POLITECNICO
MILANO 1863

Prova Finale
Progetto di Reti Logiche

Prof. Gianluca Palermo
Anno Accademico 2023/2024

Nome	Codice persona	Matricola
Nicolò Sardella	10724332	955391
Pietro Venturelli	10730295	959370

Indice

1 Introduzione

1.1 Scopo del progetto

L'obiettivo del progetto consiste nell'implementare un modulo HW, descritto in VHDL, in grado di interfacciarsi con una memoria, analizzarne il contenuto e modificarlo.

1.2 Specifiche generali

Per semplicità, la memoria, già implementata nel Test Bench, è suddivisibile in coppie di celle adiacenti. Ogni coppia contiene due valori che occupano 1 byte ciascuno:

- W: un valore compreso tra 0 e 255 che rappresenta la parola da leggere (0 viene considerato come "valore non specificato").
- C: un valore di credibilità compreso tra 0 e 31, calcolato come descritto di seguito.

Il modulo riceve due segnali in input:

- K: rappresenta il numero di parole W da leggere.
- ADDR: è l'indirizzo da cui iniziare la lettura.

In seguito inizia ad agire sulla memoria. Le celle sono lette in modo sequenziale, e vengono sovrascritte nel seguente modo:

- $\forall W_i \neq 0 \quad \Rightarrow \quad C_i = 31 \quad \text{con} \quad 0 \leq i \leq K-1,\, i \in \mathbb{N}$
- $\forall W_i = 0 \quad \Rightarrow \quad \begin{cases} C_i = C_{i-1} - 1 \quad con \quad 1 \leq i \leq K-1,\, i \in \mathbb{N} \\ C_i = 0 \quad con \quad i = 0 \quad \vee \quad C_{i-1} = 0 \end{cases}$

  Inoltre il valore W non specificato è sostituito con l'ultima parola valida letta.

1.3 Interfaccia del componente

Il progetto richiede di realizzare il componente secondo la seguente interfaccia:

```
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 down to 0);
        i_k     : in std_logic_vector(9 down to 0);

        o_done  : out std_logic;

        o_mem_addr : out std_logic_vector(15 down to 0);
        i_mem_data : in std_logic_vector(7 down to 0);
        o_mem_data : out std_logic_vector(7 down to 0);
        o_mem_we   : out std_logic;
        o_mem_en   : out std_logic
    );
end project_reti_logiche;
```

A seguire una breve descrizione dei segnali:

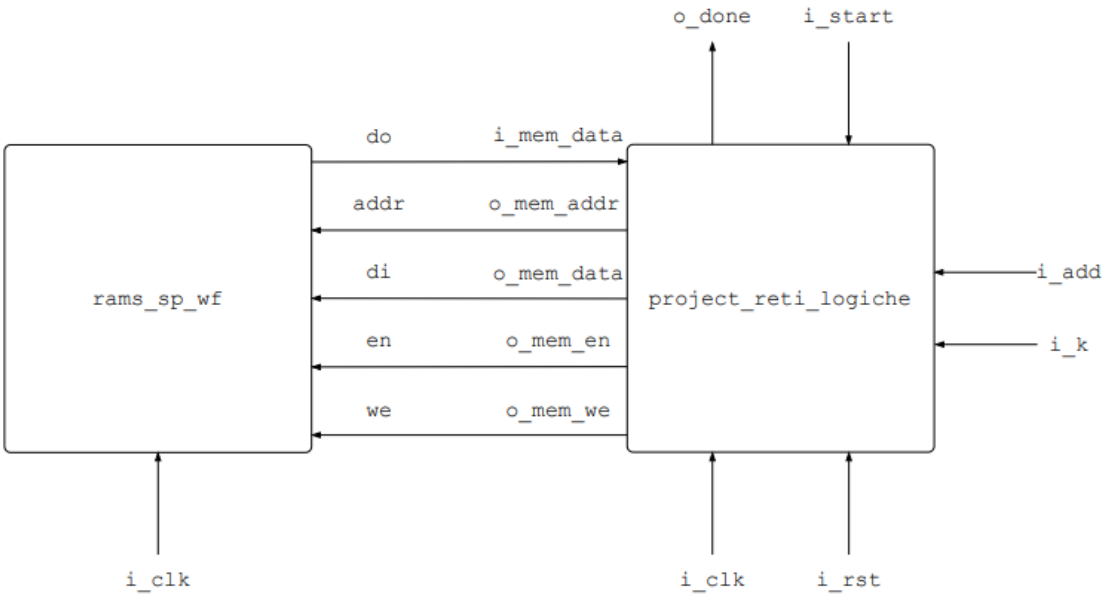
- i_clk: CLOCK in ingresso generato dal Test Bench;
- i_rst: RESET, inizializza la macchina per essere pronta a ricevere il primo segnale di START;
- i_start: START, indica l'inizio della computazione;
- i_add: segnale (vettore da 16 bit) ADD, rappresenta l'indirizzo di memoria da cui partire;
- i_k: segnale (vettore da 10 bit) K, indica la lunghezza della sequenza;
- o_done: DONE, l'uscita che comunica la fine dell'elaborazione;
- o_mem_addr: segnale (vettore da 16 bit) di uscita che passa l'indirizzo su cui agire alla memoria;
- i_mem_data: segnale (vettore da 8 bit) proveniente dalla memoria che contiene il dato letto;
- o_mem_data: segnale (vettore da 8 bit) che passa alla memoria il dato da scrivere;
- o_mem_we: WRITE ENABLE, da mandare alla memoria. Se è posto a 1 è possibile scrivere, altrimenti (segnale posto a 0) è possibile solo la lettura;
- o_mem_en: ENABLE, da inviare alla memoria. Consente di comunicarci sia in scrittura che in lettura.

1.4 Interfaccia della memoria

La memoria è di tipo "Single-Port Block RAM Write-First Mode" e presenta la seguente interfaccia:

```
entity rams_sp_wf is
    port (
        clk   : in std_logic;
        we    : in std_logic;
        en    : in std_logic;
        addr  : in std_logic_vector(15 down to 0);
        di    : in std_logic_vector(7 down to 0);
        do    : out std_logic_vector(7 down to 0);
    );
end rams_sp_wf;
```

Collegando questa memoria al modulo da progettare si ottiene il seguente schema:



1.5 Esempio

Per rendere più chiaro il funzionamento del modulo si riporta di seguito un esempio, i valori delle celle sono esadecimali.

In questo caso il nostro componente ha ricevuto i seguenti segnali:

- K = 12;
- ADDR = ADDR0.

Il modulo dovrà quindi leggere 12 parole per un totale di 24 indirizzi (considerando anche le celle C

W0	00	ADDR0	W4	00	ADDR8	W8	45	ADDR16
C0	00	ADDR1	C4	00	ADDR9	C8	00	ADDR17
W1	00	ADDR2	W5	FA	ADDR10	W9	AB	ADDR18
C1	00	ADDR3	C5	00	ADDR11	C9	00	ADDR19
W2	FF	ADDR4	W6	10	ADDR12	W10	D4	ADDR20
C2	00	ADDR5	C6	00	ADDR13	C10	00	ADDR21
W3	00	ADDR6	W7	29	ADDR14	W11	00	ADDR22
C3	00	ADDR7	C7	00	ADDR15	C11	00	ADDR23

Porzione di memoria interessata

Partiamo con l'analizzare le coppie di celle della prima colonna (in giallo le celle modificate).

W0	00	ADDR0
C0	00	ADDR1
W1	00	ADDR2
C1	00	ADDR3

La sequenza parte con $W_0 = 00$, seguendo le indicazioni della specifica non è possibile aggiornare nè W nè C, non avendo informazioni valide lette precedentemente.
Lo stesso ragionamento si può fare per la successiva coppia di celle

W2	FF	ADDR4
C2	1F	ADDR5

Troviamo il primo W diverso da 0, quindi possiamo modificare il valore C ponendolo a 31

W3	FF	ADDR6
C3	1E	ADDR7

Qua torniamo di nuovo al caso W = 0, ma questa volta abbiamo già letto dati utilizzabili per aggiornare la memoria. Poniamo il valore della parola uguale all'ultima parola valida letta (in questo caso $W_3 = W_2$) e calcoliamo la credibilità ($C_3 = C_2 - 1 = 30$)

Continuando ad applicare l'algoritmo della specifica otteniamo lo stato della memoria sotto riportato:

W0	00	ADDR0	W4	FF	ADDR8	W8	45	ADDR16
C0	00	ADDR1	C4	1D	ADDR9	C8	1F	ADDR17
W1	00	ADDR2	W5	FA	ADDR10	W9	AB	ADDR18
C1	00	ADDR3	C5	1F	ADDR11	C9	1F	ADDR19
W2	FF	ADDR4	W6	10	ADDR12	W10	D4	ADDR20
C2	1F	ADDR5	C6	1F	ADDR13	C10	1F	ADDR21
W3	FF	ADDR6	W7	29	ADDR14	W11	D4	ADDR22
C3	1E	ADDR7	C7	1F	ADDR15	C11	1F	ADDR23

2 Architettura

2.1 Struttura del componente

La rete logica realizzata può essere divisa in 5 moduli:

Modulo 1(FSM): macchina a stati finiti che si occupa della logica di controllo principale

Modulo 2 (counter): contatore che indica lo stato di esecuzione e l'indirizzo di accesso in memoria

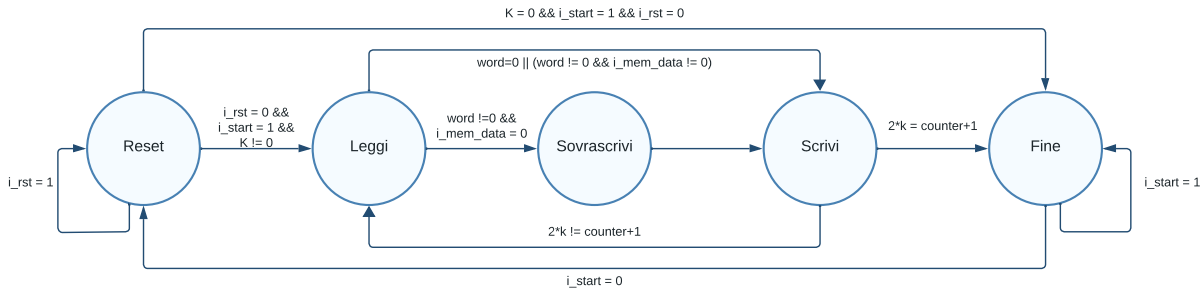
Modulo 3 (buffer): registro che memorizza il valore in ingresso dalla memoria ad ogni fronte di discesa di i_clk

Modulo 4 (word): registro che mantiene l'ultimo dato valido letto da memoria

Modulo 5 (C): registro che memorizza la credibilità

2.1.1 Modulo 1

Il primo modulo si occupa della gestione logica principale, coordinando gli altri componenti secondo le necessità imposte dalle Specifiche generali. Per memorizzare lo stato di esecuzione viene usato un registro **state**, mentre per l'operazione eseguita effettivamente il segnale **actual_state**. Per un funzionamento corretto, sono stati individuati 5 stati:



- **Reset**: è lo stato in cui si evolve se i_rst = 1 o si finisce una computazione. I registri vengono portati ai seguenti valori:
  - counter : $0_{(dec)}$
  - word : $0_{(dec)}$
  - C : $31_{(dec)}$

  mentre i segnali di controllo:
  - o_done : $0_{(dec)}$
  - o_mem_data : $0_{(dec)}$
  - o_mem_we: $0_{(dec)}$
- **Leggi**: è lo stato in cui vengono analizzati i valori del buffer e di word. A seconda dei valori di quest'ultimi verrà deciso se scrivere o sovrascrivere. Nel caso in cui word sia $0_{(dec)}$ sicuramente non è stato letto alcun valore valido per la sovrascrittura, quindi actual_state = scrivi.

Se **buffer** è uguale a $0_{(dec)}$, i registri vengono portati ai seguenti valori:

- counter : counter + 1
- word : buffer
- C : C

i segnali di controllo:

- o_done : $0_{(dec)}$
- o_mem_data : $0_{(dec)}$
- o_mem_we: $1_{(dec)}$

Nel caso contrario, un valore **valido** è stato letto in memoria, il prossimo indirizzo di memoria viene portato a $31_{(dec)}$.

- counter : counter + 1
- word : buffer
- C : C

i segnali di controllo:

- o_done : $0_{(dec)}$
- o_mem_data : C
- o_mem_we: $1_{(dec)}$

Quando il valore di **word** è diverso da $0_{(dec)}$, viene controllato il **buffer** per scegliere se sovrascrivere la memoria con l'ultimo valore valido o resettare la credibilità a 31. Se **buffer** è uguale a $0_{(dec)}$ allora actual_state = sovrascrivi e i registri vengono portati ai seguenti valori:

- counter : counter
- word : word
- C : C - 1 || C (se C = $0_{(dec)}$)

i segnali di controllo:

- o_done : $0_{(dec)}$
- o_mem_data : word
- o_mem_we: $1_{(dec)}$

Altrimenti actual_state = scrivi.

- counter : counter + 1
- word : buffer
- C : $31_{(dec)}$

i segnali di controllo:

  - o_done : $0_{(dec)}$
  - o_mem_data : C
  - o_mem_we: $1_{(dec)}$

- **Sovrascrivi**: é lo stato in cui a seguito di una sovrascrittura vengono impostati i registri per scrivere il valore aggiornato di C in memoria. actual_state = scrivi.

  - counter : counter + 1
  - word : word
  - C : C

  mentre i segnali di controllo:

  - o_done : $0_{(dec)}$
  - o_mem_data : C
  - o_mem_we: $1_{(dec)}$

- **Scrivi**: é lo stato in cui a seguito di una scrittura vengono aggiornati i registri per leggere da memoria un nuovo valore, actual_state = leggi. Se (counter + 1) = (K * 2) allora actual_state = fine. I registri vengono portati ai seguenti valori:

  - counter : counter + 1
  - word : word
  - C : C

  mentre i segnali di controllo:

  - o_done : $0_{(dec)}$
  - o_mem_data : don't care
  - o_mem_we: $0_{(dec)}$

- **Fine**: è lo stato di termine esecuzione, è l'unico stato in cui o_done = 1. Finché i_start = 1 actual_state = fine, altrimenti actual_state = reset e il modulo sarà pronto per un'altra esecuzione.

  - counter : $0_{(dec)}$
  - word : $0_{(dec)}$
  - C : $0_{(dec)}$

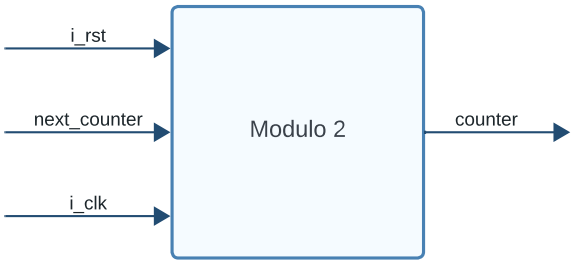
  mentre i segnali di controllo:

  - o_done : $1_{(dec)}$
  - o_mem_data : $0_{(dec)}$
  - o_mem_we: $0_{(dec)}$

Il valore del segnale di uscita o_done è collegato agli stati e non all'ingresso specifico, la FSM rispetta le caratteristiche di una **macchina di Moore**.

2.1.2 Modulo 2

Il secondo modulo svolge la funzionalità di un **contatore**. La FSM lo utilizza per tenere conto di quante parole mancano per finire la computazione, inoltre viene usato per accedere alla memoria sommandolo all'indirizzo di partenza i_addr. Poiché i_k è un vettore di 10 bit e gli indirizzi di memoria da controllare sono 2*i_k, il modulo è composto da un registro di grandezza 11 bit e può esere schematizzato nel seguente modo:



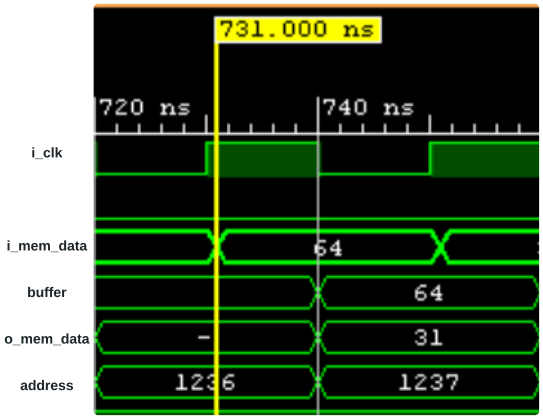
I segnali di controllo sono:

- **i_rst** : serve a resettare a $0_{(dec)}$ in modo asincrono il modulo come richiesto da specifiche
- **next_counter** : viene gestito dalla FSM. Nel caso di sovrascrittura non viene aumentato in modo da poter sovrascrivere lo $0_{(dec)}$ letto in memoria dal buffer.
- **i_clk** : è il segnale che controlla l'evoluzione del modulo al valore di next_counter

il segnale counter è il valore effettivo di uscita del modulo, usato dal resto della rete.

2.1.3 Modulo 3

Il terzo modulo è un registro usato dalla FSM per leggere dalla memoria. Poiché la memoria presenta un ritardo di uscita di 1 ns dal rising edge di i_clk, è stato scelto di lavorare sul **falling edge** in modo di evitare stati d'attesa. Un esempio di lettura è riportato qui sotto:



A partire dell'istante 731 ns il valore dell'indirizzo 1236(dec) è disponibile su i_mem_data, il buffer viene aggiornato in modo sincrono all'istante 740 ns. La FSM basandosi sul valore di buffer decide che valore scrivere nell'indirizzo 1237(dec) o se sovrascrivere l'indirizzo 1236(dec)

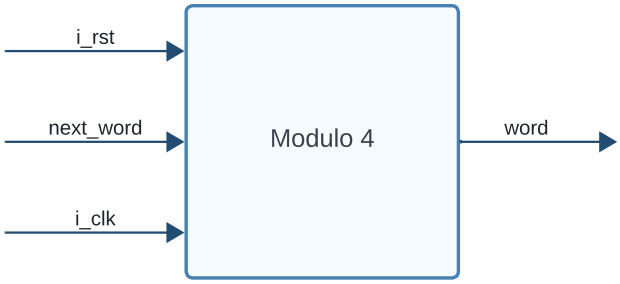
I segnali di controllo sono:

- **i_rst** : serve a resettare in modo asincrono il modulo, come richiesto da specifiche
- **i_mem_data** : è il dato letto dalla memoria
- **i_clk** : è il segnale che controlla l'evoluzione del modulo al valore di i_mem_data ogni suo falling edge

il segnale buffer è il valore effettivo di uscita del modulo, usato dal resto della rete.

2.1.4 Modulo 4

Il quarto modulo è un registro che ha il compito di salvare l'ultimo **valore valido** letto in memoria. Viene inizializzato a $0_{(dec)}$ e, ad ogni valore valido letto dal buffer, viene aggiornato sul falling edge di i_clk. Poiché la memoria contiene un byte per indirizzo, è un registro di 8 bit. Può essere schematizzato nel seguente modo:



I segnali di controllo sono:

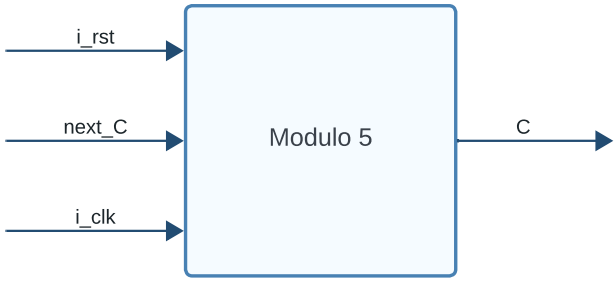
- **i_rst** : serve a resettare a $0_{(dec)}$ in modo asincrono il modulo come richiesto da specifiche
- **next_word** : viene gestito dalla FSM, contiene il valore di buffer se diverso da zero

- **i_clk** : è il segnale che controlla l'evoluzione del modulo al valore di next_counter

il segnale word è il valore effettivo di uscita dal modulo, viene usato per la sovrascrittura degli zeri.

2.1.5 Modulo 5

Il quinto modulo è un registro che ha il compito di salvare la **credibilità** del dato che viene sovrascritto. Viene inizializzato a $31_{(dec)}$ e, ad ogni sovrascrittura, è decrementato di uno, fino a raggiungere $0_{(dec)}$. Se viene letta una parola valida, il modulo viene riportato a $31_{(dec)}$. Poichè il valore massimo salvabile è 31, è un registro di 5 bit. Può essere schematizzato nel seguente modo:



I segnali di controllo sono:

- **i_rst** : serve a resettare a $31_{(dec)}$ in modo asincrono il modulo come richiesto da specifiche
- **next_C** : viene gestito dalla FSM secondo la logica descritta nel modulo 1
- **i_clk** : è il segnale che controlla l'evoluzione del modulo al valore di next_C

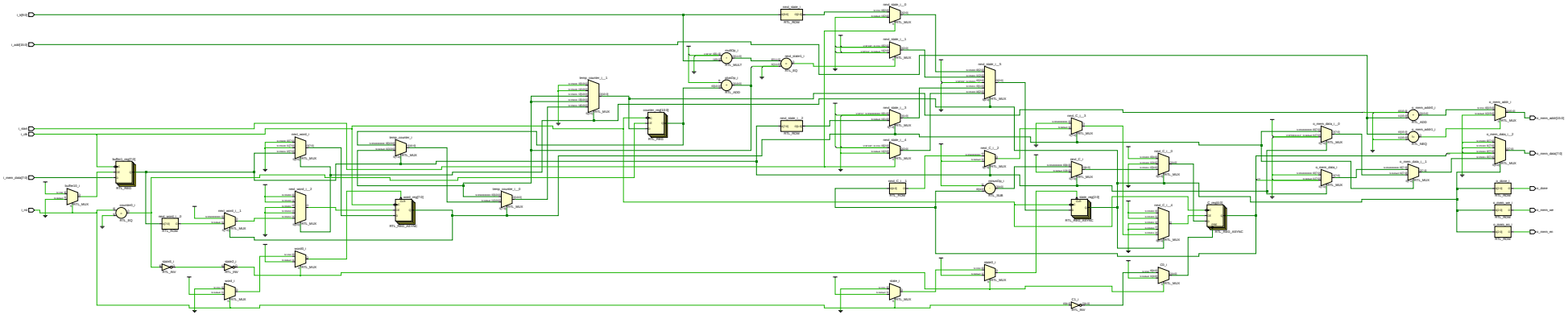
il segnale C è il valore effettivo di uscita dal modulo, viene usato per la scrittura della credibilità in memoria.

2.2 Processi

Per un corretto funzionamento sono stati implementati i seguenti processi:

- **synchronous_logic** : è il processo che governa l'evoluzione della FSM al prossimo stato, è sincronizzato sul falling edge di i_clk e presenta un controllo asincrono sul valore di i_rst.
- **output_logic** : assegna ad ogni stato i valori dei registri e dei segnali di controllo
- **memory_reader** : aggiorna il buffer ogni falling edge di i_clk, presenta un controllo asincrono su i_rst
- **address_updater** : somma ad i_addr il valore di counter per aggiornare l'indirizzo di lavoro della memoria (o_mem_addr)
- **next_state_logic** : imposta, a seconda dello stato attuale e di alcuni valori dei registri, il prossimo stato della FSM

2.3 Schematico



3 Risultati

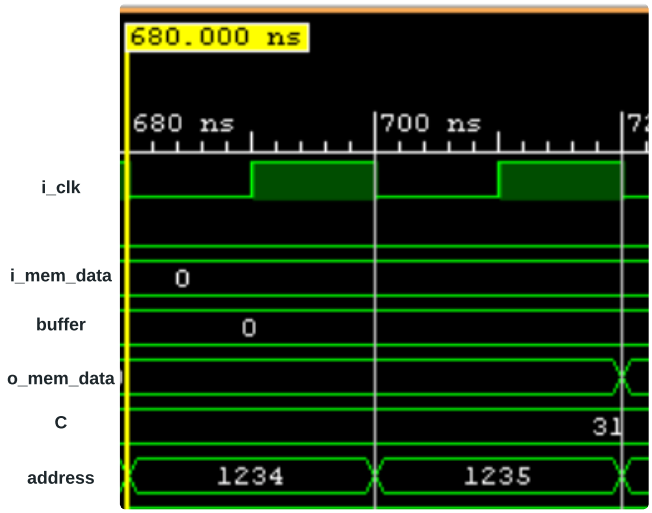
3.1 Test

Per testare il corretto funzionamento della rete sono stati testati i seguenti casi limite:

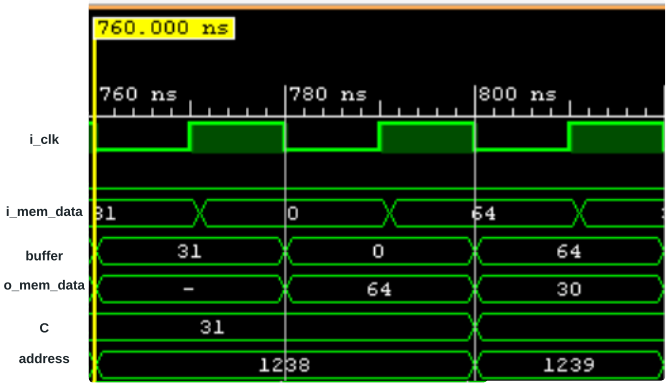
- la sequenza da analizzare è composta **solo da zeri**
- viene richiesta una **doppia sequenza** di lavoro
- la sequenza di lavoro è **interrotta da i_rst**
- viene fornito **K uguale a zero**
- in memoria è presente una sequenza di zeri di numero **maggiore** della credibilità massima

Di seguito alcuni esempi di esecuzione significativi:

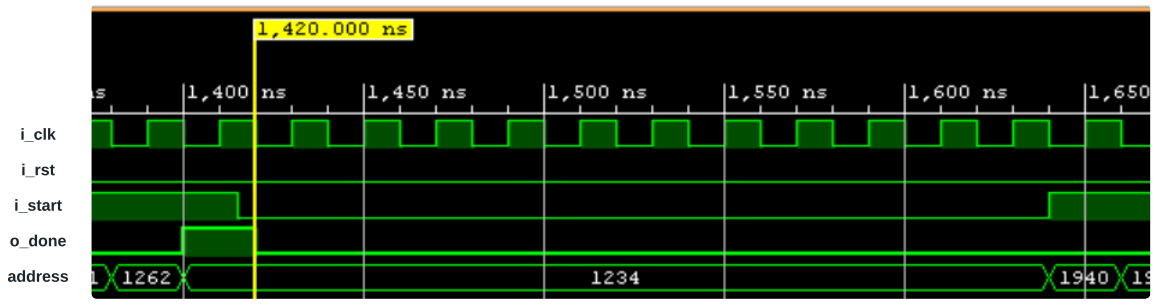
- uno zero ad inizio esecuzione



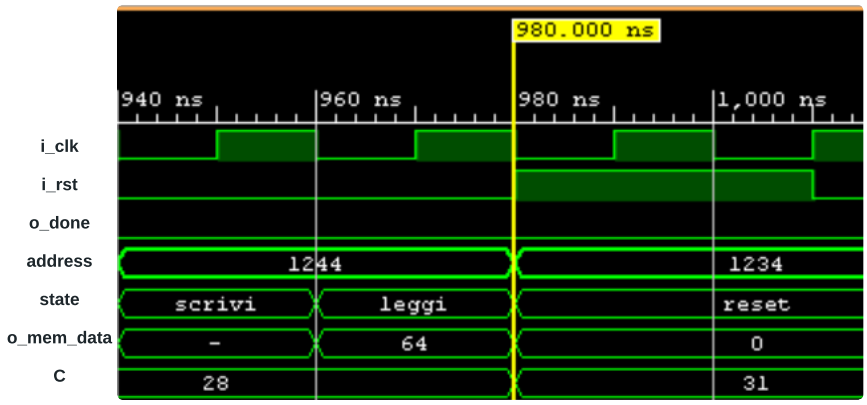
- sovrascrittura di uno zero letto dalla memoria, C viene decrementata



- un esempio di scrittura è stato affrontato qui
- viene richiesta una doppia sequenza di lavoro



- reset = 1 durante la sequenza di lavoro



3.2 Sintesi

La rete viene correttamente sintetizzata, non sono presenti warning e tutti i test sopra citati vengono portati a termine in post-sintesi functional e timing. Inoltre, dal punto di vista dei timing, il WNS (worst negative slack) è di 14,156 ns (71 % del ciclo di i_clk).

4 Commenti

La rete progettata rispetta le richieste delle specifiche generali, supera il test fornito d'esempio e i test di casi particolari. L'approccio tenuto durante la progettazione è stato quello di ottimizzare il numero degli stati della FSM, evitare stati di attesa per ridurre il tempo di esecuzione e limitare l'uso di flip-flop del componente hardware. Nonostante i risultati siano soddisfacenti, siamo consapevoli che riducendo di uno stato la FSM (da 5 a 4) avremmo risparmiato un registro passando da 3 bit a 2 bit. Anche se non richiesto, è stato testato un tempo di clock inferiore. Nonostante la post-sintesi functional esegua correttamente con clock pari a 2 ns(limite imposto dal ritardo della memoria), la post-sintesi timing incontra problemi intorno ai 15 ns. Migliorando il WNS si potrebbe aumentare la frequenza di clock, migliorando le prestazioni della rete. Inoltre bisogna considerare che i limiti della board, usata per implementare la rete, sono stati trascurati.

Clock	velocità di esecuzione senza zeri letti	velocità di esecuzione con sovrascrittura
20 ns / 50 Mhz	50 MB/s	33 MB/s
16 ns / 63 Mhz	63 MB/s	41,6 MB/s