POLITECNICO
MILANO 1863

PROVA FINALE
PROGETTO DI RETI LOGICHE
2023/2024

Prof. Gianluca Palermo

Salim Salici

Matricola: 893196

Codice persona: 10640001

Indice

1 INTRODUZIONE

1.1 SCOPO DEL PROGETTO

Il componente hardware da realizzare ha lo scopo di elaborare dati contenuti all'interno di una memoria RAM, completando eventuali valori mancanti e assegnando ad ogni dato un proprio valore di credibilità. Il componente hardware deve essere descritto in linguaggio VHDL.

1.2 SPECIFICHE GENERALI

A partire da un indirizzo ADD una memoria RAM contiene una sequenza di K parole da elaborare di valore compreso fra 0 e 255. Il valore 0 di una delle K parole segnala che quel valore è da considerarsi come "non specificato". Le K parole sono memorizzate ogni 2 byte.

Il byte compreso fra una parola W e l'altra rappresenta il valore di credibilità della parola memorizzata all'indirizzo immediatamente prima del byte.

Il compito del componente hardware è di sostituire i dati non specificati (valore 0) con l'ultimo valore valido incontrato. Inoltre, deve assegnare un valore di credibilità di 31 alle parole valide, che verrà ridotto di una unità per ogni parola non specificata successivamente incontrata.

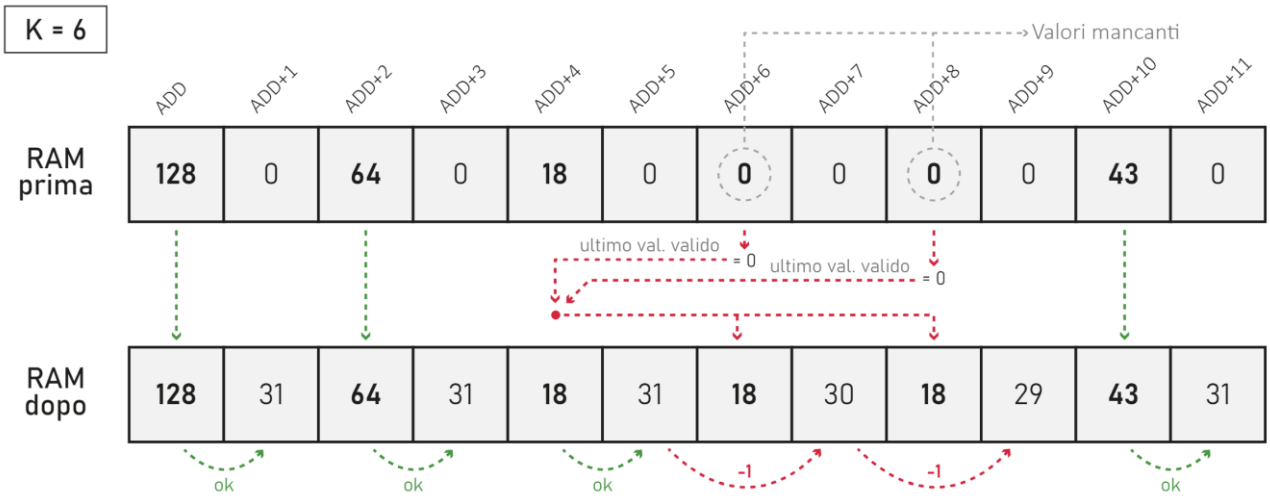


Figura 1: esempio di funzionamento desiderato del componente hardware.

Nel caso in cui le prime parole all'interno della sequenza da elaborare siano "non specificate" (ovvero abbiano un valore pari a 0), il modulo hardware deve mantenere questi valori uguali a 0. In aggiunta, per ciascuna di queste parole iniziali non specificate, il modulo dovrà assegnare un valore di credibilità pari a 0.

Esempio:

0	0	**0**	0	**0**	0	**72**	0	**243**	0	**0**	0
0	0	**0**	0	**0**	0	**72**	31	**243**	31	**243**	30

1.3 INTERFACCIA DEL COMPONENTE

L'interfaccia del componente deve essere 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  : in  std_logic_vector(7 downto 0);
            o_mem_we    : out std_logic;
            o_mem_en    : out std_logic
    );
end project_reti_logiche;
```

Qui di seguito è fornita una descrizione dei segnali che compongono l'interfaccia:

- **`i_clk`** : segnale di CLOCK;
- **`i_rst`** : segnale di RESET che serve a inizializzare o re inizializzare il componente;
- **`i_start`** : questo ingresso segnala al componente che può iniziare l'elaborazione dei dati. Rimane a 1 fino a quando il componente non porta a 1 il segnale `o_done`;
- **`i_add`** : indirizzo della memoria RAM al quale inizia la sequenza da elaborare;
- **`i_k`** : numero di parole della sequenza da elaborare (questo valore non conta i valori di credibilità di ogni parola della sequenza. Includendo anche quelli, il numero totale di byte da elaborare è `2*i_k`);
- **`o_done`** : segnale che comunica la fine dell'elaborazione da parte del componente;
- **`o_mem_addr`** : segnale di indirizzamento della memoria RAM;
- **`i_mem_data`** : segnale proveniente dalla memoria RAM e contente il dato in seguito a una richiesta di lettura;
- **`o_mem_data`**: segnale contente il dato da scrivere in memoria RAM in seguito a una richiesta di scrittura;
- **`o_mem_we`** : segnale di WRITE ENABLE da mandare alla memoria per effettuare una richiesta di scrittura;
- **`o_mem_en`** : segnale di ENABLE da mandare alla memoria per effettuare una richiesta di lettura o scrittura.

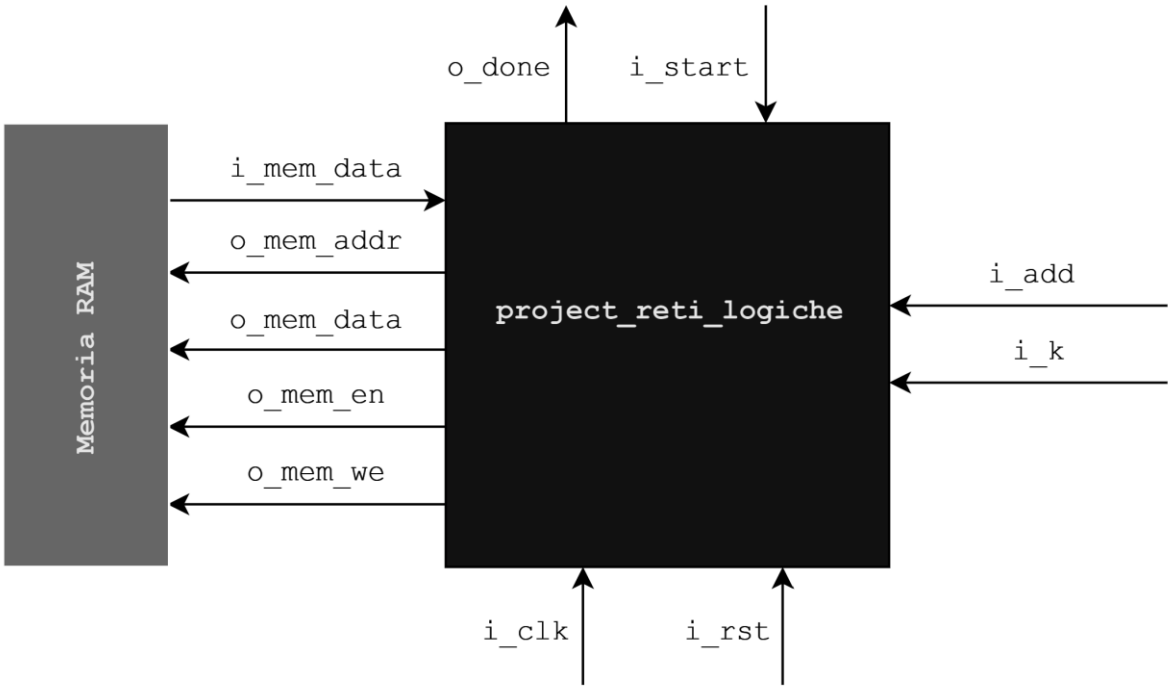


Figura 2: interfaccia del componente.

2 ARCHITETTURA E FUNZIONAMENTO

2.1 DESCRIZIONE GENERALE DELL'ARCHITETTURA

Il componente hardware realizzato è composto da tre moduli principali. Il primo modulo, chiamato **`fsm_mod`**, definisce una macchina a stati finiti che tiene traccia dei vari stati dell'elaborazione. Il secondo modulo, chiamato **`addr_mod`**, si occupa della gestione del segnale **`o_mem_addr`**, basandosi sullo stato attuale ricevuto da **`fsm_mod`**. Questo modulo è anche responsabile di portare a 1 il segnale **`o_done`** una volta raggiunto il termine dell'elaborazione. Il terzo e ultimo modulo, chiamato **`completer_mod`**, si occupa di completare la sequenza con i valori mancanti e di assegnare i relativi valori di credibilità.

Nel paragrafo 2.4 verrà fornita una spiegazione più dettagliata di questi moduli e del loro funzionamento.

2.2 CICLO DI ELABORAZIONE DEI DATI

Il componente hardware elabora la sequenza di dati in memoria RAM ciclicamente: per ciascun dato della sequenza, legge il valore nel primo ciclo di clock. Se il dato è valido, ne scrive la credibilità (31) nel ciclo successivo e procede con la lettura del dato seguente. In caso di dato non valido, nel ciclo successivo alla lettura lo sostituisce con l'ultimo valido, ne aggiorna la credibilità nel terzo ciclo, e poi passa al dato successivo. Riassumendo, il componente impiega due cicli di clock per elaborare un dato valido (lettura valore e scrittura credibilità), mentre impiega tre cicli di clock per un dato non valido (lettura valore, sovrascrittura valore e scrittura credibilità).

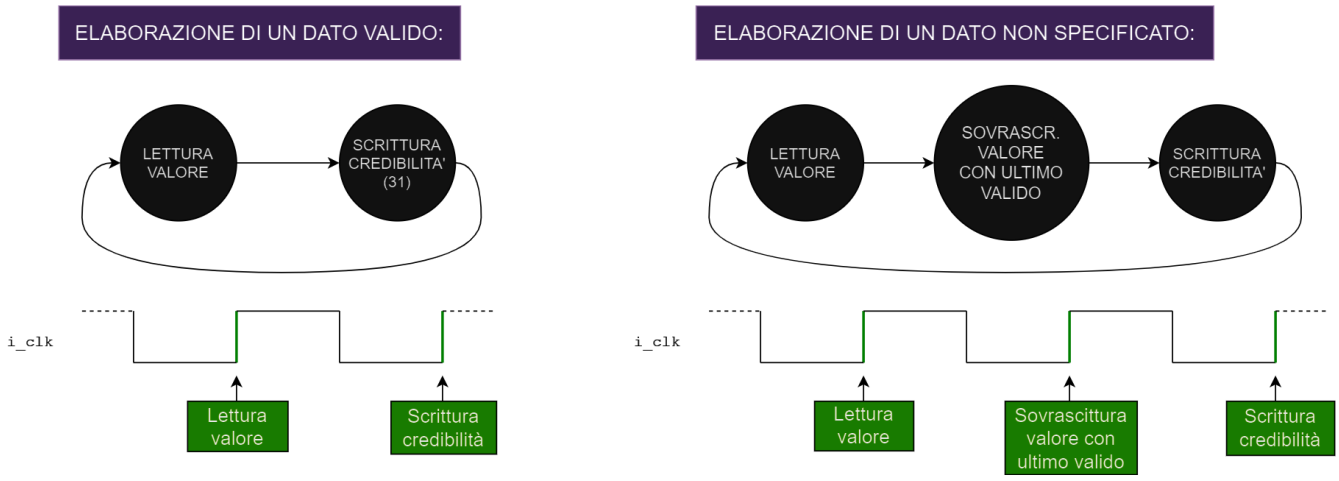


Figura 3: schema del ciclo di elaborazione dei dati nei due casi.

2.3 TEMPO DI ELABORAZIONE DEI DATI

Assumendo un clock di periodo T_{clk}, un numero K di valori da elaborare nella sequenza in ingresso e un numero Z di valori non validi all'interno della sequenza ($Z \leq K$), il tempo totale necessario all'elaborazione dei dati è

$$T_{tot} = 2 * (K - Z) * T_{clk} + 3 * Z * T_{clk} = (2K + Z) * T_{clk}$$

Dove $K - Z$ è il numero di valori validi, ognuno dei quali richiede 2 cicli di clock.
Per ogni valore "non specificato" sono necessari 3 cicli di clock.

Nel calcolo di T_{tot} viene trascurato il tempo che intercorre fra l'attivazione del segnale `i_start` e il primo fronte di salite del clock successivo.

2.4 DESCRIZIONE DEI MODULI

2.4.1 FINITE STATE MACHINE MODULE (`fsm_mod`)

Ingressi : `i_clk, i_rst, i_start, o_done, valid`
Uscite : `state, o_mem_en, o_mem_we`

Il modulo implementa una macchina a stati finiti. Oltre che a fornire agli altri moduli un segnale che codifica lo stato corrente dell'elaborazione (uscita `state`), si occupa anche di pilotare i segnali `o_mem_en` e `o_mem_we` della memoria RAM. La macchina passa da uno stato al successivo sul fronte di salita del segnale di clock (`i_clk`), mentre il reset è asincrono.

Di seguito sono riportati i vari stati della macchina e la loro descrizione:

- **RESET**: questo è lo stato iniziale in cui si trova la macchina alla sua accensione e ogni qualvolta il segnale `i_rst` viene portato a 1. La macchina passa allo stato successivo `IDLE` quando `i_rst` viene portato a `0`.
  Uscite: `state <= 000; o_mem_en <= 0; o_mem_we <= 0`

- **IDLE**: in questo stato la macchina non sta eseguendo nessuna elaborazione. La macchina rimane in attesa fino a quando non viene richiesto l'avvio di una nuova elaborazione (quando il segnale `i_start` viene portato a 1). La macchina si trova sempre in questo stato prima dell'avvio di una nuova elaborazione.
  Uscite: `state <= 001; o_mem_en <= 0; o_mem_we <= 0`

- **READ**: questo stato corrisponde alla lettura del dato dalla RAM (il modulo `addr_mod` si occuperà di impostare il corretto indirizzo sul segnale `o_mem_addr`).
  Uscite: `state <= 010; o_mem_en <= 1; o_mem_we <= 0`

- **WRITE_VAL_OR_CRED**: in questo stato viene scritto un valore nella memoria RAM. Se l'ultima parola letta nello stato `READ` è valida (diversa da 0), allora il valore scritto è la credibilità associata a quella parola. Se, invece, la parola letta è non valida, allora questa parola viene sovrascritta con l'ultima parola valida incontrata precedentemente. Il modulo `completer_mod` si occupa della scrittura del dato e dell'impostazione del segnale `valid`.
  Uscite: `state <= 011; o_mem_en <= 1; o_mem_we <= 1`

- **WRITE_CRED**: questo stato viene raggiunto solo quando l'ultima parola letta durante lo stato `READ` è non valida. Preceduto dallo stato `WRITE_VAL_OR_CRED`, che ha già sostituito il valore non valido con l'ultimo valore valido incontrato, `WRITE_CRED` si occupa di aggiornare il valore di credibilità per questa parola della sequenza.
  Uscite: `state <= 100; o_mem_en <= 1; o_mem_we <= 1`

- **DONE**: questo stato rappresenta il termine dell'elaborazione. La macchina passa da quest'ultimo stato allo stato `IDLE` quando il segnale `i_start` viene posto a 0.
  Uscite: `state <= 101; o_mem_en <= 0; o_mem_we <= 0`

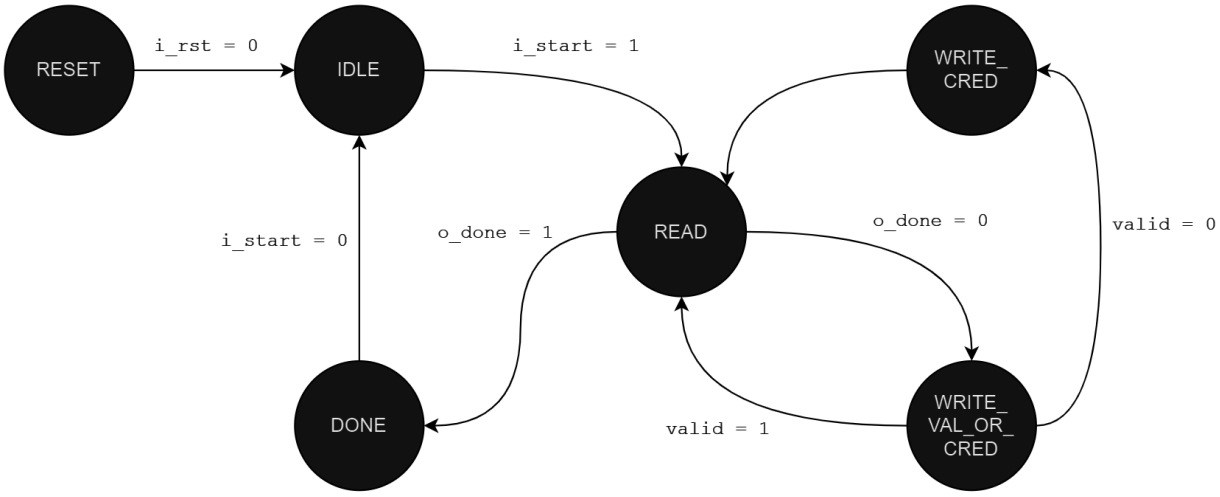


Figura 4: diagramma degli stati della FSM.

2.4.2 ADDRESS MODULE (addr_mod)

Ingressi : i_clk, i_rst, i_start, i_add, i_k, valid, state
Uscite : o_mem_addr, o_done

Questo modulo gestisce l'indirizzamento della memoria RAM, calcolando l'indirizzo di memoria o_mem_addr su cui operare durante l'elaborazione dei dati. Utilizza un registro interno chiamato offset per tenere traccia dello spostamento rispetto all'indirizzo iniziale (i_add) specificato. In base allo stato attuale ricevuto dalla macchina a stati finiti (segnale state) e alla validità del dato (segnale valid ricevuto dal prossimo modulo descritto, completer_mod), il modulo aggiusta dinamicamente l'offset per puntare alla corretta locazione di memoria per le operazioni di lettura o scrittura. Una volta completata l'elaborazione dei dati, il modulo segnala la conclusione del processo portando a 1 il segnale o_done. Al modulo viene fornito in ingresso anche il segnale i_start per potergli permettere di gestire correttamente il caso in cui i_k è uguale a 0. In questo caso, il segnale o_done viene portato a 1 immediatamente dopo l'inizio dell'elaborazione.

2.4.3 COMPLETER MODULE (completer_mod)

Ingressi : i_clk, i_rst, i_mem_data, state
Uscite : o_mem_add, valid

Queto modulo si occupa della scrittura dei dati all'interno del sistema. In base al valore della sequenza in ingresso letto nello stato READ, il modulo imposta il segnale o_mem_data per sovrascrivere eventuali valori mancanti e per assegnare le relative credibilità. Inoltre, imposta il segnale valid per comunicare agli altri due moduli la validità del valore letto durante stato READ.

Utilizza due registri interni chiamati last_value e credibility per tener traccia rispettivamente dell'ultimo valore valido letto e della credibilità associata. Il registro credibility è impostato a 31 ogniqualvolta il valore letto nello stato di READ è valido (quindi diverso da 0), mentre in caso contrario viene decrementato di 1 fino a un minimo di 0.

Questi due registri sono opportunamente inizializzati a 0 per gestire correttamente il caso in cui la sequenza inizi con valori uguali a 0.

2.4.4 SCHEMA DEL COMPONENTE

Di seguito è riportato lo schema del componente. È illustrata l'architettura interna che mette in relazione i vari moduli con i segnali di ingresso e uscita, e le interconnessioni fra i vari moduli.

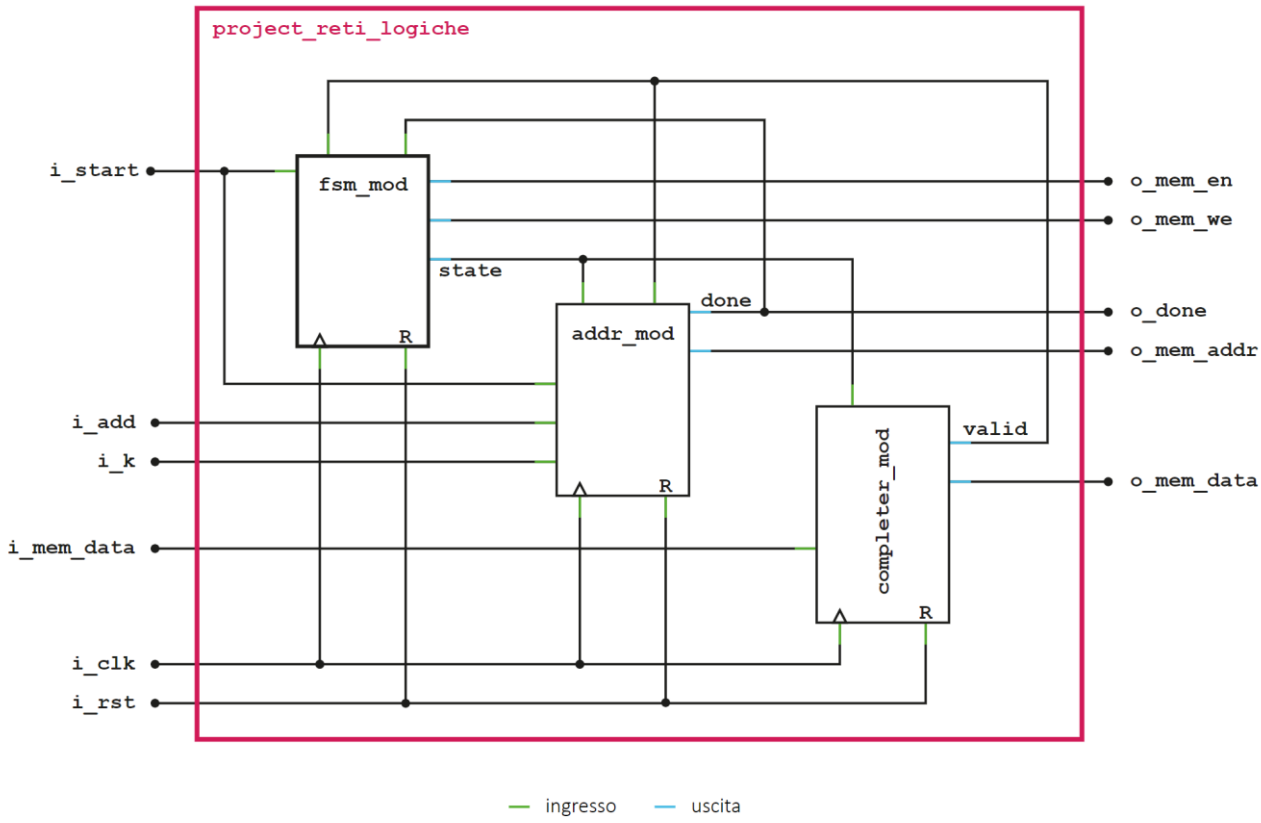


Figura 5: schema del componente.

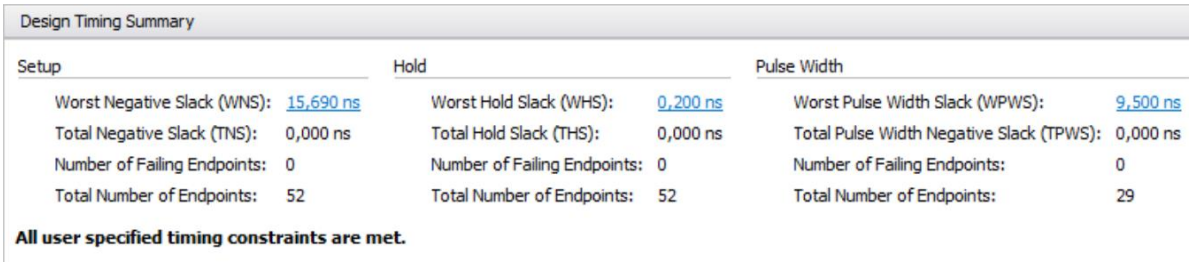
3 RISULTATI SPERIMENTALI

3.1 SINTESI

La sintesi del componente è stata eseguita tramite il software Vivado, impostando come FPGA target il dispositivo `xc7a15tcpg236-2L` della famiglia `Artix-7`.
Di seguito sono mostrate la tabella di utilizzo dei componenti della scheda e le informazioni riguardanti il timing report (impostando nei constraints un periodo di clock di 20 ns).

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	93	0	10400	0.89
LUT as Logic	93	0	10400	0.89
LUT as Memory	0	0	9600	0.00
Slice Registers	28	0	20800	0.13
Register as Flip Flop	28	0	20800	0.13
Register as Latch	0	0	20800	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

È possibile notare che vengono utilizzate 93 Look Up Tables and 28 registri come FF (tutti sincroni rispetto al clock e asincroni rispetto al segnale di reset), molti meno di quelli totali messi a disposizione dalla scheda FPGA. Non sono stati utilizzati registri come latch.



Il Worst Negative Slack (WNS) è pari a 15,690 ns, valore positivo che soddisfa il constraint di temporizzazione impostato per il periodo di clock di 20 ns. Il WNS rappresenta la differenza tra il tempo disponibile affinché un segnale si propaghi fino alla sua destinazione nel circuito e il ritardo massimo effettivamente riscontrato dal segnale che si trova lungo il percorso critico del componente.

3.2 SIMULAZIONI

Il componente è stato testato con vari test bench al fine di assicurarne il corretto funzionamento sotto diverse ipotesi. Oltre agli aspetti specifici di ogni test bench, in tutti i test sono sempre state verificate le seguenti condizioni:

- il segnale `o_done` deve essere sempre 0 prima del segnale di start;
- la memoria non deve essere scritta dopo che il segnale `o_done` viene portato a 1;
- il segnale `o_done` deve rimanere a 1 fino a quando il segnale di start non viene riportato a 0;

Di seguito sono riportati i test effettuati e la loro descrizione.

3.2.1 project_tb.vhd

Lo scopo di questo test bench è quello di verificare il funzionamento base del componente. Viene testata la seguente sequenza di 14 valori da elaborare (28 data points in tutto contando anche le credibilità), alcuni dei quali non specificati. Nella prima riga è riportata la sequenza in ingresso, nella seconda la sequenza elaborata dal componente:

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

3.2.2 project_long_zero_seq_tb.vhd

Lo scopo di questo test bench è quello di verificare che nel caso in cui ci fossero più di 32 valori “non specificati” consecutivi, il valore di credibilità dei valori “non specificati” successivi al trentunesimo siano tutti 0.

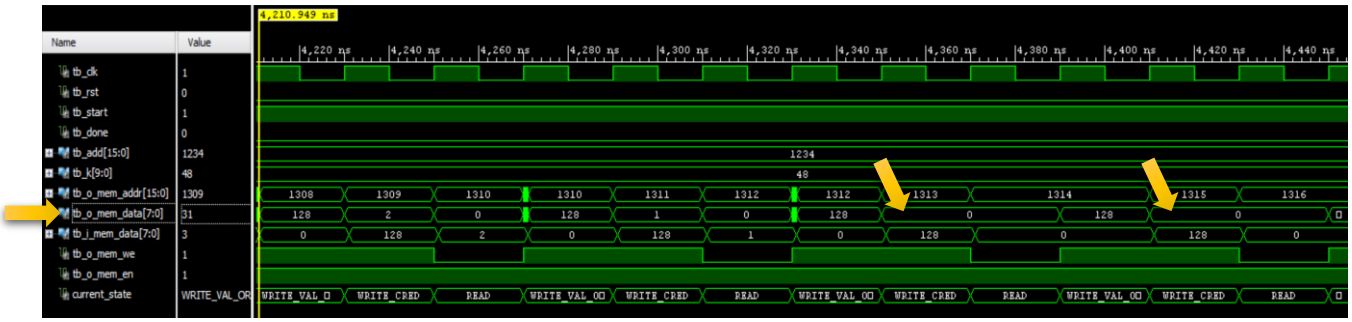


Figura 6: forme d'onda del TB nel quale si vede che una volta azzerata la credibilità, questa rimane a 0 per i successivi valori non specificati.

3.2.3 project_multiple_tb.vhd

Lo scopo di questo test bench è quello di verificare che il componente funzioni correttamente nel caso in cui venissero richieste più elaborazioni. Nello specifico sono state richieste due elaborazioni: una di 14 valori a partire dall'indirizzo di memoria RAM 1234, una di 6 valori a partire dall'indirizzo 5678. Per portare al limite il componente, il segnale di start della seconda elaborazione viene portato a 1 dopo 0.5 ns dall'abbassamento del segnale `o_done` da parte del componente.

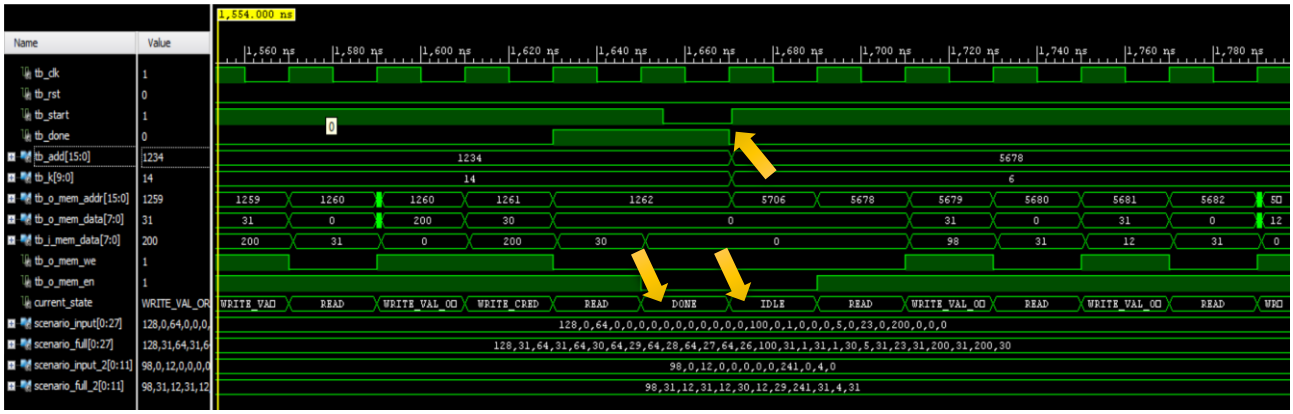


Figura 7: nella figura è possibile vedere che i segnali tb_add (i_add) e tb_k (`i_k`) sono aggiornati non appena il segnale `o_done` viene riportato a 0 dal componente. Subito dopo viene portato a 1 il segnale `i_start`. Il componente resta nello stato di IDLE per un solo ciclo di clock.

3.2.4 project_reset_tb.vhd

Lo scopo di questo test bench è quello di verificare che il componente funzioni correttamente dopo aver ricevuto il segnare di reset nel mezzo di una elaborazione. Il segnale di start è mantenuto sempre a 1, mentre il segnale i_add viene aggiornato per far ripartire una nuova elaborazione in una porzione diversa di RAM, inizializzata adeguatamente. Il segnale di reset è attivo per 1 ns.

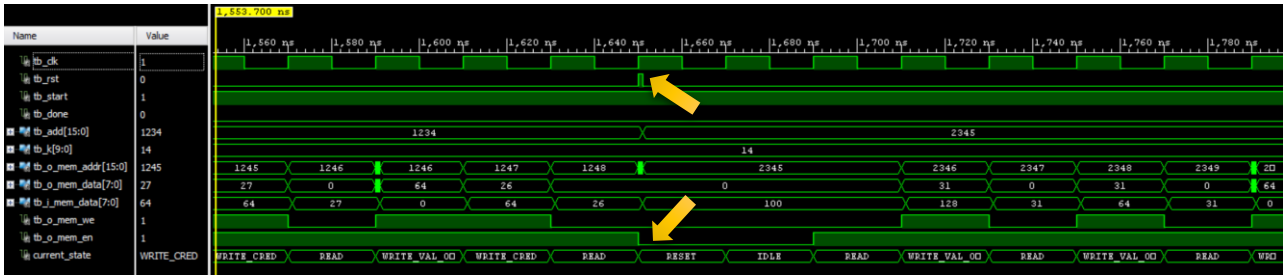


Figura 8: forme d'onda del TB. È possibile notare che dopo aver ricevuto il segnale di reset, il componente passa un ciclo di clock nello stato RESET e un ciclo di clock nello stato IDLE prima di partire con la nuova elaborazione.

3.2.5 project_starting_zeros_tb.vhd

Lo scopo di questo test bench è quello di verificare che il componente gestisca correttamente un'elaborazione in cui i valori iniziali della sequenza siano zero. Il componente deve lasciare questi valori a zero e impostare le relative credibilità a 0.

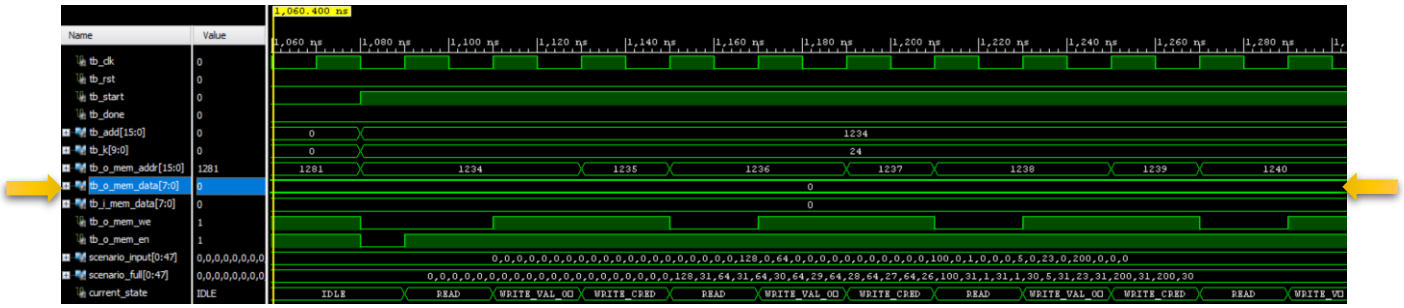


Figura 9: nella figura è possibile vedere che il componente imposta il segnale `o_mem_add` a 0 per i primi cicli di clock, scrivendo così valori corretti im memoria.

3.2.6 project_zero_data_tb.vhd

Lo scopo di questo test è verificare che il componente lasci la memoria RAM inalterata nel caso in cui il segnale `i_k` ricevuto fosse uguale a 0 (nessun valore da elaborare).

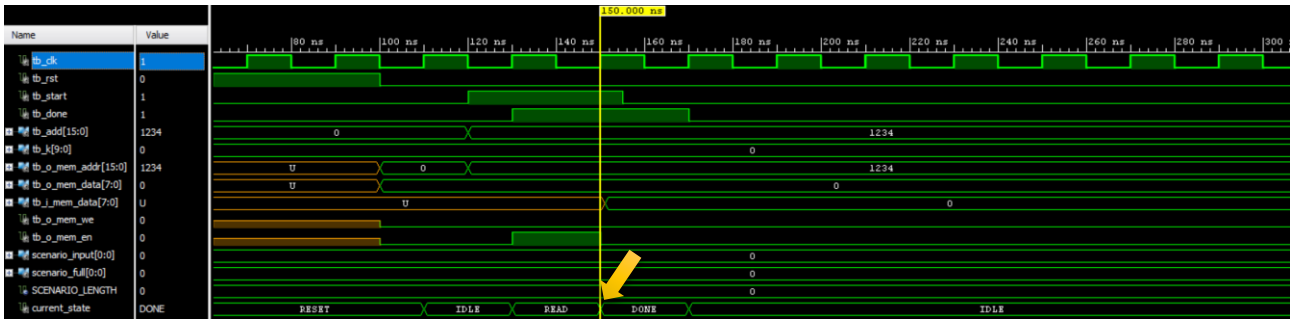


Figura 10: è possibile notare che il componente passo allo state DONE subito dopo il primo stato READ.

3.2.7 project_max_k_tb.vhd

Lo scopo di questo test è verificare che il componente funzioni correttamente nel caso in cui riceva in ingresso il valore massimo per il segnale `i_k`. Essendo `i_k` un segnale di 10 bit, il componente dovrebbe elaborare 1023 valori (2046 byte in tutto, includendo anche le credibilità).
La sequenza è stata generata tramite uno script python, avendo cura di inserire alcune catene di più di 32 zeri.

3.2.8 project_loop_around_tb.vhd

Le specifiche del progetto non menzionano come gestire il caso in cui i dati da elaborare sforino lo spazio di indirizzamento della memoria RAM. Il componente sviluppato adotta il seguente comportamento: una volta raggiunto l'ultimo indirizzo della RAM (65535) l'elaborazione prosegue ripartendo dall'inizio della memoria RAM (indirizzo 0).

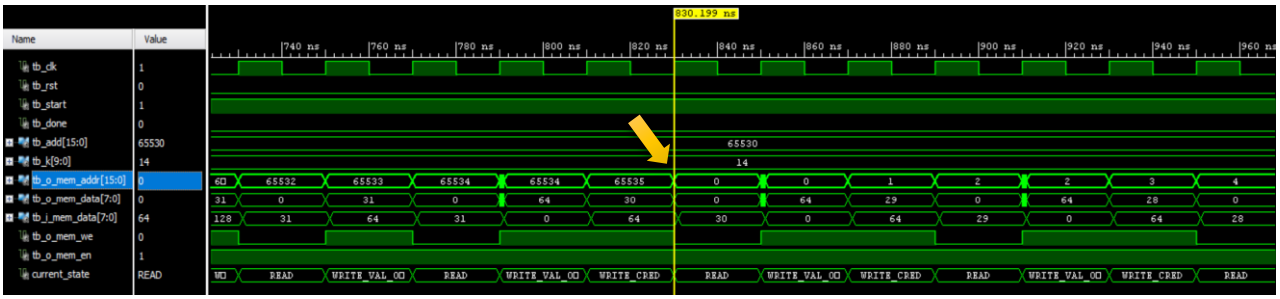


Figura 11: nella figura è possibile osservare come il segnale `o_mem_addr` passi da 65535 a 0.

3.2.9 project_long_start_after_done_tb.vhd

Lo scopo di questo test è verificare che il componente non abbassi il segnale `o_done` prima del dovuto, anche quando il segnale `i_start` è tenuto alto per molto più tempo del necessario.

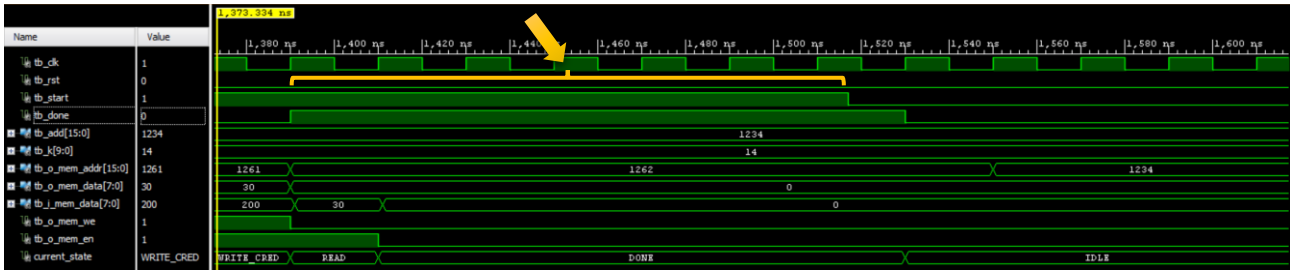


Figura 12: è possibile notare che il segnale i_start resta alto per vari cicli di clock oltre il necessario.

3.3 RISULTATI DEI TEST

Tutti i test bench effettuati sono stati superati con successo dal componente per ognuna delle seguenti tipologie di simulazione: **Behavioural** Simulation, **Post-Synthesis Functional** Simulation, **Post-Synthesis Timing** Simulation.

4 CONCLUSIONI

4.1 EVOLUZIONE DELLO SVILUPPO DEL COMPONENTE

Per soddisfare le specifiche richieste, in principio è stato sviluppato un componente composto da un singolo modulo monolitico che implementava una macchina a stati finita. Essa doveva gestire tutti gli ingressi e le uscite. In seguito, la logica del componente è stata divisa e distribuita nei tre moduli descritti nel capitolo 2. Questa scelta ha permesso un approccio più modulare e semplice rispetto a quello originale, senza però dover sacrificare performance in termini di velocità. Rispetto ad una versione precedente, inoltre, il numero di stati della FSM è diminuito, così come è diminuito il numero di cicli di clock necessari a elaborare un dato valido (in precedenza, alla ricezione di un dato valido il componente sprecava un ciclo di clock per riscriverlo nella memoria RAM tale e quale, invece di scriverne direttamente la credibilità associata).

4.2 RIFLESSIONI FINALI

Questo progetto mi ha permesso di migliorare le competenze di progettazione di componenti hardware descritti in VHDL. In particolare, ho potuto approfondire la modellazione di macchine a stati finiti, l'importanza di un'architettura modulare, le tecniche di testing e debug, e le parti principali del flusso di sviluppo, dalla codifica alla sintesi su FPGA. Nel complesso, è stata un'esperienza formativa che mi ha consentito di applicare e consolidare le conoscenze teoriche acquisite nel corso.