PROVA FINALE RETI LOGICHE

Oliva Antonio - 10819753

Sommario

1 Introduzione **2**
1.1 Specifica 2

2 Architettura **3**
2.1 Prima implementazione 3
2.2 Problemi dell'architettura iniziale 3
2.3 Implementazione definitiva 4
2.4 Scelte Progettuali 5
2.4.1 Variabili utilizzate 5

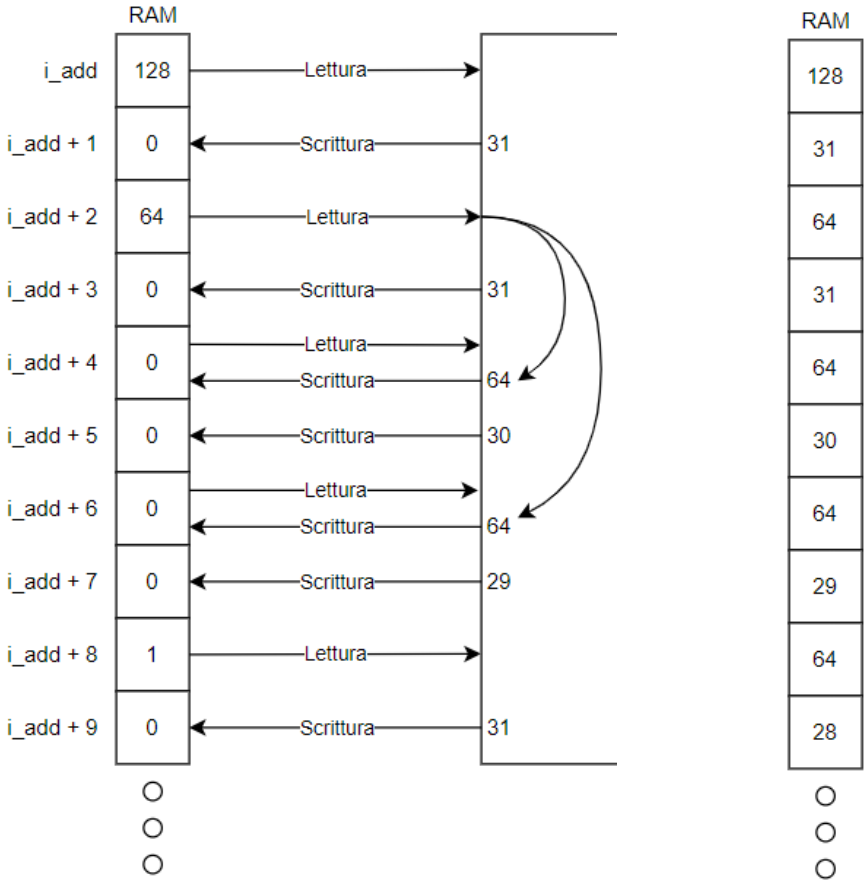
3 Risultati sperimentali **6**
3.1 Sintesi 6
3.1.1 Utilization Report 6
3.1.2 Timing Report 6
3.2 Simulazioni 6
3.2.1 Casi base 6
3.2.2 Casi particolari 7

4 Conclusioni **7**

1 Introduzione

1.1 Specifica

Il progetto consiste in un'architettura FSM che legge una parola da un indirizzo della RAM e scrive all'indirizzo successivo un valore di credibilità relativo alla parola letta, che può assumere come valore 31 o inferiore.



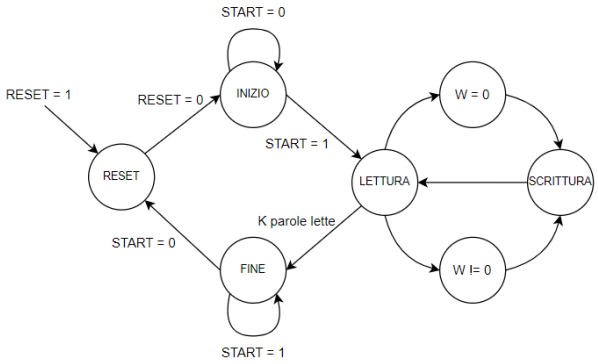
Si distinguono tre casi principali:

- **Parola letta diversa da 0**: *credibilità = 31*;
- **Prima parola letta e uguale a 0**: *credibilità = 0*;
- **Parola letta 0 e non prima parola**: *parola = parola precedente* e *credibilità = credibilità precedente - 1*.

2 Architettura

2.1 Prima implementazione

L'architettura consiste in una macchina a stati finiti deterministica, di cui una prima implementazione del tipo:



con stati:

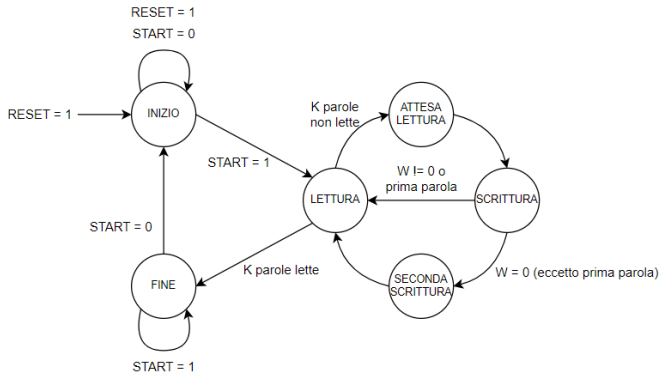
- **RESET**: Stato di reset in cui vengono impostati i valori di default quando i_rst = 1 oppure alla fine di un'elaborazione;
- **INIZIO**: Una volta impostato i valori, la macchina aspetta i_start = 1 per iniziare la lettura del nuovo scenario;
- **LETTURA**: Una volta ricevuto l'indirizzo, si legge all'interno della RAM la parola contenuta al suo interno e decide in quale stato andare;
- **W=0**: Parola letta 0; se è la prima parola letta, imposto la scrittura di 0 nell'indirizzo successivo (relativo alla credibilità), altrimenti sostituisco 0 con la parola letta precedentemente e decremento la sua credibilità di 1;
- **W!=0**: Imposto la scrittura di 31 nell'indirizzo successivo (relativo alla credibilità);
- **SCRITTURA**: Scrivo i valori impostati negli indirizzi corrispondenti;
- **FINE**: Alla fine della lettura imposto il segnale o_done = 1 e aspetto la discesa del segnale i_start.

2.2 Problemi dell'architettura iniziale

La lettura dell'informazione non avviene immediatamente ed è necessario aggiungere uno stato di attesa.
Inoltre, in caso di lettura di una parola W = 0, vanno impostate due scritture all'interno dello stato SCRITTURA. Risulta essere un processo molto scomodo da implementare e quindi si è deciso di impostare un processo diverso.

2.3 Implementazione definitiva

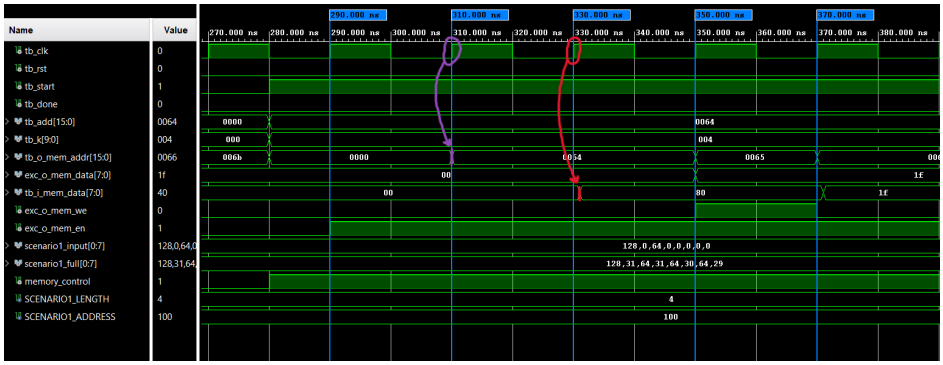
Apportando modifiche relative ai problemi dell'architettura iniziale, e ottimizzando la struttura unendo gli stati di RESET e INIZIO, la macchina a stati finiti utilizzata nel codice risulta essere composta da tali stati:



- **INIZIO**: In caso di i_rst = 1 imposta i valori di default, altrimenti aspetta l'ascesa del segnale i_start;
- **LETTURA**: Una volta ricevuto l'indirizzo, si legge all'interno della RAM la parola contenuta al suo interno;
- **ATTESA LETTURA**: Stato di attesa per aggiornare il valore letto
- **SCRITTURA**: Nel caso di parola diversa da 0 o prima parola letta uguale a 0, aggiorno il valore contenuto all'indirizzo successivo (relativo alla credibilità). In seguito imposto l'indirizzo di lettura successivo e torno allo stato di LETTURA.
  In caso di parola letta uguale a 0, sostituisco tale parola con quella letta nell'iterazione precedente e passo allo stato SECONDA SCRITTURA.
- **SECONDA SCRITTURA**: Nel caso di parola letta uguale a 0, aggiorno il valore contenuto all'indirizzo successivo (relativo alla credibilità).
- **FINE**: Alla fine della lettura imposto il segnale o_done = 1 e aspetto la discesa del segnale i_start.

2.4 Scelte Progettuali

Inizialmente, è stata implementata in VHDL la macchina a stati precedentemente ideata ed è stato associato a ogni segnale fornito il rispettivo valore assunto all'interno di ciascuno stato.
Un problema che ha richiesto particolare attenzione riguarda l'attesa di un aggiornamento di i_mem_data, dato che, come visto nell'immagine sottostante, viene aggiornato con 1ns di ritardo e, non potendo leggere sul fronte di discesa causa specifica, c'è bisogno di aspettare un ciclo di clock aggiuntivo.



2.4.1 Variabili utilizzate

Successivamente, durante l'implementazione, si è pensato alle variabili necessarie. Prima variabile ipotizzata era una semplice variabile i, utilizzata come contatore di parole lette (una volta arrivati in una situazione in cui i = i_k sarebbe terminata l'elaborazione). In seguito, è stata rimossa questa variabile e si è optato invece per l'utilizzo di una variabile **K**, inizializzata a i_k e decrementata a ogni lettura, fino a raggiungere K = 0, in modo tale da poter lavorare anche nel caso in cui i_k dovesse cambiare durante l'elaborazione.

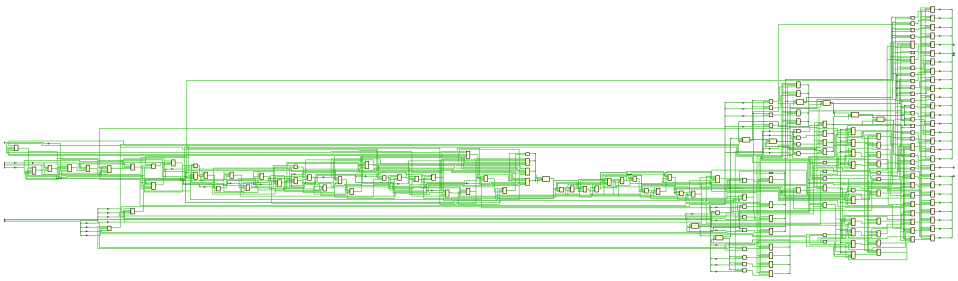
Altre variabili ausiliarie sono **LastW** e **C**, che corrispondono rispettivamente al valore della parola e al valore della credibilità all'iterazione precedente, utilizzate per non dover leggere nuovamente tali valori prima di apportare modifiche nell'iterazione considerata.

Ulteriori modifiche apportate al codice hanno portato all'eliminazione dello stato di ATTESA LETTURA, integrato nello stato di LETTURA attraverso la variabile **parola_letta**.

3 Risultati sperimentali

3.1 Sintesi

Di seguito riportata la Schematics del codice da me implementato, costituito da 249 cells, 64 I/O ports e 343 nets.



3.1.1 Utilization Report

Syte Type	Used
LUT (all as Logic)	77
Slice Registers (all as Flip Flops)	73

3.1.2 Timing Report

Slack: 17.037 ns (required time - arrival time)

3.2 Simulazioni

3.2.1 Casi base

Prima testbench simulata è quella fornita insieme alla specifica del progetto, servita per testare la correttezza del programma da me implementato, in cui vengono analizzati i casi principali di lettura e scrittura su RAM.
In questa testbench, vengono lette parole diverse da zero, per le quali viene aggiornata la credibilità a 31, e parole uguali a zero, nessuna di queste come prima parola, in cui sono stati aggiornati il valore della parola (prendendo come nuovo valore la parola precedente) e, in seguito, della credibilità (decrementando la credibilità della parola precedente).

Non essendo controllato anche il caso in cui la prima parola letta sia uguale a zero, ho creato una nuova testbench in cui vengono lette sole parole uguali a zero, le cui credibilità sono in seguito aggiornate con valore 0.

Un ultimo caso di test è relativo al controllo della corretta decrementazione del valore di credibilità. Nella testbench di riferimento, ho impostato la lettura di una parola diversa da zero e di successive 33 parole uguali a zero.

3.2.2 Casi particolari

Una volta controllato il corretto funzionamento dei casi base, ho deciso di analizzare un comportamento particolare della specifica, ovvero il caso in cui avvengano due elaborazioni successive.

4 Conclusioni

Potevano essere fatte ulteriori modifiche per rendere il codice più ottimizzato. Prima tra queste, credo sia possibile rimuovere lo stato di ATTESA_LETTURA senza aggiungere la variabile parola_letta semplicemente aspettando nello stato di SCRITTURA il fronte di discesa del clock, riuscendo così a ridurre il tempo di lettura a un unico ciclo di clock.

Altra possibile modifica (se gli stati sono codificati allo stesso modo di come venivano codificati durante il corso) è rimuovere uno dei cinque stati presenti all'interno della FSM, in modo tale da poter codificare i quattro stati rimanenti attraverso due Flip Flop, invece di usarne tre per cinque stati. L'idea era unificare lo stato di FINE con lo stato di INIZIO, ponendo ad esempio come condizione:

$$if\ K = "000000000"$$

(quando K è uguale a 0 sono state lette tutte le parole).

Altra possibile rimozione riguarda lo stato di SECONDA SCRITTURA, aspettando nello stato di LETTURA la lettura della parola e, se zero, sostituirla prima di andare nello stato di SCRITTURA.