Relazione di Progetto di Reti Logiche

Gabriele Santandrea

POLITECNICO
MILANO 1863

Anno Accademico: 2023-2024
Codice Persona: 10807746
Docente Referente: Gianluca Palermo

Indice

Introduzione

Specifica di progetto

Il progetto consiste nella modellazione di un circuito a livello *RT*. Esso si interfaccia con una memoria sincrona, sulla quale elabora una sequenza di K *parole* ($0 =< K < 1023$) di modo che a ogni lettura sia valutata se quest'ultima è diversa da zero:

- in caso positivo, allora procede a inserire nello spazio di memoria successivo un valore ben determinato, detto *credibilità*, inizializzato;
- in caso negativo, sovrascrive la cella corrente con il valore dell'ultima *parola* letta e inserisce nello spazio di memoria adiacente un valore di credibilità, decrementato di uno per ogni iterazione successiva compiuta in questo caso ($\forall t(31 > cred >= 0)$).

Definito ADD l'indirizzo della prima *parola*, da cui il modulo inizia a elaborare, allora un possibile stato della memoria prima e dopo l'elaborazione è rappresentato nelle seguenti tabelle.

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
218	0	17	0	0	31
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	0	0	0	0	0

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
218	31	17	31	17	30
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	31	77	30	77	29

Tabella 1: Stato della memoria prima e dopo l'elaborazione di una sequenza generica

Il modulo si occupa di segnalare alla memoria la fine dell'elaborazione.

Un caso particolare dell'elaborazione è dato dalla lettura di uno zero a inizio sequenza: questo comporta una scrittura di zeri in memoria, sino alla lettura di un numero diverso da zero che riporti l'elaborazione a normale regime come mostrato in Tabella 2.

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
0	31	0	2	0	0
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	0	0	12	0	0

ADD	ADD+1	ADD+2	ADD+3	ADD+4	ADD+5
0	0	0	0	0	0
ADD+6	ADD+7	ADD+8	ADD+9	ADD+10	ADD+11
77	31	77	30	77	29

Tabella 2: Stato della memoria prima e dopo l'elaborazione di una sequenza che inizia con uno zero

Interfaccia tra i componenti

La memoria e il modulo comunicano attraverso i segnali specificati in Figura 1. L'interfaccia del modulo è la seguente:

```
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);
        -- Write Enable signal: 0 read - 1 write
        o_mem_we: OUT std_logic;
        -- Enable signal
        o_mem_en: OUT std_logic
    );
END project_reti_logiche;
```

In particolare:

- **i_clk** è il segnale di clock in ingresso;
- **i_rst** è il segnale di reset che inizializza la macchina pronta per ricevere il primo segnale di start. Se il reset è alto allora il segnale d'uscita **o_done** deve essere basso;
- **i_start** è il segnale di start;
- **i_k** è il segnale rappresentante la lunghezza della sequenza;

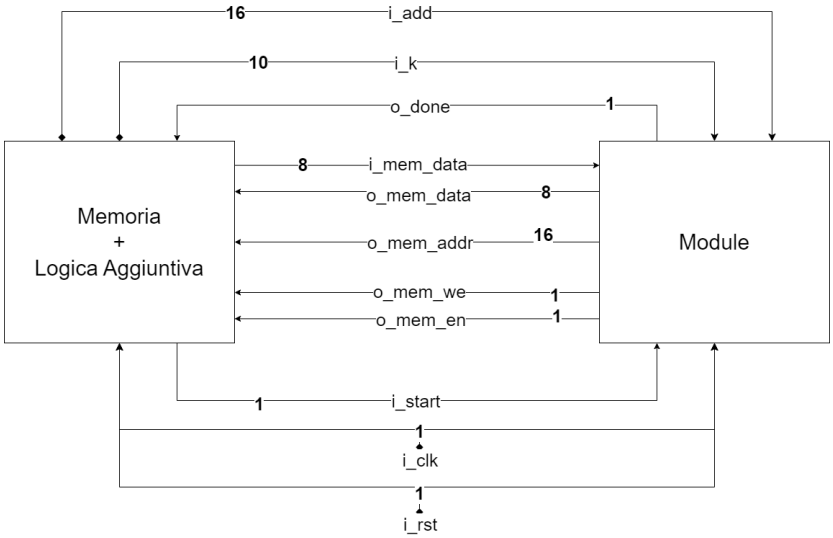


Figura 1: Schematico generale delle connessioni tra la memoria e il modulo progettato

- **i_add** è il segnale che rappresenta l'indirizzo dal quale parte la sequenza ad elaborare[1];
- **o_done** è il segnale di uscita che comunica la fine dell'elaborazione;
- **o_mem_addr** è il segnale di uscita che manda l'indirizzo alla memoria;
- **i_mem_data** è il segnale che arriva dalla memoria e contiene il dato in seguito ad una richiesta di lettura;
- **o_mem_data** è il segnale che va verso la memoria e contiene il dato che verrà successivamente scritto;
- **o_mem_en** è il segnale di *enable* che permette di comunicare con la memoria (sia in lettura sia in scrittura);
- **o_mem_we** è il segnale di *write enable*: esso permette di accedere alla memoria in scrittura se alto (`o_mem_we = '1'`) oppure in lettura se basso (`o_mem_we = '0'`).

[1]Per convenzione è chiamato ADD

Capitolo 1

Architettura del modello

1.1 Contatore

Poiché il modulo è stato sviluppato tramite un approccio strutturale bottom-up allora si presenta innanzitutto il *contatore*.
La funzionalità di questo componente è dovuta alla presenza di ingressi costanti nei sopra-moduli che ne fanno uso. Ciò permette di valutare ogni condizione di conteggio utilizzando il contatore come *offset* rispetto al segnale costante in entrata.

È costituito da un multiplexer controllato dal segnale **counter-init** la cui funzione è di inizializzare il registro ogni qualvolta necessario (`counter_init <= '1'`); altrimenti seleziona il segnale che porta il valore corrente incrementato di uno (`counter_init <= '0'`).

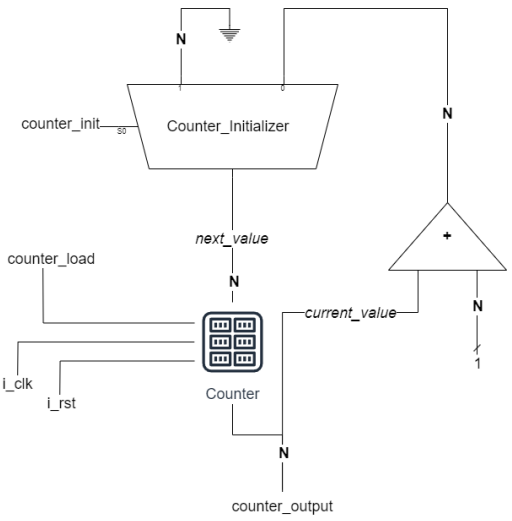


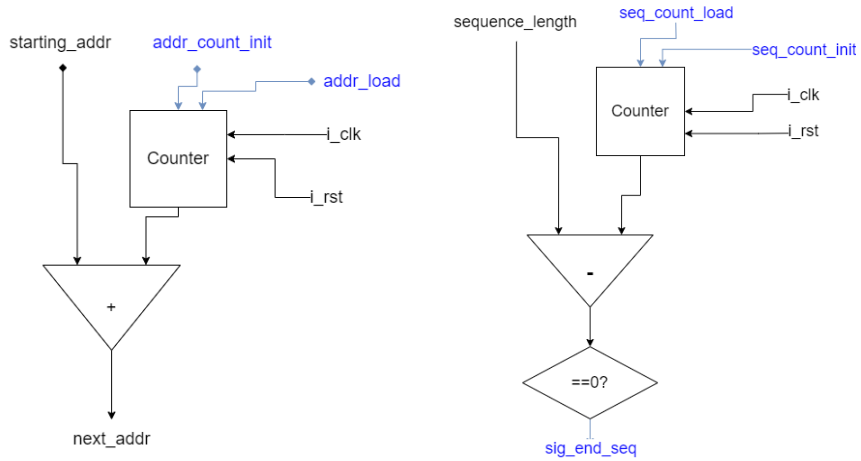
Figura 1.1: Schematico di un contatore con registro controllato da un segnale

Il registro ha un valore di reset pari a `0x0` ed è controllato attraverso un segnale asincrono **counter-load**, che permette di memorizzare il prossimo valore di conteggio (o zero), al successivo fronte di clock.

1.2 Controllore di indirizzo

Il *controllore di indirizzo* (Figura 1.2a) è il componente che si occupa di gestire l'indirizzamento della memoria.
Infatti come è possibile vedere in Figura 1.5 esso è collegato in ingresso (**starting-addr** => **i-mem-addr**) e in uscita (**next-addr** => **o-mem-addr**) ai segnali

legati agli indirizzi di memoria.
Come già sottolineato precedentemente, il contatore funziona come un *offset* rispetto allo **starting-addr**: ogni volta che il contatore viene incrementato allora $next\_addr = starting\_addr + (offset\_addr + 1)$, in cui *offset_addr* è il numero di incrementi già avvenuti.



(a) Schematico del controllore di indirizzo

(b) Schematico del controllore di sequenza

Figura 1.2: Schematici dei controllori: componenti che implementano il contatore

I segnali di **addr-count-init** e **addr-load** sono connessi ai rispettivi del contatore. Il primo segnale viene alzato (`addr_count_ini <= '1'`) solamente nello stato di reset, infatti, solo all'inizio di una nuova elaborazione è necessario ricominciare a contare a partire dall'indirizzo di memoria iniziale. Viene poi abbassato (`addr_count_init <= '0'`) se **addr-load** è alto, ovvero nel momento in cui è necessario scorrere all'indirizzo successivo.

1.3 Controllore di sequenza

Il *controllore di sequenza* (Figura 1.2b) funziona in maniera analoga al controllore di indirizzo. Differisce da quest'ultimo in quanto in entrata è connesso col valore di sequenza, ossia il numero K di parole da elaborare; in uscita genera un segnale di controllo interno al modulo.

I due segnali di controllo del contatore **seq-count-load** e **seq-count-init** si collegano nuovamente ai segnali rispettivi del contatore e permettono la gestione del conteggio delle *parole* lette. Il segnale in uscita (**sig-end-seq**) notifica il modulo che la sequenza è stata letta completamente e si deve dunque procedere alla terminazione e alla preparazione di un nuovo ciclo di elaborazione.

È possibile notare che si sarebbe potuto utilizzare una sola istanza di contatore per gestire entrambi i moduli, tuttavia ciò imporrebbe scelte di progettazione riguardo alla FSM più complesse, in quanto l'*offset* di indirizzo è incrementato in istanti diversi rispetto all'*offset* della sequenza.

1.4 Contatore di credibilità

Il *contatore di credibilità* è il componente che si occupa di tenere traccia del valore di credibilità corrente. Tale componente potrebbe essere costruito integrando il modulo contatore, tuttavia si aggiungerebbe della complessità alla gestione dei segnali da parte della FSM, in quanto è necessaria della logica di controllo aggiuntiva per il corretto funzionamento. In generale non si presta facilmente all'implementazione rispetto a un segnale costante a cui aggiungere un *offset*.

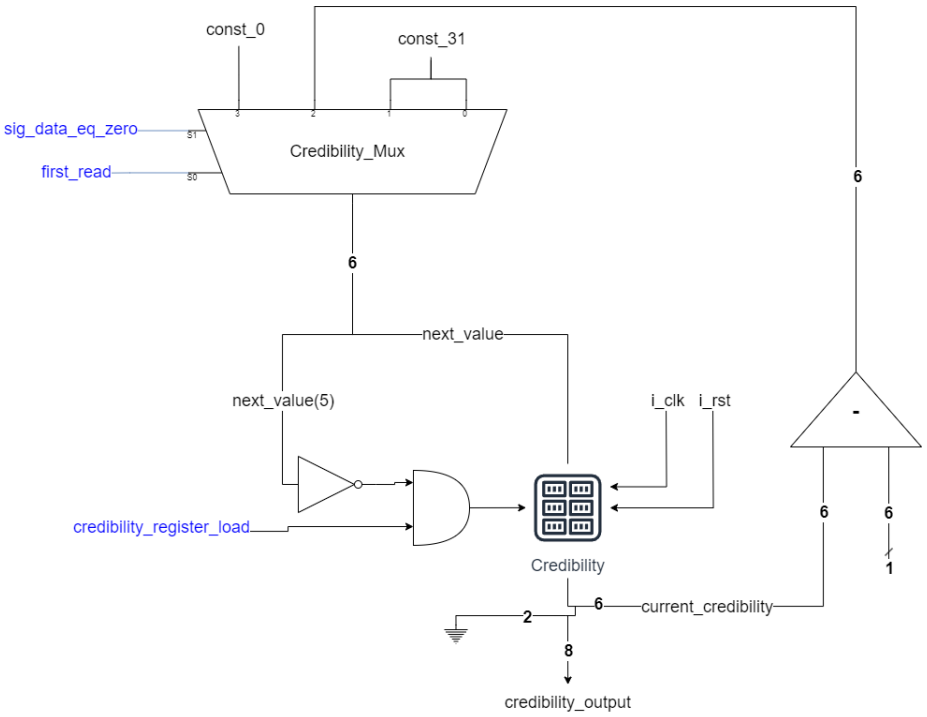


Figura 1.3: Schematico del contatore di credibilità

Il componente si compone di un multiplexer a due pin selettori controllato da due segnali:

- **sig-data-eq-zero** segnala se il dato letto in ingresso è uguale a zero;
- **first-read** segnala se si è letto almeno un valore, ossia si è compiuto almeno un ciclo di lettura.

Il primo segnale seleziona il valore di inizializzazione, `0x1f`, della credibilità (`sig_data_eq_zero = '0'`) oppure il successivo valore di credibilità (`sig_data_eq_zero = '1'`). Il secondo segnale seleziona il valore di inizializzazione (`first_read = '0'`) oppure il valore iniziale `0x0` del particolare caso di elaborazione.

In tal modo si codificano quattro opzioni ben definite, di cui due identiche, come mostrato in Tabella 1.1.

Il registro ha valore di reset pari a `0x0` ed è controllato da una porta *AND* da visualizzare come mostrato in Tabella 1.2.

sig-data-eq-zero	**first-read**	mutex selection	next value
0	0	0b00	31
0	1	0b01	31
1	0	0b10	curr_value - 1
1	1	0b11	0

Tabella 1.1: Valori in uscita dal multiplexer in base ai valori dei segnali di controllo

credibility_register_load	next_value(5)	sig_load
0	-	0
1	0	1
1	1	0

Tabella 1.2: Tabella di verità del segnale di scrittura del registro del contatore di credibilità

Dalla visualizzazione di tale tabella risulta più facile comprendere in quali casi sia possibile scrivere sul registro. Innanzitutto la scrittura è controllata dal segnale **credibility-register-load**, se questo è basso allora, in qualsiasi caso, non verrà sovrascritto il valore di credibilità corrente, altrimenti la scrittura dipende dal valore successivo di credibilità.
Poiché la specifica richiede che $\forall t(31 > cred >= 0)$, considerando una rappresentazione in complemento a due di un numero a 6 bit, allora il valore massimo rappresentabile è `0x1f` corrispondente a `0b011111`, al contrario il valore minimo rappresentabile è `0x20` corrispondente a `0b100000`. Con tale prospettiva si traduce la specifica con la condizione `if next_value >= 0`. Quest'ultima può essere circuitalmente valutata analizzando il bit più significativo del segnale **next-value**: se questo vale 0 allora il valore di credibilità è ancora compreso tra 31 e 0, altrimenti sarà un numero oltre il 31 (o in complemento a due, minore di zero).

1.5 Manipolatore di parole

L'ultimo componente del modulo è il *manipolatore di parole*. Esso ha la funzione di tenere traccia dell'ultima parola letta diversa da zero (oppure zero nel caso particolare). La parola letta all'indirizzo specificato dal *controllore di indirizzo* è il segnale d'ingresso che arriva al registro che lo compone; questo si occupa di averne memoria.

La logica che comanda il registro è la porzione più complessa di questo componente. Infatti, a meno del caso particolare, si vuole tenere conto dell'ultima parola letta diversa da zero. La scrittura del registro è regolata allora da una porta *AND*, che permette la scrittura solo nel caso in cui la parola in ingresso sia diversa da zero (`sig_data_eq_zero = '0'`) e il segnale di scrittura **word-load** sia alto.
Inoltre per tenere conto del caso particolare, è necessario permettere la scrittura di uno zero solamente nel caso in cui sia la prima parola della sequenza. Se questo è il caso allora `sig_data_eq_zero = '1'` e `first_read = '1'`; ciò implica che venga selezionato la seconda uscita del multiplexer (`0b1`), che può

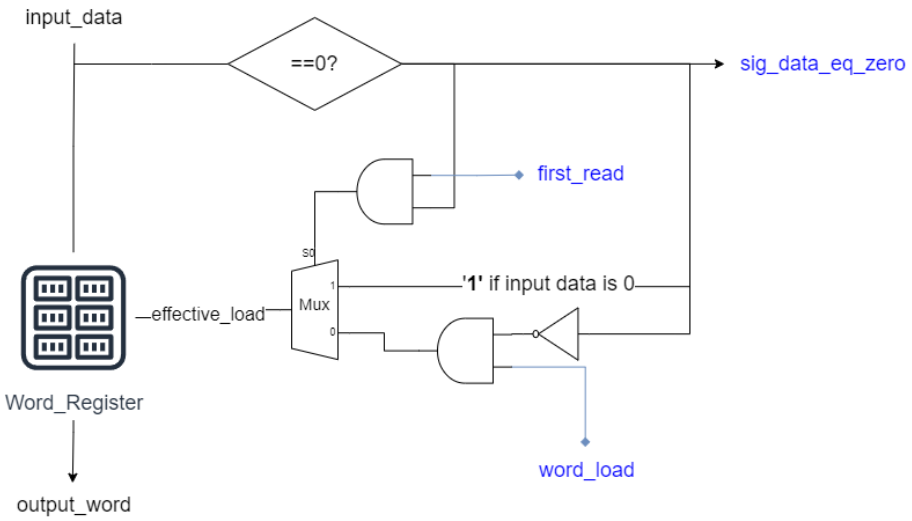


Figura 1.4: Schematico del manipolatore di parole

essere anche rappresentato da una costante a uno, perché, per costruzione, se viene scelto vale sicuramente uno.

1.6 Modulo Completo

Descrizione del modulo

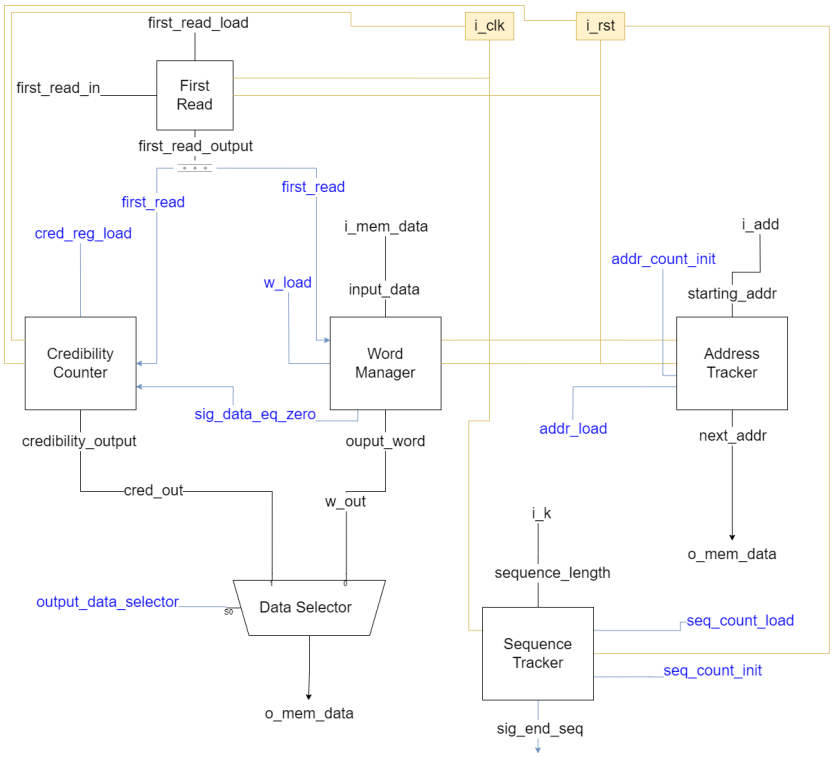


Figura 1.5: Schematico complessivo del modulo progettato, con visione esterna delle sottocomponenti

All'interno della Figura 1.5 è possibile vedere anche due altre componenti che completano il modulo:

- il flip-flop *first read* che si occupa di segnalare il manipolatore di parole e il contatore di credibilità;
- il multiplexer *data selector* che seleziona il segnale da scrivere in uscita.

Alcune considerazioni

Sebbene si sia tentato di mantenere sempre una visione modulare e di mantenere indipendenti tra loro le varie componenti del modulo, è visibile che almeno per quanto riguarda il *contatore di credibilità* (Figura 1.3) e il *manipolatore di parole* (Figura 1.4), questi siano interdipendenti e non siano effettivamente portabili in altri sistemi; questa scelta progettuale è stata dettata dal tempo e dalla maggiore facilità di progettazione.[1]

[1] Per questo si reputa che i due componenti meglio progettati siano invece i due controllori.

Capitolo 2

Macchina a stati finiti

La macchina a stati finiti (di Moore) definita su questa architettura si compone di otto stati, di cui sei vanno a formare un ciclo. Quest'ultimo è il ciclo di elaborazione effettivo composto da due stati di lettura, due stati di scrittura e due stati ausiliari che permettono lo scorrimento della memoria. Gestisce la logica di tutti i segnali interni al sistema, ma anche dei segnali in uscita, che sono segnali di controllo di comunicazione con la *RAM*.

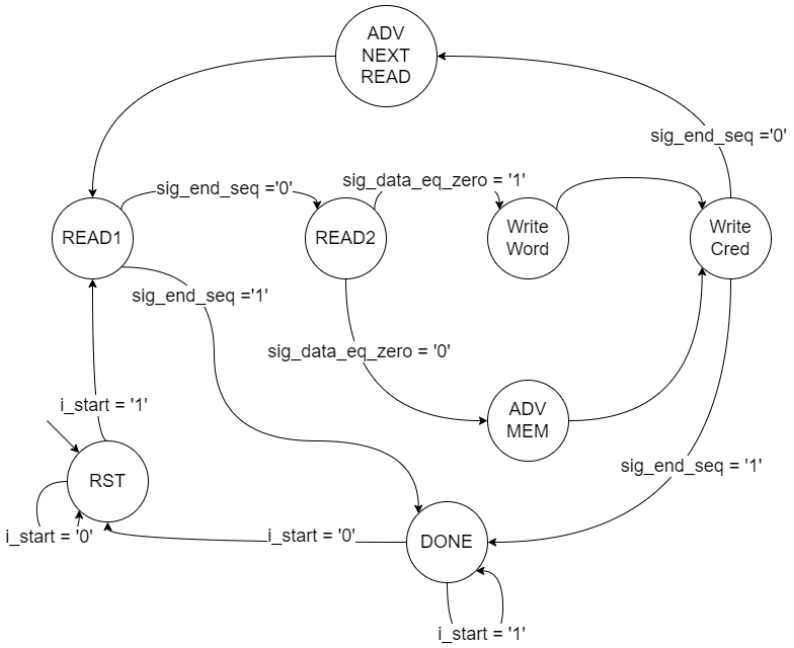


Figura 2.1: Schema delle transizioni della macchina a stati

2.1 Percorso di prima lettura

Dallo stato di reset (*RST_S*), che ha il compito di abbassare **o-done** e di inizializzare tutti i componenti, ovvero azzerare i contatori dei due controllori e

alzare il segnale di **first-read**, tale percorso raggiunge per la prima volta i due stati di lettura.

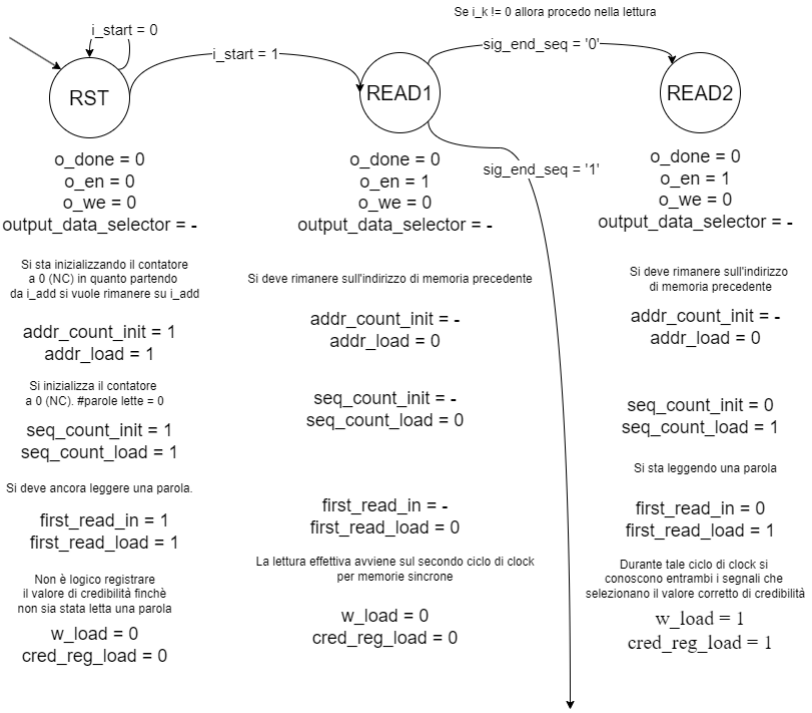


Figura 2.2: Stati percorsi durante la prima lettura

Nel primo, il modulo richiede l'accesso in lettura alla memoria, nel secondo memorizza il valore ricevuto in input, aumenta il contatore di sequenza (parole lette + 1) e abbassa il segnale di **first-read**.[1]

Inoltre nel primo stato di lettura è già possibile notare un bivio: se il segnale di fine sequenza è alto, allora si è verificato il caso in cui il numero di parole da elaborare fosse nullo; è possibile allora terminare l'elaborazione senza procedere all'effettiva lettura.

2.2 Percorso di scrittura

Il percorso di scrittura dipende dalla lettura del valore di input. Se questo è nullo (`sig_data_eq_zero = '1'`) allora è necessario sovrascrivere lo zero all'indirizzo di memoria corrente prima di poter scrivere il valore di credibilità (decrementato). In caso contrario, è possibile procedere direttamente alla scrittura del valore di credibilità[2].

Lo stato *WRITE WORD* si occupa di sovrascrivere lo zero presente in memoria con l'ultima parola letta; per fare ciò si seleziona la prima uscita dell'*output selector* (`output data selector <= '0'`).

[1] Si noti che a questo punto sono disponibili tutti i segnali che permettono l'inizializzazione del contatore di credibilità

[2] Per controllare il valore di credibilità cfr. Tabella 1.1

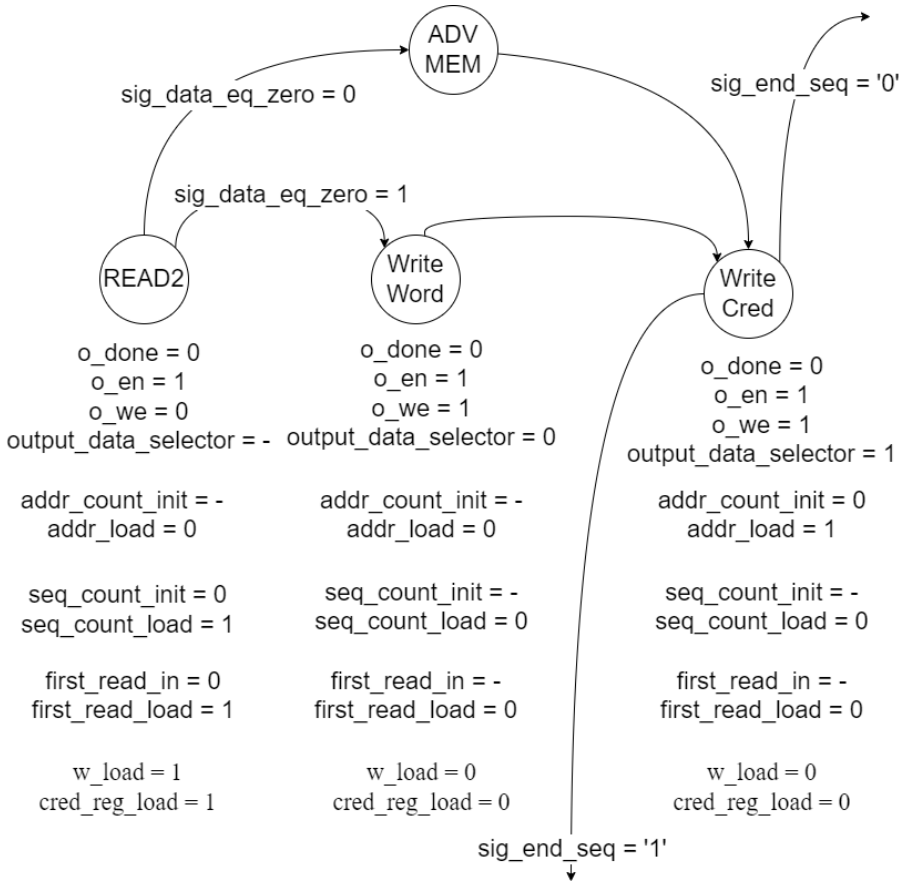


Figura 2.3: Stati del percorso di scrittura

Analogamente nello stato *WRITE CRED* si seleziona l'uscita dell'*output selector* corrispondente al valore di credibilità corrente.

Da qui, se la sequenza è finita (`sig_end_seq = '1'`) allora si transita nello stato finale *DONE*, altrimenti il modulo si prepara per la lettura di una nuova parola.

2.3 Stati rimanenti

Lo stato di *DONE* ha il solo compito di abbassare tutti i segnali e alzare il segnale **o-done**, finché non si verifichi l'abbassamento del segnale **i-start**.
Gli stati *ADV MEM* e *ADV NEXT READ* incrementano il contatore del controllore di indirizzo, scorrendo in tal modo la memoria.

Capitolo 3

Risultati Sperimentali

3.1 Report di sintesi

Dal report di sintesi è possibile acquisire alcune informazioni riguardo alla sintesi del modulo. Innanzitutto in Tabella 3.1[1] sono state codificati gli stati della FSM in modalità *sequential*.[2]

State	Encoding
rst-s	000
read-1	001
read-2	010
ww	011
advmem	100
wc	101
adv-next-read	110
done	111

Tabella 3.1: Trasposizione degli stati in codifica sequenziale

Dopodiché sono riferite le informazioni rispetto ai sotto moduli che compongono il circuito sintetizzato. In particolare si riporta in Tabella 3.2 l'informazione riguardo ai registri sintetizzati.[3]

Registers:	10 Bit Registers := 1
	8 Bit Registers := 1
	6 Bit Registers := 1
	1 Bit Registers := 1

Tabella 3.2: Tabella dei registri sintetizzati

Facendo un confronto con lo schematico sull'FPGA e utilizzando il report approfondito, si è controllata la corrispondenza dei registri con le componenti

[1] Cfr. Figura 3.2

[2] Questo nome è riportato rispetto al report di Vivado. Un'altro nome per tale codifica è *gray coding*.

[3] Cfr. Figura 3.3

descritte. Infatti è possibile notare un registro mancante: il registro del contatore a 16 bit del controllore di indirizzo[4] è stato sostiuito da un gruppo di LUTs che implementano la funzione di conteggio.

Il registro da 1 bit corrisponde al flip-flop che manipola il segnale di **first-read**, il registro da 8 bit è quello presente all'interno del manipolatore di parole, il registro da 6 bit memorizza la credibilità.

Tali registri sono tutti implementati per mezzo di flip-flop, come si può notare dalla Tabella 3.3.[5]

Site Type	Used	Util%
Slice Registers	44	0.05
Registers as Flip Flop	44	0.05
Registers as Latch	0	0.00

Tabella 3.3: Tabella di tipologia registri

Infine, come è possibile vedere in Figura 3.1, il modulo rispetta il vincolo imposto sul clock di 20 ns, con gran margine.

```
Timing Report

Slack (MET) :             17.368ns  (required time - arrival time)
  Source:                 SeqTrack/SeqCount/current_value_reg[1]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@
  Destination:            FSM_sequential_current_state_reg[0]/D
                            (rising edge-triggered cell FDCE clocked by clock  {rise@
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns - clock rise@0.000ns)
  Data Path Delay:        2.489ns  (logic 1.174ns (47.168%)  route 1.315ns (52.832%))
  Logic Levels:           5  (CARRY4=2 LUT2=1 LUT5=1 LUT6=1)
```

Figura 3.1: Report dello slack

3.2 Testing delle componenti

Al disegno e alla progettazione delle componenti sono seguiti implementazione (con controllo dello schematico) e testing sia logico-comportamentale sia post-sintesi.

3.2.1 Test del contatore

Il test del contatore è un piccolo test che controlla semplicemente che vi sia un effettivo incremento del registro una volta scelto l'ingresso del multiplexer corretto. Tale test è risultato subito positivo ed ha posto le fondamenta di tutti gli altri.

[4]Cfr. Figura 1.2a
[5]Cfr. Figura 3.4

```
-----------------------------------------------------------------------------------------------------
                 State |                     New Encoding |               Previous Encoding
-----------------------------------------------------------------------------------------------------
                 rst_s |                              000 |                              000
                read_1 |                              001 |                              001
                read_2 |                              010 |                              010
                    ww |                              011 |                              011
                advmem |                              100 |                              101
                    wc |                              101 |                              100
          adv_next_read |                              110 |                              110
                  done |                              111 |                              111
-----------------------------------------------------------------------------------------------------
INFO: [Synth 8-3354] encoded FSM with state register 'current_state_reg' using encoding 'sequential' in module 'project_reti_logiche'
```

Figura 3.2: Tabella della codifica degli stati

```
+---Registers :
                      10 Bit    Registers := 1
                       8 Bit    Registers := 1
                       6 Bit    Registers := 1
                       1 Bit    Registers := 1
```

Figura 3.3: Tabella dei Registri sintetizzati

```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   79 |     0 |     41000 |  0.19 |
|   LUT as Logic          |   79 |     0 |     41000 |  0.19 |
|   LUT as Memory         |    0 |     0 |     13400 |  0.00 |
| Slice Registers         |   44 |     0 |     82000 |  0.05 |
|   Register as Flip Flop |   44 |     0 |     82000 |  0.05 |
|   Register as Latch     |    0 |     0 |     82000 |  0.00 |
| F7 Muxes                |    0 |     0 |     20500 |  0.00 |
| F8 Muxes                |    0 |     0 |     10250 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figura 3.4: Tabella della slice logic

3.2.2 Test del controllore di indirizzo

Subito dopo il contatore si è testato il controllore di indirizzo. Si è controllato che si scorresse sulla memoria nella maniera corretta e che il modulo rispettasse dunque i suoi requisiti.

Insieme al testing del modulo effettivo si è anche fatto un test di scrittura per valutare come interagissero il modulo definito e la memoria, rispetto all'interfaccia data. Si sono dapprima tratte le conclusioni sbagliate: si era ipotizzato, da una lettura del testbench fornito, che il cambio di indirizzo di memoria fosse più rapido della scrittura e che dunque il valore sarebbe stato effettivamente scritto nel ciclo di clock successivo e sull'indirizzo di memoria calcolato successivamente. Questo errore di valutazione è stato corretto poi in fase finale di completamento del modulo, in quanto si è notato che effettivamente lo scarto non potesse essere utilizzato in tal modo.

3.2.3 Test del controllore di sequenza

Per controllare in maniera corretta il controllore di sequenza si sono iniziate a descrivere le prime FSM, in modo tale da poter controllare il corretto decremento e valutazione della fine della sequenza.

Tramite tale test si è iniziato ad annotare il comportamento effettivo dei flip-flop e come questo avrebbe impattato sul funzionamento del modulo.

3.2.4 Test del controllore di credibilità

Il controllore di credibilità è stato il componente più complesso da progettare. Ciò si deve all'implementazione per mezzo di un multiplexer comandato da due segnali di controllo interni.

Durante la prima scrittura del test, prima di valutare i risultati, si è notato che il comportamento desiderato non poteva essere rispettato dalla prima versione del componente disegnata[6]. Si è dovuto procedere allora a una ri-progettazione del componente che ha prodotto il sotto modulo descritto[7].

Di tale componente si è testato che tutti gli ingressi del multiplexer funzionassero come desiderato e che la scrittura del registro fosse coerente con quanto richiesto.

3.2.5 Test del manipolatore di parole

Il manipolatore di parole non è stato testato in un testbench personalizzato e a se stante. Tuttavia se n'è valutato il corretto funzionamento sia dal controllo dello schematico, che rispecchiava il circuito desiderato, sia dai testbench fatti successivamente per il modulo completo.

3.3 Testing del modulo

Una volta composto il modulo completo si è testato nella sua interezza, a partire dal testbench fornito, che ha subito mostrato l'errore di valutazione fatto durante il testing del controllore di indirizzi.

[6] Cfr. Figura 3.5
[7] Cfr. Figure 1.3

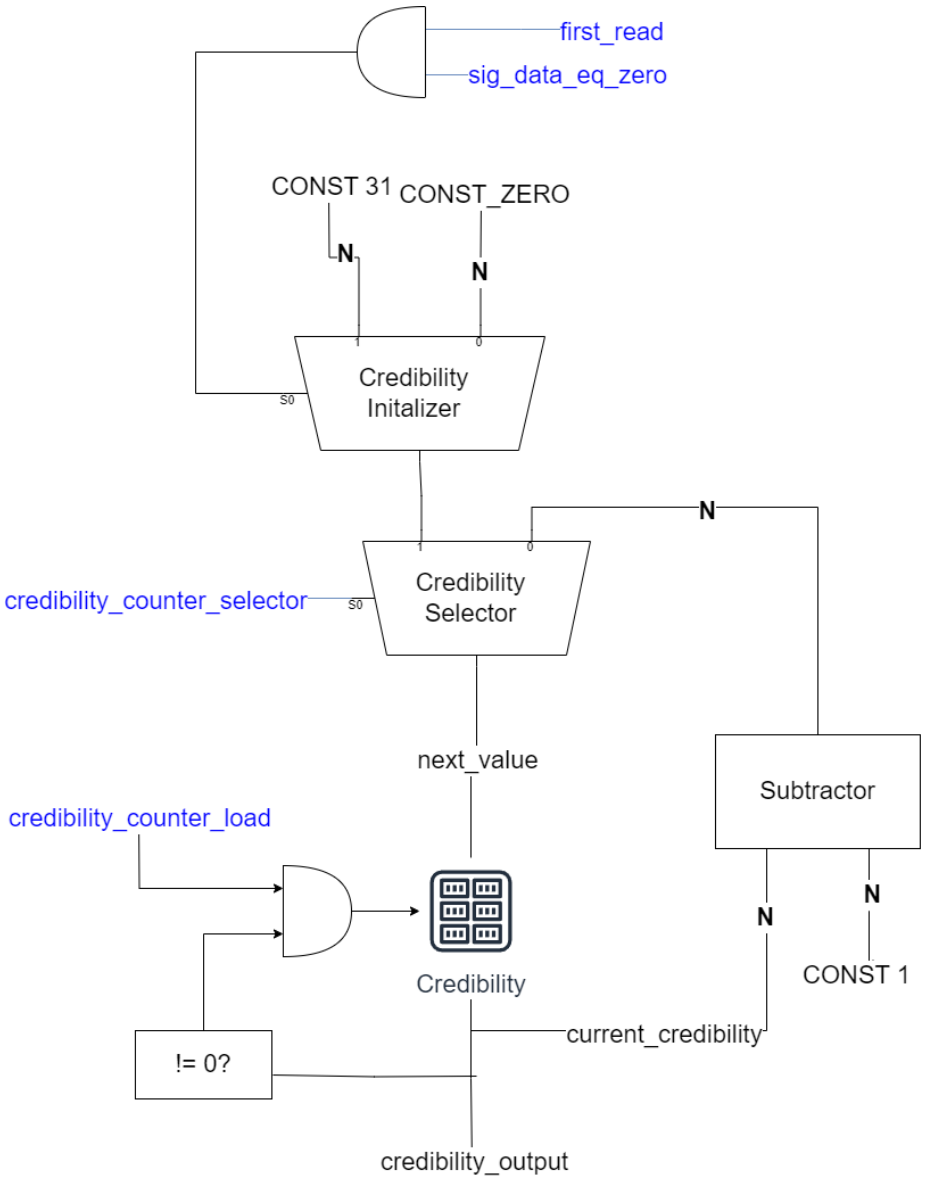


Figura 3.5: Prima versione del contatore di credibilità

3.3.1 Testbench base

Il testbench fornito ha rivelato che la scrittura in memoria rispetto al cambio di indirizzo è più veloce e dunque era necessario spostare l'azione di avanzamento della memoria su di uno stato diverso. Tuttavia tale risultato ha mostrato anche come fosse possibile scrivere sull'indirizzo corrente e spostarsi all'indirizzo di memoria successivo all'interno dello stesso stato: poiché il contatore necessita di aspettare il fronte di salita per registrare un incremento, allora preparando la scrittura del contatore nello stato di scrittura della parola (`addr_count_init <= '0'` e `addr_load <= '1'`), si è constatato possibile accedere in scrittura, allo stato seguente, nello spazio di memoria del valore di credibilità.

3.3.2 Testbench sequenze multiple

Tale testbench testa due sequenze consecutive. Verifica che si abbassi correttamente il segnale **o-done** e che il modulo di prepari correttamente a una nuova elaborazione.

Testa inoltre il caso particolare di lettura di una serie di zeri a inizio elaborazione:

```
constant SCENARIO_LENGTH : integer := 14;
type scenario_type is array (0 to SCENARIO_LENGTH*2-1) of integer;

signal scenario_input : scenario_type := (128, 0, 64, 0, 0, 0, 0, 0,
                                          0, 0, 0, 0, 0, 0, 100, 0, 1, 0,
                                          0, 0, 5, 0, 23, 0, 200, 0, 0, 0);

constant SCENARIO2_LENGTH : integer := 15;
type scenario2_type is array (0 to SCENARIO2_LENGTH*2-1) of integer;
signal scenario2_input : scenario2_type := (0, 0, 0, 0, 64, 0, 0, 0, 0, 0,
                                            0, 0, 0, 0, 0, 0, 150, 0, 1, 0,
                                            0, 0, 5, 0, 23, 0, 0, 0, 0, 0);
```

Una seconda variazione di tale test utilizza un reset tra le due sequenze.

3.3.3 Test di conteggio massimo

Tale test valuta il limite massimo di parole che si possano contare, ovvero 1023 (**i-k** è un segnale a 10 bit).

3.3.4 Test di credibilità

Si è definito un test che porti il limite di credibilità a zero per verificare se il modulo limiti la credibilità a zero.

3.3.5 Test di lettura nullo

Il testbench impone un numero di parole da elaborare pari a zero, all'inizio della sequenza ma anche in una sequenza di elaborazioni. Serve a verificare che il componente non effettui una scrittura se il segnale interessato è zero.

Conclusioni

La progettazione del modulo qui sopra descritto si è mossa per passi successivi secondo un procedimento bottom-up, individuando per prima cosa i comportamenti necessari per soddisfare le specifiche e procedendo poi con il disegno delle componenti.

Infine la progettazione della FSM e i testbench fatti su ogni componente hanno portato alla luce alcuni dei difetti di progettazione di questi. In particolare è stata una sfida, progettare il contatore di credibilità senza utilizzare un *process* o un segnale indipendente, in quanto tutta la complessità elaborazionale si riduce a tale componente.

Un aspetto negativo è stato l'abbandono di una filosofia di progettazione modulare per una maggiore semplicità di costruzione.