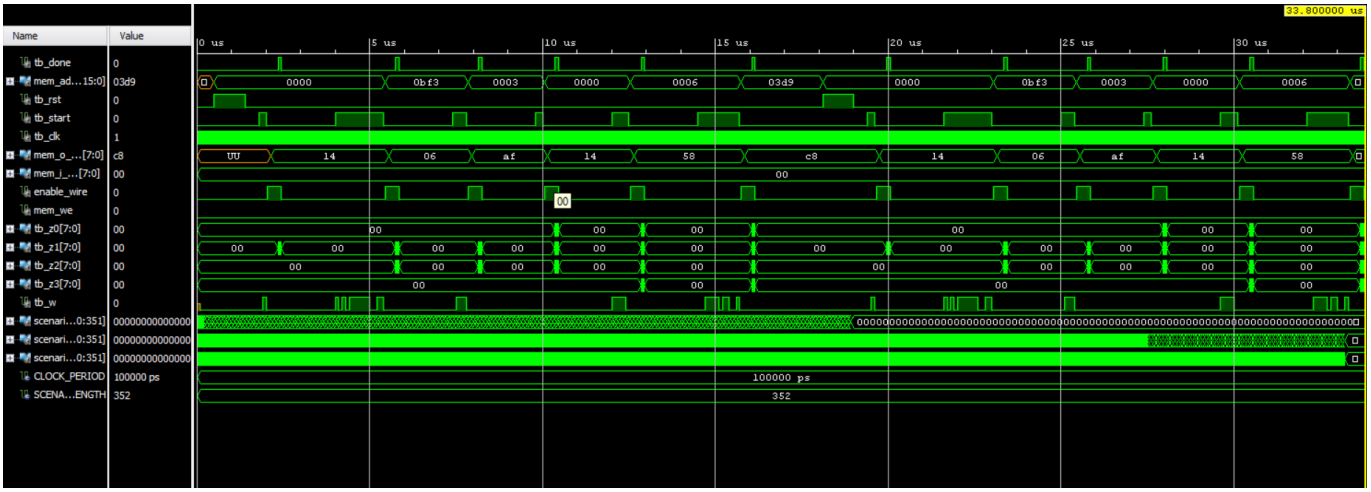

Figura 4: forme d'onda dei risultati di tb_1.vhd

<u>tb_1.vhd</u>: contiene il caso particolare in cui i_start rimane al livello alto soltanto per due cicli di clock, indicando in questo modo "0000000000000000" come indirizzo di memoria in cui leggere il dato da portare in uscita. (N.B. come da specifica, è garantito che i_start sia al livello '1' per almeno 2 cicli di clock e non più di 18)

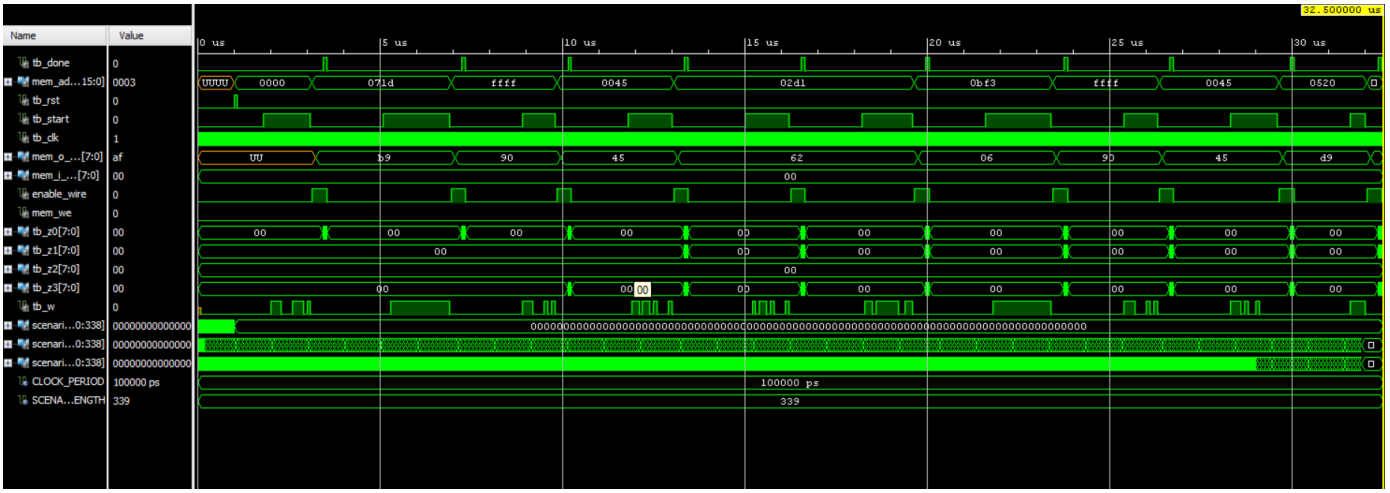

Figura 5: forme d'onda dei risultati di tb_2.vhd

tb_2.vhd: *contiene il caso particolare in cui i_start rimane al livello alto per 18 cicli di clock, indicando in questo modo "ffffffffffffffff" come indirizzo di memoria in cui leggere il dato da portare in uscita.*

Figura 6: forme d'onda dei risultati di tb_3.vhd

tb_3.vhd: *contiene il caso particolare che unisce i primi due test bench, ovvero i_start rimane al livello alto sia per soli 2 cicli di clock, che per tutti i 18, indicando in questo modo "0000000000000000" e "ffffffffffffffff" come indirizzi di memoria in cui leggere il dato da portare in uscita.*

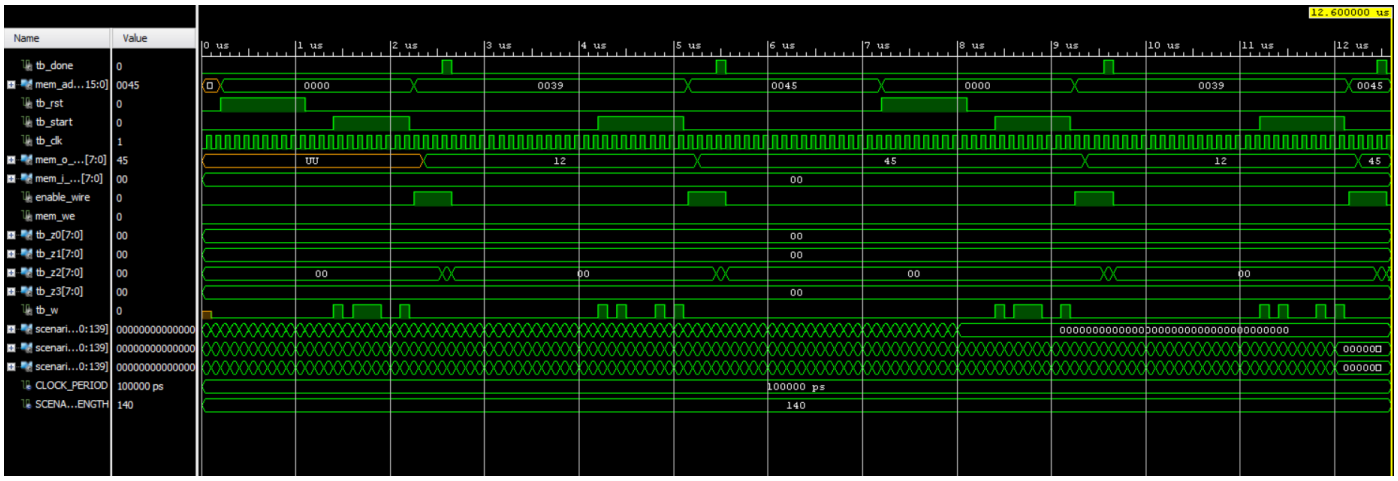

Figura 7: forme d'onda dei risultati di tb_4.vhd

tb_4.vhd: *contiene il caso in cui i_rst sale al livello alto per due volte, mostrando il corretto funzionamento della implementazione anche di fronte a segnali di RESET asincroni dove si inizializzano nuovamente tutti i segnali, e l'output viene sempre scritto sul canale o_z2.*

Figura 8: forme d'onda dei risultati di tb_5.vhd

tb_5.vhd: *contiene nuovamente il caso in cui i_rst sale al livello alto per due volte, e il segnale i_start rimane ad '1' soltanto per 2 cicli di clock.*

tb_6.vhd: *contiene il caso analogo a tb_2.vhd.*

tb_7.vhd: *contiene il caso analogo a tb_1.vhd.*

4. Conclusioni

Il progetto soddisfa la specifica per cui si debba produrre in output il risultato della computazione in meno di 20 cicli di clock.

La presente implementazione riesce passare tutti i test bench in simulazione Behavioral in pre-sintesi, ma non può essere sintetizzato a causa della presenza del segnale di clock nella sensitivity list del processo combinatorio, necessaria però per evitare loop infiniti e blocchi nella esecuzione del programma.