Prova Finale Reti Logiche

Simone Ronzoni, Tommaso Tognoli

1. Introduzione

Il sistema, per come descritto nella specifica, prende dall'ingresso W una sequenza di bit che rappresentano l'uscita su cui mostrare il dato e l'indirizzo di memoria dal quale prelevare suddetto dato.
Serve quindi un processo che elabori gli ingressi (START e W) in modo da distinguere i bit necessari al calcolo dell'uscita e ne tenga memoria.
Similmente un altro processo opererà per ricavare l'indirizzo di memoria.
In entrambi i processi abbiamo pensato di usare dei registri, rispettivamente a due e sedici bit, che permettono di concatenare i bit che arrivano uno ad uno in ingresso.
Il risultato del processo che calcola l'uscita è un segnale a due bit, mentre quello del processo che calcola l'indirizzo di memoria è un segnale a sedici bit.
Il dato ricavato dalla memoria deve essere mostrato sulla opportuna uscita solo in concomitanza di DONE = 1 e deve essere ricordato, ma non mostrato, mentre il segnale DONE è basso. Per fare ciò abbiamo pensato di realizzare quattro registri, uno per ogni uscita, in cui salvare il dato prelevato dalla memoria.
Il dato arriva al registro corretto grazie al controllo dei due bit precedentemente isolati.
Per coordinare le operazioni dei processi utilizzati abbiamo trovato opportuno creare una macchina a stati finiti, la quale genera in ogni stato dei segnali con cui si controlla il funzionamento dei vari processi.
La macchina a stati è anche responsabile della generazione del segnale di uscita DONE.

2. Architettura

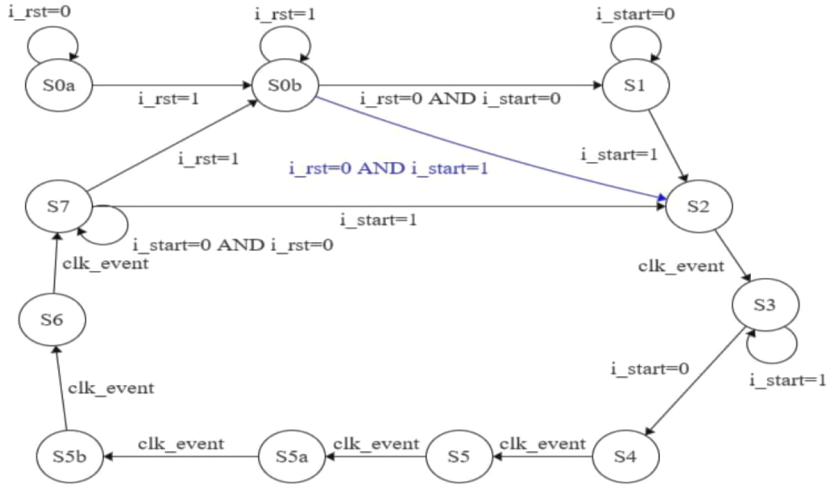
1 - FSM (macchina a stati finiti)

La nostra macchina a stati distingue undici stati:

- S0a è lo stato in cui ci si trova prima che arrivi il primo RESET = 1. Lo stato corrente rimane S0a fino a che non ricevo RESET = 1, una volta ricevuto vado nello stato S0b e non si tornerà più in questo stato.

- S0b è lo stato raggiunto dopo un RESET = 1 e in cui rimango fino a che RESET non è tornato a 0. Quando la macchina riceve RESET = 0 e START = 0 lo stato muta in S1, se ricevo il primo RESET = 0 e contemporaneamente il primo START = 1 vado in S2.
- S1 è lo stato in cui si attende il primo START = 1, rimane lo stato corrente finché START = 0. Quando ricevo START = 1 vado in S2 e salvo nel registro che calcola l'uscita il primo bit di W.
- S2 è lo stato in cui leggo il secondo bit di W che inserisco nel registro a due bit necessario al calcolo dell'uscita. In ogni caso da S2 si procederà raggiungendo lo stato S3.
- S3 è lo stato in cui si continua a leggere l'ingresso W e si cominciano a salvare nel registro a 16 bit tutti i bit ricevuti. S3 rimane lo stato corrente finché START = 1, quando START passa a 0 vado in S4
- S4 è lo stato in cui accedo alla memoria e dopo un ciclo di clock passo allo stato successivo S5
- S5, S5a, S5b vengono usati per prelevare il dato dalla memoria e avere il tempo necessario per scriverli sul corretto registro di uscita. Da S5b dopo un ciclo di clock mi sposto in S6.
- S6 è lo stato in cui viene dato il segnale di DONE = 1 e i dati salvati nei registri vengono mostrati in uscita. Dopo un ciclo di clock mi porto in S7.
- S7 è lo stato in cui DONE è tornato a 0 e mi metto in attesa di un segnale di START o di RESET. Se prendo START = 1 vado in S2, se ricevo RESET = 1 mi sposto in S0b.

La macchina a stati controlla i valori di 7 segnali:
- *o_mem_we, o_mem_en* sono i segnali che abilitano l'accesso in memoria;
- *done, o_done* calcolano quando mostrare il contenuto dei registri;
- *exit_en* attiva il processo che calcola il registro di uscita corretto;
- *registry_en* permette di scrivere il dato prelevato dalla memoria nel registro;
- *addr_calc_en* attiva il processo che calcola l'indirizzo di memoria;



2 - Exit Calculator (*exit_calculator)*

Si tratta del processo che permette di calcolare su quale uscita sarà il mostrato il dato che arriva dalla memoria. Si tratta di un registro a due bit che viene inizializzato a ‘00’ quando il segnale di DONE = 1 o il segnale di RESET = 1. Nel momento in cui DONE = 0, RESET = 0, START = 1 e *exit_en* = 1 inserisco il bit letto in ingresso nel registro. Per fare in modo di salvare il bit letto in ingresso all’indice giusto del vettore abbiamo aggiunto una condizione grazie al segnale *exit_position.*

3 - Memory Address Calculator (*mem_addr_calc*)

Questo è il processo in cui si calcola l’indirizzo di memoria da cui estrarre il dato. Dovendo in ogni caso produrre un indirizzo a 16 bit, con l’eventuale aggiunta di zeri a sinistra in caso di ricezione di un numero minore di 16 segnali START = 1, dopo i primi due necessari per il calcolo dell’uscita, abbiamo deciso di creare un vettore, *mem_addr*, inizializzato a sedici zeri sul quale operare e poi assegnarlo al segnale della nostra entity *o_mem_addr.*
Le operazioni effettuate su *mem_addr* sono delle concatenazioni di ‘1’ o ‘0’ in base al valore del segnale di ingresso W. Aggiungendo ogni volta un bit in fondo alla stringa ottenuta precedentemente, nell’operazione di concatenazione consideriamo tutti i bit di *mem_addr* tranne il più significativo.
Una volta terminata la lettura del segnale W, il risultato delle operazioni viene assegnato a *o_mem_addr.*
Entrambi i segnali vengono settati a sedici ‘0’ quando DONE = 1 o RESET = 1.

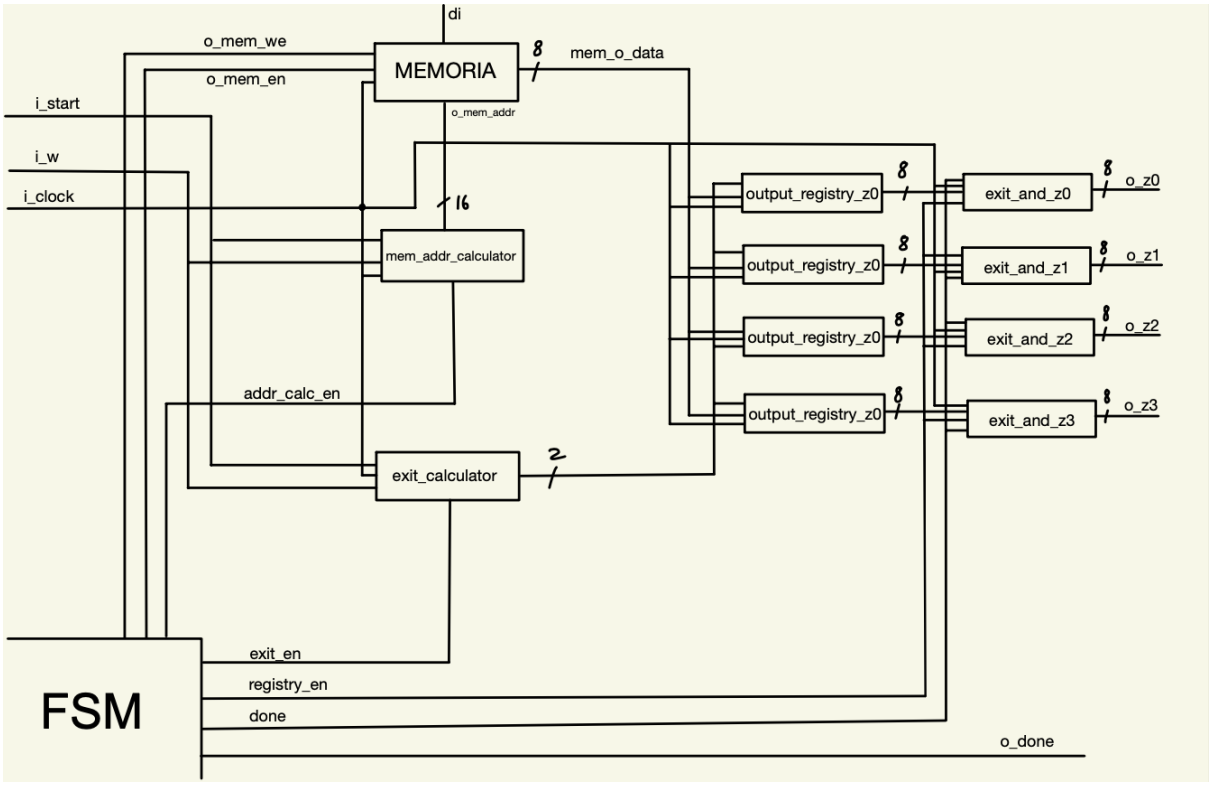
4 - Output Registry (*output_registry_z*)

I quattro output registry sono i componenti utilizzati per avere memoria del dato ricevuto e non perderlo quando il segnale DONE è basso.
Il dato salvato viene poi fornito dal registro al rispettivo componente *exit_and_z*.
Questo registro permette di mostrare correttamente ‘0000 0000’ in uscita quando DONE = 0 tenendo però il dato corretto disponibile per il successivo DONE = 1.
I segnali vengono settati a ‘0000 0000’ quando si riceve un segnale RESET = 1.

5 - Exit (*exit_and_z*)

Si tratta dei quattro componenti in cui assegnamo ai segnali di uscita (*o_z0, o_z1, o_z2, o_z3*) i dati salvati nei registri.
Per mostrare nel momento corretto i dati sulle uscite usiamo il segnale *done* che assume gli stessi valori del segnale di uscita *o_done.*

Se *done* assume valore ‘1’ allora assegnamo ai segnali di uscita i valori dei segnali salvati nei registri, altrimenti in uscita si mostra ‘0000 0000’.



3. Risultati Sperimentali

Per quanto riguarda la fase di simulazione le nostre preoccupazioni principali dopo aver scritto il codice seguendo lo schema che ci eravamo immaginati erano:

- aver prodotto un codice che permettesse di scrivere più di un dato sulla stessa uscita, e di conseguenza anche sullo stesso registro, nella stessa simulazione;
- realizzare il modulo in modo tale che superasse la simulazione anche in caso si ricevesse in ingresso una sequenza di RESET = 1 seguita immediatamente da una sequenza di START = 1, senza cicli di clock in cui entrambi i segnali fossero a ‘0’.

Procedendo alla simulazione dei testbench forniti abbiamo cercato di analizzare quali casi andassero a stressare. Ci siamo inoltre chiesti se i casi limite che ci preoccupavano venissero ben affrontati dal nostro codice.

1 - Testbench 1 (tb_1)
Il primo testbench utilizzato ci ha permesso di verificare che il sistema progettato permette la riscrittura sullo stesso registro di uscita. Inoltre ha confermato che il funzionamento è corretto anche nel caso in cui si riceva una sequenza di RESET = 1 oltre a quella iniziale.

2 - Testbench 2 (tb_2)
Questo testbench ci ha garantito che la macchina a stati finiti regola correttamente l'esecuzione delle operazioni anche nel caso in cui l'inizializzazione del modulo avvenga con un solo ciclo di clock in cui RESET = 1.

3 - Testbench 3 (tb_3)
Il testbench in questione ha esaminato il caso limite in cui il segnale START vale '1' per 18 cicli di clock, il massimo consentito dalla specifica, e ci ha confermato che anche in questa situazione il modulo simula correttamente.

4 - Altri Testbench
I testbench 4,5,6,7 non aggiungono casi di interesse a quelli già analizzati; infatti anche in queste occasioni la simulazione è andata a buon fine.
A nostro parere nei testbench forniti non erano considerati alcuni casi limite, per questo abbiamo deciso di creare alcuni testbench per verificare anche questi ultimi. Abbiamo quindi testato il caso in cui dopo una sequenza di RESET = 1 si presenta immediatamente in ingresso una sequenza di START = 1. Oltre a questo abbiamo testato il caso in cui il segnale W = 1 mentre il segnale START = 0, per controllare che il segnale venisse giustamente ignorato. In particolare questo processo è stato ripetuto modificando il momento in cui questa condizione si verifica in modo da testarla in diversi stati della FSM per assicurarci che i segnali di controllo funzionino correttamente.

Per quanto riguarda la sintesi, il modulo viene sintetizzato correttamente come indicato dal report di sintesi, che comunica un solo *warning* riguardo al fatto che il segnale *o_mem_we* è costante a '0'.
Nelle simulazioni post-sintesi abbiamo riscontrato dei problemi nella *timing simulation,* mentre nella *functional simulation* il sistema funziona correttamente completando tutti i test. Più precisamente con il nostro codice non riusciamo a leggere correttamente i bit corrispondenti alla sequenza di START = 1. Vengono letti correttamente tutti i bit di W ad eccezione dell'ultimo e questo non ci permette di calcolare correttamente l'indirizzo di memoria (o l'uscita nel caso l'indirizzo di memoria venga lasciato a tutti '0').

4. Conclusioni

In conclusione abbiamo realizzato un modulo funzionante in simulazione. In post-sintesi immaginiamo che il problema sia nel ritardo di propagazione dei segnali all’interno del sistema, questo perché la simulazione funzionale ci porta a dei risultati corretti, ma non siamo riusciti a scovare la radice del problema nella simulazione temporale.