Prova Finale - Progetto di Reti Logiche

Andrea Torti (10730470) && Jonatan Sciaky (10741047)

Anno Accademico 2022/2023

Contents

1 Specifiche di Progetto **3**
1.1 Scopo del progetto 3
1.2 Interfaccia del componente da realizzare 3
1.3 Comportamento atteso del componente 3

2 Processo di sviluppo e Design **4**
2.1 Processo di sviluppo 4
2.2 Scelte progettuali 4
2.3 Descrizione della FSM 4

3 Testing **6**
3.1 Testing dei casi limite 6

4 Risultati della sintesi **6**

1 Specifiche di Progetto

1.1 Scopo del progetto

Il Progetto di Reti Logiche di quest'anno prevede di realizzare la descrizione VHDL di un componente hardware che si interfacci con una memoria, ricevendo indicazioni da vari input seriali circa la locazione di memoria da cui prendere i dati e sulla porta su cui mostrarli in output.

1.2 Interfaccia del componente da realizzare

L'interfaccia del componente da realizzare *project_reti_logiche* prevede cinque input: *clk*, *reset*, *w*, *start* e *mem_data*.

- Il segnale da 1 bit *w* contiene informazioni sulla porta di output da utilizzare sulle quattro disponibili e sull'indirizzo di memoria da cui prelevare i dati
- *start* si alza quando *w* contiene informazioni utili
- *clk* regola il circuito sui fronti di salita
- *reset* funziona in modo asincrono rispetto al clock
- *mem_data*, input da 8 bit in parallelo, riporta il dato in uscita dalla memoria

I sette output del componente sono *z0*, *z1*, *z2*, *z3*, *done*, *mem_en* e *mem_addr*.

- *z0*, *z1*, *z2* e *z3* sono i quattro output da 8 bit su cui devono comparire i dati in output dalla memoria quando viene alzato il segnale di *done*
- *done* è il segnale, generato internamente al circuito, da alzare quando si attivano gli output
- *mem_en* è il segnale da alzare per attivare la comunicazione con la memoria
- *mem_addr* è un segnale da 16 bit che indica l'indirizzo di memoria che va letto

Esiste un ulteriore output, *mem_we*, che verrà d'ora in poi ignorato e impostato costantemente a zero: serve ad abilitare la scrittura su memoria, azione mai richiesta dalla specifica.

1.3 Comportamento atteso del componente

Il sistema, nello stato di reset, deve mostrare 0 sia sulle uscite che su *done*. Dal momento in cui *start* viene impostato ad '1', i dati letti dall'ingresso seriale *w* sui fronti di salita del clock sono organizzati nel seguente modo: i primi 2 bit indicano la porta di uscita, con mappatura *00->z0, 01->z1, 10->z2, 11->z3*, e i bit successivi, da 0 a 16, indicano la parte meno significativa dell'indirizzo di memoria, sempre da 16 bit, da cui prelevare i dati. I bit più significativi, eventualmente non specificati dall'ingresso seriale, hanno valore 0.

Il segnale *start* è garantito rimanere a livello logico alto da 2 a 18 cicli di clock, ed è garantito rimanere a livello logico basso per almeno 21 cicli di clock: 20 a disposizione per il circuito per interrogare la memoria e attendere una risposta (garantita arrivare in meno di 1 ciclo di clock), e 1 per mostrare il dato ricevuto sulla porta corretta contemporaneamente al segnale di *done*. Se i dati sono pronti prima di 20 cicli, il segnale *done* può essere impostato ad un livello logico alto anche in anticipo, ma deve rimanere alto solo per un ciclo di clock. È garantito inoltre esserci un reset prima che si alzi *start* per la prima volta, ma non ne verrà dato uno ogni successiva elaborazione: tuttavia, ogni volta che si dovesse ricevere nuovamente il reset, l'intero modulo andrà re-inizializzato.

Quando il segnale *done* è impostato a 0, tutte le uscite devono valere 0, mentre quando viene impostato a 1, la porta scelta deve essere sovrascritta con il valore in output dalla memoria per un solo ciclo di clock, mentre le altre devono mostrare l'ultimo output visto sulle relative porte.

2 Processo di sviluppo e Design

2.1 Processo di sviluppo

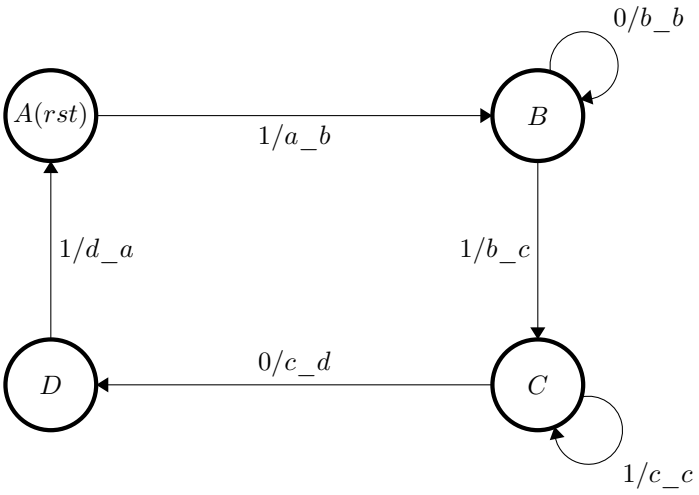
Per giungere alla versione finale del nostro modello, abbiamo attraversato tre iterazioni di design molto diversi tra loro. Nel primo e secondo design abbiamo utilizzato diversi moduli interconnessi tra loro per parallelizzare il segnale seriale in input, leggere la memoria e indirizzare il messaggio alla corretta porta di output. Tuttavia, alla fine abbiamo optato per una soluzione più snella ed elegante, che implementa una macchina di Mealy ed esegue tutte le operazioni all'interno di un unico modulo.

Durante lo sviluppo del progetto, abbiamo scelto una strategia di progettazione basata su comportamento. Tale approccio si basa sulla descrizione di alto livello del componente, permettendo di concentrarsi sulla funzionalità e di trascurare temporaneamente l'implementazione fisica. Ciò ha permesso di pensare il componente a un livello di astrazione superiore e semplificare il processo di debugging.

2.2 Scelte progettuali

La nostra macchina ha un funzionamento sequenziale, in cui i primi due cicli di clock sono dedicati alla lettura dell'indirizzo della porta d'uscita corretta. Successivamente, essa procede con la lettura dell'indirizzo di memoria da cui prelevare i dati. Durante questo processo, il segnale *o_mem_enable* è sempre impostato su 1: ciò significa che la memoria leggerà anche celle non richieste. Tuttavia, il messaggio sarà indirizzato solo alla porta da noi richiesta sul fronte di salita del clock successivo a quando il segnale *i_start* viene rilevato a 0: ciò avviene quando viene completata la lettura dell'indirizzo corretto. Allo stesso tempo, il segnale *o_done* viene impostato su 1 per un solo ciclo di clock e poi torna a 0 insieme a tutte le altre porte d'uscita.

2.3 Descrizione della FSM



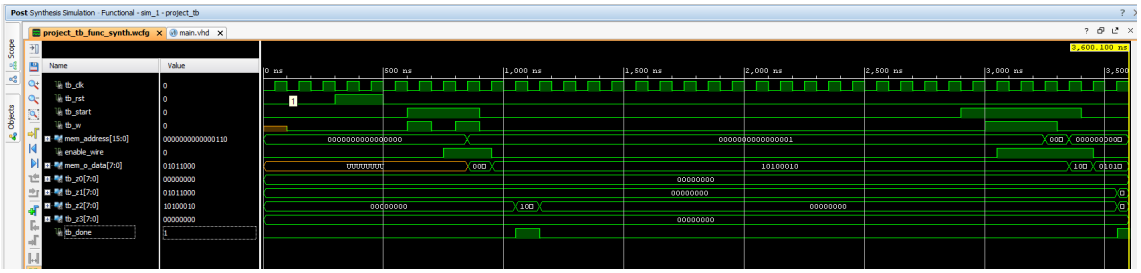
L'intero modulo implementa una macchina di Mealy composta da quattro stati. Durante la transizione dallo stato di reset (A) allo stato successivo (B), e poi dallo stato B allo stato C, l'automa assegna al vettore *out_port* i bit dell'indirizzo della porta di uscita utilizzando l'operazione di concatenazione '&' per unirli in un unico vettore. Durante la transizione dallo stato B allo stato C, viene anche impostata la porta *o_mem_en* ad '1' per abilitare la memoria: è questo il motivo per cui durante la decodifica dell'indirizzo di memoria richiesto, essa legge anche porzioni non essenziali; è anche grazie a questa scelta che siamo in grado di riportare in uscita il messaggio corretto in un solo ciclo di clock.

Nell'autoanello che porta dallo stato C a se stesso viene decodificato l'indirizzo (precedentemente inizializzato a 0 durante lo stato di reset) della cella di memoria che deve essere letta, utilizzando nuovamente l'operatore di concatenazione. Poiché la porta *o_mem_en* è già stata impostata ad '1', la memoria darà in output il messaggio presente nell'indirizzo intermedio ad ogni ciclo in cui la macchina si trova nello stato C. Tuttavia, poiché l'indirizzo potrebbe non essere completo finché la macchina si trova nello stato C, il messaggio viene riportato alla porta corretta solo durante la transizione dallo stato D allo stato di reset. Questo è possibile perché quando la macchina si trova nello stato D, l'indirizzo sarà già stato letto per intero. Difatti la macchina esegue la transizione dallo stato C allo stato D solo quando viene letto il valore '0' sulla porta d'ingresso *w*, ovvero quando l'input utile sarà già stato letto per intero. In questo modo, si è garantito che nel vettore contenente l'address di memoria verrà scritto l'indirizzo corretto.

Infine, durante la transizione dallo stato D allo stato di reset, viene impostato il segnale i_done a '1', che indica il completamento dell'operazione di lettura dalla memoria. Il messaggio letto dalla memoria viene assegnato alla porta corretta per un solo ciclo di clock, dopodiché tutte le uscite vengono azzerate nella transizione successiva. La porta di uscita corretta è stata determinata nelle prime due transizioni attraverso la lettura dell'indirizzo a 2 bit.

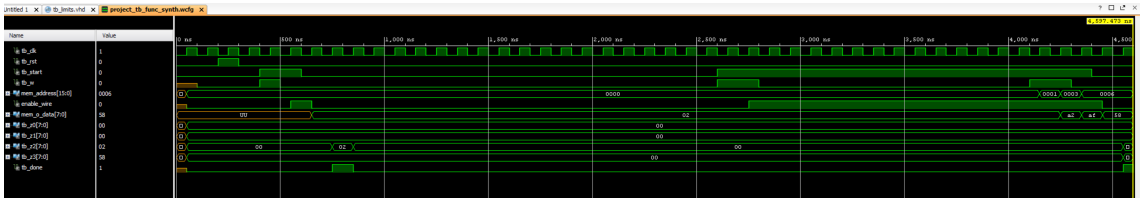
3 Testing

Per garantire il corretto funzionamento del componente descritto, abbiamo condotto diverse attività di testing. In particolare, abbiamo utilizzato gli otto testbench forniti per verificare il comportamento dell'automa di Mealy in vari scenari di utilizzo. Come si puó vedere nell'immagine qui sotto, il segnale *o_done* e l'output sulle porte si comportano come da specifica. Inoltre è stato possibile riportare, grazie alla nostra scelta di architettura, il segnale *o_done* e i messaggi di output sulle porte dopo un solo ciclo di clock a partire dal momento in cui viene rilevato il segnale *i_start* a 0.



3.1 Testing dei casi limite

Oltre agli otto testbench forniti abbiamo scritto una simulazione personalizzata che ha consentito di testare il componente nelle situazioni limite una dopo l'altra, come ad esempio quando vengono rilevati solamente i due bit dell'indirizzo della porta di uscita, o quando viene impostato un indirizzo per la memoria di lunghezza 16 bit. Questo assicura che il componente sia in grado di funzionare sempre in modo corretto e affidabile. In entrambi i casi, i test sono stati eseguiti con successo.



4 Risultati della sintesi

Il componente che abbiamo sviluppato è stato sottoposto a due diversi tipi di simulazione, ovvero *Behavioral* e *Post-Synthesis Functional*, superando in entrambi i casi tutti i testbench con esito positivo. Come detto in precedenza l'output viene correttamente inviato alla porta specificata dopo un ciclo di clock a partire dal momento in cui il segnale *i_start* viene rilevato come 0: non è possibile ridurre ulteriormente questa tempistica poiché sul fronte di salita in cui *i_start* viene visto a 0 non è possibile propagare istantaneamente in uscita il segnale letto dalla memoria.

Come si può notare dal *Design Timing Summary*, il valore riportato per il *Worst Negative Slack* è positivo, il che significa che il circuito sintetizzato sta soddisfacendo i time constraints con un certo margine di tempo; inoltre avere un *WNS* positivo rende il circuito più robusto anche in casi non ottimali di operazione.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	97.513 ns	Worst Hold Slack (WHS):	0.139 ns	Worst Pulse Width Slack (WPWS):	49.500 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:	197	Total Number of Endpoints:	197	Total Number of Endpoints:	102

All user specified timing constraints are met.