Progetto di Reti Logiche

POLITECNICO
MILANO 1863

Camilla Angela Bongiovanni (Codice Persona 10766833 - Matricola 984247)

Riccardo Cadario (Codice Persona 10788669 - Matricola 982317)

Indice

1 Introduzione **3**
1.1 Scopo del progetto . . . 3
1.2 Specifiche generali . . . 3
1.3 Interfaccia del modulo da progettare . . . 5

2 Architettura e design **6**
2.1 Address register . . . 6
2.2 Counter K register . . . 6
2.3 Word register . . . 6
2.4 Credibility register . . . 7
2.5 Data selector mux . . . 7
2.6 Finite State Machine (FSM) . . . 7
2.6.1 Descrizione dettagliata degli stati: . . . 7

3 Esiti sperimentali **10**
3.1 Sintesi . . . 10
3.2 Simulazioni . . . 10
3.2.1 caso generale: . . . 10
3.2.2 Bit di credibilità a 0: . . . 11
3.2.3 Multiarrivi di 0: . . . 11
3.2.4 Conclusioni: . . . 11

1 Introduzione

Il documento mostra tutte le nostre scelte architetturali prese per risolvere la prova finale di reti logiche, riassumendo la specifica del progetto ed esponendo i risultati delle simulazioni e della sintesi.

1.1 Scopo del progetto

Lo scopo del progetto è quello di sviluppare un modulo hardware in VHDL che possa gestire una sequenza di parole memorizzate in una memoria. La sfida principale consiste nel completare questa sequenza in modo intelligente, sostituendo i valori nulli con l'ultimo valore valido letto precedentemente e calcolando un valore di "credibilità" associato a ciascuna parola.

1.2 Specifiche generali

Il modulo richiesto presenta 3 ingressi principali (figura 1): START (1 bit), ADD (16 bit) e K (numero di parole, 10 bit) e deve essere in grado di interagire con una memoria RAM. Sono inoltre presenti altri due segnali sincroni e unici per l'intero sistema, CLK (segnale di clock) e RESET (segnale di reset), necessari per, rispettivamente, sincronizzare le operazioni del modulo e reinizializzarlo. L'uscita principale è DONE (1 bit).

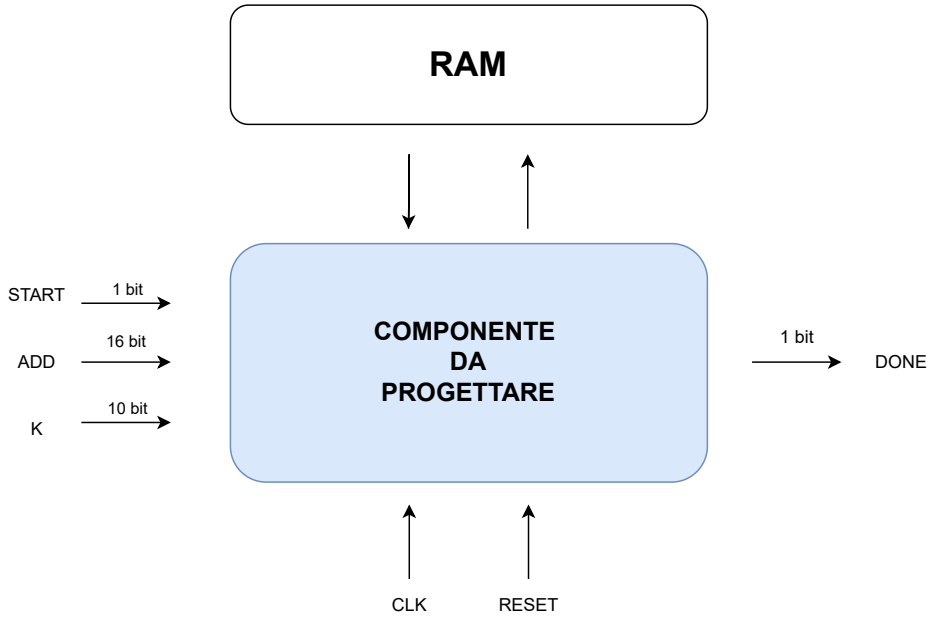


Figura 1:

All'inizio, il segnale di reset (RESET) riporta DONE a 0. Quando START viene attivato dopo il reset, il modulo legge una sequenza di K parole W (il cui valore è compreso tra 0 e 255) a partire dall'indirizzo specificato (ADD). Il modulo calcola anche un valore di "credibilità" C per ogni parola. In particolare, la "credibilità" è inizializzata a 31 ogni volta che si incontra un valore non zero nella sequenza. Al contrario quando la parola letta dal modulo è 0 la "credibilità" viene decrementata di uno rispetto al suo valore precedente, fino a un minimo di 0. Il valore di "credibilità" viene memorizzato nel byte successivo alla parola stessa. Quando la sequenza è completata, il modulo alza il segnale DONE, indicando il termine dell'elaborazione. DONE rimane alto fino a quando START non ritorna a 0. Durante questo periodo, il modulo non può accettare un nuovo START. Prima di ogni nuova elaborazione, il modulo viene reinizializzato con il segnale di reset (RESET).

Ecco riportati alcuni esempi esplicativi:

- Sequenza di partenza 1:

  ```
  128 0 64 0 0 0 0 0 0 0 0 0 0 0 100 0 1 0 0 0 5 0 23 0 200 0 0 0
  ```

  Sequenza finale:

  ```
  128 31 64 31 64 30 64 29 64 28 64 27 64 26 100 31 1 31 1 30 5 31 23 31
  200 31 200 30
  ```

- Sequenza di partenza 2:

  ```
  0 0 0 0 0 0 0 0 0 0
  ```

  Sequenza finale:

  ```
  0 0 0 0 0 0 0 0 0 0
  ```

- Sequenza di partenza 3:

  ```
  100 0 0 0 0 0 0 0 0 0
  ```

  Sequenza finale:

  ```
  100 31 100 30 100 29 100 28 100 27
  ```

- Sequenza di partenza 4:

  ```
  121 0 171 0 244 0 11 0 101 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0
  ```

  Sequenza finale:

  ```
  121 31 171 31 244 31 11 31 101 31 101 30 101 29 101 28 101 27 101 26 101 25
  101 24 101 23 101 22 101 21 101 20 101 19 101 18 101 17 101 16 101 15 101 14
  101 13 101 12 101 11 101 10 101 9 101 8 101 7 101 6 101 5 101 4 101 3 101 2
  101 1 101 0 101 0 101 0
  ```

1.3 Interfaccia del modulo da progettare

Il componente da progettare ha un'interfaccia così definita (tabella 1):

```
entity project_reti_logiche is
    port (
        -- INPUTS
        i_clk  : in std_logic;
        i_rst  : in std_logic;
        i_start: in std_logic;
        i_add  : in std_logic_vector(15 downto 0);
        i_k    : in std_logic_vector(9 downto 0);

        -- OUTPUT
        o_done: out std_logic;

        -- MEMORY
        i_mem_data: in std_logic_vector(7 downto 0);

        o_mem_en  : out std_logic;
        o_mem_we  : out std_logic;
        o_mem_addr: out std_logic_vector(15 downto 0);
        o_mem_data: out std_logic_vector(7 downto 0)
    );
end entity;
```

Tabella 1: Entity.

In particolare nella seguente tabella è riportata una descrizione più accurata del ruolo svolto da ciascun segnale coinvolto:

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 (e la prepara ad una nuova esecuzione)
i_start	È il segnale di START generato dal Test Bench per iniziare l'esecuzione
i_add	È il segnale (vettore) ADD generato dal Test Bench che rappresenta l'indirizzo dal quale parte la sequenza da elaborare
i_k	È il segnale (vettore) K generato dal Test Bench che rappresenta la lunghezza della sequenza
o_done	È il segnale DONE 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 e contiene il dato in seguito ad una richiesta di lettura

Tabella 2: Signals' description.

2 Architettura e design

L'architettura del componente è organizzata in maniera strutturata, sfruttando cinque moduli principali, ognuno dei quali svolge una funzione specifica. Questa suddivisione, oltre ad essere stata consigliata a lezione, permette una gestione chiara e un'individuazione veloce di eventuali errori.

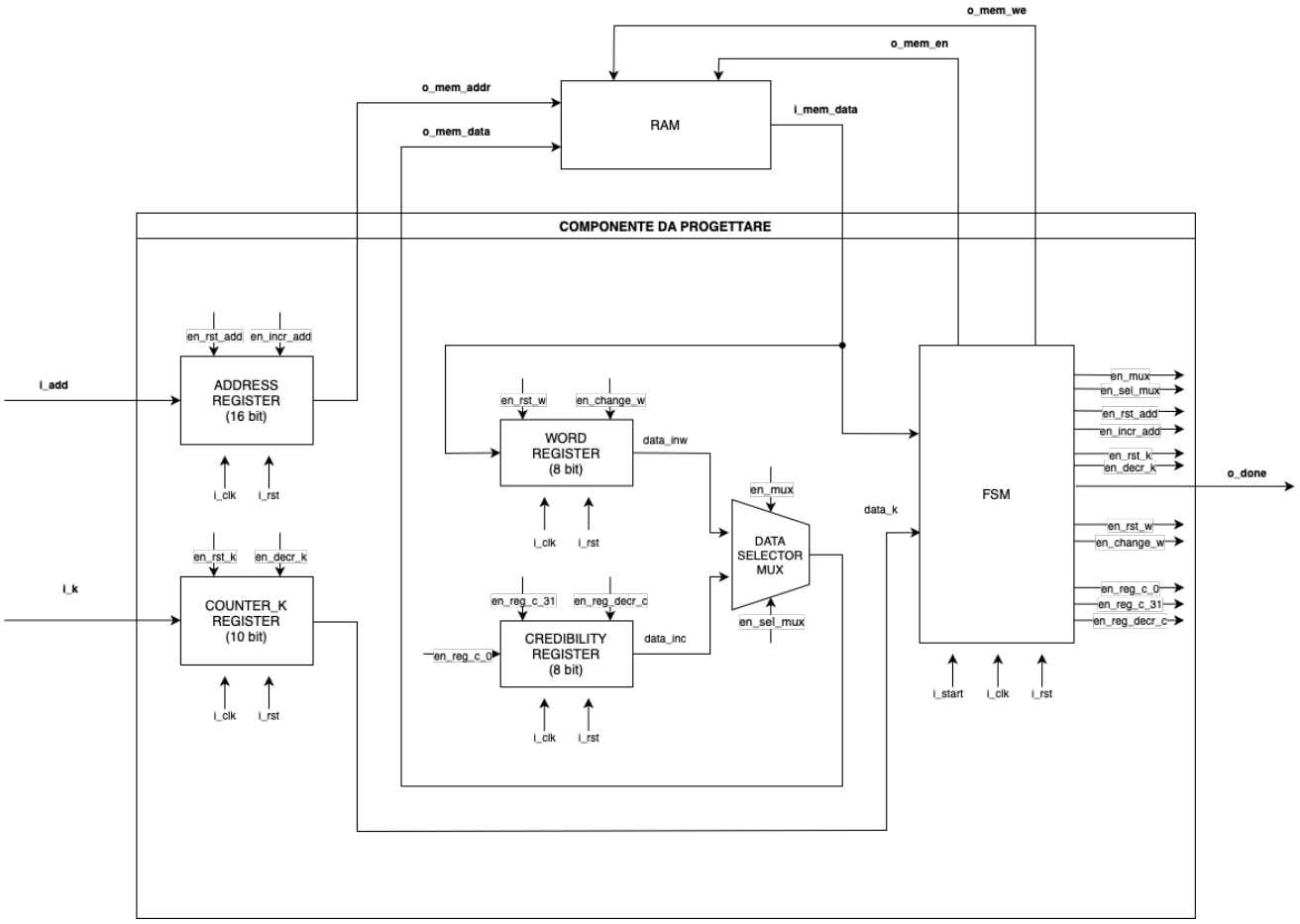


Figura 2: Datapath completo.

2.1 Address register

Questo modulo rappresenta un registro a 16 bit che, una volta attivo il segnale i_start e abilitato (dalla FSM) en_rst_add, riceve e memorizza il valore di memoria iniziale (i_add). In seguito, qualora la FSM attivi en_incr_add, il dato precedentemente immagazzinato viene incrementato di 1 e salvato direttamente nel segnale di uscita o_mem_addr (che lo indirizza direttamente alla RAM).

2.2 Counter K register

Questo modulo rappresenta un registro a 10 bit che, una volta attivo il segnale i_start e abilitato (dalla FSM) en_rst_k, riceve e memorizza il valore che rappresenta il numero di parole presenti nella sequenza da elaborare (i_k). In seguito, qualora la FSM attivi en_decr_k, il dato precedentemente immagazzinato viene decrementato di 1 e salvato direttamente nel segnale di uscita data_k (che lo indirizza direttamente alla FSM).

2.3 Word register

Questo modulo rappresenta un registro a 8 bit che, una volta attivo il segnale i_start e abilitato (dalla FSM) en_rst_w, viene inizializzato salvando al suo interno il valore 0. Successivamente, abilitato il segnale en_change_w, il registro riceve dalla RAM, tramite i_mem_data, il valore della nuova parola. Se quest'ultima è diversa da 0 (o comunque dal valore salvato in precedenza), data_inw assume tale valore per indirizzarlo al multiplexer.

2.4 Credibility register

Questo modulo rappresenta un registro a 8 bit che, una volta attivo il segnale i_start e abilitato (dalla FSM) en_rst_c_0, viene inizializzato salvando al suo interno il valore 0. Successivamente, abilitato il segnale en_rst_c_31, il registro salva il valore massimo che la credibilita può assumere, ossia 31; invece quando viene attivato en_reg_decr_c, se il valore attuale del registro è maggiore di 0, quest'ultimo viene decrementato. Ci siamo assicurati che non assuma mai valori negativi. Infine il valore, tramite data_inc viene indirizzato al multiplexer.

2.5 Data selector mux

Questo modulo rappresenta un multiplexer che, ricevuti in input due valori: credibilità (data_inc) e l'ultima parola memorizzata (data_inw), seleziona quale dei due inoltrare all'uscita in base al segnale di controllo. Una volta abilitato il segnale en_mux, se en_sel_mux assume il valore 0, il valore di data_inw viene memorizzato in o_mem_addr. Altrimenti, en_sel_mux sel assume il valore 1, viene memorizzato data_inc. L'uscita o_mem_addr è direttamente collegata alla RAM.

2.6 Finite State Machine (FSM)

Questo modulo è responsabile della gestione delle operazioni sui componenti interni e sulla memoria, coordinandole tramite i vari segnali di output. Si tratta di una FSM di Moore, in quanto le uscite sono determinate esclusivamente dallo stato corrente della macchina, garantendo così un controllo sequenziale e preciso delle operazioni indipendentemente dagli ingressi.

La FSM è progettata per passare attraverso 10 diversi stati, ognuno dei quali ha uno scopo ben preciso: inizializzazione e reset dei registri, lettura e attesa dei dati dalla RAM, modifica della credibilità, incremento dell'indirizzo di memoria, scrittura dei dati aggiornati nella memoria.

2.6.1 Descrizione dettagliata degli stati:

INIT STATE (S0): Questo stato rappresenta la fase iniziale, in cui il sistema attende il segnale di start. Se il reset (i_rst) è attivo, il sistema viene mantenuto in questo stato fino a quando non viene disattivato. Una volta che il segnale i_start è impostato a '1', la FSM si sposta al prossimo stato operativo.

RESET REGISTERS (S1): In questo stato, i registri e i contatori necessari per l'elaborazione vengono resettati e preparati utilizzando i seguenti segnali di controllo: en_rst_k (per il registro del contatore k), en_rst_add (per il registro degli indirizzi), en_rst_c_0 (per il registro della credibilità) e en_rst_w (per il registro delle parole). È importante notare che la credibilità viene inizialmente impostata a 0, permettendo alla FSM di gestire in modo efficiente anche il caso limite in cui la sequenza in ingresso sia composta esclusivamente da zeri, senza la necessità di aggiungere stati aggiuntivi nella FSM.

READ DATA (S2): Una volta impostato l'indirizzo da cui leggere nella RAM (S1), la FSM in questo stato abilita la lettura dei dati dalla memoria attraverso l'attivazione del segnale o_mem_en. Se il contatore data_k è diverso da 0, il sistema inizia a leggere i dati, altrimenti si sposta nello stato iniziale, in quanto la dimensione della sequenza da leggere risulta nulla.

WAIT FOR DATA (S3): In questo stato, la FSM attende dato letto dalla memoria (i_mem_data). Se un dato valido (ossia diverso da 0) viene ricevuto, si passa allo stato S4, altrimenti allo stato S5.

SET CREDIBILITY (S4): In questo stato, il valore della credibilità viene aggiornato a 31, poiché l'ultima parola letta dalla memoria è risultata valida, ossia diversa da 0.

MODIFY CREDIBILITY and WRITE the right W (S5): Se nel passo precedente non è stato ricevuto un dato valido (i_mem_data = 0), la FSM procede a modificare il valore di credibilità, attivando en_decr_cred per decrementare la credibilità, a condizione che sia ancora maggiore di zero; in caso contrario, non viene eseguita alcuna modifica. Successivamente, vengono attivati i segnali o_mem_we e o_mem_en per scrivere i dati aggiornati nella memoria.

ADDRESS INCREMENT (S6): In questo stato, l'indirizzo di memoria viene incrementato, attivando il segnale en_incr_add, preparandosi a scrivere il valore della credibilità.

WRITE CREDIBILITY (S7): La FSM abilita la scrittura dei nuovi valori di credibilità nella memoria. Attraverso l'attivazione del multiplexer (en_mux) e del suo selettore (en_sel_mux), viene selezionato il valore corretto da inviare alla RAM. Entrambi i segnali vengono dunque attivati per garantire la corretta scelta.

NEW SEQUENCE TO READ (S8): Dopo aver scritto i dati, la FSM verifica se data_k è ancora maggiore di zero. Se è vero, il ciclo riparte dallo stato READ DATA (S2) per elaborare una nuova parola. In caso contrario, si passa allo stato di completamento (S9).

COMPLETE SEQUENCE STATE (S9): In questo stato finale, la FSM segnala il completamento del processo di elaborazione di una sequenza impostando o_done a '1'. Se il segnale i_start viene riportato a '0', il sistema torna allo stato iniziale, pronto per una nuova esecuzione.

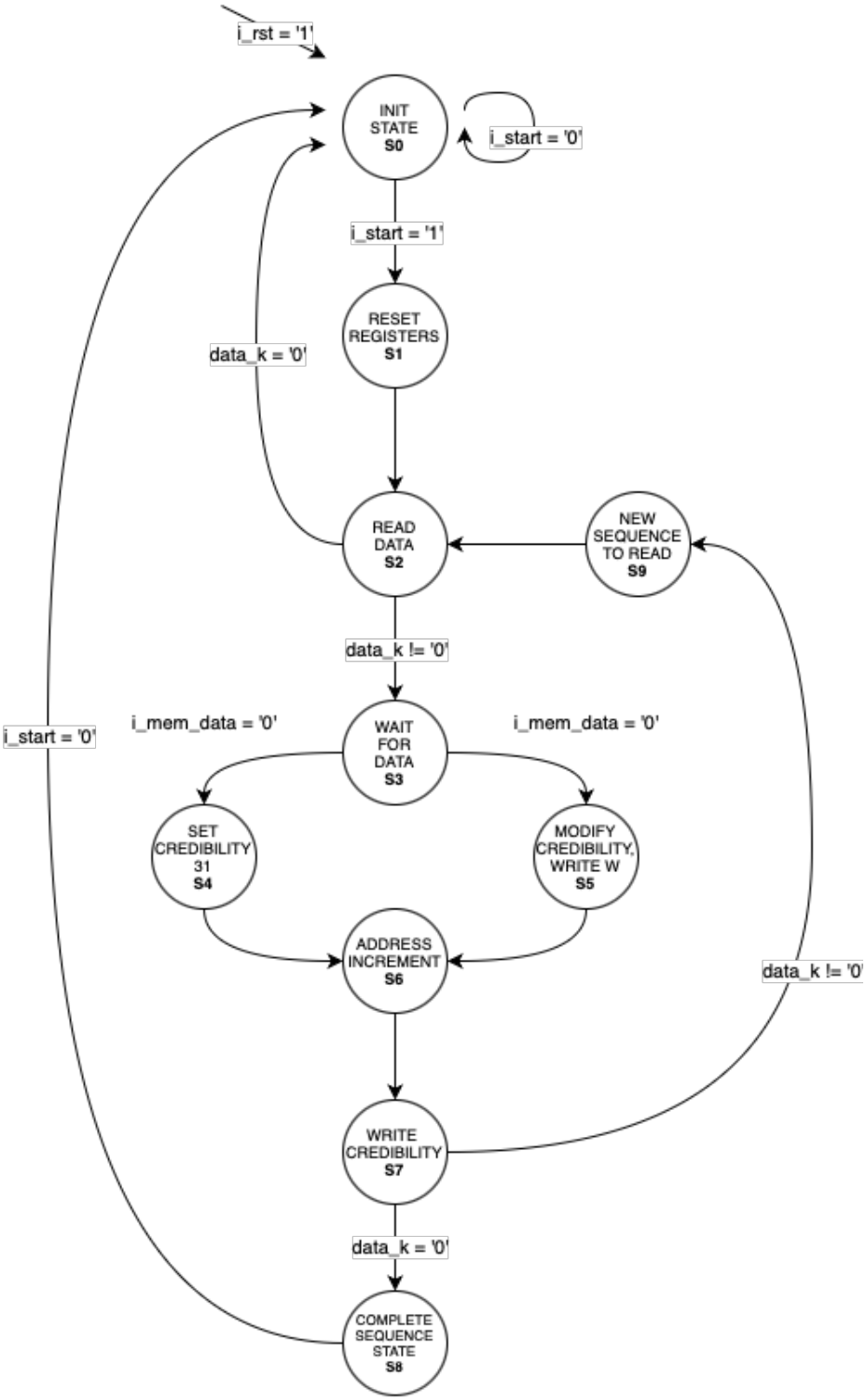
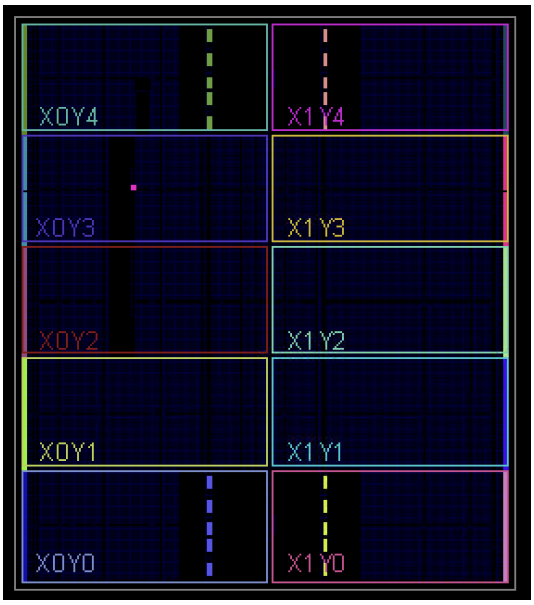


Figura 3: Finite state machine.

3 Esiti sperimentali

3.1 Sintesi

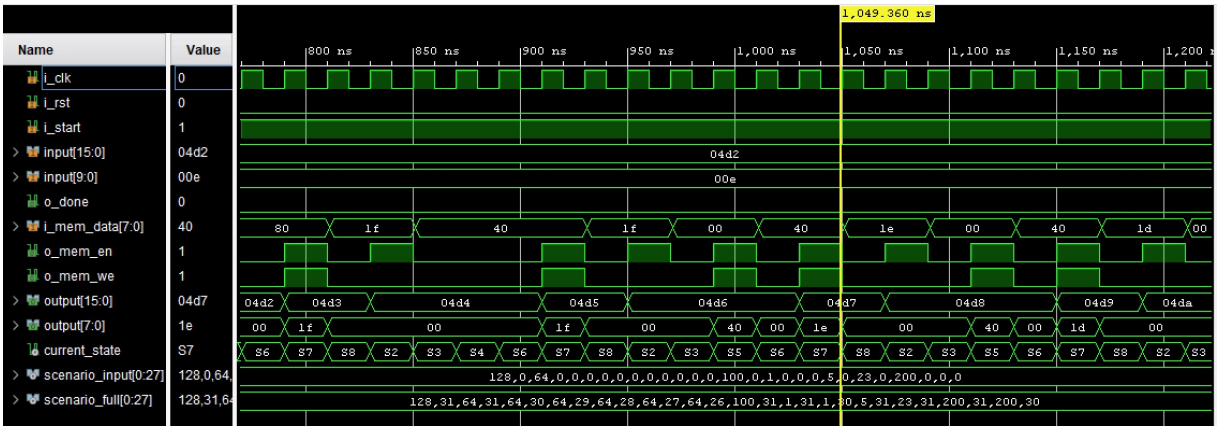
Il programma si è dimostrato funzionante in tutti i casi da noi testati, sia in fase di pre-sintesi che in post-sintesi. A testimonianza dei risultati ottenuti, riportiamo qui sotto le immagini relative alla sintesi dei componenti e degli esempi di simulazioni.



3.2 Simulazioni

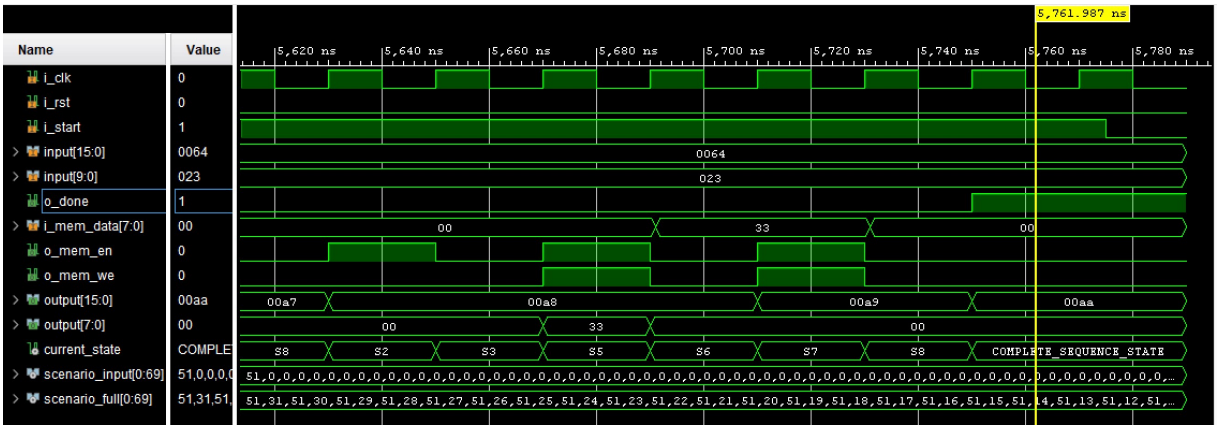
Tutte le simulazioni che abbiamo condotto hanno portato a un risultato positivo, in particolare oltre ai test forniti dal prof abbiamo sottoposto i test degli altri scaglioni e un generatore di test che fornisce piu di 30 sequenze. Forniamo qui sotto alcuni esempi di casistiche:

3.2.1 caso generale:



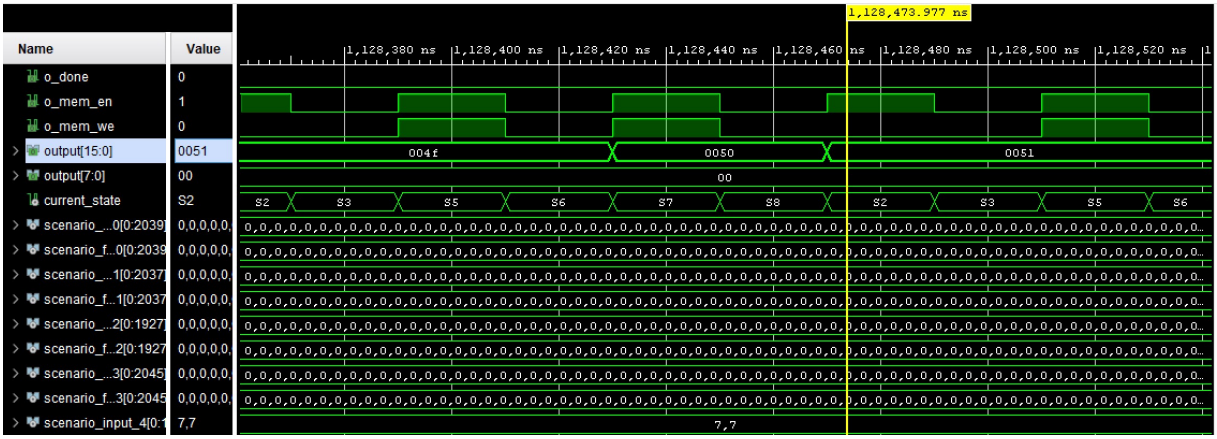
Questa immagine mostra il comportamento di una simulazione post sintesi in un caso generico. Si puo notare, confrontando l'output con scenario-full, che il programma si comporta correttamente.

3.2.2 Bit di credibilità a 0:



Questa immagine mostra il comportamento di una simulazione post sintesi nel caso in cui il bit di credibilità venga decrementando fino a zero, senza però diventare negativo, come precisato nelle specifiche.

3.2.3 Multiarrivi di 0:



Questa immagine invece mostra il comportamento sempre in post sintesi nel caso in cui la sequenza sia fatta di soli 0, si nota anche l'arrivo di piu sequenze differenti.

3.2.4 Conclusioni:

Utilizzando il comando report-utilization possiamo andare piu in profondità e raccogliere più informazioni, riportiamo qui sotto la tabella slice logic:

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	66	0	134600	0.05
LUT as Logic	66	0	134600	0.05
LUT as Memory	0	0	46200	0.00
Slice Registers	46	0	269200	0.02
Register as Flip Flop	46	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Si puoi notare il numero nullo di latch e l'utilizzo dei Flip Flop, nel nostro caso 46. In conclusione, la specifica è stata soddisfatta progettando un'architettura efficiente, come testimoniato rispettivamente dalle simulazioni sopra mostrate.