Prova Finale di Reti Logiche

Federico Grandi [10802488]

Settembre 2024

Indice

1 Introduzione **1**
1.1 Struttura del componente 1
1.2 Comportamento del componente 2

2 Architettura **4**
2.1 Moduli 4
2.2 Processi 4

3 Dati sperimentali **6**
3.1 Sintesi 6
3.2 Simulazione 7
3.2.1 Sequenza di esempio 7
3.2.2 Sequenza di lunghezza 0 7
3.2.3 Elaborazioni consecutive 7
3.2.4 Gestione del segnale di reset 7

4 Conclusioni **8**

1 Introduzione

La consegna richiede di progettare un componente hardware che vada ad eseguire delle operazioni sulla memoria ad esso collegata, basandosi sui segnali che riceve in entrata e sul contenuto della memoria stessa.

1.1 Struttura del componente

Il componente deve possedere le porte per i seguenti segnali:

- 3 ingressi primari: `START` (1 bit), `ADD` (16 bit), `K` (10 bit).
- 1 uscita primaria: `DONE` (1 bit).
- 1 segnale di `CLOCK`, unico per tutto il sistema, con periodo di almeno 20 ns.

- 1 segnale di `RESET`, unico per tutto il sistema.
- I segnali per comunicare con la memoria: `O_MEM_ADDR` (16 bit), `I_MEM_DATA` (8 bit), `O_MEM_DATA` (8 bit), `O_MEM_WE` (1 bit), `O_MEM_EN`(1 bit).

1.2 Comportamento del componente

Il componente deve manipolare i dati contenuti in memoria negli indirizzi da `ADD` a $\texttt{ADD} + \texttt{K} * 2 - 1$ (inclusi), secondo i seguenti criteri:

1. Tutti i dati in memoria nelle posizioni $\texttt{ADD} + k * 2, k \in [0, \texttt{K})$ sono considerati valori di una sequenza, ad eccezione degli 0, che vengono trattati come non-valori.
2. Per ogni valore salvato in memoria in posizione $\texttt{ADD} + i$, se il dato in posizione $\texttt{ADD} + i + 1$ non è 0, allora esso va sostituito con un valore di "credibilità" 31.
3. Per ogni valore salvato in memoria in posizione $\texttt{ADD} + i$, se il dato in posizione $\texttt{ADD} + i + 1$ è 0, allora esso va sostituito con un valore di "credibilità" pari al valore di credibilità precedente diminuito di 1, se esiste ed è maggiore di 0, altrimenti 0.
4. Ogni non valore nella sequenza va sostituito con il valore precedente, se esso esiste.

L'uscita `DONE` dev'essere impostata ad 1 solamente quando il componente ha terminato tutte le elaborazioni e scritture in memoria.

`RESET` è l'unico segnale asincrono, e quando viene portato a 1 il componente dev'essere reimpostato allo stato originale. Tutti gli altri segnali sono da considerarsi sincroni, da interpretare sul fronte di salita di `CLOCK`.

L'elaborazione deve iniziare quando `START` viene portato a 1, momento dopo il quale `ADD` e `K` possono essere considerati costanti. `START` rimarrà a 1 finché il componente non porrà `DONE` a 1.

Riassumendo, sono stati presi in considerazione i seguenti elementi:

- La credibilità deve rimanere a 0 finché non si incontra il primo valore.
- È necessario andare a scrivere in memoria solamente se il dato presente non è 0.
- La credibilità ha 0 come valore minimo.
- Il componente deve poter processare più sequenze tramite un corretto utilizzo del segnale `START`.

Esempio di sequenza Di seguito (figura 1) viene riportata la sequenza di dati nella porzione di memoria interessata prima e dopo l'elaborazione da parte del componente. I dati sono scritti in valori decimali; per brevità, ho riportato come [...] una serie di "0 0" nella sequenza di dati in ingresso.

Nella figura seguente viene evidenziato, tramite colorazione:

- In verde, il modo in cui i valori vengono riportati.
- In blu, i punti dove il valore di credibilità viene diminuito.
- In rosso, un punto dove il valore di credibilità non viene mostrato anche se viene diminuito, perché il dato in entrata non è 0.
- In giallo, i punti in cui il valore di credibilità viene resettato a 31.

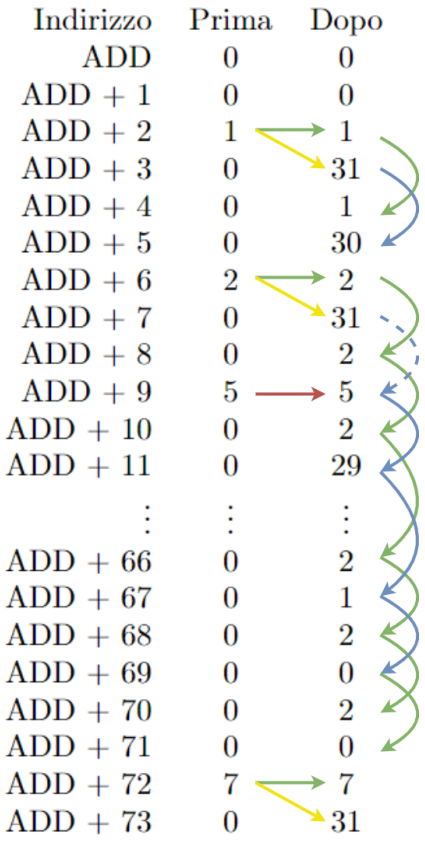

Figura 1: Sequenza di esempio

2 Architettura

2.1 Moduli

Per quanto riguarda l'architettura si è preferito optare per una soluzione mono-componente multi-processo: non essendo le operazioni da eseguire particolarmente articolate, si è scelto di mantenere una minore complessità e quindi di evitare di aggiungere ulteriori moduli.
Lo schema dei moduli che ne risulta è riportato di seguito nella figura 2. Nello schema è anche riportato il modulo di memoria, con cui vengono mostrati i collegamenti, che è però descritto nei testbench e non fa quindi parte del sorgente VHDL.

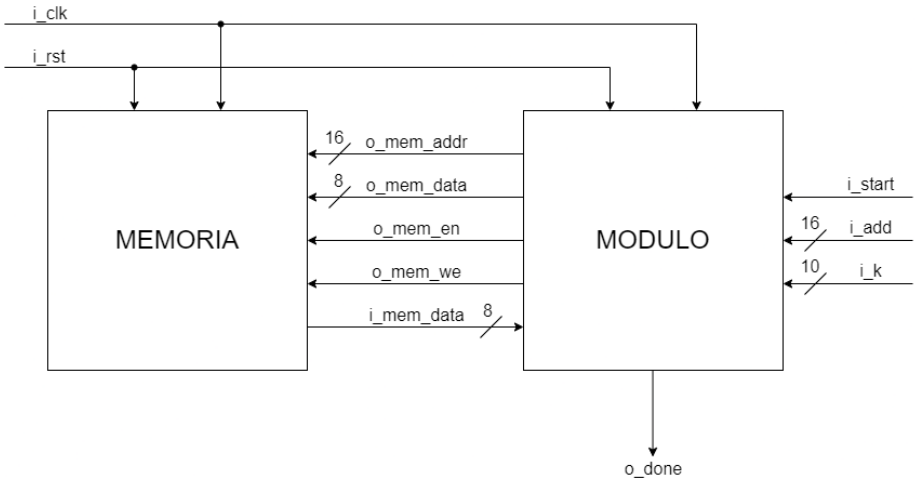

Figura 2: Moduli

2.2 Processi

All'interno dell'architettura del componente vengono descritti 6 processi in tutto:

- 4 per gestire i registri:
  1. Stato della Macchina a Stati Finiti.
  2. Contatore per tener traccia di quanti elementi della sequenza sono stati processati.
  3. Valore di credibilità C.
  4. Ultimo valore letto.
- 2 per rappresentare le funzioni della MSF:

5. Funzione di transizione δ, che si occupa di determinare lo stato successivo in base allo stato corrente, al numero di valori processati, e alle entrate del componente
6. Funzione di uscita λ, che si occupa di determinare i valori da assegnare alle uscite e ai registri, basandosi sui valori dei registri e delle entrate del componente

Sono stati inoltre dichiarati 8 segnali interni, 2 per ciascun registro.

I 6 stati della MSF sono rappresentati nel diagramma in figura 3. Si possono notare gli autoanelli sugli stati `INIT` e `DONE`, che bloccano la macchina in base al valore di `i_start`, e la biforcazione del flusso in `WRITE_CRED`, in base al fatto che tutti i valori della sequenza siano già stati letti o meno.

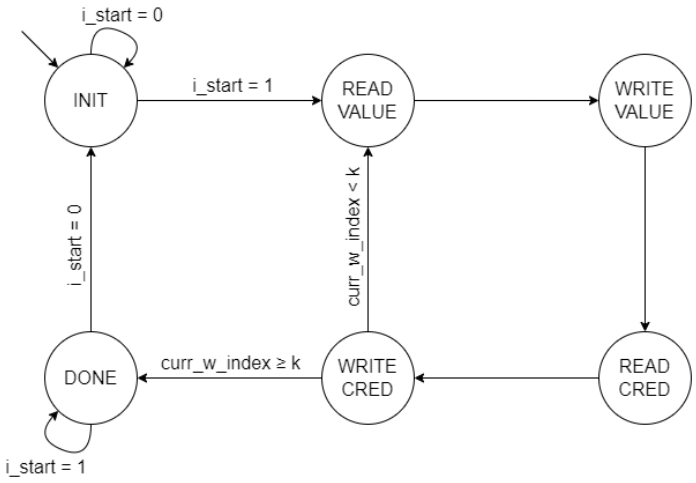

Figura 3: Diagramma degli stati della MSF

Negli stati vengono effettuate le seguenti operazioni:

- `INIT`: reimposta i valori dei registri `cred`, `w_index`, e `last_w` a 0.
- `READ_VALUE`: imposta l'indirizzo del valore della sequenza da leggere (partendo da `i_add`) e abilita la lettura della memoria.
- `WRITE_VALUE`: se il dato letto non è 0, lo salva in `last_w` e imposta `cred` a 31; in caso contrario, diminuisce `cred` e scrive in memoria l'ultimo dato letto.
- `READ_CRED`: imposta l'indirizzo successivo a quello del dato appena letto, predisponendone la lettura.
- `WRITE_CRED`: incrementa `w_index` e, se il dato presente in memoria è 0, allora lo sovrascrive con il valore di credibilità.

- `DONE`: imposta `o_done` a 1.

Gli stati `WRITE_VALUE` e `WRITE_CRED` vengono sempre percorsi, anche quando non è necessario scrivere in memoria, perché devono svolgere anche dei compiti aggiuntivi (come aggiornare i registri di `cred` e `w_index`).

3 Dati sperimentali

3.1 Sintesi

Per ottenere i dati riguardanti la sintesi del componente, è possibile fare riferimento a due report, uno relativo all'utilizzo delle risorse (`report_utilization`) e uno relativo alle tempistiche (`report_timing`).

Da quello sull'utilizzo delle risorse è possibile notare (vedi tabella "1. Slice Logic" riportata sotto) che non sono presenti latch nella sintesi del design. Da ciò è possibile trarre che, all'interno di ciascun processo, i segnali sono assegnati in modo completo per tutti i percorsi logici, evitando comportamenti non deterministici.

Site type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	78	0	0	134600	0.06
LUT as Logic	78	0	0	134600	0.06
LUT as Memory	0	0	0	46200	0.00
Slice Registers	26	0	0	269200	<0.01
Register as Flip Flop	26	0	0	269200	<0.01
Register as Latch	0	0	0	269200	0.00
F7 Muxes	0	0	0	67300	0.00
F8 Muxes	0	0	0	33650	0.00

Di seguito vengono riportati altri dati degni di nota, estratti dalle altre tabelle non riportate per brevità:

1. Non è stata utilizzata la BRAM disponibile.
2. Non è stato utilizzato nessun DSP.
3. Risulta utilizzato un solo segnale di clock.

Dall'analisi delle tempistiche si può notare che il componente opera con uno slack di 16.756ns e che il data path delay, ovvero il massimo tempo di propagazione lungo i percorsi del design, è di 3.093ns. Il requisito di 20ns è quindi ampiamente soddisfatto.

3.2 Simulazione

Per simulare l'operato del componente è stato fatto uso di diversi testbench, tutti strutturati in modo simile, che si occupano di simulare le operazioni di memoria e di controllare il componente collegandosi ad input e output dello stesso. Per ciascuna elaborazione che si vuole testare, si occupano di caricare in memoria i vari "scenari" e, ad elaborazione conclusa, di verificare che in memoria siano presenti i valori attesi, specificati a priori nel codice del testbench.

Di seguito sono riportati i risultati di alcune simulazioni significative al fine di mostrare il funzionamento del componente.

3.2.1 Sequenza di esempio

In questa simulazione (figura 4) è possibile vedere chiaramente i 5 momenti in cui l'accesso alla memoria (`tb_o_mem_en`) è disabilitato:

- Nel primo caso, la MSF è nello stato `WRITE_VALUE` e il dato letto è 0, ma non è ancora stato letto nessun valore
- Nel secondo e nel terzo caso, la MSF è in `WRITE_VALUE`, ma è stato letto un nuovo valore
- Nel quarto caso, la MSF è in `WRITE_CRED`, ma il valore letto è diverso da 0
- Nell'ultimo caso, la MSF è in `DONE`, e quindi non sono necessarie più operazioni sulla memoria

È anche possibile notare delle linee di colore più intenso all'interno del segnale tb_o_mem_en, dovute a rapidi cambiamenti nel valore, causati dai tempi di propagazione dei segnali. Questi cambiamenti avvengono comunque non a cavallo del fronte di salita e quindi non rappresentano un problema per il funzionamento del componente.

3.2.2 Sequenza di lunghezza 0

Vedi figura 5.

3.2.3 Elaborazioni consecutive

Vedi figura 6.

3.2.4 Gestione del segnale di reset

Vedi figure 7 e 8.

4 Conclusioni

I risultati sperimentali riguardanti la sintesi sono soddisfacenti, perché evidenziano un'assenza di latch non necessari e un tempo di propagazione massimo molto al di sotto del limite richiesto.

Sono soddisfacenti anche gli esiti delle simulazioni, che dimostrano che il componente è in grado di gestire diversi scenari correttamente. In particolare, il componente:

- produce una sequenza finale rispettando la specifica, tenendo in considerazione i casi notevoli delineati nell'introduzione.
- è in grado di gestire più elaborazioni consecutive.
- gestisce correttamente i segnali di reset indipendentemente dallo stato in cui si trova internamente.
- non accede a zone di memoria al di fuori del range della sequenza né in lettura né in scrittura.

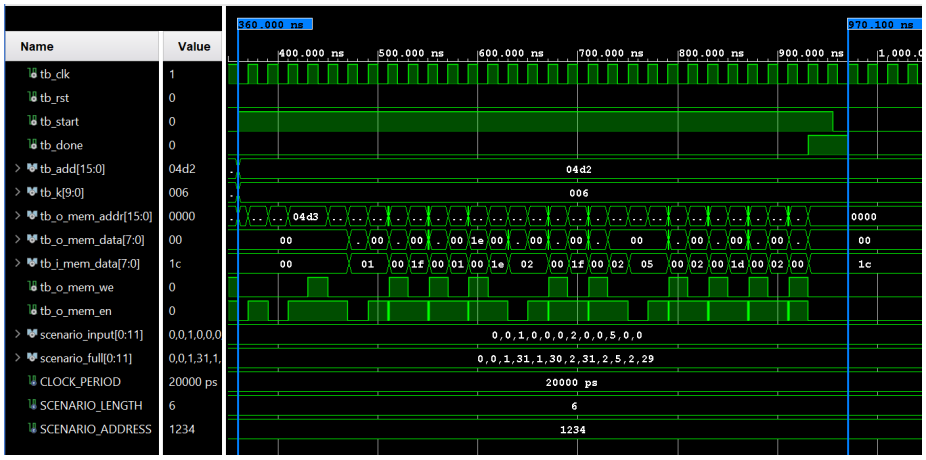

Figura 4: Oscilloscopio - sequenza di esempio

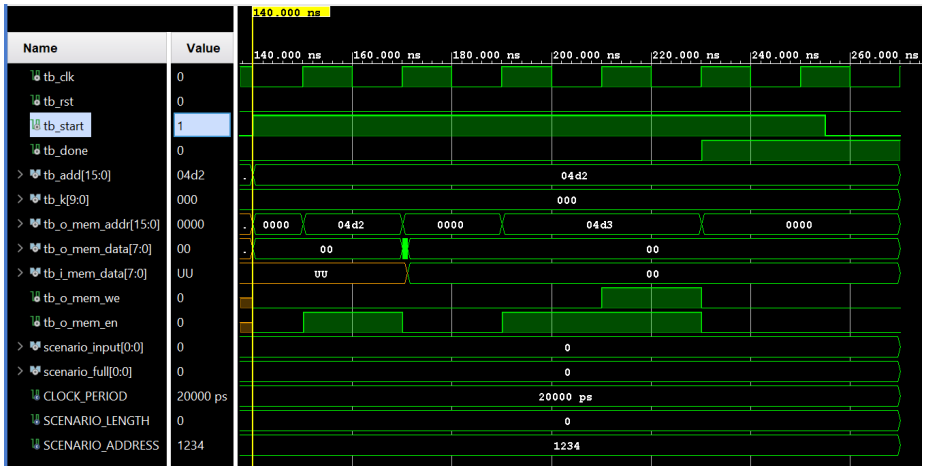

Figura 5: Oscilloscopio - sequenza vuota

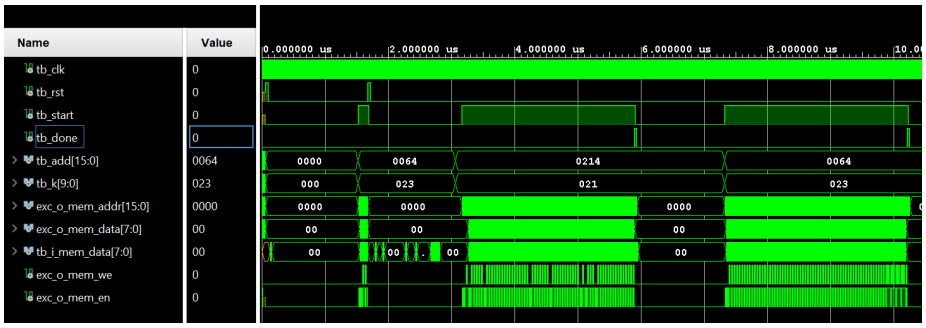

Figura 6: Oscilloscopio - elaborazioni consecutive

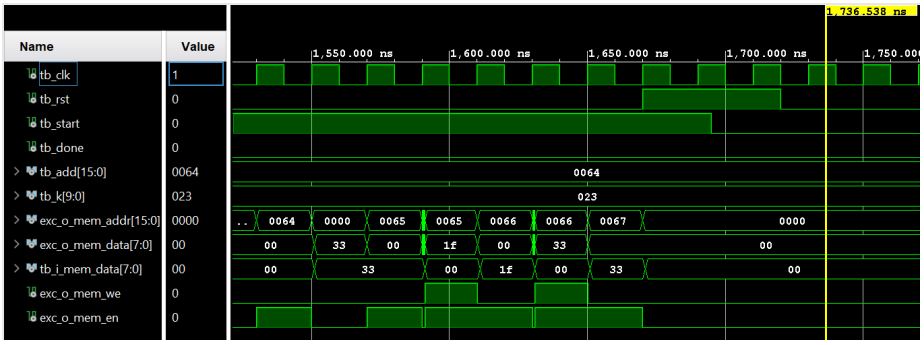

Figura 7: Oscilloscopio - reset durante lettura

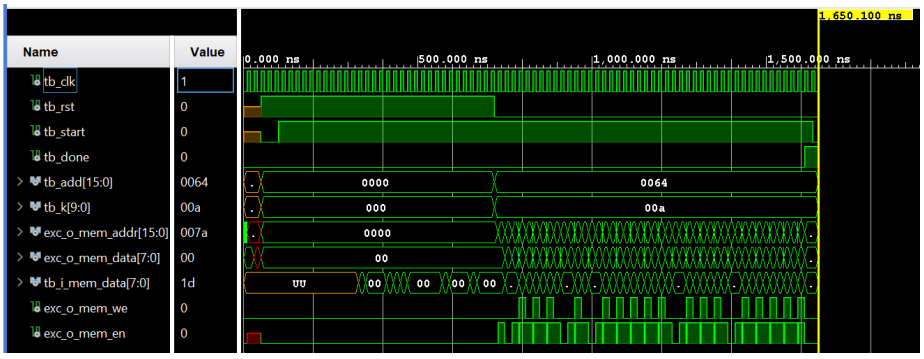

Figura 8: Oscilloscopio - start durante reset