Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo – Anno 2022/2023

POLITECNICO
MILANO 1863

Daniel Shala (Codice Persona 10710181 – Matricola 957780)

Jurij Diego Scandola (Codice Persona 10709931 – Matricola 956223)

Indice

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto prevede l'implementazione di un modulo hardware descritto in VDHL, in grado di interfacciarsi con una memoria: ad un elevato livello di astrazione, il sistema riceve un segnale in ingresso, i primi due bit indicanti uno dei quattro canali di uscita di un multiplexer, mentre i restanti indicano il blocco di memoria dalla quale si estrarrà il dato da mostrare in output, sul canale specificato dai bit precedenti.

1.2 Specifiche generali

Il componente ha una serie di ingressi:
I) l'ingresso seriale i_start
II) l'ingresso seriale i_w
III) l'ingresso per il segnale i_clk - clock
IV) l'ingrsso per il segnale i_rst - reset

Il segnale i_w trasmette da un minimo di due bit ad un massimo di diciotto. La sequenza di ingresso è valida quando il segnale i_start è alto (=1) e termina quando il segnale i_start è basso (=0). Il segnale i_start rimane alto per almeno di 2 cicli di clock e per non più di 18 cicli. Quindi a i_start alto, inizia la trasmissione del segnale i_w: i due bit iniziali vengono salvati, concatenati in un primo vettore, il quale scopo sarà quello di andare a identificare uno dei quattro canali da 8 bit di uscita, di un multiplexer, posto alla fine della rete. I successivi n bit, con n compreso tra zero e sedici, verranno concatenati in un secondo vettore di dimensione sedici, inizializzato a zero. Viene effettuata l'estensione di segno a bit più significativo nei casi in cui il segnale i_w trasmetta meno di diciotto bit in totale. Il secondo vettore contiene così l'identificativo di un indirizzo di memoria, inviato poi al componente esterno, il quale restituirà l'informazione contenuta nel blocco di memoria avente indirizzo pari ai sedici bit inviati. Quindi il dato verrà poì indirizzato sul canale di output del multiplexer precedentemente specificato e diventerà visibile solo quando il segnale o_done verrà posto ad uno - il canale associato al messaggio cambierà il suo valore, mentre gli altri canali mostreranno l'ultimo valore trasmesso derivato dai messaggi ad essi associati. Quando il segnale o_done è 0 tutti i canali devono essere a zero. Le uscite del multiplexer sono chiamate Z0, Z1, Z2 e Z3 e sono inizializzate a 0 - i loro valori rimangano inalterati eccetto il canale sul quale viene mandato il messaggio letto in memoria. Contemporaneamente alla scrittura del messaggio sul canale, il segnale o_done passa da 0 passa a 1 e rimane attivo per un solo ciclo di clock. Il segnale i_start è garantito rimanere a 0 fino a che il segnale o_done non è tornato a 0, e prima del primo START (START=1) verrà sempre dato il RESET (RESET=1). Una seconda (o successiva) elaborazione con START=1 non dovrà invece attendere il reset del modulo.

1.3 Interfaccia del componente

Il componente descritto ha 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;
```

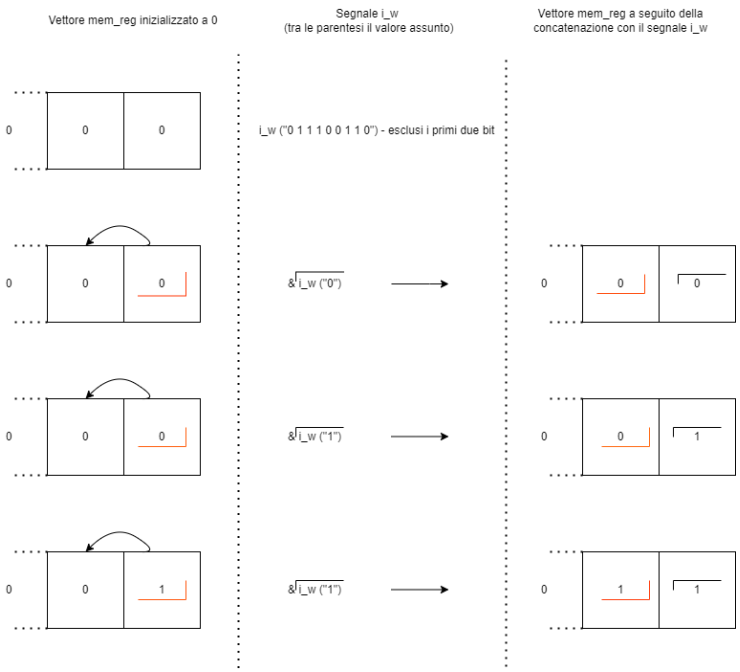
Nel dettaglio:

- 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 potercomunicare (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.

1.4 Dati e descrizione memoria

I dati in output sono memorizzati da una memoria già istanziata nel testbench. Le celle di memoria hanno indirizzi di 16 bit e contengono dati codificati in 8 bit. Il componente chiede alla memoria di accedere e recuperare l'informazione dalla cella di indirizzo tale a quello costruito a partire da i_w.

Esempio i_w : 0 1 0 1 1 1 0 0 1 1 0
Operazione sul vettore di modulo 16 relativo al salvataggio dell'indirizzo di memoria:



La freccia arcuata indica lo shift logico a sinistra eseguito dal bit più significativo a seguito della concatenazione. Dalla lettura del segnale i_w vengono elaborati due vettori, selected_out, che memorizza l'uscita, e mem_reg, che memorizza l'indirizzo di memoria. Ad ogni rising edge del clock, il bit trasmesso dal segnale viene sostituito nel vettore uscita piuttosto che concatenato nel vettore indirizzo - a seconda dello stato in cui si trova la macchina. Alla fine del processo, con l'estensione del segno, il vettore rappresentante l'indirizzo di memoria risulterà il seguente:

0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0

2 Design

Il componente si presenta inizialmente in attesa che il segnale di i_start in ingresso sia portato ad 1. Quando questo evento avviene, il modulo comincia l'elaborazione, salvando il primo bit del segnale i_w per poi passare dallo stato di IDLE al primo stato di computazione. Alla fine di questa il segnale o_done di output viene posto ad 1 e il dato di output risulta visibile sul canale precedentemente decodificato. Il segnale di o_done resta 1 soltanto per un ciclo di clock, il tempo necessario a mostrare l'output, dopodiche la macchina viene riportata in stato di IDLE, le variabili temporanee azzerate e i segnali re-inizializzati. Quindi la macchina torna in attesa del segnale di i_start. La differenziazione delle operazioni che la macchina compie rende molto vantaggioso a livello progettistico l'utilizzo di una FSA, in particolare, il nostro approccio iniziale prevedeva l'utilizzo di un contatore, contatore che si è rivelato obsoleto utilizzando il giusto numero di stati.

2.1 Stati dell macchina

La macchina è composta da 8 stati, brevemente presentati nelle successive righe.

2.1.1 IDLE

Stato iniziale in cui si attende il segnale i_start. Quando questo viene portato ad 1 inizia anche la trasmissione del segnale i_w. Viene eseguito il salvataggio nella prima cella di un vettore di modulo 2, inizializzato a zero. In caso di RESET la macchina ricomincia la computazione da questo stato.

2.1.2 HEADER

In questo stato viene salvato il secondo bit trasmesso da i_w.

2.1.3 GET ADDRESS

Questo stato è ricorsivo, esternamente può essere visto come un ciclo while avente come condizione di terminazione il passaggio del segnale i_start da 1 a 0. Lo stato corrente viene mantenuto uguale fino a quando questa condizione non si verifica, e ad ogni ripetizione viene concatenato il bit trasmesso da i_w ad un vettore di modulo 15. Con la concatenazione a destra, viene automaticamente eseguito lo shifting verso sinistra dei bit del vettore originario. Quando i_start viene portato a 0, il vettore risultato viene salvato nella variabile che verrà poi data in input alla memoria; mem_en viene portato ad 1, per consentire la lettura della memoria e viene cambiato lo stato corrente.

2.1.4 WAIT RAM

In questo stato la macchina attende risposta dalla memoria.

2.1.5 GET DATA

In questo stato della compilazione, la macchina riceve il dato nel blocco di memoria precedentemente specificato, utilizza il vettore creato dai primi due bit di i_w per scegliere tramite un CASE quale sarà l'uscita del multiplexer utilizzata. Quindi salva il valore restituito dalla memoria sul canale individuato.

2.1.6 WAIT DATA

Stato di attesa per la riallineazione dei dati. La scelta ingegneristica che ci ha portato alla creazione di uno stato del genere, a prima vista obsoleto, è stata quella di lasciare un ciclo di clock alla macchina, in modo da gestire meglio casi di ritardo dei segnali.

2.1.7 WRITE OUT

In questo stato le variabili di controllo della memoria vengono riportate ai loro valori iniziali, vengono riscritti i valori precedenti delle uscite del multiplexer, salvate in precedenza in appositi registri, e si cambia stato, passando all'ultimo.

2.1.8 DONE

Stato in cui si riporta il bit di o_done a 0, si resettano le variabili e i registri utilizzati nello stato precedente, si reimposta la macchina allo stato di IDLE.

2.2 Scelte Progettuali

Il nostro primo approccio alla progettazione del componente prevedeva l'utilizzo di un unico stato per la gestione del segnale i_w. Per distinguere i due vettori dati da realizzare partendo dal segnale, veniva utilizzato un contatore, che nel range tra 0 e 1 salvava i bit nel vettore canale, e nel range successivo nel vettore indirizzo, fino a quando il segnale i_start tornava a 0. Questo approccio comportava diverse problematiche in fase di test - spesso accadeva che il bit letto nella transizione tra i due vettori venisse perso, comportando così l'accesso ad un errato blocco di memoria. Considerata la dimensione finita dei vettori, abbiamo poi optato per una divisione degli stati, per gestire meglio la raccolta delle informazioni. Siamo stati molto rigorosi nella gestione dei Warnings segnalati dall'ambiente di sviluppo VIVADO: l'errore 8-327, "infering latch for variable" ci ha portati a riconsiderare l'utilizzo di Latch per il salvataggio temporaneo delle informazioni. Di fatto abbiamo cercato di limitare il numero di registri, risolvendo così errori in fase di simulazione.
Utilizzando stati "cuscinetto" come "WAIT DATA", della durata massima di un ciclo di clock, siamo riusciti infine a risolvere i problemi di timing.

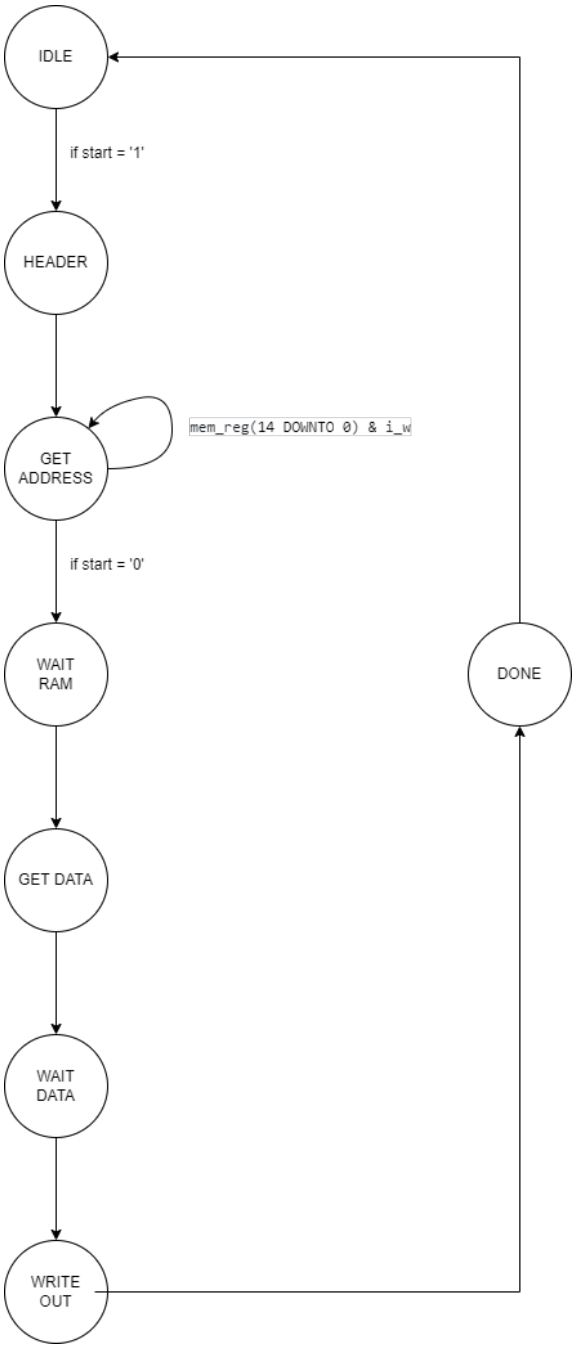


Figure 1: FSA con relativi stati e condizioni di transizione

3 Risultati dei test

Per verificare il corretto funzionamento del componente sintentizzato, abbiamo eseguito l'analisi utilizzando prima il test bench di esempio, e in seguito i restanti 7. In particolare ci siamo focalizzati sull'osservazione dei test numero 6 e 7, i quali rappresentavano i due casi limite del segnale i_w, rispettivamente maschera completa di 16 bit tutti ad 1 e maschera di soli 2 bit, tutto impostato a 0. Di seguito vengono mostrati gli screenshot dei test con l'andamento durante i cicli di clock.

I test bench per la verifica dei casi limite sono due:

3.1 6 - Segnale di Start pari ad 1 per 18 cicli di clock

Il segnale di Start resta alto per 18 cicli, con il segnale i_w formato da soli 1. Viene considersto come il caso peggiore in quanto la macchina resta nello stato GET_ADDRESS per il tempo massimo da specifica. L'uscita selezionata, la Z3, conterrà il dato della cella di memoria di indirizzo 65535.

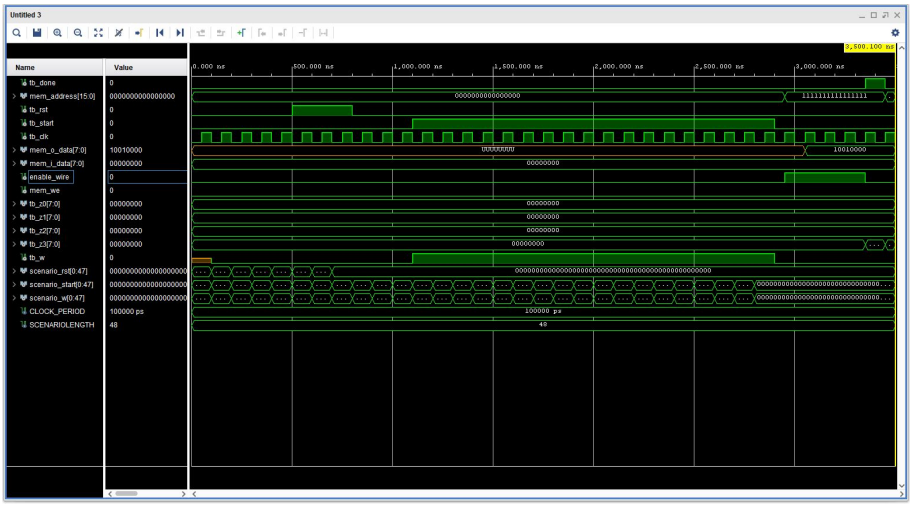


Figure 2: risultato del test bench 6

3.2 7 - Segnale di start pari ad 1 per 2 cicli di clock

Il segnale di Start resta alto solo per 2 cicli, con il segnale i_w formato da soli 0. Viene considerato come il caso migliore in quanto la macchina resta nello stato GET_ADDRESS per il tempo minimo da specifica. L'uscita selezionata, la Z0, conterrà il dato della cella di memoria di indirizzo 0.

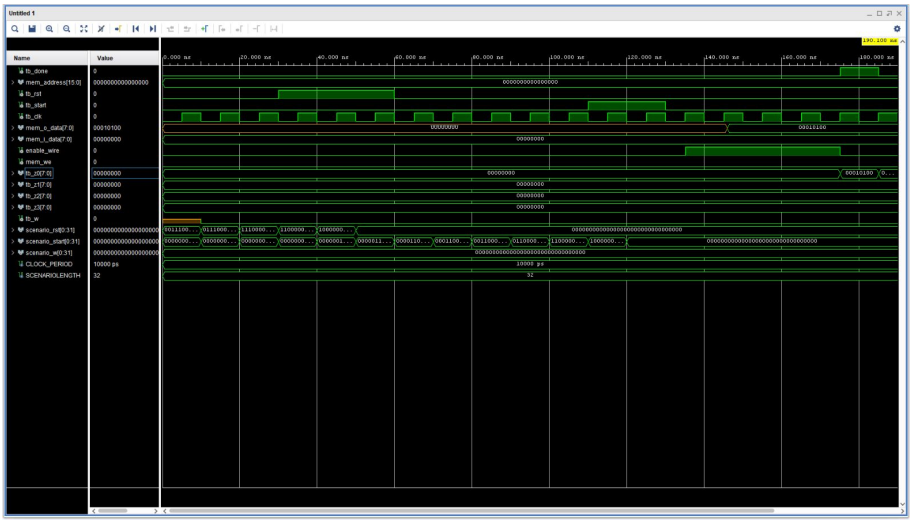


Figure 3: risultato del test bench 7

I test bench che verificano il corretto funzionamento dei segnali sono 2:

3.3 Reset asincrono

Il test verifica che un segnale di reset inaspettato non comprometta la computazione e che essa ricominci facendo ritornare l'automa nello stato di IDLE.

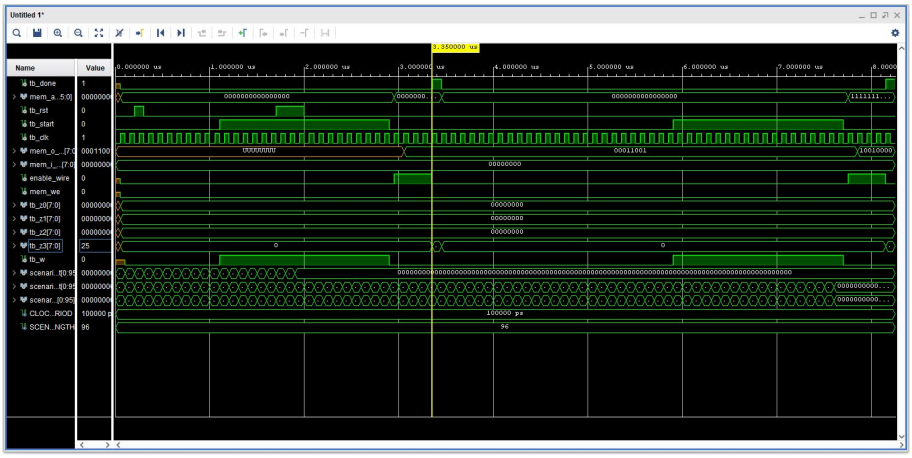


Figure 4: risultato del test bench con reset asincrono

3.4 Test bench con molteplici casi

Il test verifica la capacità del componente di processare diversi input in sequenza, così come la corretta sincronizzazione dei segnali di start e reset. Abbiamo utlizzato un test bench di

lunghezza 1276, ideato da noi.

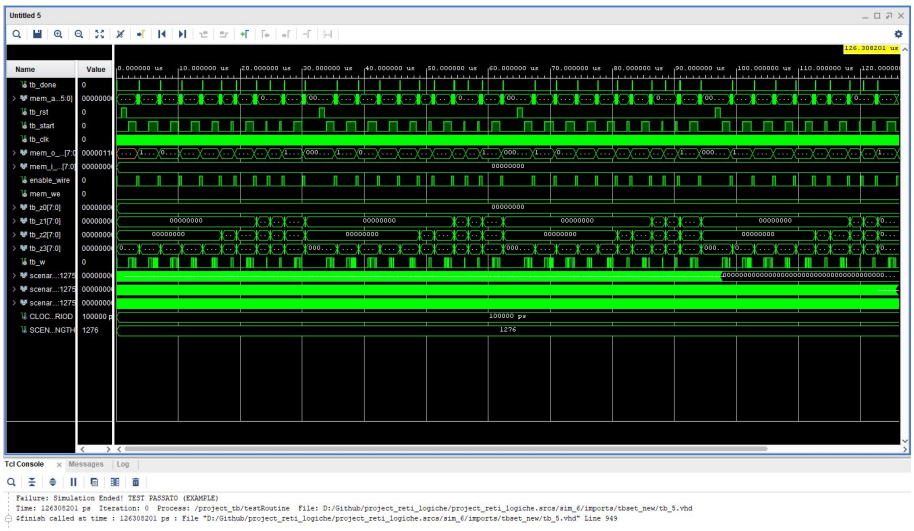


Figure 5: risultato del test bench

4 Conclusioni

4.1 Risultato della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle tre simulazioni: Behavioral, Post-Synthesis Functional e Post-Synthesis Timing. Qui di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner case che portano la macchina verso la computazione più breve e quella più lunga.

1. Computazione più breve - Failure: Simulation Ended! TEST PASSATO (EXAMPLE) Time: 190100 ps Iteration: 0 Process: /project_tb/testRoutine

2. Computazione più lunga - Failure: Simulation Ended! TEST PASSATO (EXAMPLE) Time: 3500100 ps Iteration: 0 Process: /project_tb/testRoutine

4.2 Ottimizzazioni

Le ottimizzazioni che abbiamo apportato consistono in una quantità di stati non minima ma ottima per evitare errori di ritardo a livello di trasmissione del segnale. Il progetto inizialmente prevedeva solo 5 stati ma a seguito di errori in fase di simulazione abbiamo optato per l'aggiunta di stati intermedi. A livello computazionale ci riteniamo soddisfatti in quanto comunque l'output viene reso a disposizione dopo pochi cicli di clock, rispettando ampiamente il limite dei 20.