Politecnico di Milano

Report Progetto Reti Logiche

Autore:

Deshan Tharindu Edirisinghe Mudiyanselage Edirisinghe

C. Persona: 10668753

N°.matricola: 937003

Anno Accademico 2023–2024

Indice

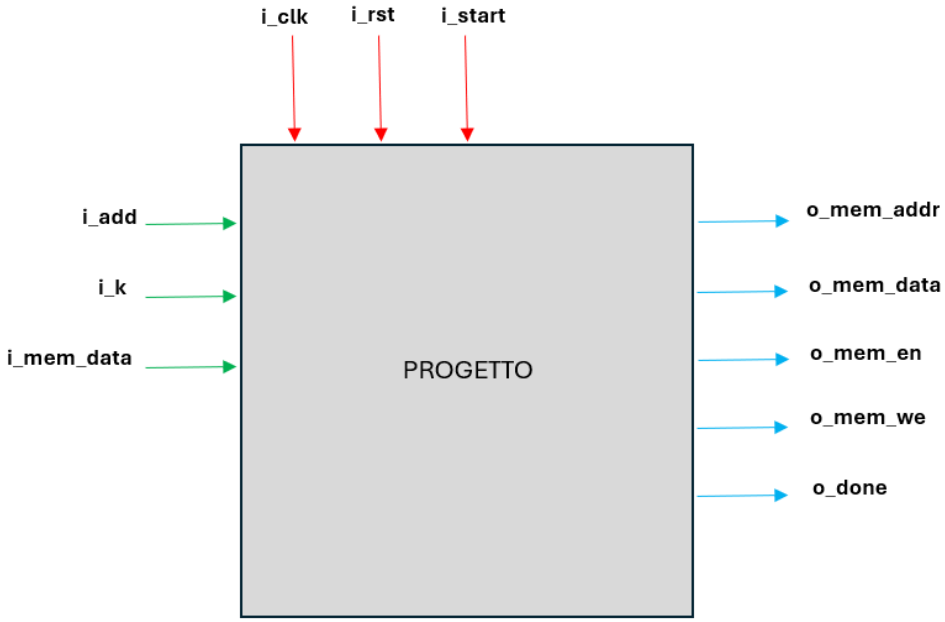
1 INTRODUZIONE

Questo documento contiene il report del progetto di Ingegneria Informatica, del corso di Reti Logiche, relativo all'anno accademico 2023/2024. L'obbiettivo consiste nel progettare una rete logica che, prendendo dati in input e avendo la possibilità di leggere valori su una memoria, deve produrre una sequenza in output, ben definita da alcune specifiche, direttamente in memoria e un valore di uscita o_done sincronizzato al valore di alcuni segnali.

La rete si avvierà o resetterà con appositi segnali i_start e i_rst, non sempre in maniera sincrona, ma il fulcro del problema resta nella gestione della lettura e della scrittura in memoria (e di eventuali casi limite), che richiederà uno specifico ciclo lettura/scrittura calibrato in base al timing di risposta della memoria alle richieste di lettura/-scrittura. Vedremo in questo report come sono stati risolti questi problemi e in particolare le scelte progettuali fatte anche per risolvere i casi limite.

Il progetto è stato simulato (pre e post sintesi) sul software Vivado (Ver. 2022.1), specializzato nella programmazione di hardware.

Di seguito un generico schema con tutti i segnali di input e output, come da specifica, per la rete progettata.



Segnali blu => segnali di output che escono dalla rete
Segnali verdi e rossi => segnali di input esterni alla rete

2 ARCHITETTURA RETE

Le frecce sono segnali la cui direzione esplicita la relazione input-output.

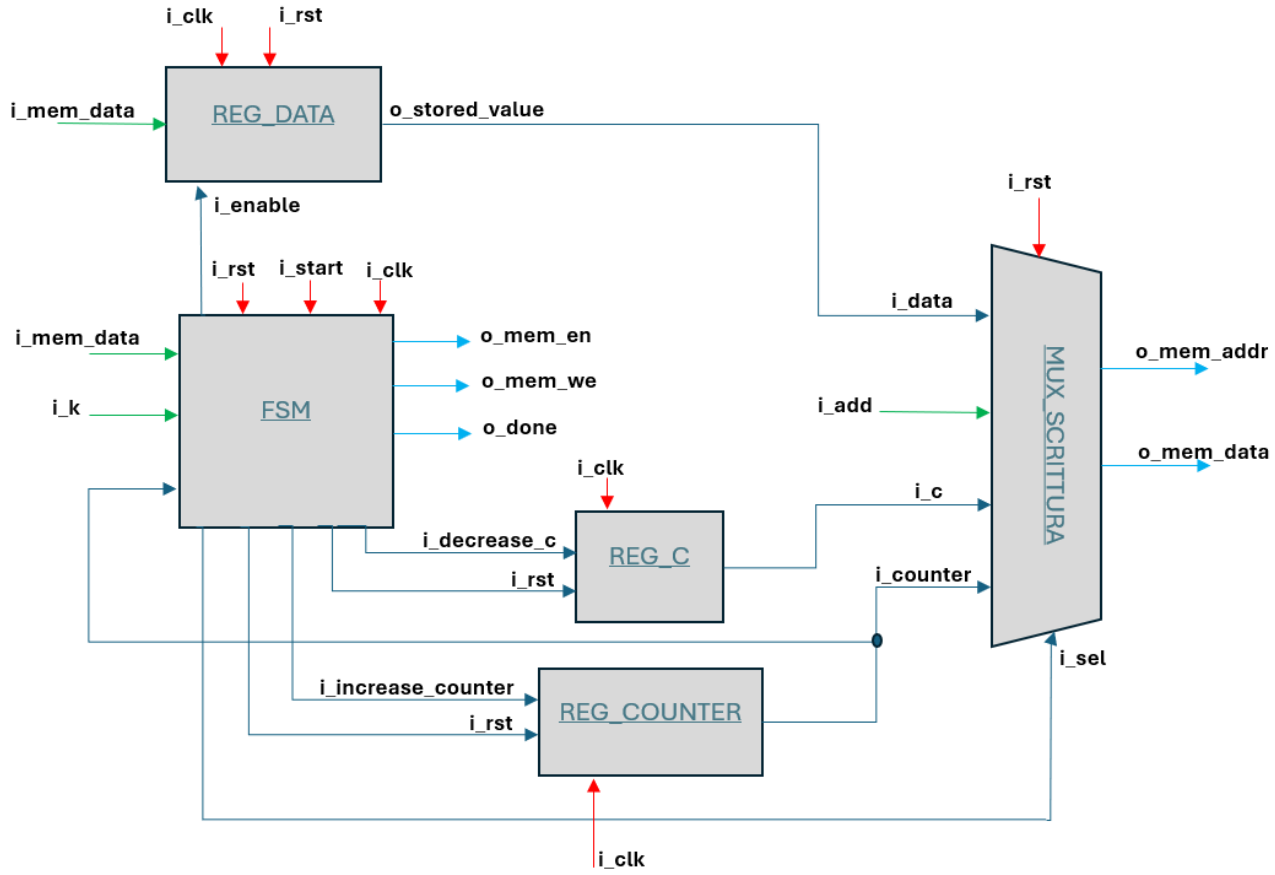


Figura 1: architettura completa della rete

Segnali blu => segnali di output che escono dalla rete
Segnali verdi e rossi => segnali di input esterni alla rete

2.1 MODULO_1: REG_DATA

Questo modulo legge il valore del dato letto **i_mem_data** e lo salva nel segnale **stored_value** (8 bit) se e solo se è diverso da zero e se il segnale i_enable = '1'. Il modulo ha anche la duplice funzione di offrire il valore salvato come output **o_stored_value**. Il valore stored_value e il segnale o_stored_value sono aggiornati solo sul fronte di salita del segnale di clock (i_clk fa parte della sensitivity list dell'unico process del modulo).

Il segnale i_rst permette di resettare il valore di stored_value a 0 in maniera asincrona (i_rst fa parte della sensitivity list dell'unico process del modulo)

2.2 MODULO_2 : REG_C

Questo modulo contiene il **valore di credibilità C** (8 bit) inizializzato a 31 nello stato S0 della FSM. Se **i_decrease_c** = "1" il valore di credibilità verrà decrementato di 1, altrimenti rimarrà inalterato. Se il segnale i_rst = "1" il valore di credibilità tornerà al suo valore iniziale 31. Il modulo fornisce il valore di credibilità salvato come output.

Questo modulo è sincronizzato con il clock di sistema e i suoi segnali sono aggiornati al fronte di salita del clock.

Il segnale **i_rst** non è il segnale di reset esterno alla rete bensì un segnale interno, ricevuto dal modulo FSM, e necessario in quanto, durante il ciclo di lettura e scrittura, il valore di credibilità potrebbe dover essere resettato più volte. Dato che l'uscita è un segnale interno non ho bisogno di un reset asincrono.

2.3 MODULO_3 : REG_COUNTER

Questo modulo contiene il valore di iterazione **counter** (11 bit) raggiunto dal ciclo di lettura e scrittura. Se **i_increase_counter** = "1" il valore del counter verrà incrementato di 1, altrimenti rimarrà inalterato. Se il segnale i_rst = "1" il valore del counter tornerà al suo valore iniziale. Il modulo fornisce il valore counter salvato come output.

Questo modulo è sincronizzato con il clock di sistema e i suoi segnali sono aggiornati al fronte di salita del clock.

Come per il modulo_2 il segnale **i_rst** è un segnale interno, il motivo però è che nel caso di multiple elaborazioni non è detto che ci sia un valore alto di reset, nonostante sia necessario resettare il counter alla fine di ogni sotto-elaborazione. Dato che l'uscita è un segnale interno non ho bisogno di un reset asincrono.

2.4 MODULO_4 : MUX_SCRITTURA

La logica del mux è la selezione del tipo di uscita in base al valore di **i_sel**. Quest'ultimo può essere considerato come un segnale di "enable" poiché se i_sel assume un valore tra 1 e 5 (in codifica binaria) il modulo sarà attivo e darà un certo output, altrimenti le uscite sono portate a '0' e il modulo sarà "disabilitato". Per evitare ridondanze, il valore i_sel = "000" è usato per "disabilitare" il modulo (così come **i_rst** = '1'). L'output è costruito utilizzando i valori dei 4 ingressi (i_data, i_add, i_c, i_counter).

La particolarità di questo modulo è che, diversamente da tutti gli altri, non è sincronizzato con il clock di sistema, in particolare i segnali del mux si aggiornano solo in concomitanza di un cambiamento del segnale i_sel (uno dei due segnali nella sensitivity list del process del modulo). Questa è una scelta progettuale che permette di avere un segnale di uscita "diretto" con il segnale di selezione, ossia detto in parole povere, chiunque aggiorni il segnale di selezione sa perfettamente che subito dopo avrà l'aggiornamento delle uscite del mux_scrittura e così rimarranno fino ad un cambiamento del segnale di selezione.

Una delle motivazioni che ha portato all'uso di questo tipo di modulo era la necessita di evitare che più moduli avessero la stessa uscita in comune, che avrebbe richiesto un modulo extra con un sistema per rendere "invisibile" tutti i segnali tranne quello voluto. Per questo motivo in questo progetto si hanno segnali di uscita unici per ogni modulo.

Esempio: se modulo_1 e modulo_2 avessero entrambe un'uscita collegata a o_mem_addr, portandone una al livello logico basso l'altra non riuscirebbe a portare al livello logico alto il segnale o_mem_addr in quanto direttamente collegato al valore logico basso ergo ho bisogno del terzo modulo sopracitato.

Questo modulo NON ha nessun elemento di memoria.

2.5 FSM

2.5.1 RAPPRESENTAZIONE DELLA FSM TRAMITE GRAFO

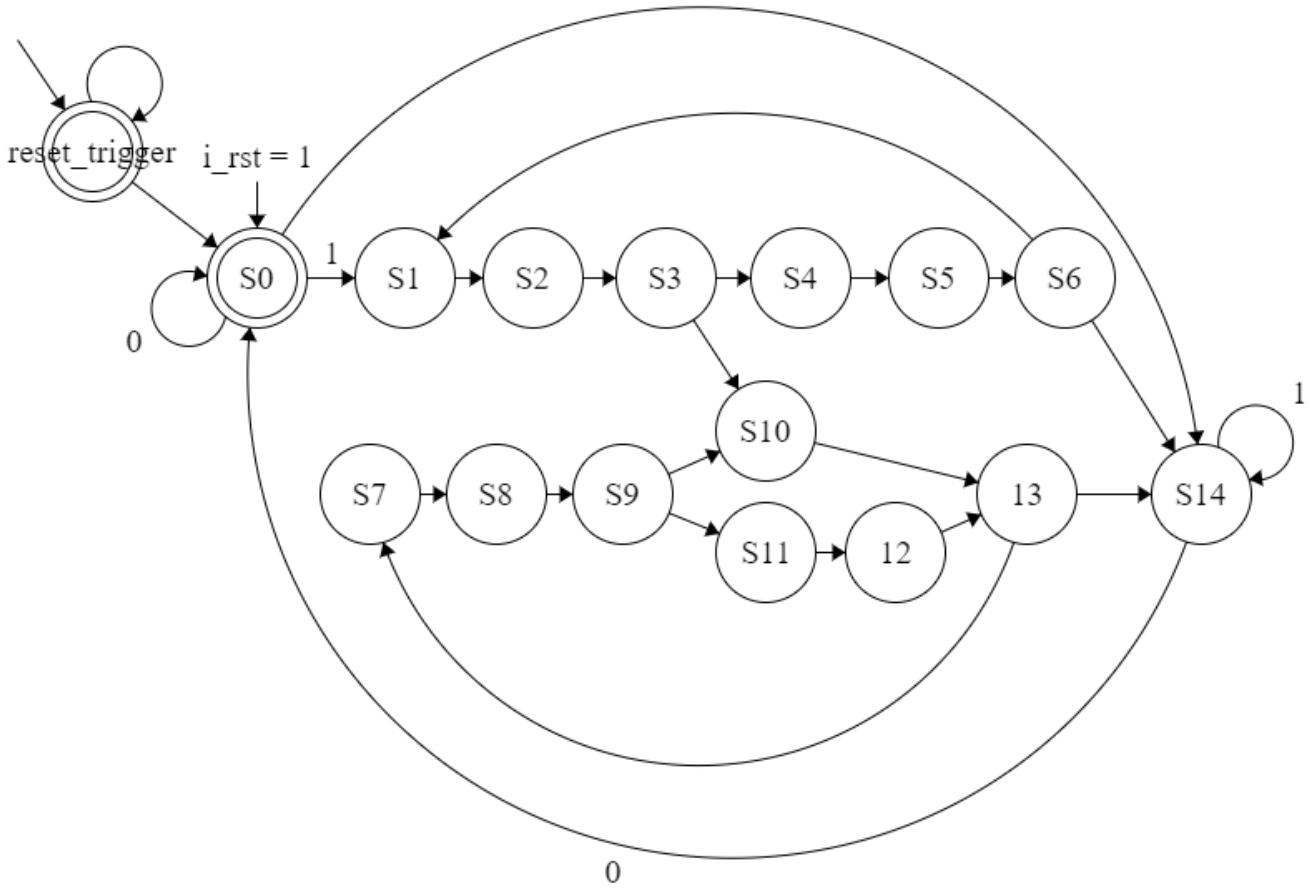


Figura 2: fsm con tutti gli stati, solo i valori di start sono stati specificati sugli archi

2.5.2 INIZIO ELABORAZIONE

(RESET_TRIGGER) Lo stato reset trigger ha lo scopo di aspettare che il valore reset diventi "1", così da passare nello stato S0. Questo stato non è mai più raggiungibile una volta entrati in S0 e permette di eliminare problemi di lettura di start = "1" con reset mai andato a "1". Se il valore di reset e start vanno entrambi a "1" nello stesso momento si entrerà direttamente in S0 senza perdere un ciclo di clock. **Tutti i segnali di uscita vanno a 0.**

(S0) In questo stato viene letto il valore di start, se è "1" inizia il ciclo di lettura/scrittura altrimenti ritorno in S0. **Viene fatto il reset di counter e C.** Questo stato è anche lo stato di ritorno in caso di **RESET ASINCRONO**. Caso speciale: se i_start = "1" ma i_k = "0" si passa direttamente in S14 (si salta il ciclo di lettura/scrittura).

2.5.3 CICLO DI LETTURA/SCRITTURA

Questo ciclo è definito tra lo stato S1 e lo stato S13 ed è diviso in 2 casi particolari. La transizione tra i due sotto-cicli è evidente dal grafo (vedasi stato `S3 -> S10`). Idea generale del ciclo (C inizializzato a 31 e si presupponga di tenere traccia del counter del ciclo):

1) leggo il primo valore dalla sequenza
2)
- se leggo un valore diverso da zero, C = 31, salvo il valore letto, inserisco C nel indirizzo del valore letto + 1.
- altrimenti, decremento C, scrivo l'ultimo valore letto nel ciclo precedente al posto del valore letto adessso, scrivo C nell'indirizzo del valore letto + 1.

3) se sono a fine sequenza termino, altrimenti ritorno al punto 1 passando all'indirizzo attuale + 2.(Quindi la lettura è fatta saltando il valore di credibilità)
FINE

Da S1 a S6 ho un sotto-ciclo di lettura/scrittura nel caso limite in cui leggo degli zeri all'inizio della sequenza. Sequenza in ordine di stato:

(S1) abilito memoria, stato di "wait" necessario per aggiornare correttamente tutti i segnali prima della lettura a seguito di riinizio ciclo;

(S2) tramite mux_scrittura (**i_sel = "001"**) faccio richiesta per leggere il valore in (indirizzo + counter*2);

(S3) se ho letto valore = "0" vado in S4 altrimenti salvo il valore in reg_data e vado in S10;

(S4) tramite mux_scrittura (**i_sel = "101"**) scrivo il valore di credibilità **C = "0"** in (indirizzo + counter*2 + 1);

(S5) incremento counter di 1 (**i_increase_counter = "1"**) e vado in S6.

(S6) se il valore di counter salvato in reg_counter è uguale a i_k vado in S14 altrimenti ricomincio il ciclo in S1. **Tutti i segnali di uscita vanno a 0.**

Da S7 a S13 ho un sotto-ciclo di lettura/scrittura dal primo valore letto diverso da zero fino all'ultimo valore della sequenza. Una volta usciti da questo ciclo non si potrà più rientrare se non dopo uno start = '0' e un successivo start = '1' (ossia la transizione S14 -> S0 -> S1). Sequenza in ordine di stato:

(S7) stato di "wait" necessario per aggiornare correttamente tutti i segnali prima della lettura a seguito di riinizio ciclo;

(S8) Tramite mux_scrittura (**i_sel = "001"**) faccio richiesta per leggere il valore in (indirizzo + counter*2);

(**S9**) se ho letto valore = 0 vado in S11 altrimenti vado in S10;

(S10) tramite mux_scrittura (**i_sel = "100"**) scrivo il valore di credibilità C in (indirizzo + counter*2 + 1), pongo il segnale **i_increase_counter = "1"** e vado in S13;

(S11) tramite mux_scrittura (**i_sel = "011"**) scrivo l'ultimo valore salvato in reg_data in (indirizzo + counter*2), pongo il segnale **i_decrease_c = "1"** e vado in S12.

(S12) tramite mux_scrittura (**i_sel = ""**) scrivo il valore di credibilità C in (indirizzo + counter*2 + 1), pongo il segnale **i_decrease_c = "0"** e vado in S13.

(S13) se il valore di counter salvato in reg_counter è uguale a i_k vado in S14 altrimenti ricomincio il ciclo in S7. **Tutti i segnali di uscita vanno a 0.**

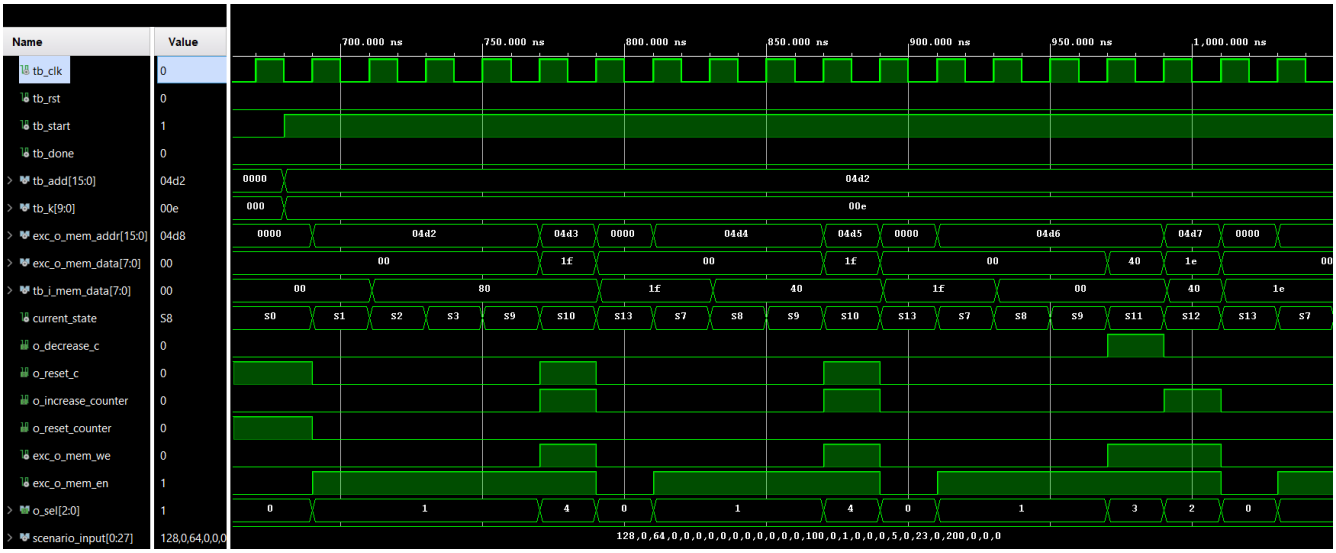


Figura 3: Waveform relativo ad una simulazione pre-sintesi in cui compaiono le due sequenze principali del cilco da S7 a S13

2.5.4 FINE ELABORAZIONE

(S14) Una volta usciti dal ciclo di lettura/scrittura la fsm pone **o_done = "1"** e aspetta che il valore di i_start torni basso. Se i_start = "1" ritorno in S14 altrimenti vado in S0. Sia il valore di credibilità C che del counter vengono resettati (**o_reset_c = "1"** e **o_reset_counter = "1"**) e tutte le altre uscite sono poste a "0".

2.5.5 AGGIORNAMENTO DEI SEGNALI

La transizione degli stati della FSM avviene sul fronte di salita del clock oppure in caso di variazione del segnale di reset o del segnale di start.
Il segnale di start è costante durante tutta l'elaborazione per specifica e quindi non interferisce con il ciclo di lettura/scrittura ma permette di avere una transizione di stato veloce per inizio e termine dell'elaborazione (quindi entrata in S0 e uscita da S14).
La transizione di stato deve necessariamente essere sensibile al reset poichè è asincrono e per specifica impone di avere tutte le uscite a "0".

2.5.6 MOTIVAZIONI ARCHITETTURA GENERALE

La rete descritta è formata da 5 moduli, di cui 1 FSM e 4 moduli di supporto. La gran parte del "lavoro" della rete è svolto dalla FSM, infatti quest'ultima ha il compito di assicurare la corretta esecuzione dei moduli e degli output della rete. Di fatto tutti i moduli sono elementi passivi che reagiscono solo ed esclusivamente quando lo richiede la FSM. Questa è una scelta progettuale dettata dalla volontà di avere un sistema di controllo più centralizzato

possibile e sopratutto più chiaro possibile nella definizione di ogni "passo" della rete. Questo non solo permette di avere un debugging più rapido, in quanto si può isolare direttamente il problema ad una determinata transizione della FSM e quindi agli output dati ai moduli negli stati in questione, ma permette di garantire anche maggiore flessibilità nell'elaborazione poiché si possono manipolare facilmente le uscite della FSM per cambiare i valori dei moduli e quindi gli output. Difficilmente si raggiungerebbe la stessa flessibilità con dei moduli con funzioni complesse che potrebbero richiedere una difficile gestione dei casi particolari.

3 RISULTATI SPERIMENTALI

3.1 SINTESI

Dal report utilization e dallo schematics della rete sintetizzata risultano:

	Flip Flop	Latch
MOD_1	8	0
MOD_2	8	0
MOD_3	11	0
MOD_4	0	0
FSM	15	0
TOT	42	0

Questi valori sono coerenti con le caratteristiche dei moduli. Non ci sono Latch poichè tutti i moduli che hanno elementi di memoria sono sincronizzati al clock mentre quelli che non hanno memoria hanno gli output definiti per ogni possibile configurazione degli input. I Moduli 1, 2 e 3 hanno un elemento di memoria rispettivamente da 8, 8 e 11 bit => un numero corrispondente di registri FLIP FLOP in sintesi. Mentre la FSM possiede 15 stati sintetizzati in 15 registri FLIP FLOP. In totale 42 registri FLIP FLOP.

La rete soddisfa ampiamente i requisiti di tempo.

Slack time (MET) : 7.675ns.

3.2 SIMULAZIONI

Di seguito una collezione di testbench con i casi limite analizzati:

(testbench_0) Contiene gli **esempi** forniti dai docenti.
Nel testbench di prova e nell'esempio 1, 4 e 5 non ci sono casi particolari da analizzare.

Nell'esempio 2 c'è un singolo valore iniziale seguito da zero consecutivi fino alla fine della sequenza, inoltre la sequenza è più lunga di quanto necessario per portare il valore di credibilità a zero. Quando questo avviene la macchina a stati finiti continua a dare il segnale per decrementare il valore di credibilità per i prossimi valori, ma il reg_c NON decrementa il valore in quanto ha già raggiunto il valore minimo. Quindi in questo caso il controllo è fatto da reg_c e non dalla FSM, si tratta di una scelta progettuale fatta per evitare di dover leggere il valore di credibilità dalla FSM e quindi aggiungere un segnale interno.

Nell'esempio 3 c'è un valore iniziale nullo che porta la FSM nel primo sotto-ciclo di lettura descritto nella sezione di architettura generale per la FSM.

(testbench_1) Contiene il caso in cui il valore i_k = 0 e quindi una **sequenza nulla**.
In questo caso la FSM rileva il valore nullo di i_k nello stato S0 e si sposta nello stato S14, dove porterà o_done a "1" e aspetterà che i_start torni a '0', per poi ritornare a S0 dove aspetterà la prossima elaborazione.

(testbench_2) Contiene il caso di una **sequenza di i_k valori costituita da soli zero**, con i_k maggiore di zero. Anche in questo caso la FSM entrerà nel primo sotto-ciclo di lettura/scrittura e non entrerà mai nel secondo sotto-ciclo passando direttamente a S14 a fine ciclo.

(testbench_3) Contiene il caso di **reset asincrono** durante l'elaborazione della sequenza.
Tutti i moduli che hanno segnali di output esterni (FSM e mux_scrittura) sono collegati al segnale i_rst e sono progettati in modo tale da portare le uscite a zero quando i_rst = "1". Nel caso dei moduli 2 e 3, ossia i registri che salvano il valore di credibilità C e del counter, questo reset è prodotto direttamente dal segnale della FSM. Quando i_rst = "1" la FSM va subito in S0 dove i segnali di i_reset_c e i_reset_counter vanno a "1" e quindi, nel fronte di salita del clock subito successivo al reset, C e counter verranno resettati. Nel caso del reg_data il valore di stored_value non deve essere necessariamente resettato (è comunque resettato) in quanto il modo in cui è fatto il ciclo di lettura/scrittura permette di non usare vecchi valori di stored_data di un'elaborazione precedente.

(testbench_4) Contiene il caso di **elaborazioni multiple**.
Come già spiegato nel paragrafo di architettura principale, una volta finita una elaborazione la FSM tornerà nello stato S0 dove resetterà il valore di C e counter e aspetterà un i_start = "1" per poi iniziare un nuovo ciclo di lettura/scrittura.

(testbench_5) Contiene il caso di una **sequenza con i_k massimo** ossia i_k = 1023.
In questo caso il counter nel mux_scrittura può raggiungere un valore massimo di 2046 ossia il doppio di i_k. Questo avviene perché per inserire i valori nell'indirizzo corretto, nel mux_scrittura, è fatto uno shift a sinistra di 1 bit di i_counter => moltiplicazione per 2 di i_counter. Questo significa che il valore massimo che raggiunge i_counter NON è 1023 come i_k ma 2046 e quindi i_counter deve essere un segnale da 11 bit e tutti i segnali ad esso collegato sono a 11 bit. Quindi il valore di counter nel reg_counter è a 11 bit.

(testbench_6) Contiene il caso di **start alto prima di reset iniziale**.
In questo caso il valore di i_start è alto fin dall'inizio, ancora prima che i_rst sia alto, quindi prima che venga resettato il componente. Dopo che viene resettato il componente e il valore di i_rst torna basso la rete inizia ad elaborare la sequenza. Questo è possibile grazie allo stato reset_trigger che aspetta che la rete sia correttamente resettata prima che sia possibile "leggere" il valore di start nello stato successivo.

(Altri casi limite considerati ma senza testbench perchè banali o impossibili)

- Reset fuori dal ciclo lettura/scrittura (identico al reset dentro al ciclo di lettura scrittura)

- Valore i_add = massimo indirizzo memoria o out_of_bound (impossibile poiché si considerano valori in input sempre corretti)

- Valore di i_start che torna a "0" durante l'elaborazione (impossibile per specifica)

- Elaborazioni multiple con sovrascrittura di parti di memoria precedentemente elaborate, esempio: due elaborazioni, una che parte da i_add e l'altra che parte da i_add+1 (non ci sono dettagli nella specifica che impongono di verificare che questo non avvenga).

4 CONCLUSIONI

La rete progettata è funzionante poiché passa tutte le simulazioni sui Testbench prodotti (anche post sintesi). Tutti i requisiti della specifica sono stati raggiunti e i possibili casi limite sono stati analizzati e testati. Le prestazioni della rete non sono state analizzate poiché non richiesto. Si consideri per esempio che la FSM non è minima, in quanto non era richiesto di ottimizzare la rete. Ho optato invece per un approccio di rete semplice e il più possibile facile da capire con meno diramazioni possibili ad ogni stato (i due sottocicli principali e il ciclo "main" sono così evidenti ad occhio).