POLITECNICO
MILANO 1863

Prova finale
Progetto di Reti Logiche

Matteo Gallo
Codice Persona: [Redacted]
Matricola: [Redacted]

Corso di Reti Logiche
Anno accademico 2023/2024
Professore: Fabio Salice

Indice

1 Introduzione **2**

2 Architettura **3**
2.1 Algoritmo . . . 3
2.2 Macchina a stati finiti . . . 3
`INIT` . . . 4
`READ_EVEN` . . . 4
`WAIT_READ` . . . 4
`WRITE_EVEN & WRITE_ODD` . . . 4
Ciclo e Termine . . . 4
2.3 Implementazione . . . 5
`fsm` . . . 5
`counter` . . . 6
`credibility` . . . 6
`last_value` . . . 6
`mux` . . . 7

3 Risultati sperimentali **7**
3.1 Sintesi . . . 7
3.1.1 Timing Report . . . 8
3.2 Testing . . . 8
3.2.1 Test sui Componenti . . . 8
3.2.2 Test sul Progetto . . . 9

4 Conclusioni **11**

1 Introduzione

Si vuole implementare un modulo hardware descritto in VHDL che si interfacci con memoria e che abbia 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 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;
```

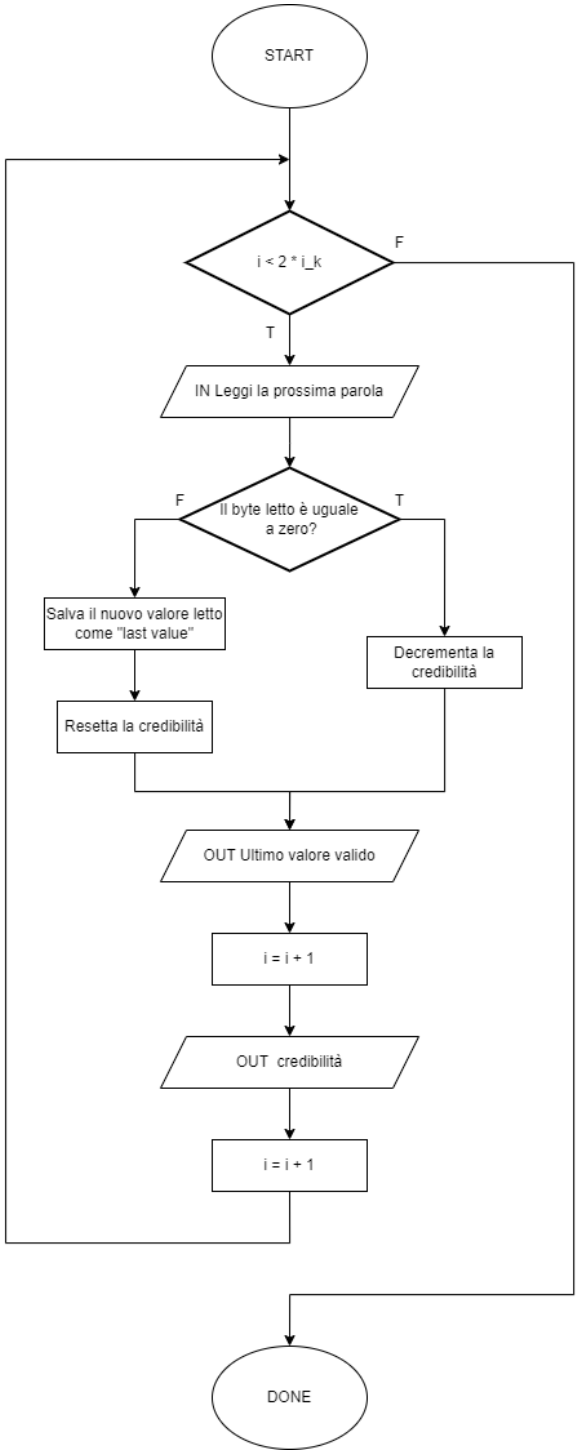
Il sistema deve leggere un messaggio composto da una sequenza di `i_k` parole i cui valori sono compresi tra 0 e 255. Il valore 0 deve essere considerato come un'informazione che indica "il valore non è specificato". La sequenza da elaborare è memorizzata a partire da un indirizzo specificato (`i_add`), con ogni parola occupante 2 byte. Il modulo deve completare la sequenza sostituendo gli zeri nel primo byte con l'ultimo valore valido letto (non zero) precedente e appartenente alla sequenza. Inoltre, deve inserire un valore di "credibilità" nel byte mancante per ogni valore della sequenza. Questo valore di credibilità è inizializzato a 31 ogni volta che un valore della sequenza è non zero e viene decrementato rispetto al valore precedente ogni volta che si incontra uno zero. Il valore di credibilità associato ad ogni parola viene memorizzato nella memoria nel byte successivo. Quando C raggiunge il valore 0, non viene ulteriormente decrementato.

Nel dettaglio il componente riceverà un segnale `i_start` per iniziare l'elaborazione e dovrà comunicare alla memoria la soluzione, usando gli opportuni segnali. Una volta terminata l'elaborazione della sequenza, il componente dovrà attivare il segnale `o_done` per comunicarlo al processore.

2 Architettura

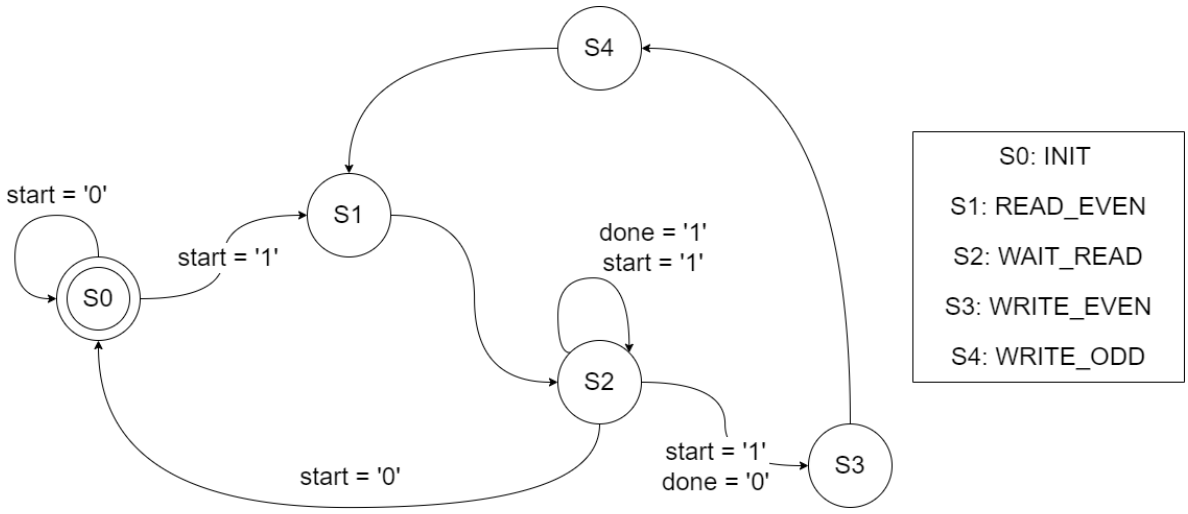
2.1 Algoritmo

L'algoritmo prevede un ciclo che legge una parola dalla sequenza, la sostituisce con l'ultimo valore letto e scrive la credibilità nel byte successivo. Se il byte letto è diverso da zero, l'ultimo valore valido viene aggiornato e la credibilità resettata a 31. Al contrario, se il byte letto è zero, l'ultimo valore valido rimane invariato e la credibilità viene decrementata. Durante ogni iterazione del ciclo, viene scritto l'ultimo valore valido al posto del valore letto, mentre la credibilità viene scritta nel byte successivo. Il contatore del ciclo viene incrementato di uno ad ogni operazione di scrittura. I valori da leggere e scrivere si trovano nella posizione `i_add + 8 * i`, dove `i` è il contatore del ciclo. Il ciclo continua finché il contatore non raggiunge il valore `2 * i_k`.



2.2 Macchina a stati finiti

Sotto sono mostrate il grafico della FSM e le relative tabelle degli stati e uscite. Gli ingressi sono rispettivamente `start` e `done`, mentre le uscite corrispondono a `en_reg | sel_mux_en_cred | mem_we | mem_en | done`. Le condizioni di indifferenza sono presenti quando i valori delle uscite non influenzano il comportamento della macchina, ma vengono comunque impostati a `'0'` per evitare la formazione di Latch.



	00	**01**	**11**	**10**
S0	S0	S0	S1	S1
S1	S2	S2	S2	S2
S2	S0	S0	S2	S3
S3	S4	S4	S4	S4
S4	S1	S1	S1	S1

	00	**01**	**11**	**10**
S0	- - - 00	- - - 00	- - - 00	- - - 00
S1	00010	00010	00010	00010
S2	- - - 00	- - - 01	- - - 01	10010
S3	00110	00110	00110	00110
S4	01110	01110	01110	01110

`INIT`

Lo stato di reset della macchina a stati è `INIT`.

`READ_EVEN`

Una volta ricevuto il segnale `start`, la FSM passa allo stato `READ_EVEN`, il quale richiede alla memoria di leggere il primo byte.

`WAIT_READ`

A causa dei ritardi, la memoria impiegherà più di un ciclo di clock per restituire il byte letto. Questo fa sì che, una volta raggiunto il successivo fronte di clock, la macchina a stati non sia ancora pronta per continuare con l'elaborazione del dato, poiché non ha ancora ricevuto il dato stesso. Anche se il ritardo è minimo, si è ritenuto necessario considerare l'operazione di lettura come se impiegasse 2 cicli di clock. Pertanto, la soluzione adottata è ritardare la FSM facendola passare da uno stato ausiliario `WAIT_READ`, in modo che inizi ad elaborare il dato solo al secondo fronte di clock. Alla fine di `WAIT_READ`, avviene l'effettiva lettura del byte, e quindi il segnale `en_reg` viene settato a `'1'`, comunicando al registro che contiene l'ultimo valore valido `last_value` che può considerare di salvare un nuovo valore. Questo avviene solo se il valore letto è diverso da zero, come specificato dall'algoritmo. In tal caso, viene anche resettato il valore di credibilità.

`WRITE_EVEN & WRITE_ODD`

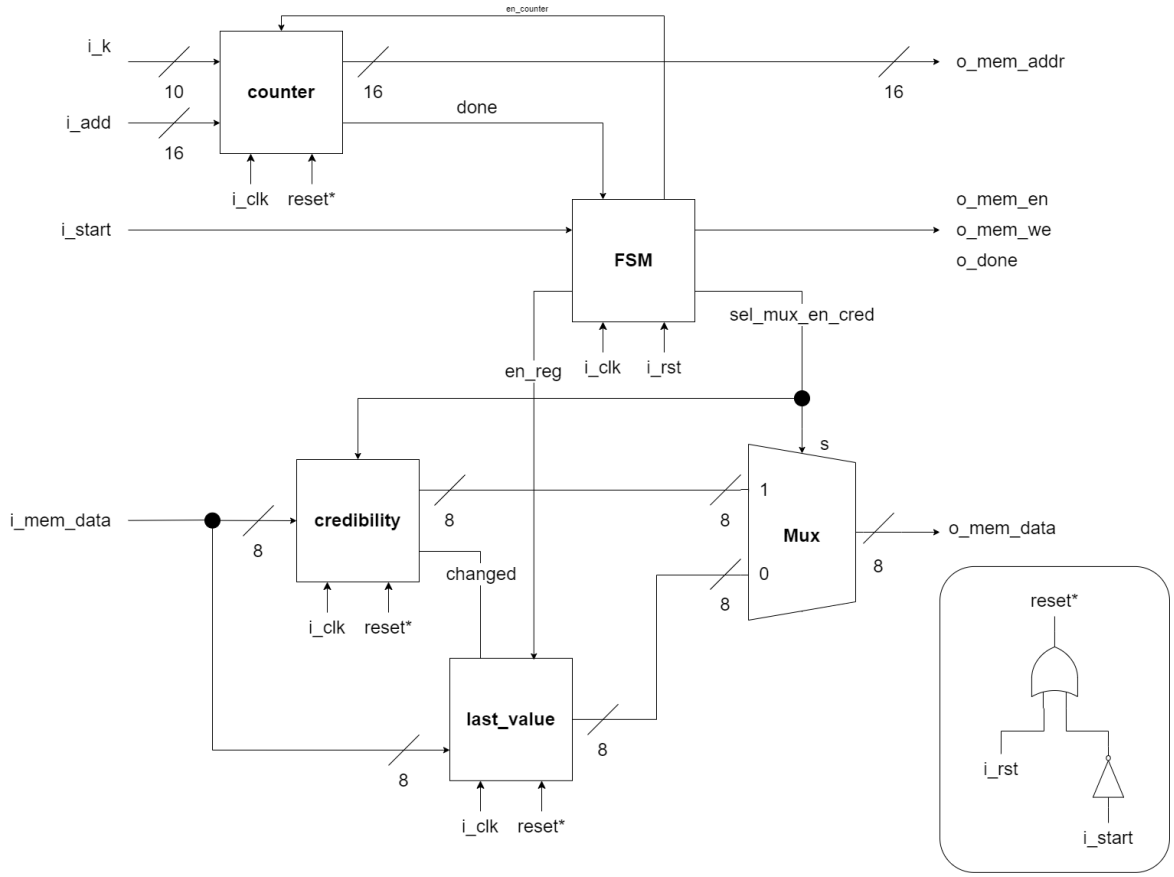
Il prossimo stato è `WRITE_EVEN`, il quale comunica alla memoria di scrivere l'ultimo valore valido letto. In modo simile opera l'ultimo stato `WRITE_ODD`, con la differenza che scrive il valore di credibilità corrente. I due stati differiscono unicamente per il segnale di selezione inviato al multiplexer `sel_mux_en_cred`, che seleziona il valore corretto da inviare alla memoria in base allo stato corrente. Tale selezione coincide anche con il segnale di enable per permettere il decremento della credibilità, la quale avviene solo nello stato `WRITE_ODD`, cioè dopo aver scritto il valore di credibilità.

Ciclo e Termine

Dopo `WRITE_ODD`, la FSM ritorna a `READ_EVEN`, completando un ciclo, che termina solo quando vengono letti `K` byte. In questo caso, il registro contenente il contatore dei byte letti comunica alla macchina tramite un segnale interno `done`. La FSM, una volta ricevuto questo segnale, ritarda di un ciclo di clock il reale segnale `done` da comunicare al processore, per dare alla memoria il tempo di salvare l'ultimo valore scritto. Siccome l'ultimo valore scritto è la credibilità, il segnale `done` viene comunicato una volta raggiunto `WAIT_READ`.

Infine la FSM rimane in attesa che il segnale `start` ritorni a `'0'`, e in quel caso ritorna nello stato `INIT` in attesa di un'altra esecuzione.

2.3 Implementazione



`fsm`

```
entity fsm is
    port (
        i_start : in std_logic;
        i_done : std_logic;

        clk, rst : in std_logic;

        o_en_reg : out std_logic;
        o_sel_mux : out std_logic;
        o_mem_we : out std_logic;
        o_mem_en : out std_logic;
        o_done : out std_logic
    );
end fsm;
```

La macchina a stati è definita da due processi: uno che sincronizza con il segnale di clock, gestisce lo stato di reset e definisce le funzione degli stati successivi. Il secondo è un processo per formalizzare gli output. Oltre ai segnali di clock e di reset, la macchina a stati riceve in input il segnale `i_start` e `i_done`, quest'ultimo proveniente dal componente `counter` una volta che la sequenza è completata. Gli output includono:

- `o_en_reg`: impostato su `'1'` se il registro contentente l'ulitmo valore valido *può* essere modificato, altrimenti su `'0'`.
- `o_sel_mux_en_cred`: un unico segnale utilizzato sia per selezionare il multiplexer sia da segnale `enable` per il registro di `credibility`, consentendo il decremento della credibilità.
- `o_mem_we`: impostato su `'1'` per scrivere in memoria, su `'0'` per leggere.
- `o_mem_en`: impostato su `'1'` per comunicare con la memoria, altrimenti su `'0'`.
- `o_done`: impostato su `'1'` quando il componente ha completato l'esecuzione, altrimenti su `'0'`.

`counter`

```
entity counter is
    port (
        i_offset : in std_logic_vector(15 downto 0);
        i_len : in std_logic_vector(9 downto 0);
        i_enable : in std_logic;
        clk, rst : in std_logic;
        o_curr : out std_logic_vector(15 downto 0);
        o_done : out std_logic
    );
end counter;
```

Il compito principale del componente è quello di determinare l'indirizzo di memoria attuale da cui leggere o scrivere. Per fare ciò, utilizza un contatore che viene incrementato ogni volta che il segnale di abilitazione è attivo. L'output del componente è dato dalla somma tra il valore del contatore e l'offset iniziale della memoria. Quando il contatore raggiunge il valore finale, determinato dal segnale `i_len`, il componente attiva il segnale `o_done`.

`credibility`

```
entity reg_credibility is
    port(
        enable : in std_logic;
        restart : in std_logic;
        clk, rst : in std_logic;
        output : out std_logic_vector(7 downto 0)
    );
end reg_credibility;
```

Viene utilizzato un registro per memorizzare la credibilità associato al segnale di output. Inoltre, il componente si occupa anche di decrementare la credibilità ogni volta che il segnale di abilitazione è `'1'`. Se il segnale `restart` è attivato, la credibilità viene ripristinata al suo valore massimo, ovvero `31`. Simile a quanto avviene con il segnale di reset, con la differenza che la credibilità viene impostata a `0`.

`last_value`

```
entity reg_last_value is
    port(
        input : in std_logic_vector(7 downto 0);
        enable : in std_logic;
        clk, rst : in std_logic;
        changed : out std_logic;
        output : out std_logic_vector(7 downto 0)
    );
end reg_last_value;
```

Viene utilizzato un registro per memorizzare l'ultimo valore valido associato al segnale di output. Quando il segnale di abilitazione è `'1'` e il valore in ingresso è diverso da zero, il valore memorizzato viene aggiornato con il valore in ingresso. In tal caso, il segnale `changed` sarà impostato a `'1'` per segnalare a `credibility` di ricominciare il decremento.

`mux`

```
entity multiplexer is
    port (
        in_0 : in std_logic_vector(7 downto 0);
        in_1 : in std_logic_vector(7 downto 0);
        sel : in std_logic;
        output : out std_logic_vector(7 downto 0)
    );
end multiplexer;
```

Il multiplexer viene utilizzato per indirizzare uno dei due ingressi, corrispondenti a `credibility` e `last_value`. Come segnale di controllo viene utilizzato `sel_mux_en_cred` proveniente dalla FSM.

3 Risultati sperimentali

3.1 Sintesi

```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs              |   49 |     0 |          0 |    134600 |  0.04 |
| Slice Registers         |   32 |     0 |          0 |    269200 |  0.01 |
|   Register as Flip Flop |   32 |     0 |          0 |    269200 |  0.01 |
|   Register as Latch     |    0 |     0 |          0 |    269200 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

```
Detailed RTL Component Info :
+---Adders :
     2 Input   16 Bit       Adders := 1
     2 Input   11 Bit       Adders := 1
     2 Input    8 Bit       Adders := 1
+---Registers :
                 11 Bit     Registers := 1
                  8 Bit     Registers := 2
                  1 Bit     Registers := 2
```

Sono stati sintetizzati tre Adder, di cui due sono impiegati dal modulo `counter` per incrementare il valore dell'indirizzo di memoria corrente, e uno per decrementare la credibilità. In alternativa, si sarebbe potuto considerare l'opzione di incrementare direttamente l'indirizzo di partenza, evitando così l'utilizzo di un Adder aggiuntivo da 11 bit. Tuttavia, al fine di mantenere la chiarezza e la comprensibilità del codice, ho scelto di adottare questa soluzione.

I registri creati sono interamente costituiti da Flip-Flop, senza Latch come previsto. Di sotto sono elencati i Flip-Flop utilizzati per verificare che il risulato sia coerente con quanto implementato. In particolare, sono stati impiegati 3 Flip-Flop per rappresentare gli stati della macchina a stati finiti e altri 2 Flip-Flop probabilmente utilizzati per le connessioni tra i componenti:

$+\,11$	*valore corrente indirizzo*
$+\,8$	*reg. credibilita*
$+\,8$	*reg. ultimo valore valido*
$+\,2$	*ausiliari*
$+\,\lceil \log_2 5 \rceil$	*stati della FSM*
$=\,32$	

3.1.1 Timing Report

```
Timing Report

Slack (MET) :             16.171ns  (required time - arrival time)
```

Dal Timing Report si evince che la differenza tra il tempo di clock ($20ns$) e il tempo effettivamente utilizzato per produrre un output è di $16.171ns$, quindi il componente impiega complessivamente $3.829ns$ per eseguire, valore ben al di sotto del limite prestabilito.

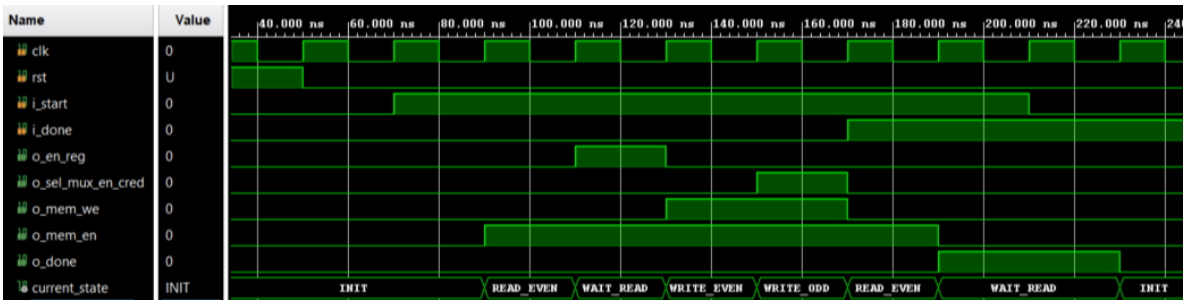
3.2 Testing

3.2.1 Test sui Componenti

I test sui componenti del progetto hanno riguardato principalmente la verifica del corretto funzionamento di ciascun elemento che compone il sistema.

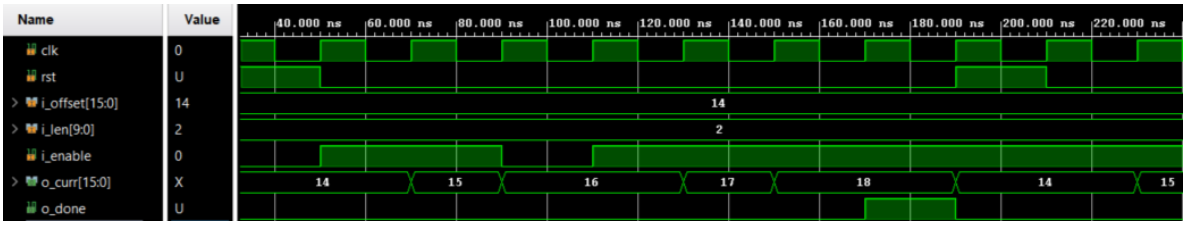
`fsm`

Per il componente simulante la macchina a stati finiti, è stato eseguito un test percorrendo ogni percorso della macchina e confermando che la successione degli stati e gli output corrispondessero con la tabella degli stati.



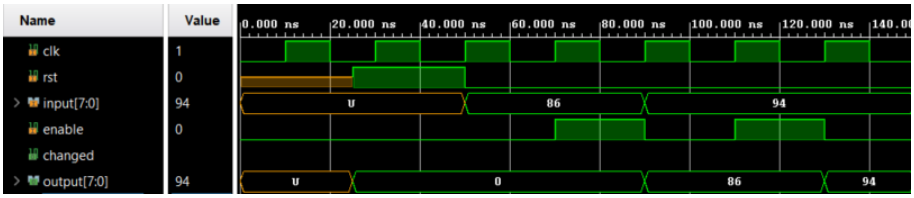
`counter`

Successivamente, sono stati eseguiti test specifici per ciascun elemento del progetto. Ad esempio, per il contatore dell'indirizzo corrente, è stata verificata l'incremento solo quando il segnale `i_enable` era attivo. Inoltre, è stato verificato che il segnale `o_done` venisse attivato una volta raggiunto il termine, garantendo così il corretto funzionamento del contatore.



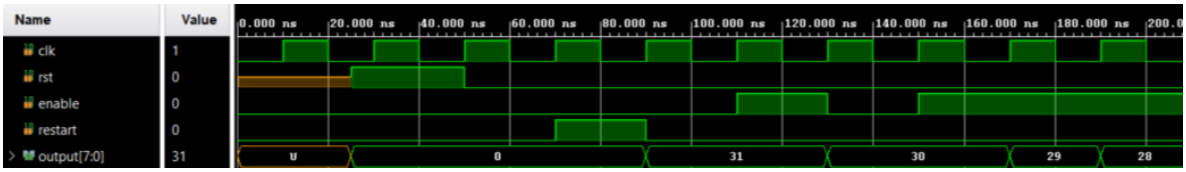
`credibility`

Analogamente, per il registro che memorizza la credibilità, è stato verificato il decremento corretto quando il segnale `enable` era attivo, e il ripristino sia a zero (con il segnale di reset) che al suo valore massimo (con il segnale di restart).



`last_value`

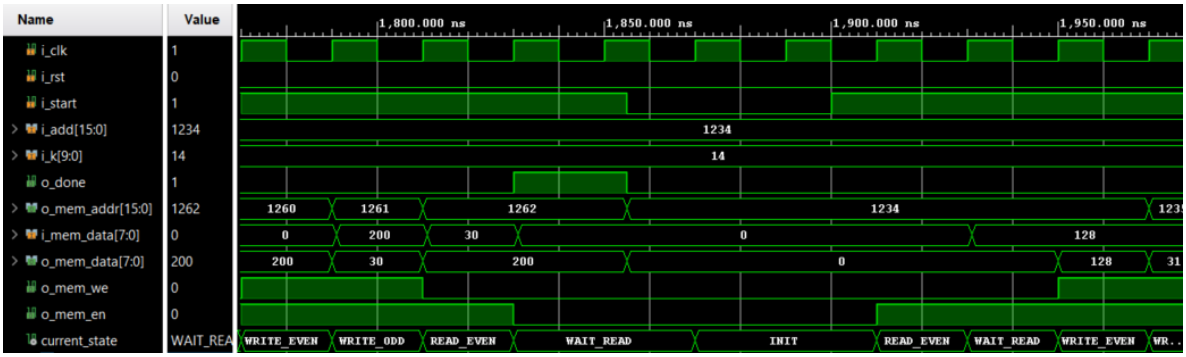
Infine, per il registro che contiene l'ultimo valore valido, è stata verificata la corretta corrispondenza dell'uscita con il valore memorizzato. Si è inoltre accertato che questo venisse modificato solo quando il segnale di abilitazione era attivo e, in caso di modifica, che il segnale `changed` venisse attivato.



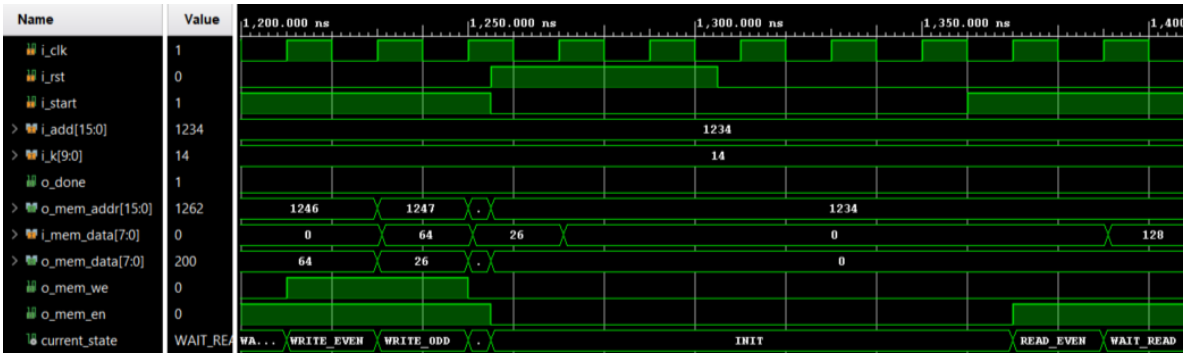
3.2.2 Test sul Progetto

Durante l'esecuzione dei test sul progetto, sono stati eseguiti una serie di test mirati a verificare il corretto funzionamento del componente:

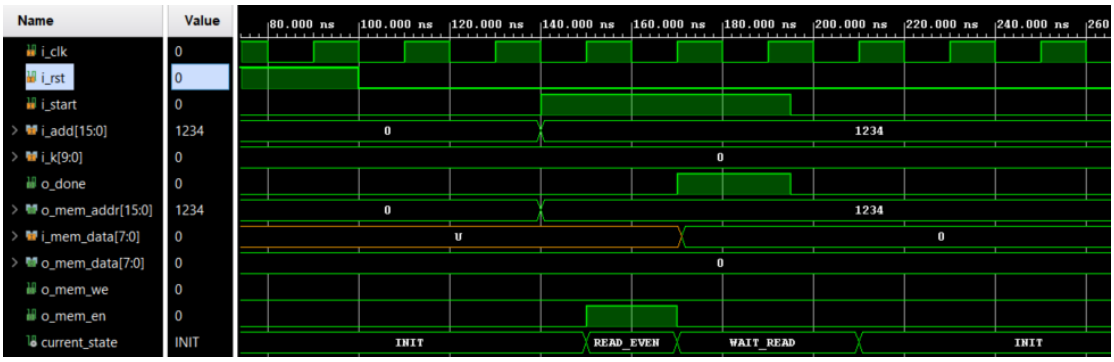
1. Inizialmente, sono state eseguite esecuzioni consecutive, simili a quanto proposto nel test bench fornito dal docente, con la differenza che dopo il primo segnale di `done` l'esecuzione non viene terminata, ma ripetuta una seconda volta per verificare che non comprometta il funzionamento del componente.



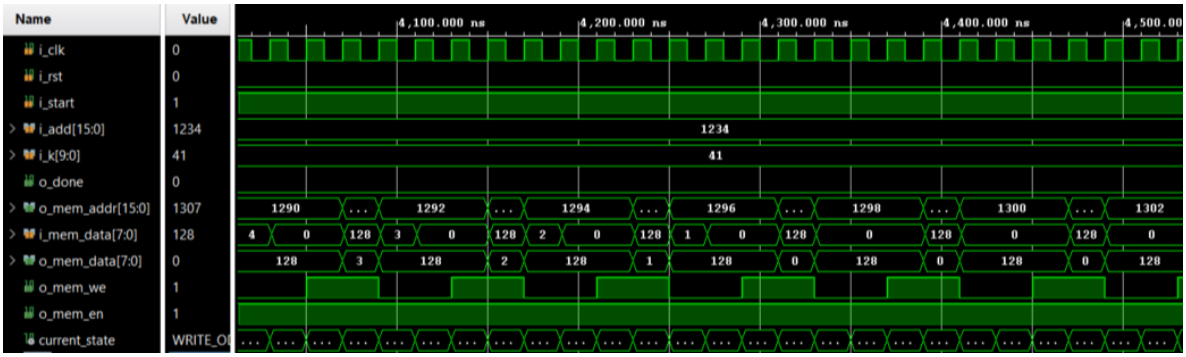
2. Nel test successivo è stato fornito un segnale di reset per permettere al componente di riavviarsi e continuare l'esecuzione senza problemi, garantendo così una corretta gestione delle situazioni di reset durante il funzionamento.



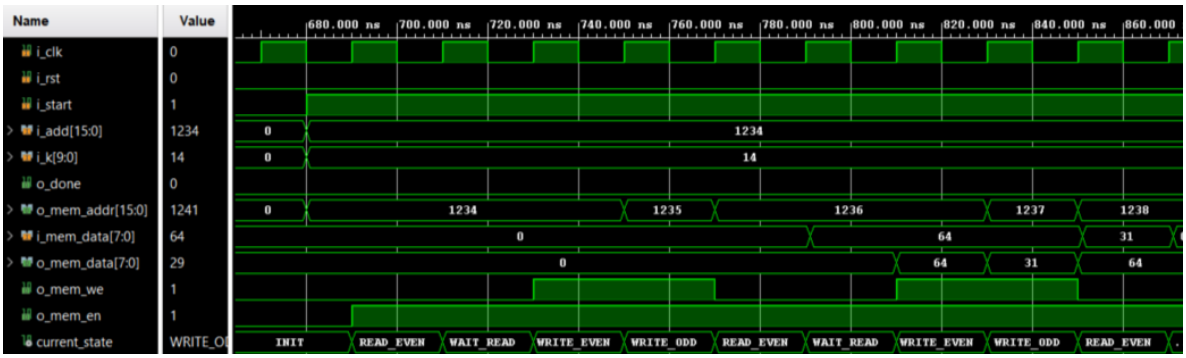
3. Inoltre, è stato eseguito un test specifico per il caso limite in cui il segnale `i_k` è impostato a 0, cioè la lunghezza della parola è nulla. Questo test è stato eseguito per verificare che il componente terminasse immediatamente l'esecuzione senza scrivere sulla memoria, come previsto.



4. Un altro test eseguito è stato quello di decrementare la credibilità a 0, che consisteva nell'eseguire una sequenza di 31+ valori non specificati per decrementare la credibilità a 0 e verificare che rimanesse costante per valori successivi.



5. Infine, viene eseguito un test nel caso in cui il primo valore della sequenza sia zero. In questa situazione, la credibilità deve essere impostata a zero.



4 Conclusioni

Il presente progetto ha rappresentato un'importante opportunità per approfondire la progettazione e l'implementazione di un modulo hardware in VHDL. Attraverso le conoscenze apprese nel corso di Reti Logiche, sono riuscito a realizzare un componente che gestisce efficacemente la sequenza di dati secondo le specifiche richieste. Ritengo importante sottolineare che il processo di sviluppo non si è limitato alla fase di implementazione, ma ha coinvolto anche attività di analisi, progettazione e verifica, per ottenere una soluzione più ottimale, coesa e coerente con quanto visto durante il corso. In definitiva, questo progetto ha rappresentato un'esperienza formativa e stimolante, consentendomi di acquisire nuove conoscenze e competenze che saranno preziose per il mio percorso accademico e professionale futuro.