Progetto di Reti Logiche

Anno Accademico 2023-2024

Ingegneria informatica

Studenti

Giovanni Pachera 10727536

Orlando Francesco Parise 10735922

Docente

Gianluca Palermo

POLITECNICO
MILANO 1863

Indice

Introduzione

VHDL è un linguaggio per la progettazione di sistemi elettronici digitali e rappresenta uno strumento fondamentale per la realizzazione di circuiti integrati digitali.

Il progetto richiede di implementare un modulo HW che si interfaccia con una memoria e che, fornita una sequenza di parole di ingresso (specificata dall'indirizzo del primo elemento e dalla lunghezza), la completi sostituendo eventuali zeri con gli ultimi valori letti e associando a ciascuna parola un valore di credibilità adeguato. Questo sarà pari a 31 nel caso in cui non ci sia stata alcuna sostituzione, altrimenti verrà calcolato decrementando il valore della credibilità della parola precedente. La struttura della sequenza di ingresso è illustrata in *Figura 1*.

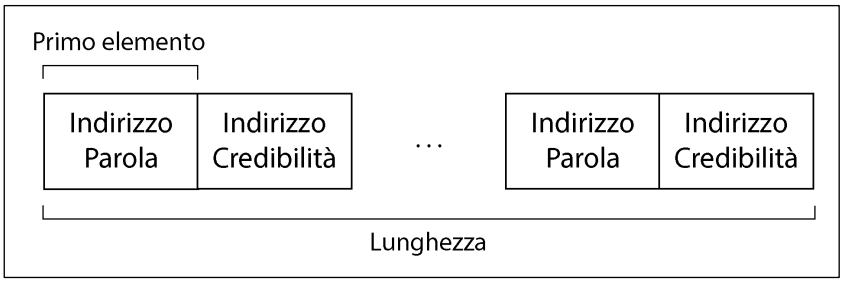

Figura 1: struttura della sequenza

Il valore della credibilità non può essere negativo, dunque nel caso in cui il valore precedente sia uguale a zero non viene effettuato il decremento. Può però succedere che la sequenza inizi con delle parole nulle. Tale eventualità richiede che non vengano fatte sostituzioni e che i valori di credibilità siano posti anch'essi a zero. Inoltre, le parole possono avere valore compreso tra 0 e 255 poiché vengono rappresentate da segnali di un Byte. In *Figura 2* è mostrato un esempio di comportamento del sistema appena descritto che, data la sequenza di ingresso fornisce la sequenza finale attesa.

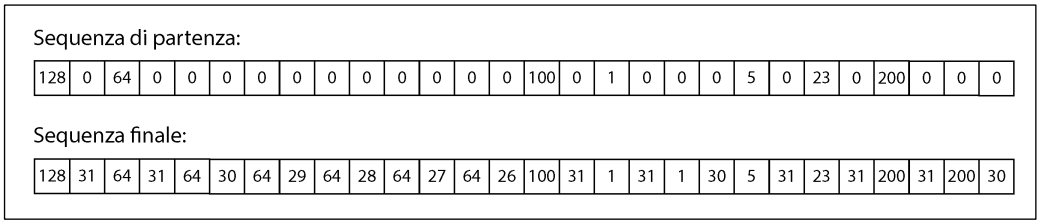

Figura 2: esempio del comportamento del sistema

Nell'esempio riportato le prime due parole sono diverse da zero, perciò, i valori di credibilità vengono posti a 31. La terza parola invece risulta nulla quindi, di conseguenza, il valore di credibilità viene calcolato decrementando il valore della credibilità della parola precedente e quindi risulta pari a 30. Lo stesso accade per le quattro parole successive finché non si riscontra un'altra parola diversa da zero che viene associata a 31 e in questo modo viene ripristinato il valore da decrementare per il calcolo della credibilità associata agli zeri successivi.

Architettura

La specifica richiede che il modulo abbia tre ingressi principali: uno di inizio, *i_start* (1 bit), uno con l'indirizzo della prima parola, *i_add* (16 bit), e uno con il numero di parole, *i_k* (10 bit). L'uscita primaria è rappresentata da un unico segnale di terminazione, *o_done* (1 bit), che definisce il termine della computazione. Un segnale di clock, *i_clk* (1 bit), scandisce sul fronte di salita l'interpretazione dei precedenti segnali, sincroni, e un ultimo segnale di reset asincrono, *i_rst* (1 bit), consente la re-inizializzazione del modulo. Questo viene sempre inizializzato al primo istante, dopodiché tra computazioni successive è sufficiente alzare e abbassare lo start. L'accesso ai dati è garantito da ulteriori segnali tra il modulo e la memoria che permettono la comunicazione, *o_mem_en* (1 bit), in lettura/scrittura, *o_mem_we* (1 bit), e consentono di inviare e ricevere dati, *o_mem_data* e *i_mem_data* (8 bit), per specifici indirizzi, o_mem_addr (16 bit). Uno schema funzionale di quanto appena descritto è illustrato in *Figura 3*.

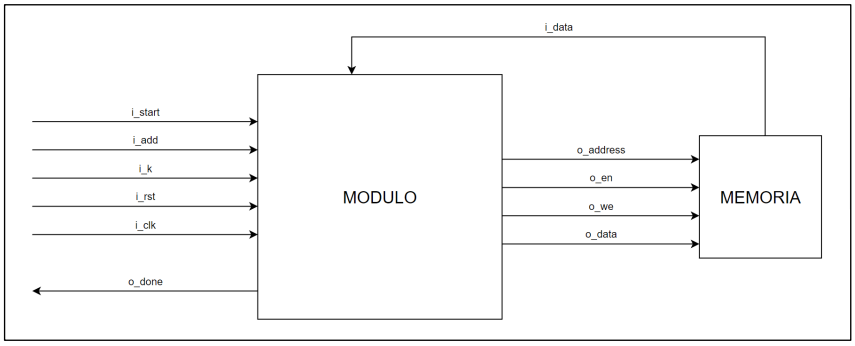

Figura 3: schema funzionale del sistema

L'implementazione della specifica ha portato alla progettazione di una macchina a stati finiti, la quale sequenzialmente completa la sequenza di ingresso. Inizialmente, il numero di stati era limitato a quattro fasi: reset del sistema, attesa del segnale di start, lettura della parola, scrittura della credibilità. In un momento successivo, la necessità di poter gestire le parole nulle e la difficoltà nel comunicare con la memoria in singoli cicli di clock ha arricchito il numero di stati. In *Figura 4* è mostrata la struttura della FSM.

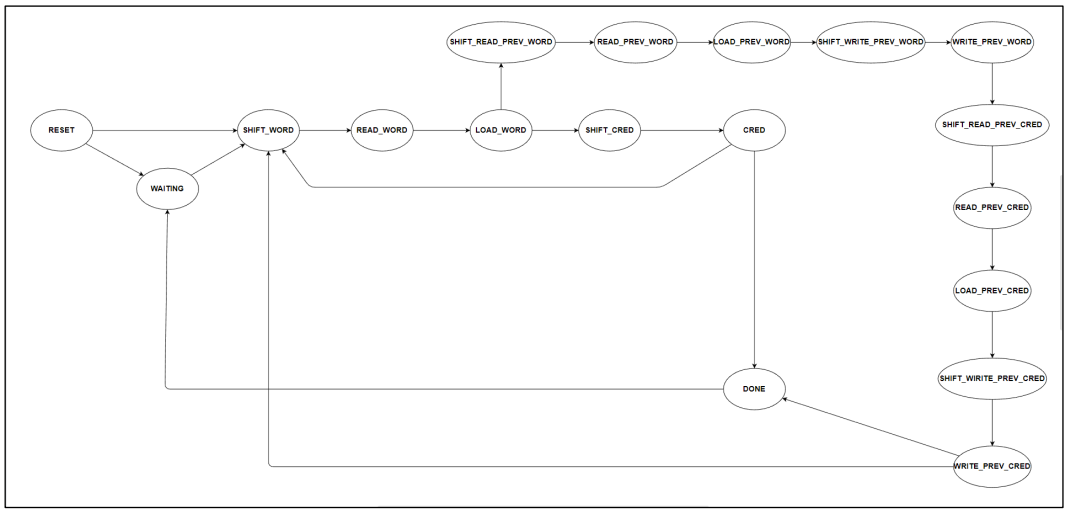

Figura 4: struttura della FSM (per semplicità, sono state omesse le frecce che collegano ogni stato a quello di reset)

I 18 stati della FSM sono brevemente descritti a seguire:

RESET
Stato iniziale nel quale tutti i segnali di uscita vengono inizializzati.

WAITING
Stato nel quale si attende che il segnale di start si alzi.

SHIFT_WORD
Stato di accesso all'indirizzo della parola.

READ_WORD
Stato di lettura della parola.

LOAD_WORD
Stato di accesso al valore della parola.

SHIFT_READ_PREV_WORD
Stato di accesso all'indirizzo della parola precedente.

READ_PREV_WORD
Stato di lettura della parola precedente

LOAD_PREV_WORD
Stato di accesso al valore della parola precedente.

SHIFT_WRITE_PREV_WORD
Stato di accesso all'indirizzo della parola nulla da sostituire.

WRITE_PREV_WORD
Stato di scrittura della parola precedente nell'indirizzo della parola nulla.

SHIFT_READ_PREV_CRED
Stato di accesso all'indirizzo della credibilità della parola precedente.

READ_PREV_CRED
Stato di lettura della credibilità della parola precedente.

LOAD_PREV_CRED
Stato di accesso al valore della credibilità della parola precedente.

SHIFT_WRITE_PREV_CRED
Stato di accesso all'indirizzo della credibilità associata alla parola sostituita.

WRITE_PREV_CRED
Stato di scrittura della credibilità associata alla parola sostituita.

SHIFT_CRED
Stato di accesso all'indirizzo della credibilità associata alla parola.

CRED
Stato di scrittura della credibilità associata alla parola.

DONE
Stato di termine della computazione.

Per garantire la sincronizzazione dei segnali di uscita sono stati creati dei segnali interni che permettono di tenere traccia dello stato attuale, del riscontro di parole diverse da zero, dell'indice di spostamento nella sequenza e delle uscite ad ogni stato corrente e prossimo. Il modulo è stato progettato in un'unica architettura costituita da due processi, uno per gestire la sincronizzazione con il clock e il reset e l'altro per determinare i valori dei segnali al ciclo di clock successivo. Tali segnali sono sempre inizializzati all'inizio di quest'ultimo processo in modo tale da prevenire la generazione di latch indesiderati. I segnali interni sono elencati in *Figura 5*.

```
signal current_state, next_state : state_type; -- stato attuale (aggiornato sul fronte di salita del clock) e stato successivo
signal non_zero, next_non_zero: std_logic; -- riscontro di una parola diversa da zero, allo stato attuale e allo stato successivo
signal index, next_index: integer; -- indice nella sequenza, allo stato attuale e allo stato successivo
signal o_mem_addr_tmp : std_logic_vector(15 downto 0); -- o_mem_addr allo stato successivo
signal o_mem_data_tmp : std_logic_vector(7 downto 0); -- o_mem_data allo stato successivo
signal o_done_tmp, o_mem_en_tmp, o_mem_we_tmp : std_logic; -- o_done, o_mem_en e o_mem_we allo stato successivo
```

Figura 5: segnali interni in VHDL

Risultati sperimentali

Utilization report

L'inizializzazione dei segnali all'inizio del secondo processo ha permesso di azzerare il numero di latch. Tale eliminazione ha consentito di realizzare con successo la sintesi del componente. In *Figura 6* sono mostrati gli elementi di memoria utilizzati.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	280	0	134600	0.21
LUT as Logic	280	0	134600	0.21
LUT as Memory	0	0	46200	0.00
Slice Registers	65	0	269200	0.02
Register as Flip Flop	65	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

Figura 6: utilization report

Numerosi testbench, ciascuno focalizzato su un aspetto importante, sono stati simulati in pre-sintesi (behavioral) e in post-sintesi (functional e timing) con esito positivo. A seguire i risultati dei più significativi.

Verifica di funzionamento

Tale testbench analizza il funzionamento dell'implementazione verificando che rispetti le indicazioni della specifica. Viene testata la sequenza di ingresso in *Figura 2*. Parte della simulazione che mostra l'inizio della computazione è mostrata in *Figura 7*.

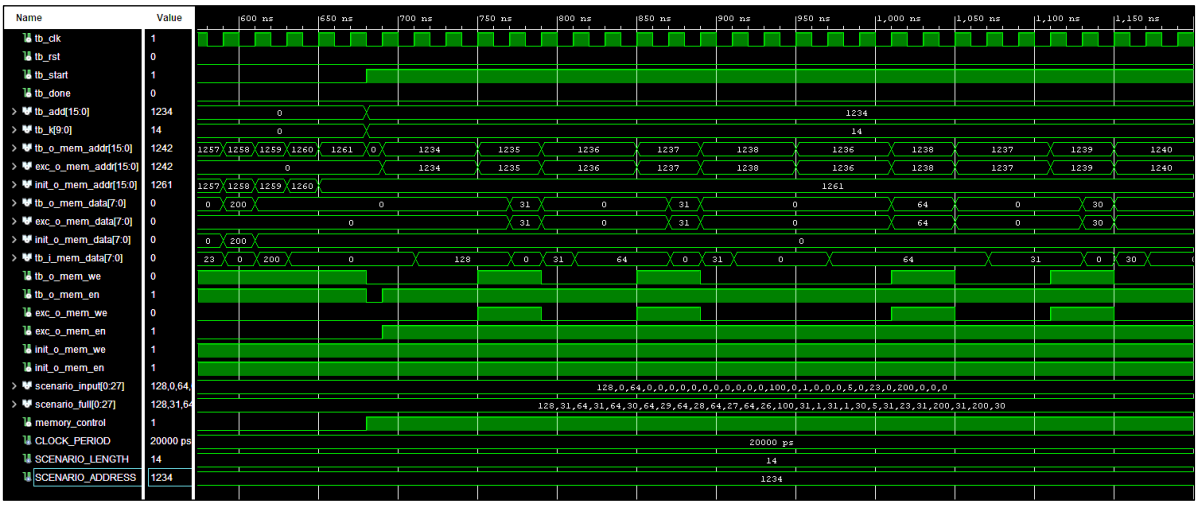

Figura 7: simulazione che verifica il funzionamento del modulo

Verifica di funzionamento con sequenza diversa

Analogamente a quello precedente, questo testbench verifica il funzionamento corretto dell'implementazione con una diversa sequenza di ingresso. In *Figura 8* viene mostrata la parte finale della computazione nella quale si alza il segnale done.

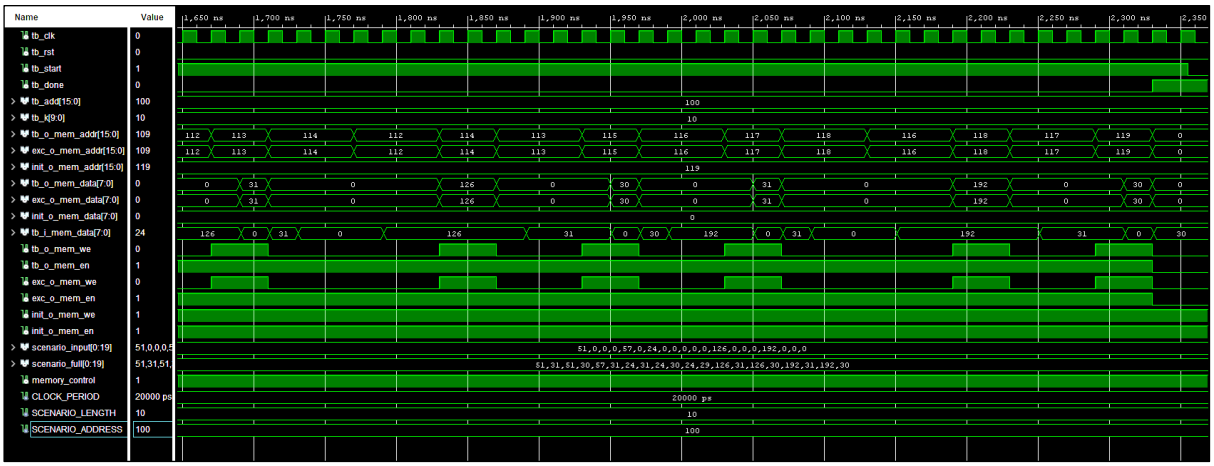

Figura 8: simulazione che verifica il funzionamento con sequenza diversa

Caso particolare: la prima parola è zero

Tale testbench verifica il funzionamento corretto dell'implementazione nel caso in cui la prima parola della sequenza di ingresso dovesse essere nulla. Il risultato della simulazione è mostrato in *Figura 9*.

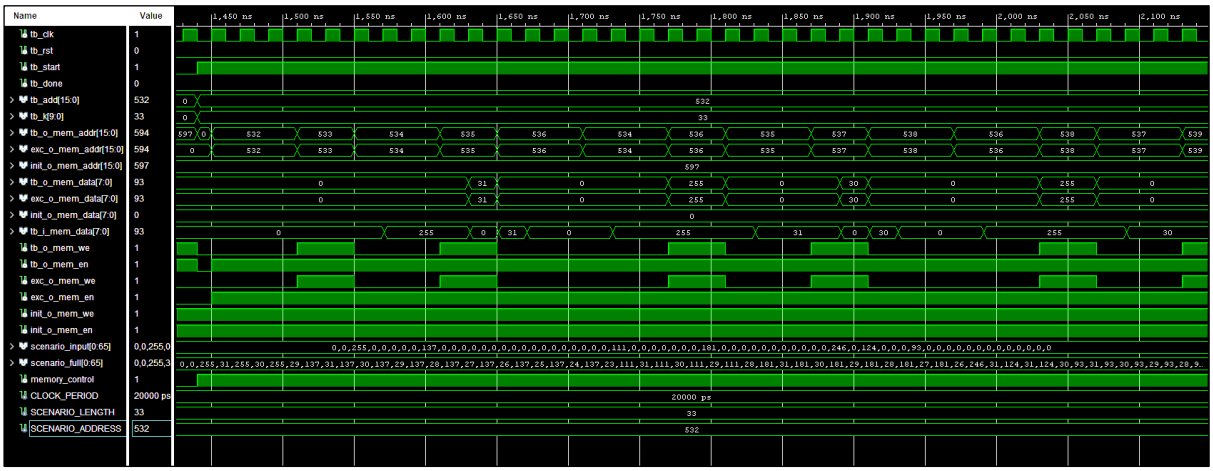

Figura 9: caso particolare in cui la sequenza inizia con zero

Caso particolare: credibilità raggiunge zero

Tale testbench verifica il funzionamento corretto dell'implementazione nel caso in cui gli zeri in sequenza dovessero portare la credibilità a zero. Essendo la credibilità, da specifica, un valore positivo non potrà essere decrementata ulteriormente. In *Figura 10* è mostrato il momento più critico della simulazione.

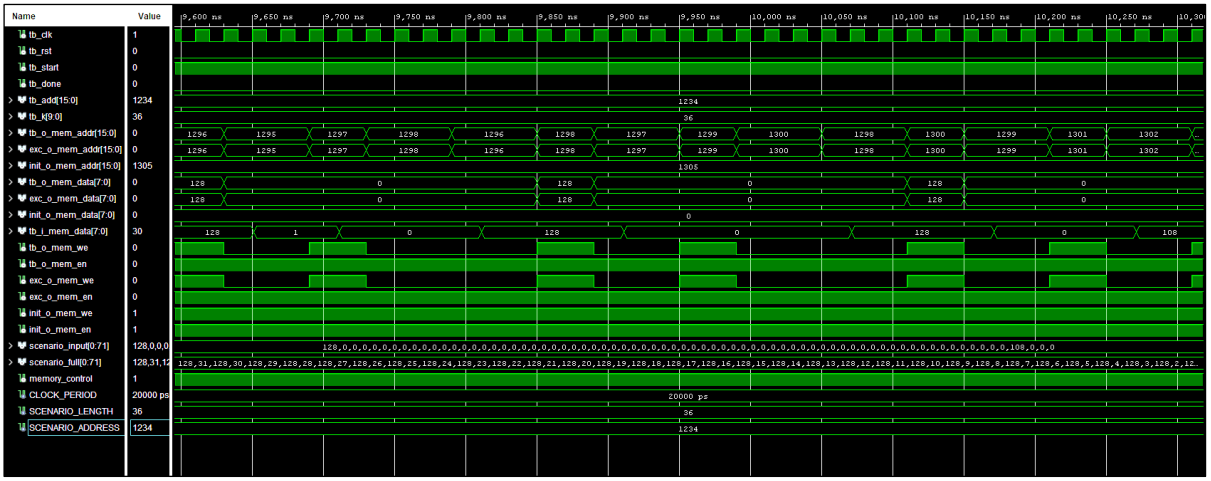

Figura 10: caso particolare in cui la credibilità raggiunge zero

Verifica di comportamento in caso di reset

Tale testbench verifica che il modulo risponda correttamente nel momento in cui si alza il segnale di reset. In *Figura 11* è possibile vedere il comportamento del modulo in caso di reset.

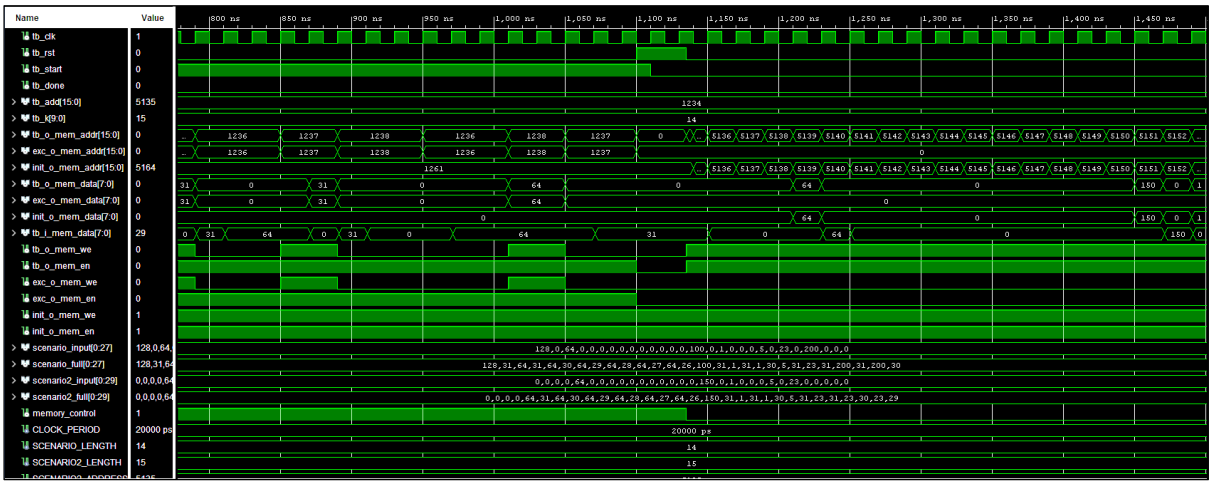

Figura 11: simulazione che verifica il comportamento in caso di reset

Verifica di comportamento in caso di sequenze multiple

Tale testbench verifica che il modulo riesca ad eseguire più computazioni in successione. Secondo specifica, è necessario che il reset del sistema venga fatto al primo istante. Successivamente tra una computazione e l'altra è sufficiente alzare e abbassare il segnale di start. In *Figura 12* è mostrato l'inizio di una successiva computazione.

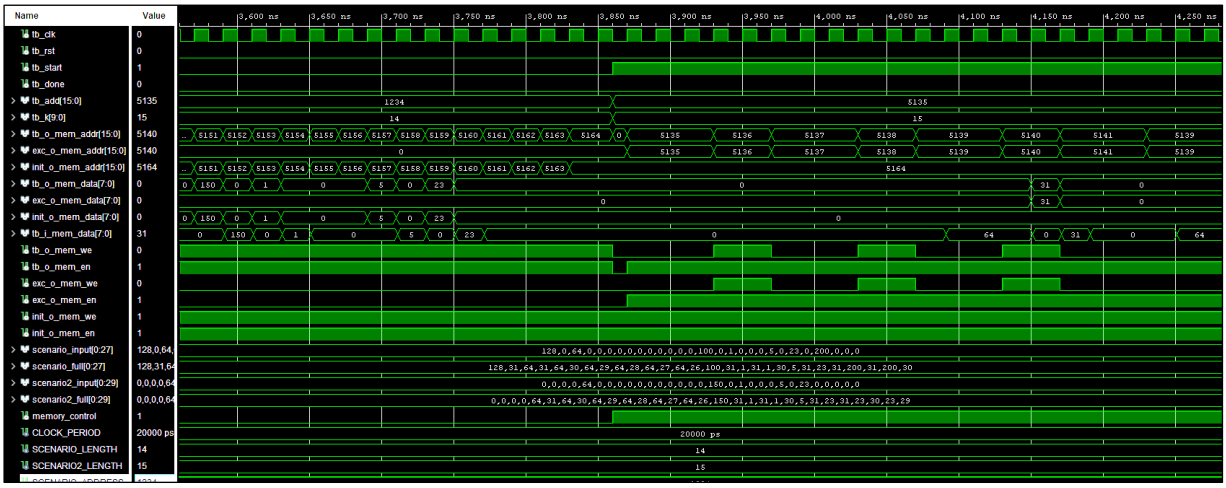

Figura 12: simulazione che verifica il comportamento in caso di sequenze multiple

Verifica di comportamento in caso di lunghezza nulla

Tale testbench verifica che la computazione termini subito nel caso in cui il numero di parole sia uguale a zero. Il risultato della simulazione è mostrato in *Figura 13*.

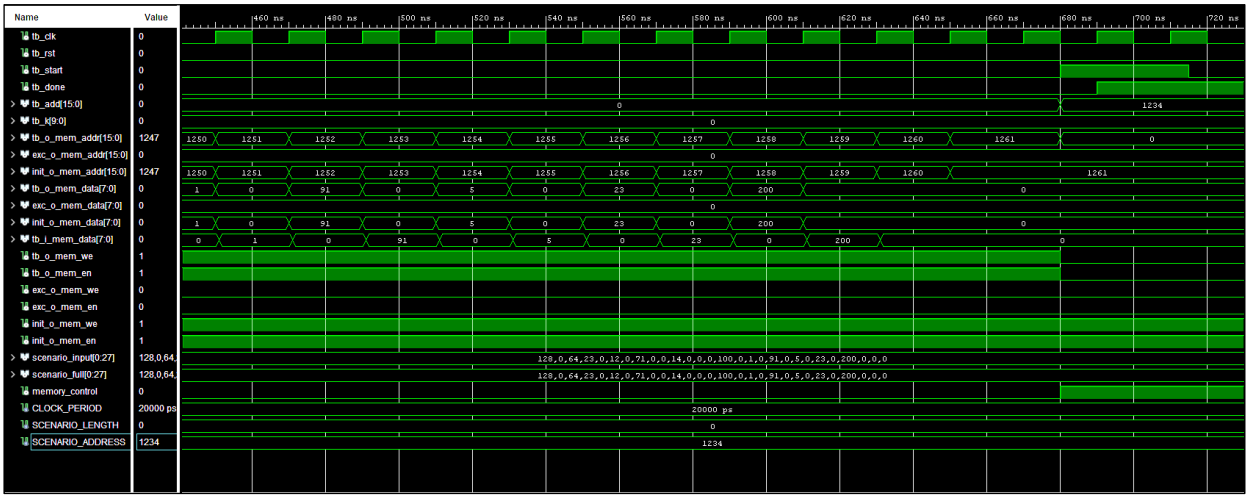

Figura 13: simulazione che verifica il comportamento in caso di lunghezza nulla

In *Figura 14* è mostrato lo schematico del modulo dopo la sintesi.

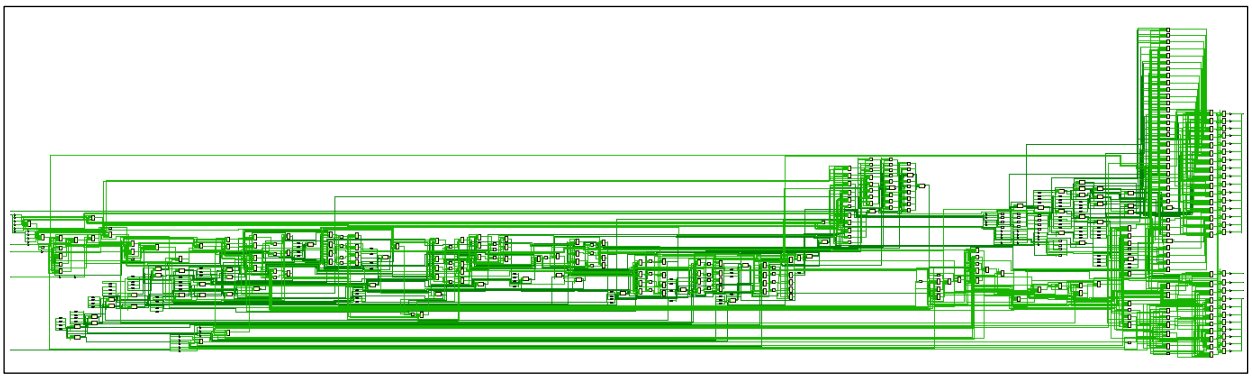

Figura 14: schematico post-sintesi

Conclusioni

Un'accurata fase di pianificazione ha permesso di determinare quali passi seguire nella realizzazione del progetto e ha consentito di affrontare le problematiche emerse senza grandi difficoltà. Il codice risulta così scalabile e mantenibile. Utilizzando una FSM è stato possibile separare le operazioni in diversi stati facilitandone la comprensione e la modifica e limitando l'utilizzo di segnali interni.
Il progetto, infine, è stato molto utile per apprendere il funzionamento di Vivado e l'utilizzo di VHDL come strumento per la sintesi di componenti HW.