Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo - Anno 2023/2024

Yana Siao (Codice Persona ***- Matricola***)

Veronica Viceconti (Codice Persona ***- Matricola ***)

1. Introduzione

1.1 Scopo del progetto

La prova finale per l'anno accademico 2023/2024 richiede lo sviluppo di un modulo hardware in VHDL in grado di interfacciarsi con una memoria e di eseguire specifiche operazioni su un insieme di dati. In particolare, il modulo deve:

- Accedere a una porzione di memoria contenente K parole,
- Elaborare e scrivere valori numerici da 0 a 255 nelle celle pari,
- Basandosi sulle elaborazioni precedenti, scrivere la credibilità della parola corrispondente nelle celle dispari,
- Gestire inizio e fine dell'elaborazione e segnalare la fine.

1.2 Specifiche generali

La sequenza di K parole W da elaborare è memorizzata a partire da un indirizzo specificato, ogni 2 BYTE. Il byte mancante dovrà essere completato sostituendo lo zero e inserendo un valore di credibilità C, per ogni valore della sequenza. Questo avviene copiando l'ultimo valore valido letto precedente e appartenente alla sequenza.
La credibilità è pari a 31 ogni volta che il valore W della sequenza è non zero, mentre viene decrementato rispetto al valore precedente ogni volta che si incontra uno zero in W. Il valore C è sempre >=0 e torna ad essere 31 ogni volta che si incontra un valore W diverso da zero. Una volta raggiunto 0 non viene ulteriormente decrementato.

ESEMPIO

-Sequenza di input:

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 di output:

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

1.3 Interfaccia del componente

Il componente da descrivere ha un'interfaccia così definita:

```
entity project_reti_logiche is
    port (
        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);

        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_data : out std_logic_vector(7 downto 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

Dove troviamo:

- i_clk, segnale di clock in ingresso generato dal test bench;
- i_rst, segnale di reset in ingresso che inizializza la macchina per ricevere il primo segnale di start;
- i_start, segnale in ingresso generato dal test bench che determina l'inizio dell'elaborazione della sequenza in memoria;
- i_add, segnale in ingresso generato dal test bench che rappresenta l'indirizzo da cui inizia la sequenza da elaborare;
- i_k, segnale in ingresso generato dal test bench e rappresenta la lunghezza della sequenza da elaborare;
- o_done, segnale di output che comunica la fine dell'elaborazione della sequenza corrente;
- o_mem_addr, segnale (vettore) di output che comunica l'indirizzo alla memoria;
- i_mem_data, segnale (vettore) in ingresso che arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura;
- o_mem_data, segnale (vettore) in uscita che viene mandato alla memoria e rappresenta il dato che verrà successivamente scritto;
- o_mem_we, segnale di output che permette di comunicare in scrittura con la memoria;
- o_mem_en, segnale di output che permette di comunicare con la memoria sia in lettura che in scrittura;

2. Design

2.1 Metodo risolutivo

Il componente principale è una macchina a stati finiti (FSM) che implementa il seguente ciclo di elaborazione:

1. **Attesa segnale Start:**
    - Attende il segnale di avvio i_start = 1,
2. **Ciclo di Elaborazione:**
    - Inizializzazione dei moduli X, Y, CELLA, DATA,
    - Per ogni coppia di dati in memoria (Wi, Ci), dove con Wi si intende la parola i-esima e con Ci la credibilità corrispondente i-esima:
        - Se Wi = 0:
            - Sostituisce Wi con X,
            - Decrementa Y di 1, salva il risultato e modifica la memoria in posizione corrispondente alla credibilità Ci,
        - Altrimenti:
            - Aggiorna X con la nuova Wi,
            - Imposta Y e Ci a 31,
    - Alza il segnale o_done, quando serve, per indicare la fine dell'elaborazione,
3. **Attesa Nuovo Avvio:**
    - Attende l'abbassamento del segnale i_start,
    - Torna nello stato di attesa di nuovo i_start,

Per permettere ad FSM di lavorare correttamente abbiamo anche un altro processo che si occupa di gestire il RST e il CLK:

1. Se in input si ha un rst= 1 allora FSM verrà riportato allo stato di partenza,
2. Se rst = 0 e abbiamo un rising_edge del clk, allora si passa al prossimo stato della macchina.

Moduli:

- X, contiene l'ultima parola (Wi) diversa da 0 che viene copiata in memoria quando successivamente si incontrano parole Wi=0,
- Data, contiene la parola appena letta (non contiene mai la credibilità),
- Y, memorizza la credibilità dell'ultima parola elaborata,
- Cella, è un contatore che va da 0 fino a K*2 e tiene conto della cella di memoria corrente da cui prendere o su cui memorizzare i dati.

I moduli verranno ulteriormente discussi nella parte di architettura.

2.2 Stati della FSM

2.2.1 SApettoStart

La macchina torna nello stato iniziale dopo i_rst = 0 e rimane in questo stato finché non legge un segnale i_start = 0 per garantire che Reset si è abbassato ed in ingresso abbiamo un nuovo segnale i_start.

2.2.2 SInizializzazione

Inizializzazione moduli X, Y, Cella, Data. FSM è in attesa di i_start = 1 per passare nello stato SValutFine, altrimenti rimane nello stato SInizializzazione.

2.2.3 SValutFine

Dopo l'elaborazione di una coppia la macchina paragona il contatore e K*2. Se tale condizione viene soddisfatta, allora la fase di elaborazione è finita e FSM alza il segnale o_done e passa nello stato di attesa di abbassamento di i_start.
Altrimenti la macchina manda una richiesta di lettura alla memoria. Lo stato successivo è SLettura.

2.2.4 SLettura

Stato di attesa di risposta dalla memoria dopo la richiesta dei dati. Siccome il segnale con il quale la memoria comunica con il componente progettato ha un ritardo di propagazione, una delle soluzioni possibili è usare uno stato di attesa della propagazione.

2.2.5 SLetturaVera

FSM salva il nuovo valore letto dentro il modulo Data e passa allo stato di elaborazione.

2.2.6 SScrivoParola

Se la parola letta è 0, allora sovrascrive la cella di memoria con il valore del modulo X, aumenta il contatore ed avvisa Y di dover diminuire il valore di Credibilità. Il prossimo stato è SMenoCredibilita.

Se la parola letta è diversa da 0, allora non ha bisogno di sovrascrivere la parola, però deve salvare il nuovo valore nel modulo X, aggiornare il valore di Y ed aumentare il contatore. Il prossimo stato è SCredibilita31.

2.2.7 SCredibilita31

FSM sovrascrive nella cella di memoria, sostituendo il valore 0 (presente in memoria) con 31.
L'elaborazione di una parola è finita e quindi il prossimo stato è SValutFine.

2.2.8 SMenoCredibilita

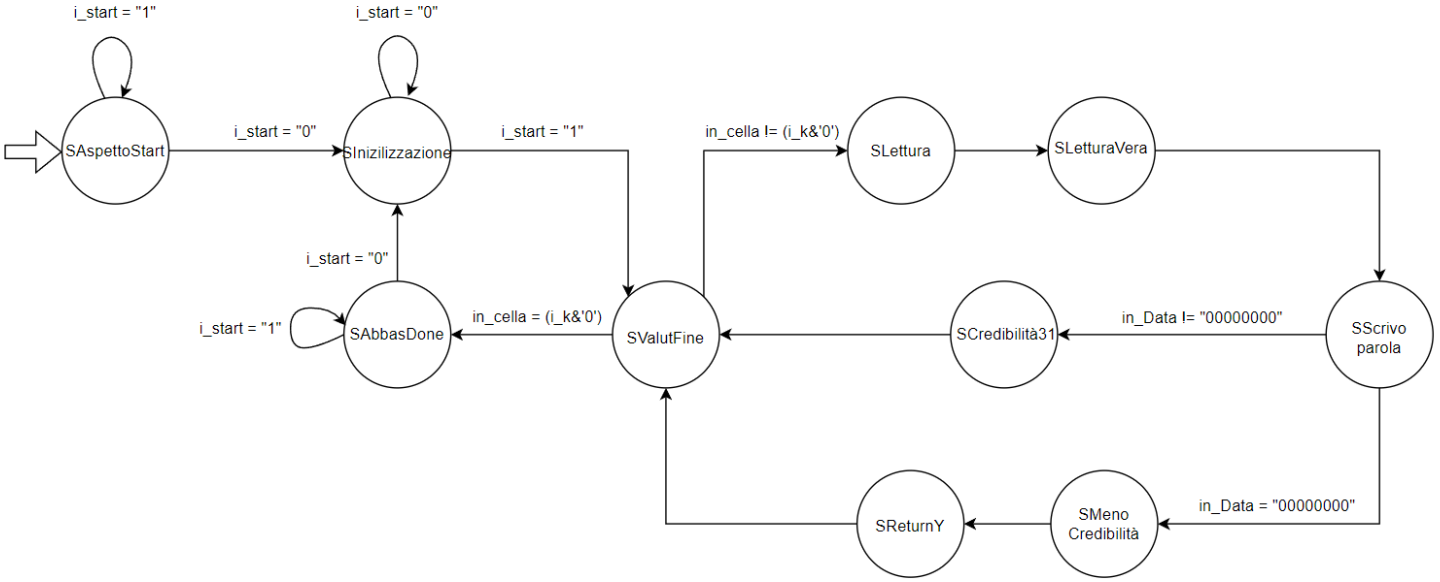
FSM aspetta la propagazione del valore appena scritto. Il prossimo stato è SReturnY.

2.2.9 SReturnY

FSM sovrascrive la cella di memoria, sostituendo il valore 0 (presente in memoria) con il valore del modulo Y.
L'elaborazione di una parola è finita e quindi il prossimo stato è SValutFine.

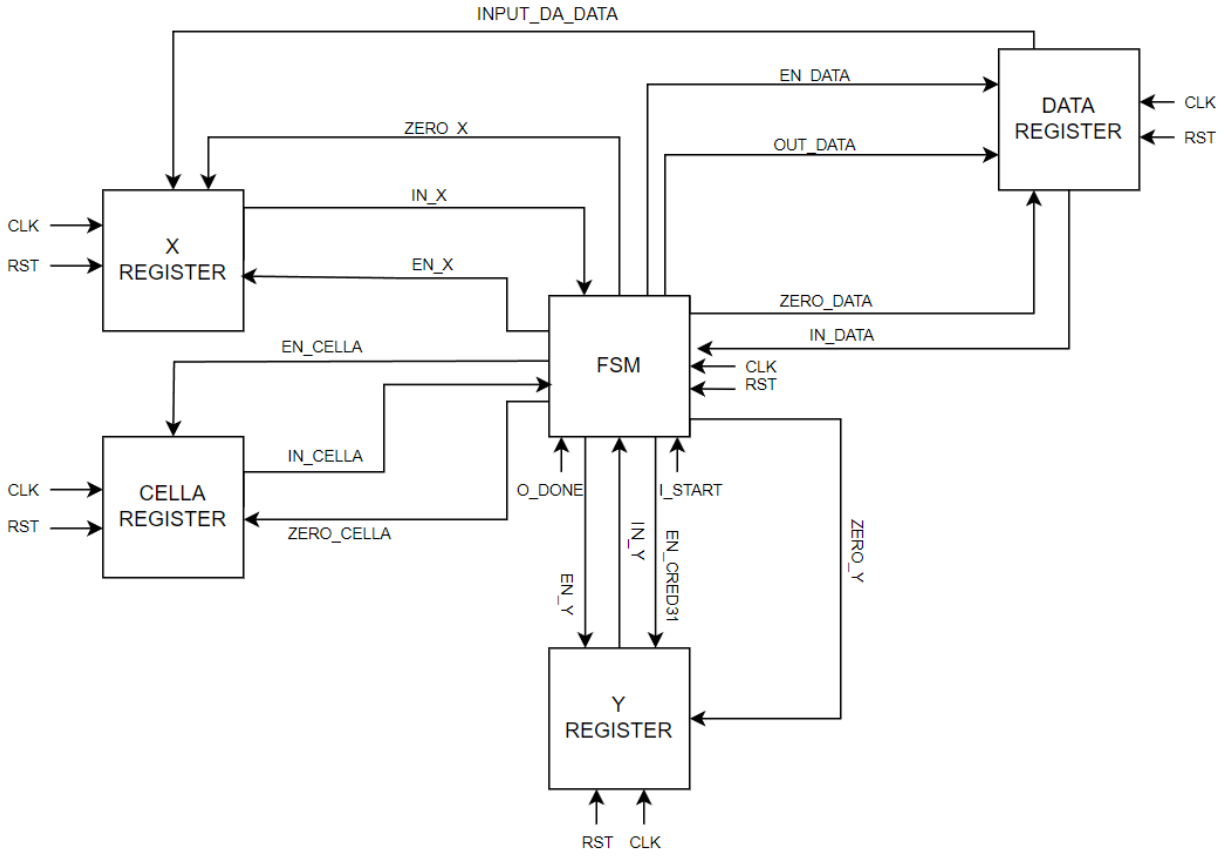
2.2.10 SAbbasDone

Finché i_start = 1 FSM rimane in questo stato. Quando i_start = 0 la macchina abbassa o_done = 0 ed è pronta per un nuovo ciclo di elaborazione e passa allo stato SInizializzazione.



Rappresentazione della Macchina a stati

2.3 Architettura



Rappresentazione dell'architettura

2.3.1 FSM

Componente principale del progetto, si occupa di attivare e disattivare gli altri componenti e di controllare che tutto funzioni correttamente, di seguito vengono spiegati alcuni segnali aggiuntivi:

- in_data: segnale in input dal modulo data che rappresenta la parola Wi presente in memoria all'indirizzo specificato,
- zero_data: segnale di enable per reinizializzare il modulo DATA,
- en_data: segnale di enable per attivare il modulo DATA,
- out_data: segnale che rappresenta la parola Wi ritornata dalla memoria ad un indirizzo specificato,
- zero_y: segnale di enable per reinizializzare il modulo Y,
- en_Cred31: segnale di enable per indicare al modulo Y che deve reinizializzarsi al valore massimo 31,
- in_y: segnale in input dal modulo Y che rappresenta il valore aggiornato di Y,
- en_Y: segnale di enable per attivare il modulo Y,
- zero_cella: segnale di enable per reinizializzare il modulo CELLA,
- in_Cella: segnale di input dal modulo CELLA che rappresenta il valore aggiornato di CELLA,
- en_cella: segnale di enable per attivare il modulo CELLA,
- en_x: segnale di enable per attivare il modulo X,
- in_x: segnale di input dal modulo X che rappresenta il valore aggiornato di X,
- zero_x: segnale di enable per reinizializzare il modulo X

2.3.2 X module

Modulo che si occupa di tenere aggiornato qual è l'ultima parola Wi presente in memoria diversa da 0. Di seguito i segnali che non sono già stati precedentemente descritti:

- input_da_data: segnale di input dal modulo DATA, è lo stesso segnale in_data presente in ingresso al FSM

2.3.3 Y module

Modulo che si occupa di tenere aggiornato il valore della credibilità della parola Wi.
I segnali di questo modulo sono già stati tutti precedentemente descritti.

2.3.4 DATA module

Modulo che si occupa di salvare la parola che arriva dalla memoria.
I segnali di questo modulo sono già stati tutti precedentemente descritti.

2.3.5 CELLA module

Modulo che rappresenta un contatore crescente del numero di celle di memoria già elaborate, che permette di tenere conto di quale sia la prossima cella da elaborare.
I segnali di questo modulo sono già stati tutti precedentemente descritti.

2.3.6 Progetto_reti_logiche

Modulo responsabile del collegamento e della comunicazione tra i vari moduli del progetto.
Contiene un componente e un portmap per ogni modulo che elabora dati. L'utilizzo del progetto_reti_logiche dedicato per la gestione solo della comunicazione facilita la manutenzione del progetto e rende il codice più chiaro e leggibile.

3. Scelte progettuali

1. La scelta principale è stata quella di realizzare il componente assegnando la maggior parte del lavoro al FSM che si occupa del coordinamento dei vari componenti e contiene la logica di evoluzione dell'elaborazione. Inizialmente avevamo pensato di realizzare il progetto interamente nel FSM ma, documentandoci, abbiamo capito che questo avrebbe compromesso le prestazioni del nostro componente e quindi abbiamo deciso di realizzare vari moduli più piccoli di supporto per evitare la creazione di LATCH indesiderati,
2. Per evitare la creazione di latch indesiderati abbiamo deciso di settare i default di tutti i segnali all'inizio del processo del FSM in modo da non poter creare dubbi su quale valore dare nei vari stati,
3. Per capire quando siamo arrivati alla fine dell'elaborazione della sequenza, abbiamo deciso di utilizzare la condizione "in_cella = i_k&'0' ", ovvero quando il contatore CELLA diventa uguale al doppio di i_k allora abbiamo terminato la sequenza e possiamo passare ad alzare o_done. Utilizzando uno shift a sx, e di conseguenza creando cella di un bit più grande rispetto i_k, abbiamo evitato problemi nel caso in cui i_k è abbastanza grande da avere overflow in CELLA,
4. Per evitare di salvare memoria inutilmente, abbiamo deciso di sovrascrivere in memoria a ogni parola elaborata, senza dover prima salvare in memoria tutti i dati aggiornati per poi, solo alla fine, aggiornare la memoria,
5. Abbiamo deciso di aggiungere uno stato di attesa di buona avvenuta della lettura dalla memoria in quanto il componente progettato ha un ritardo di propagazione e altre soluzioni, come l'aggiunta di I_Mem_Data nella sensitivity list, crea ulteriori complicazioni del codice, che volevamo evitare.

4. Risultati dei test

4.1. Risultati delle simulazioni

Supponiamo che il comportamento della memoria e i blocchi esterni ai componenti progettati funzionino in modo predefinito dalla specifica, in particolare: il segnale i_start, i_k, i_mem_addr rimangono invariati durante una singola elaborazione della sequenza e modificati solo tra diverse elaborazioni. Il modulo non prevede la segnalazione e la gestione in casi di violazione delle regole predefinite.

Per verificare il corretto funzionamento del componente, dobbiamo fare 8 casi di test:

1. Testbench base di una sequenza da elaborare

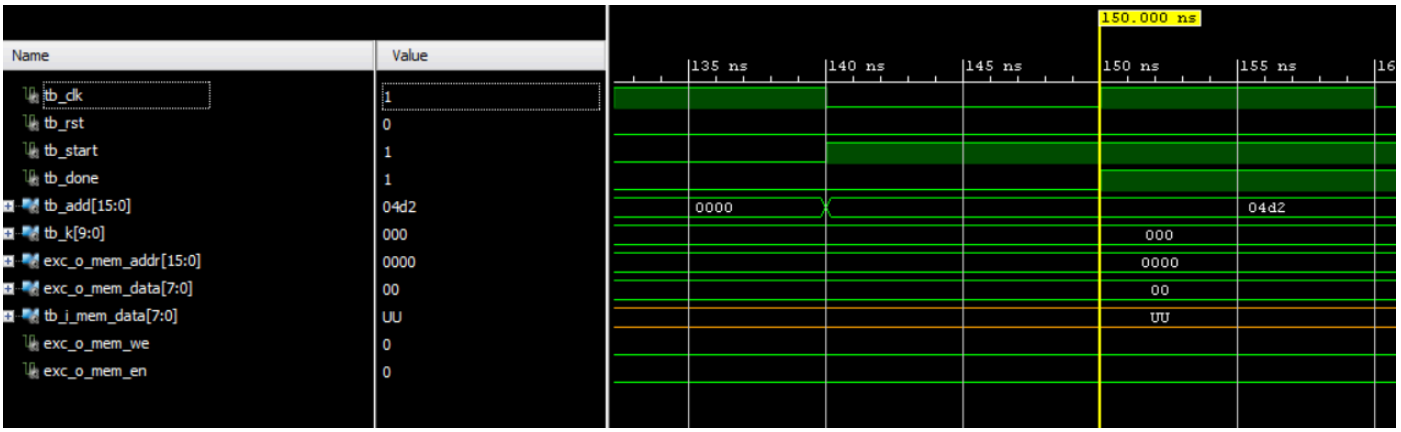
Test che ci è stato fornito dal professore come primo testbench, dove abbiamo una singola sequenza da elaborare.
Risultato atteso:

- Controllare che per ogni parola =0 venga associata la parola !=0 corretta,
- Per ogni parola deve essere associata la credibilità corretta corrispondente,
- Le parole che sono già !=0 vanno mantenute uguali,
- Al termine dell'esecuzione o_done deve essere alzato.

2. Numero di parole I_K = 0

Il test verifica la capacità del componente di gestire l'assenza di dati da elaborare (K = 0) e di attivare correttamente il segnale o_done per indicare la fine dell'elaborazione.
Risultato atteso:

- Il segnale o_done deve salire a 1 dopo 1 ciclo di clock. FSM richiede un altro ciclo di clock per tornare nello stato SInizializzazione.



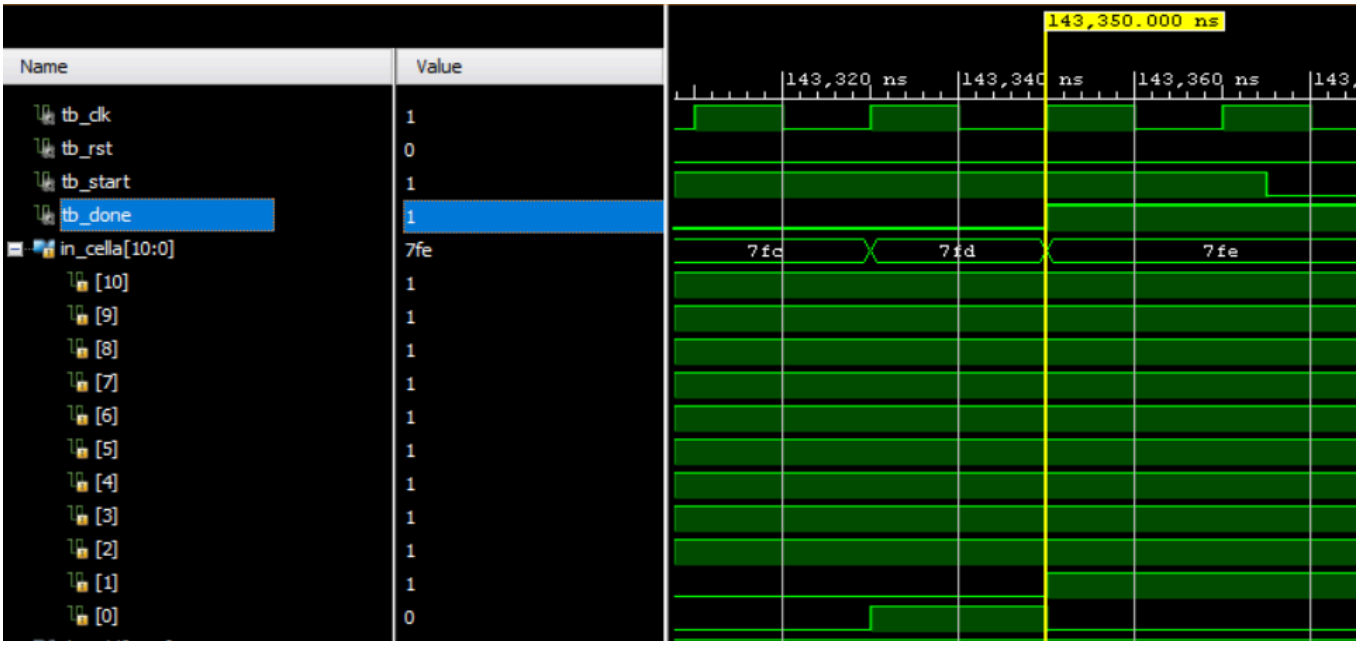
Behavioural del test 2

Dall'immagine riportata notiamo che quando inizia l'elaborazione (i_start = 1) si ha che al primo clock viene posto o_done = 1, questo conferma il fatto che avendo i_k=0 si passa direttamente alla fine dell'elaborazione, per poi andare nello stato di attesa di un nuovo i_start per una nuova eventuale elaborazione.

3. Numero di parole massimo K = 1023

Il test verifica la capacità del contatore (cella) di raggiungere tutti i valori possibili (da 0 a 1023) e di gestire correttamente il caso limite del numero massimo di parole (K = 1023).
Risultato atteso:

- Il segnale o_done deve salire a 1 dopo elaborazione di 1023 parole,
- Il contatore interno (cella) deve raggiungere il valore 2046 prima di attivare o_done



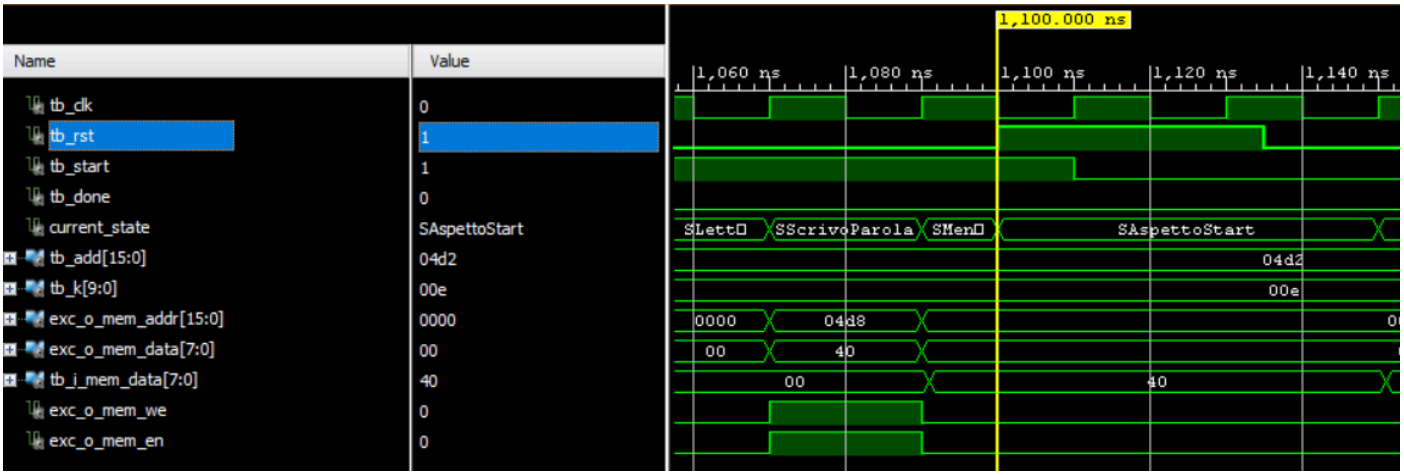
Behavioural test 3

Dall'immagine riportata notiamo che abbiamo che quando arriviamo alla fine della sequenza, in_cella vale esattamente 2046 e o_done viene alzato per indicare la fine dell'elaborazione.

4. Reset durante l'elaborazione di una sequenza

Il test verifica la capacità del componente di gestire un reset durante l'elaborazione e di reinizializzarsi correttamente in qualsiasi momento.
Risultato atteso:

- Il componente deve arrestare l'elaborazione in corso dopo il reset,
- FSM deve tornare nello stato SAspettoStart,
- Il componente deve essere in grado di avviare una nuova elaborazione dopo il reset in caso di un nuovo segnale di i_start.



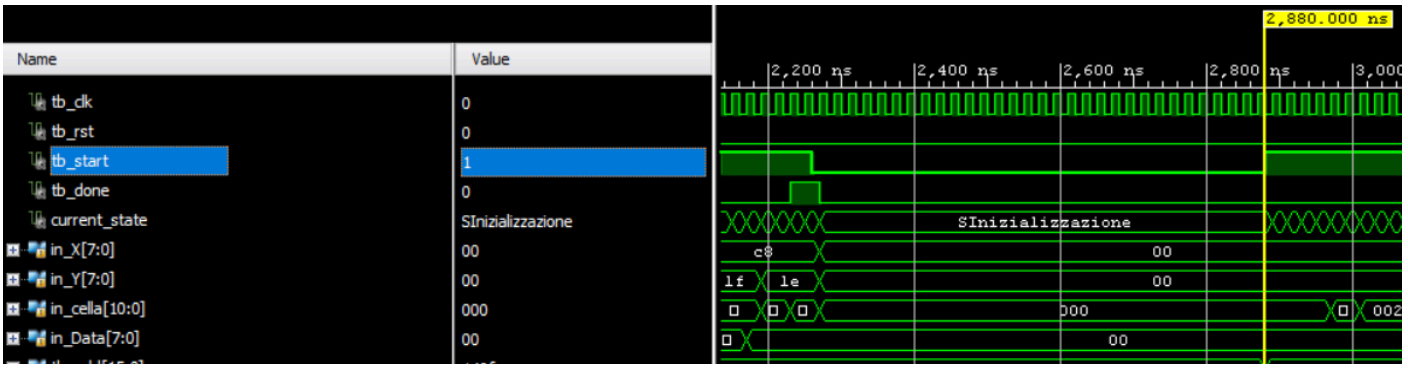
Behavioural test 4

Dall'immagine riportata notiamo che appena arriva un segnale di i_rst allora FSM passa allo stato SAspettoStart e viene atteso un nuovo segnale i_start per poter ripartire con l'elaborazione.

5. Due sequenze da elaborare in una stessa esecuzione

Valutare il comportamento del segnale o_done e la capacità della macchina a stati finiti (FSM) di gestire due cicli di elaborazione consecutivi, verificando:

- Corretto innalzamento di o_done: al termine dell'elaborazione della prima sequenza, il segnale o_done deve salire a 1 per indicare il completamento,
- Ritorno allo stato di attesa: dopo la prima sequenza, FSM deve ritornare allo stato di attesa (SAspettoStart) in modo da poter gestire ulteriori sequenze in ingresso,
- Elaborazione di sequenze successive: FSM deve essere in grado di elaborare correttamente sequenze di dati successive senza errori,
- Reinializzazione dei moduli: al termine di ogni ciclo di elaborazione, tutti i moduli devono essere reinizializzati.



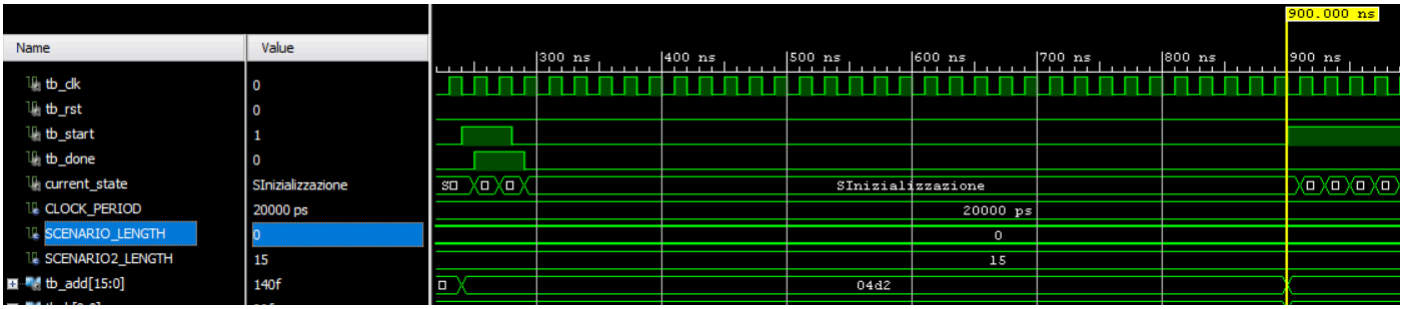
Behavioural test 5

Dall'immagine riportata notiamo che si hanno più i_start; quando il primo o_done viene posto a 1 allora i_start ha la possibilità di tornare a 0 e di permettere la reinizializzazione dei moduli a 0. Quando i_start torna a 1 allora si può ripartire con la prossima sequenza da elaborare.

6. Due sequenze da elaborare in una stessa esecuzione. Prima sequenza con K = 0

Valutare il comportamento del segnale o_done e la capacità della macchina a stati finiti (FSM) di gestire due cicli di elaborazione consecutivi. A differenza del test precedente, in questo caso ci concentriamo su due aspetti:

- Assenza di blocchi: Vogliamo verificare che il programma non si blocchi dopo aver elaborato zero parole. In altre parole, il programma non dovrebbe terminare anche se non ci sono dati da elaborare, FSM dovrebbe tornare nello stato di attesa SInizializzazione.

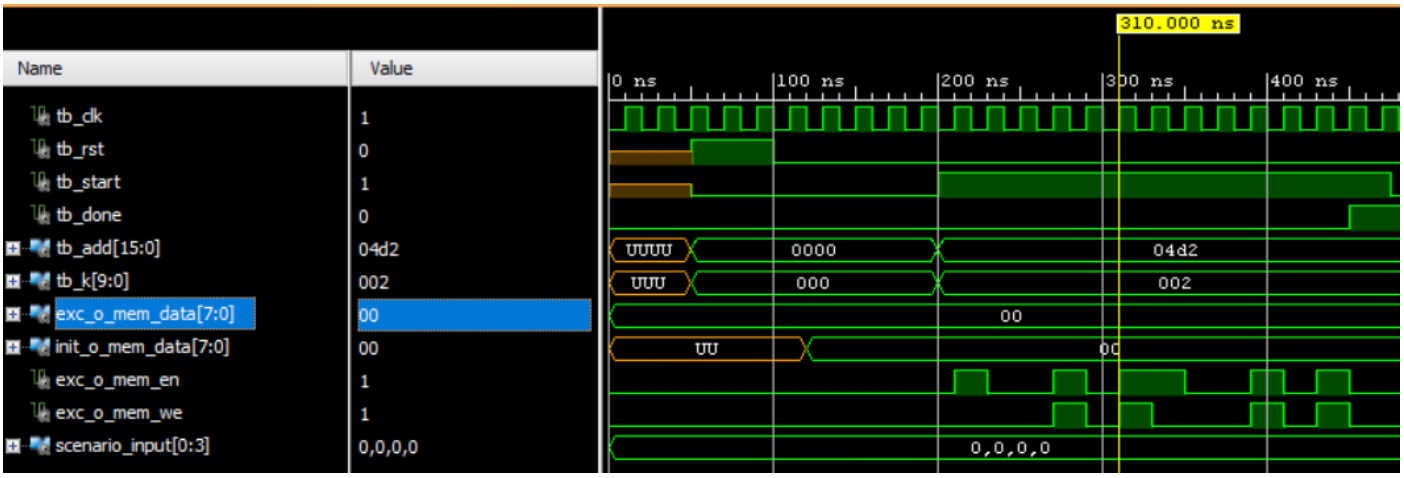


Dall'immagine riportata notiamo che all'inizio si ha i_k=0 e o_done si alza subito, correttamente, non avendo parole da elaborare e dopo diversi cicli di clock abbiamo un nuovo start che permette di partire con l'elaborazione di una nuova sequenza.

7. Prima parola con valore 0

Il test verifica il comportamento del componente quando la prima parola da elaborare ha valore 0. Lo scopo è di assicurarsi che:

- Il valore della parola viene mantenuto inalterato, ovvero rimanga 0,
- La credibilità rimanga a 0,
- Il componente non vada in errore o si blocchi durante l'elaborazione di parole con credibilità 0.



Behavioural test 7

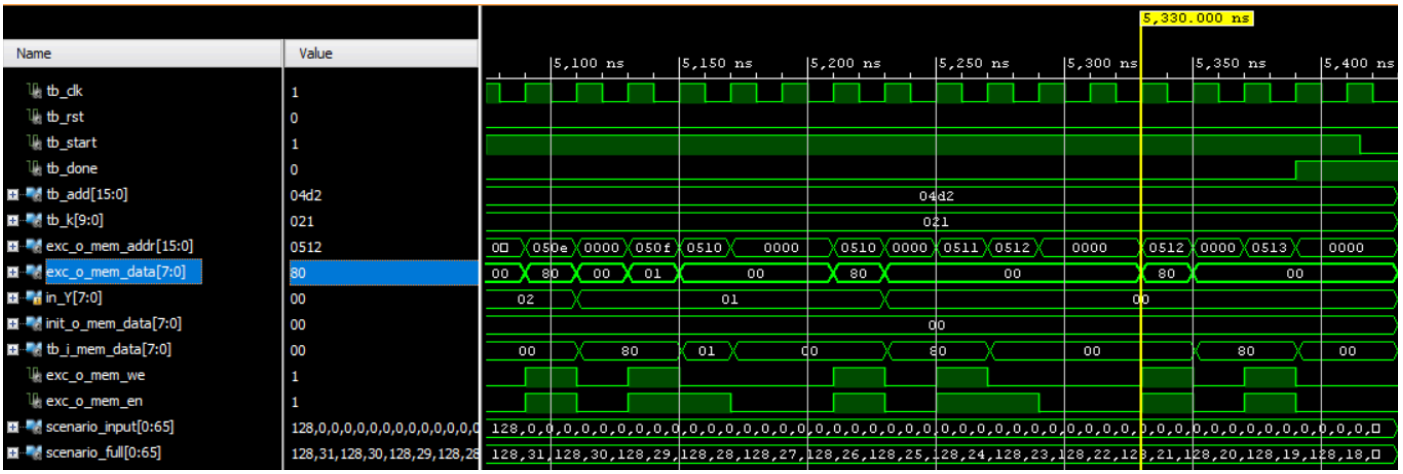
Dall'immagine riportata notiamo che quando lo scenario parte con 0 allora o_mem_data continua a mandare 0 in memoria, correttamente, sia per la parola sia per la credibilità.

8. Sequenza con prima parola !=0 e 33 parole con valore 0

Il test verifica il comportamento del modulo responsabile di tener conto della credibilità.

Risultato Atteso:

- Tutte le parole 0 vengono sostituite con l'unica parola diversa da zero,
- La credibilità di ogni parola è decrementata di uno rispetto alla credibilità dell'ultima parola, finché non arriva al valore 0,
- Quando la credibilità raggiunge il valore 0, rimane 0.



Behavioural test 8

Dall'immagine riportata notiamo che in_cella continua a decrementare e quando arriva a 0, rimane a 0. Tutte le parole vengono sostituite con l'unica diversa da 0, 80hex nel nostro caso.

4.2 Risultati della sintesi

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	89	0	41000	0.22
LUT as Logic	89	0	41000	0.22
LUT as Memory	0	0	13400	0.00
Slice Registers	45	0	82000	0.05
Register as Flip Flop	45	0	82000	0.05
Register as Latch	0	0	82000	0.00
F7 Muxes	0	0	20500	0.00
F8 Muxes	0	0	10250	0.00

report_utilization

Un elemento che si può utilizzare per analizzare le prestazioni è il numero di latch, in quanto sono elementi di memoria aggiuntivi creati da vivado quando non tutti i segnali hanno un valore definito. Come si può notare dall'immagine, abbiamo realizzato un componente che evita la creazione di latch indesiderati grazie all'utilizzo di moduli esterni alla logica del FSM, che permettono di salvare dati in maniera corretta.

required time	21.695
arrival time	-4.332
slack	17.363

report_timing

Un altro elemento che utilizziamo per analizzare le prestazioni è lo slack, differenza tra required time ed arrival time, rappresenta quanto tempo è disponibile rispetto al tempo massimo richiesto per completare l'elaborazione. Se è positivo allora si stanno rispettando i requisiti di timing. Nel nostro caso dovevamo anche ottenere uno slack <20ns, cosa che abbiamo ottenuto.

4.3 Ottimizzazioni

Per ottimizzare FSM abbiamo messo tutti i segnali utili all'interno della sensitivity list, che permettono di ridurre il numero di stati della macchina, ed evitato di mettere quelli che non servivano, come per esempio i_k e i_add. Inoltre, durante la progettazione del codice abbiamo pensato che fosse buona pratica utilizzare un modulo che si occupasse di aggiornare l'ultima parola Wi valida, uno che si occupasse della credibilità e uno che mantenesse in un modulo la parola Wi letta da memoria. Abbiamo pensato di aggiungere quest'ultimo modulo perchè abbiamo notato che dovendo utilizzare la parola Wi anche per alcuni costrutti condizionali, allora era buona pratica salvarla, invece di chiederla più volte alla memoria, processo che richiede più tempo, e per evitare di leggere memoria inutilmente abbiamo deciso di leggere ogni 2Byte, in quanto la credibilità di ogni parola non ci interessa sapere che valore abbia prima dell'elaborazione, perchè verrà sovrascritta in ogni caso.

5. Conclusioni

Il modulo VHDL progettato è in grado di interfacciarsi con una memoria per processare una sequenza di K parole (W). Le sue funzioni principali sono:

1. Comunicazione con componenti esterni:

Il modulo gestisce la comunicazione con componenti esterni attraverso segnali dedicati. I segnali includono dati di input, dati di output, segnali di controllo e segnali di clock.

2. Richiesta di accesso alla memoria:

Il modulo gestisce le richieste di accesso alla memoria per leggere e scrivere dati, specificando l'indirizzo di memoria, dati da scrivere e segnali di accesso.

3. Identificazione dei byte da elaborare:

Il modulo distingue i byte che contengono i dati da elaborare e i byte con valori non validi.

4. Elaborazione e memorizzazione delle elaborazioni precedenti:

Il modulo elabora i dati letti dalla memoria secondo le regole predefinite. L'elaborazione include operazioni logiche, aritmetiche e di confronto. Il modulo scrive i risultati dell'elaborazione nella memoria esterna e memorizza i valori delle elaborazioni precedenti per ottimizzare le prestazioni.