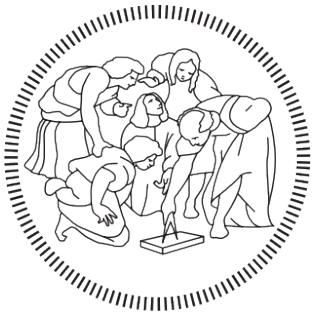
Prova finale di reti logiche

Prof. Gianluca Palermo

Anno accademico 2023 - 2024

Studente: Matteo Sabino



POLITECNICO
MILANO 1863

Indice

1. Introduzione

L'obiettivo del progetto è l'implementazione di un modulo hardware (HW) in grado di interfacciarsi con una memoria. La realizzazione è stata effettuata mediante un linguaggio di descrizione dell'hardware, nello specifico il **VHDL** (Very High Speed Integrate Circuit - Hardware Description Language).
Il progetto è stato supportato dall'utilizzo della suite di design **Vivado**, la quale ha permesso la simulazione e la sintesi durante lo sviluppo del modulo HW.
Il componente, a seguito di segnali fornitigli in ingresso, esegue operazioni di lettura e scrittura nella memoria secondo la politica specificata in seguito.

1.1 Interfaccia del componente

```
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 : out  std_logic_vector(7 downto 0);
        o_mem_we : out  std_logic;
        o_mem_en : out std_logic
    );
end project_reti_logiche;
```

1.1.1 Segnali del componente

- **i_clk** : segnale di clock per la sincronizzazione **CLK**;
- **i_rst** : segnale di reset **RST** asincrono;
- **i_start** : segnale di **START** per iniziare l'elaborazione;

- **i_add** : segnale di **ADD** che indica l'indirizzo da cui iniziare l'elaborazione;
- **i_k** : segnale per indicare quante parole $\{W_i\}_{i=0}^{K-1}$ sono da elaborare;
- **o_done** : segnale di terminazione **DONE**;
- **o_mem_addr** : indirizzo a cui il modulo vuole accedere;
- **i_mem_data** : input fornito dalla memoria al modulo;
- **o_mem_data** : output fornito dal modulo alla memoria;
- **o_mem_we** : segnale per determinare se il modulo vuole scrivere in memoria;
- **o_mem_en** : segnale per determinare se il modulo vuole accedere alla memoria.

1.2 Specifiche complete

A seguito di un primo segnale di **RESET**, il modulo deve rimanere in attesa del segnale di **START** per avviare l'elaborazione. Quando il segnale **START** è posto a 1, in ingresso al modulo i seguenti segnali presenteranno valori stabili: **ADD** e **K**.

- Il segnale **ADD** indica l'indirizzo di memoria da cui deve iniziare l'elaborazione.
- Il segnale **K** rappresenta il numero di parole $\{W_i\}_{i=0}^{K-1}$ da dover elaborare.

Le parole W_i assumono valori compresi tra 0 e 255. In particolare, il valore nullo ($W_i = 0$) indica che *"il valore non è specificato"*.

Partendo dall'indirizzo **ADD**, le **K** parole W_i sono disposte all'interno della memoria a una distanza di 2 byte. Il range di indirizzi utilizzato è quindi:

$$\{ADD, ADD + 2, ADD + 4, \ldots, ADD + 2(K - 1)\}$$

A ciascuna parola W_i deve essere associato un numero, detto valore di credibilità **C**. L'indirizzo immediatamente successivo a quello di ogni parola W_i è dedicato alla scrittura del valore di credibilità C_i corrispondente.

Il modulo HW ha il compito di leggere la sequenza di parole W_i, assegnare loro un valore di credibilità **C** e sostituire eventuali parole nulle con l'ultima parola non nulla letta nella sequenza. Il valore di credibilità viene determinato seguendo queste regole:

1. $C_i = 31$, se la parola W_i è non nulla.
2. $C_i = C_{i-1} - 1$, se la parola W_i è nulla, dove C_{i-1} è il valore di credibilità della parola precedente nella sequenza.
3. Se la prima parola della sequenza è nulla, essa rimane tale e il suo valore di credibilità sarà 0.
4. Deve essere garantito che $\forall i \in \{0, \ldots, K - 1\}, C_i \geq 0$, ovvero, per ogni parola W_i, il valore di credibilità non può essere negativo.

5. ***NOTA:*** Se $C_{i-1} = 0$ e W_i è nullo, per evitare che il valore di credibilità diventi negativo, C_i rimarrà a 0.

Una volta terminata l'elaborazione, il modulo HW deve impostare il segnale **DONE** a 1 e mantenerlo tale finché il segnale di **START** non è posto a 0. Un nuovo segnale di **START** non può essere fornito finché **DONE** rimane alto.

Per quanto riguarda le successive elaborazioni, non è necessario ricevere altri segnali di **RESET**.

1.3 Esempio di elaborazione

Ad esempio esplicativo di ciò che è richiesto dal modulo si riporta l'esempio di elaborazione nella *Figure 1.1.* In questa casistica sono stati utilizzati i valori **ADD** = 1024 e **K** = 5

INIZIALE	
Indirizzo (ADD)	Parola (W)
1024	0
1025	0
1026	1
1027	0
1028	2
1029	0
1030	0
1031	0
1032	0
1033	0
1034	0

FINALE	
Indirizzo (ADD)	Parola (W)
1024	0
1025	0
1026	1
1027	31
1028	2
1029	31
1030	2
1031	30
1032	2
1033	29
1034	0

Figure 1.1: Esempio: indirizzi e dati in memoria prima (*sinistra*) e dopo (*destra*) l'elaborazione (**ADD** = 1024, **K** = 5)

In questo esempio, una volta che la macchina riceve il segnale di **START**, inizia l'elaborazione leggendo la prima parola all'indirizzo 1024. Poiché W_0 è nullo, ma è il primo della sequenza, esso rimane tale e il suo valore di credibilità $C_0 = 0$, viene memorizzato all'indirizzo 1025. La lettura successiva avverrà all'indirizzo 1026. Poiché $W_1 = 1$ è non nullo, il valore di credibilità assegnatogli è $C_1 = 31$. La parola successiva, $W_2 = 2$, essendo maggiore di zero, avrà anch'essa $C_2 = 31$. Quando la macchina arriva alla parola $W_3 = 0$, all'indirizzo 1028, riconosce che il valore è nullo, quindi, memorizza $W_3 = W_2 = 2$ e $C_3 = C_2 - 1 = 30$. L'elaborazione della parola W_4 segue lo stesso processo di quella precedente. Terminata la sequenza la macchina segnalerà la fine dell'elaborazione tramite il segnale **DONE**.

2. Architettura

Il modulo hardware è stato realizzato implementando un **Datapath** e un **Automa a stati finiti** (Finite State Machine - FSA).

2.1 Datapath

Il **Datapath** rappresenta l'insieme delle unità di calcolo del componente dedicate all'elaborazione dei segnali. Esso è formato da tre moduli principali:

- **Modulo per la gestione degli indirizzi**
- **Modulo per il conteggio delle parole rimanenti**
- **Modulo per la scrittura in memoria dei dati e della credibilità**

2.1.1 Interfaccia del Datapath

```
entity datapath is
    port (
        i_clk : in  std_logic;
        i_rst : in  std_logic;
        i_add : in  std_logic_vector(15 downto 0);
        i_k : in  std_logic_vector(9 downto 0);

        is_starting_k_zero : out std_logic;
        is_k_zero : 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 : out  std_logic_vector(7 downto 0);

        r1_load : in STD_LOGIC;
        r2_load : in STD_LOGIC;
        r3_load : in STD_LOGIC;
        r4_load : in STD_LOGIC;
        r1_sel : in STD_LOGIC;
        r2_sel : in STD_LOGIC;
        r3_sel : in STD_LOGIC;
```

```
mux_starting_credibility : in STD_LOGIC;
mux_addr : in STD_LOGIC;
mux_data_credibility : in STD_LOGIC;

is_data_zero : out STD_LOGIC;
is_credibility_zero : out STD_LOGIC
);
end project_reti_logiche;
```

2.1.2 Segnali del Datapath

- **i_clk** : segnale di clock per la sincronizzazione **CLK**;
- **i_rst** : segnale di reset **RST** asincrono;
- **i_start** : segnale di **START** per iniziare l'elaborazione;
- **i_add** : segnale di **ADD** che indica l'indirizzo da cui iniziare l'elaborazione;
- **i_k** : segnale per indicare quante parole $\{W_i\}_{i=0}^{K-1}$ sono da elaborare;
- **is_starting_k_zero** : segnale che indica se il valore fornito in ingresso di **i_k** è nullo;
- **is_k_zero** : segnale che indica se è terminato il numero di parole da dover elaborare;
- **o_mem_addr** : indirizzo a cui il modulo vuole accedere;
- **i_mem_data** : input fornito dalla memoria al modulo;
- **o_mem_data** : output fornito alla memoria;
- **rx_load** : segnale utilizzato per salvare nel registro $x \in \{1, 2, 3, 4\}$ l'input fornito;
- **ry_sel** : segnale di selezione del multiplexer $y \in \{1, 2, 3\}$;
- **mux_starting_credibility** : segnale di selezione del multiplexer per il valore di credibilità massimo o minimo;
- **mux_addr** : segnale di selezione del multiplexer per selezionare l'indirizzo in da fornire in uscita;
- **mux_data_credibility** : segnale di selezione del multiplexer per fornire in uscita il dato o il valore di credibilità;
- **is_data_zero** : segnale per determinare se il valore letto in ingresso è nullo;
- **is_credibility_zero** : segnale per determinare se il valore di credibilità è nullo.

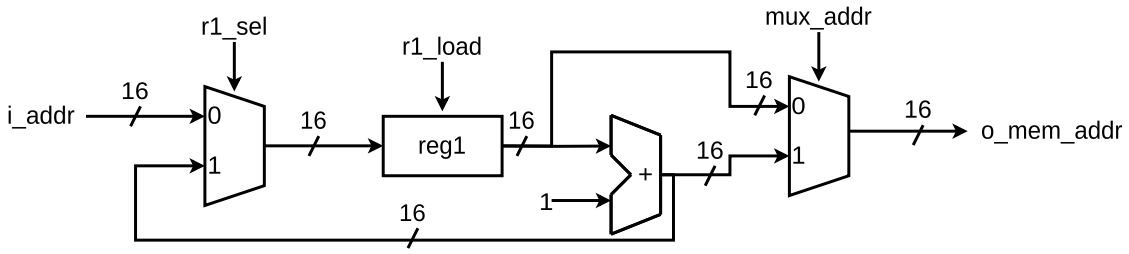


Figure 2.1: Modulo per la gestione degli indirizzi

2.1.3 Modulo per la gestione degli indirizzi

Questo modulo (vedi *Figure 2.1*) ha il compito di salvare l'indirizzo **ADD** nel registro `reg1` e, una volta acquisito, utilizza un sommatore per calcolare l'indirizzo successivo a quello memorizzato. Il primo multiplexer può selezionare quest'ultimo e porlo in ingresso al registro, mentre il segnale `r1_load` consente il salvataggio in `reg1` il valore in ingresso. In uscita, il modulo genera il segnale determinato dal secondo multiplexer, che può corrispondere all'indirizzo memorizzato o quello successivo.

2.1.4 Modulo per il conteggio delle parole rimanenti

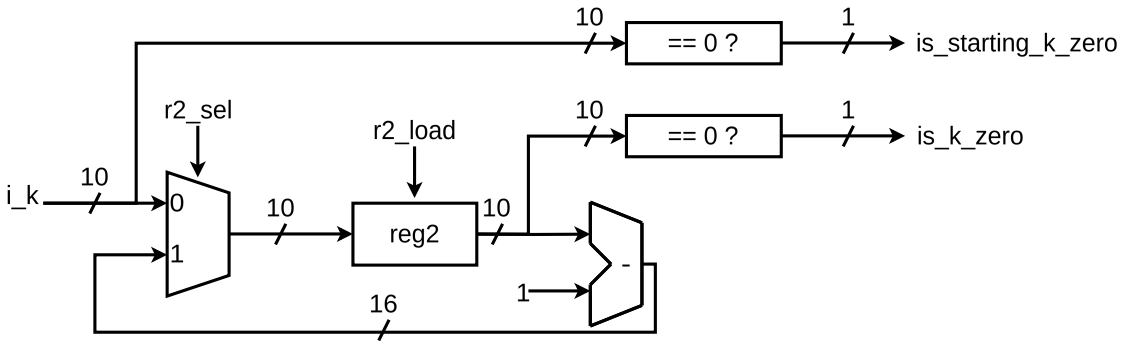


Figure 2.2: Modulo per il conteggio delle parole rimanenti

La funzione di questo componente (vedi *Figure 2.2*) è quella di tenere traccia di quante parole rimangono da elaborare nella sequenza. Nella fase iniziale, il modulo salverà il numero **K** di parole da elaborare nel registro `reg2`. Tramite un blocco sottrattore il valore può essere decrementato di una unità per volta. Inoltre, il modulo presenta in uscita due segnali: `is_starting_k_zero` e `is_k_zero`. Il primo si attiva se il valore **K** è nullo, mentre il secondo si attiva quando il valore memorizzato nel registro, ossia il numero di parole rimanenti da elaborare, `reg2` è nullo.

2.1.5 Modulo per la scrittura in memoria dei dati e della credibilità

Questo modulo è il più complesso dei tre e si suddivide in due sottoparti. La porzione superiore (vedi *Figure 2.3*) presenta il registro `reg4`, dedicato alla memorizzazione

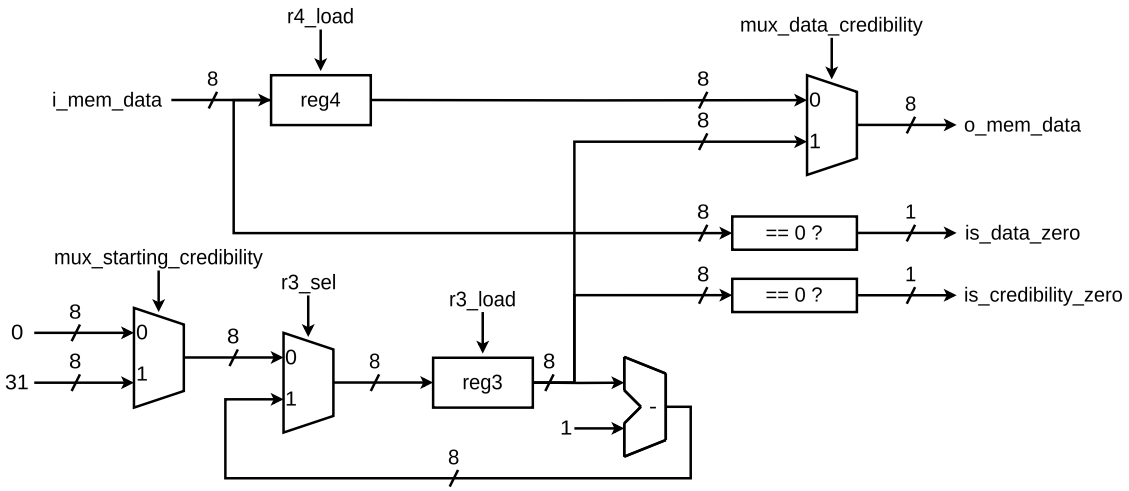


Figure 2.3: Modulo per la scrittura in memoria dei dati e della credibilità

dell'input fornito dalla memoria, rappresentato dalle parole W_i. Sempre in questa sezione si trova anche un blocco combinatorio che si occupa del controllo del segnale in ingresso `i_mem_data`. Se questo è nullo, il segnale `is_data_zero` viene posto alto.

La seconda porzione del modulo utilizza il registro `reg3` per memorizzare un valore di credibilità **C**. Il contenuto del registro può essere inizializzato a partire da due costanti: 0 e 31, le quali rappresentano, rispettivamente, il valore minimo e il valore massimo di credibilità **C**. Un blocco sottrattore consente di decrementare il contenuto del registro `reg3` di un'unità per volta. Un blocco combinatorio monitora il valore memorizzato nel registro. Se questo valore è nullo, pone alto il segnale `is_credibility_zero`.

Le due sezioni del modulo descritto presentano i rispettivi registri (`reg3` e `reg4`) collegati a un multiplexer (vedi *Figure 2.3*), che consente di scrivere in memoria il contenuto di uno dei due registri. Il registro sorgente è selezionato tramite il segnale di comando `mux_data_credibility`.

2.2 Automa a stati finiti

Il secondo macro-componente del progetto è l'**automa a stati finiti** (Finite State Machine - FSA), realizzato come una *macchina di Moore*[1].

L'automa ha il compito di coordinare i vari componenti del **datapath** assicurando che eseguano le loro funzioni in modo corretto e conforme alla specifica precedentemente descritta.

L'**automa a stati finiti** implementato si caratterizza per:

- un insieme di 13 stati;
- una funzione di transizione δ, responsabile del passaggio tra stati consecutivi;
- la funzione d'uscita λ, dedicata alla generazione dei segnali in uscita.

[1]Una macchina di Moore è un automa a stati finiti in cui le uscite sono determinate in funzione dei soli stati correnti

Nel linguaggio di descrizione dell'hardware utilizzato (**VHDL**) le due funzioni, δ e λ, sono state implementate mediante due `process`. La funzione di transizione determina il prossimo stato (`next_state`) in base allo stato corrente (`cur_state`) e ai segnali in ingresso all'automa. Un terzo `process` ha il compito di aggiornare `cur_state` assegnandogli il valore di `next_state` a ogni ciclo di clock **CLK**.

2.2.1 Diagramma degli stati

Una rappresentazione schematica dell'automa è mostrata in *Figure 2.4*. Nelle sezioni successive verranno descritte le funzioni associate ai singoli stati.

2.2.2 Reset [S0]

L'automa non è pronto ad eseguire alcuna elaborazione finché non riceve un primo segnale di **RESET**. Ricevuto il segnale, passa allo stato `S0`, nel quale inizializza i registri dei vari moduli del **datapath** a valori nulli e rimane in attesa di un segnale di **START**. Questo è anche lo stato finale di una elaborazione completa, nel quale la macchina attende un segnale di **START** per avviare una nuova elaborazione.

(**NOTA:** si ricorda che il segnale di **RESET** non è strettamente necessario per eseguire elaborazioni successive alla prima.)

δ : `S0` $\rightarrow$ `S1`, se `i_start = '1'` **and** `i_rst = '0'`;

2.2.3 Inizializzazione [S1]

Una volta ricevuto un segnale di **START**, l'automa entra nello stato `S1`, durante il quale invia i segnali al **datapath** per eseguire le operazioni successive:

- salva l'indirizzo **ADD** nel registro `reg1`;
- salva il valore **K** nel registro `reg2`;
- salva in `reg3` il valore minimo di credibilità, cioè 0.

Per passare allo stato successivo, l'automa deve verificare se il valore **K** di parole da analizzare è nullo. Questo controllo avviene tramite il segnale `is_starting_k_zero`. Se il segnale è alto, la macchina transita nello stato `S2`; se è basso, passa allo stato `S3`.

δ : `S1` $\rightarrow$ `S2`, se `is_starting_k_zero = '1'`;
δ : `S1` $\rightarrow$ `S3`, se `is_starting_k_zero = '0'`;

2.2.4 Termine elaborazione [S2]

L'automa si trova in questo stato al termine di un'elaborazione e vi rimane finché il segnale di **START** non viene posto a zero. Durante questo periodo, il segnale **DONE** è impostato a uno per segnalare il completamento dell'elaborazione.

δ : `S2` $\rightarrow$ `S2` , se `i_start = '1'`;
δ : `S1` $\rightarrow$ `S0`, se `i_start = '0'`;
λ : `o_done = '1'`;

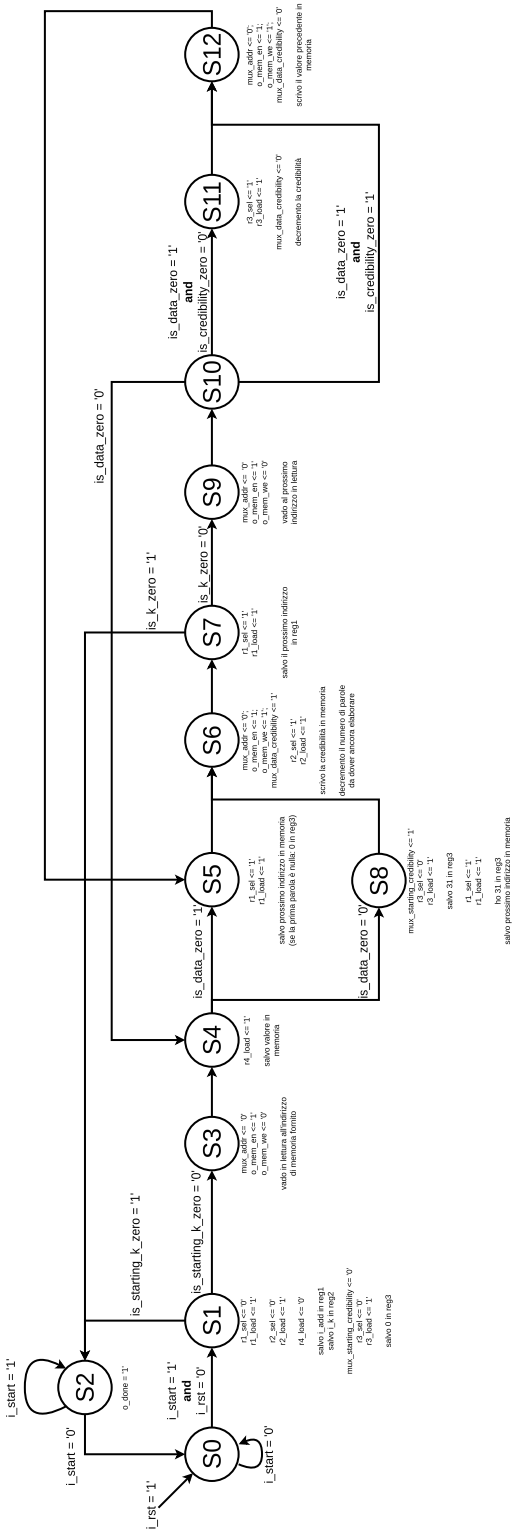


Figure 2.4: Diagramma degli stati della macchina implementata

2.2.5 Inizio elaborazione [S3, S4]

Se il numero di parole da elaborare, **K**, è non nullo, la macchina transita nello stato `S3`, dove richiede alla memoria il valore salvato all'indirizzo **ADD**. Al ciclo di clock **CLK** successivo, la funzione di trasferimento δ porta la macchina nello stato `S4`, nel quale il valore richiesto viene memorizzato nel registro `reg4`. In questa fase viene controllato se il valore letto è nullo tramite il segnale `is_data_zero`, e la macchina decide come procedere l'elaborazione in base a questo controllo.

δ : `S4` $\rightarrow$ `S5` , se `is_data_zero = '1'`;

δ : `S4` $\rightarrow$ `S8`, se `is_data_zero = '0'`;

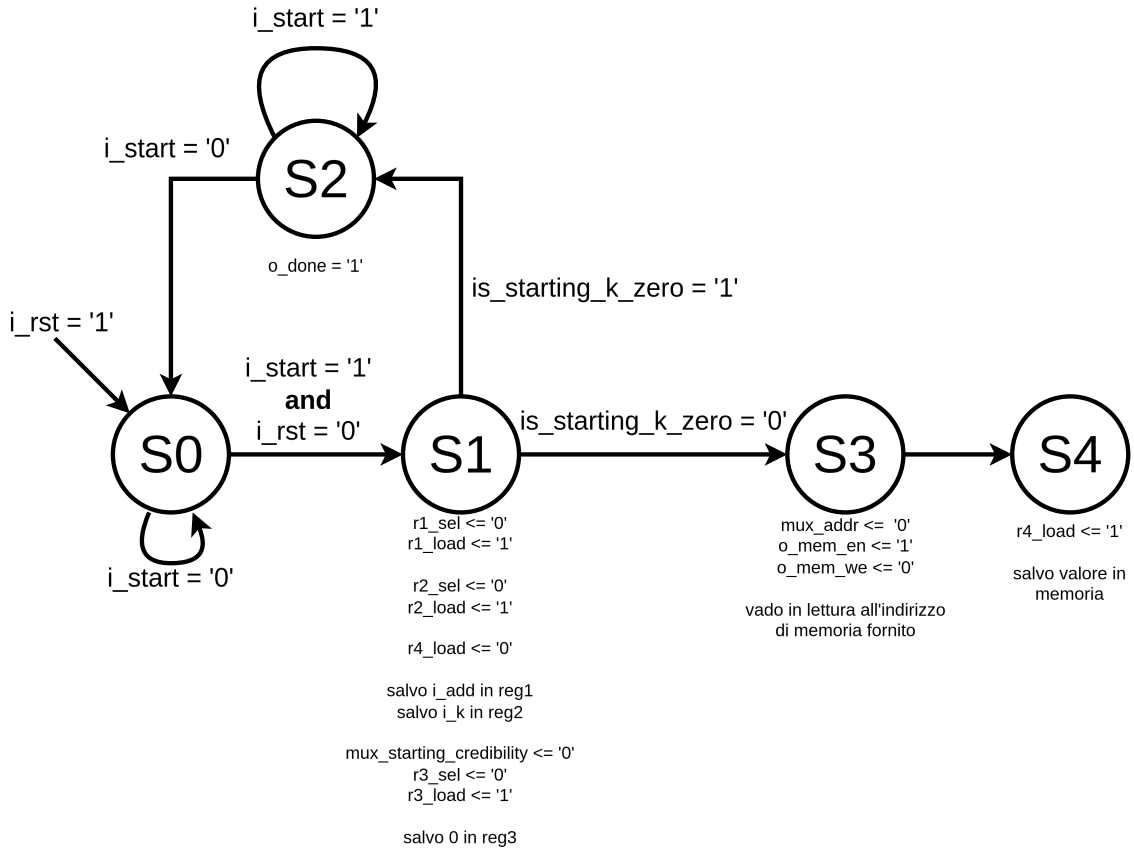


Figure 2.5: Diagramma degli stati di Inizializzazione, Termine elaborazione ed Inizio elaborazione

2.2.6 Incremento indirizzo per parola nulla [S5]

Se la parola W_i, letta nello stato precedente, è nulla, la macchina incrementa l'indirizzo salvato nel registro `reg1`, al fine di puntare la cella di memoria in cui bisogna scrivere la credibilità.

NOTA: si sottolinea che, nel caso in cui la parola letta sia la prima (W_0), il valore di credibilità salvato nel registro `reg3` è 0, poiché rimane invariato dopo l'inizializzazione nello stato `S1`.

δ : `S5` $\rightarrow$ `S6`;

2.2.7 Parola non nulla [S8]

Se la parola W_i, letta nello stato precedente, è non nulla, la macchina salva il valore di credibilità massima (31) nel registro `reg3` e, contemporaneamente, incrementa il valore dell'indirizzo salvato nel registro `reg1`.

δ : S8 $\rightarrow$ S6;

2.2.8 Scrittura della credibilità [S6, S7]

Nello stato S6, la macchina utilizza l'indirizzo salvato in `reg1` per scrivervi in memoria il valore di credibilità C_i, memorizzato nel registro `reg3`. Al ciclo di clock **CLK** successivo, la funzione di transizione δ porterà la macchina nello stato S7, dove incrementerà l'indirizzo salvato nel registro `reg1`. Questa azione conclude la manipolazione della parola W_i e prepara la macchina per la lettura della parola successiva.

δ : S6 $\rightarrow$ S7;

δ : S7 $\rightarrow$ S9;

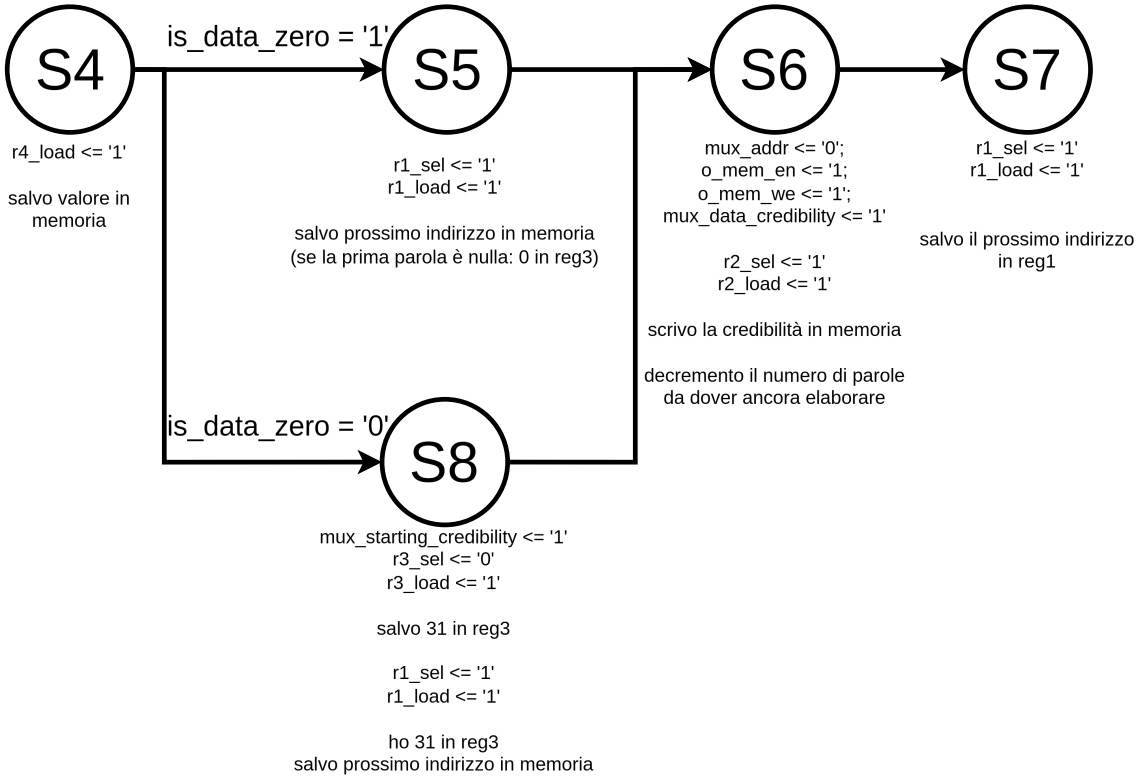


Figure 2.6: Diagramma degli stati per la scrittura della credibilità

2.2.9 Lettura di una nuova parola [S9, S10]

Nello stato S9, la macchina utilizza il registro `reg1` per richiede alla memoria la nuova parola da elaborare. Successivamente, nello stato S10, verifica se la parola W_i è nulla tramite il segnale `is_data_zero`. La transizione allo stato successivo dipende dal valore della parola e dal segnale `is_credibility_zero`, che indica se il valore di credibilità salvato nel registro `reg3` è nullo.

δ : S9 $\rightarrow$ S10;
δ : S10 $\rightarrow$ S4, se is_data_zero= '0';
δ : S10 $\rightarrow$ S11, se is_data_zero= '1' **and** is_credibility_zero= '0';
δ : S10 $\rightarrow$ S12, se is_data_zero= '1' **and** is_credibility_zero= '1';

2.2.10 Credibilità rimanente non nulla [S11]

La macchina si trova nello stato S11 quando il valore di credibilità rimanente, memorizzato in reg3, è strettamente maggiore di zero. In questo stato, il valore viene decrementato di un'unità prima che l'automa transiti allo stato successivo.

δ : S11 $\rightarrow$ S12;

2.2.11 Sovrascrittura parola nulla [S12]

Se in precedenza la macchina ha incontrato una parola nulla $\{W_i = 0\}_{i=1}^{K-1}$, questa viene sovrascritta con la parola letta precedentemente, memorizzata nel registro reg4.

δ : S12 $\rightarrow$ S5;

NOTA: si evidenzia che la sovrascrittura delle parole nulle avviene esclusivamente per parole successive alla prima.

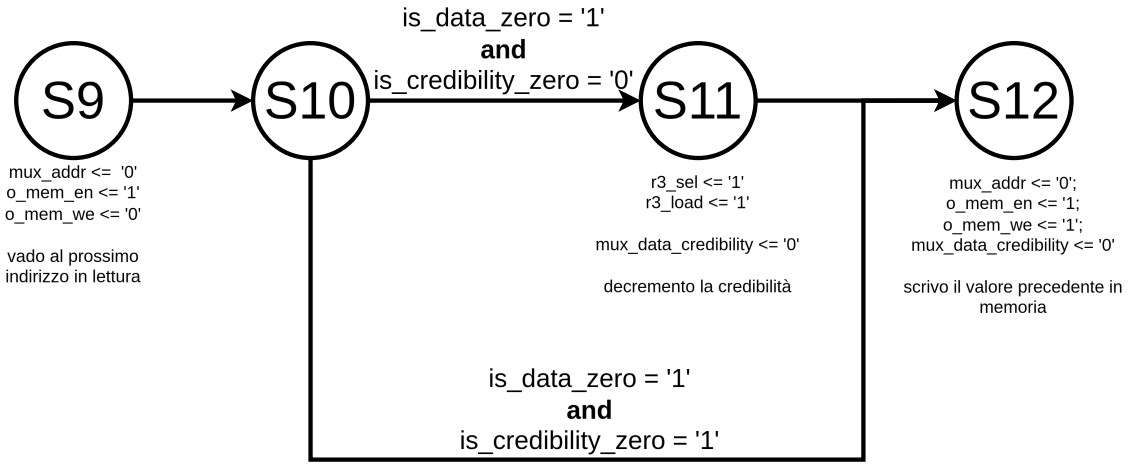


Figure 2.7: Diagramma degli stati per il decremento della credibilità e la sovrascrittura delle parole nulle

3. Risultati Sperimentali

Il componente è stato sottoposto a una serie di test bench finalizzati a verificare la conformità del suo comportamento alle specifiche definite. Particolare attenzione è stata rivolta alla gestione dei possibili casi limite, al fine di assicurare il corretto funzionamento.

3.1 Sintesi

Il modulo hardware è risultato correttamente sintetizzabile. Tramite il comando: `report_utilizazion` si può osservare la seguente tabella:.

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   75 |     0 |    134600 |  0.06 |
|   LUT as Logic          |   75 |     0 |    134600 |  0.06 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   46 |     0 |    269200 |  0.02 |
|   Register as Flip Flop |   46 |     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 |
+-------------------------+------+-------+-----------+-------+
```

Da questo report si evince che nel modulo non sono stati inferiti latch, ma esclusivamente flip-flop.

Tramite il comando: `report_timing` si può ottenere la seguente informazione:

```
Timing Report

Slack (MET) :           16.866ns  (required time - arrival time)
```

Il modulo hardware presenta uno slack di 16.866ns su un ciclo di clock di 20ns, indicando che il componente realizzato utilizza meno del 16% del tempo a disposizione per effettuare un'elaborazione.

3.2 Simulazioni

Per valutare il funzionamento del modulo hardware, sono stati eseguiti numerosi test bench, i quali sono stati simulati sia in modalità "Behavioral" sia in modalità "Post-synthesis". Di seguito vengono presentate alcune delle casistiche utilizzate per i test.

3.2.1 Test bench d'esempio

Tra i vari test bench forniti come esempio, alcuni di essi sono progettati per riprodurre un comportamento generale della macchina, fornendo un numero arbitrario di parole da dover elaborare. Tra questi, vi sono anche test bench che presentano scenari più complessi, i quali valutano il comportamento della macchina in condizioni limite di esecuzione.

In particolare, un test verifica come viene gestito il valore di credibilità, assicurandosi che non possa assumere valori negativi, mentre un altro analizza il caso in cui il primo valore sia nullo, valutandone la corretta gestione.

3.2.2 Test per valori limite di K

Sono stati realizzati dei test bench per verificare il corretto funzionamento del modulo nei casi in cui il numero di parole da dover elaborare **K** assume valore minimo (0) e massimo (1023).

3.2.3 Test con reset asincrono

Sono stati sviluppati dei test in cui viene fornito un segnale di **RESET** durante un'elaborazione, al fine di verificare che il modulo interrompa l'elaborazione corrente e che la macchina torni nello stato `S0`. Successivamente, viene fornita una una nuova richiesta di elaborazione per assicurarsi che l'esecuzione seguente venga eseguita correttamente.

Durante lo sviluppo è emersa una problematica legata al reset asincrono. In particolare, se il segnale di **RESET** veniva attivato durante un'elaborazione, mentre il segnale di **START** rimaneva alto, la macchina manifestava un comportamento anomalo. Per risolvere questo problema è stata introdotta una condizione nella transizione dallo stato `S0` allo stato `S1`, imponendo che **RESET** sia nullo per poter consentire l'avvio di una nuova elaborazione.

3.2.4 Test con molteplici elaborazioni

Sono stati realizzati dei test bench per valutare il corretto funzionamento del modulo nei casi in cui venga richiesto di eseguire più elaborazioni successive.

4. Conclusioni

Il componente risulta sintetizzabile e simulabile, sia in modalità pre-sintesi che post-sintesi (Behavioral e Functional), mostrando un comportamento conforme alle specifiche in tutti i test effettuati.

4.1 Ottimizzazioni

Il modulo HW presentato è stato ottenuto come risultato di una raffinazione di una versione più elaborata, costituita da un'automa a 26 stati. La soluzione precedente presentava diversi stati indistinguibili fra loro, che sono stati accorpati, consentendo una riduzione del numero di stati pari al 50%.

Nonostante il modulo risulti corretto, si ritiene che possano essere apportate ulteriori ottimizzazioni, sia a livello di **datapth** sia nell'**automa a stati finiti**. In particolare, una riduzione ulteriore del numero di stati potrebbe essere ottenuta ottimizzando i moduli del **datapath**. L'utilizzo di contatori per gestire l'indirizzo di memoria, il valore di credibilità e il numero di parole da elaborare permetterebbe di semplificare ulteriormente l'automa, riducendo il numero di stati necessari.