POLITECNICO
MILANO 1863

Progetto di Reti Logiche

Prof. Gianluca Palermo

Francesco Simone
10719948

Fabio Vokrri
10738386

A.A. 2023 - 2024

Indice

1 Introduzione

Il progetto del corso di Reti Logiche (a.a. 2023-2024), consiste nell'implementazione di un componente hardware, che si interfacci con una memoria di tipo *Single-Port Block RAM Write-First Mode* e che svolga le seguenti azioni:

- leggere dalla memoria una sequenza di `K` parole `W`, il cui valore é compreso tra 0 e 255.
- il valore 0 all'interno della sequenza deve essere interpretato come *"valore non specificato"*.
- la sequenza di `K` parole da elaborare é presente in memoria a partire dall'indirizzo `i_add` specificato, ogni 2 byte (e.g. `ADD`, `ADD+2`, `ADD+4`, ...).
- completare la sequenza, sostituendo gli zero, laddove presenti, con l'ultimo valore valido letto. Nel byte mancante (e.g. `ADD+1`, `ADD+3`, `ADD+5`, ...), dovrà essere inserito il valore di credibilità, il quale sarà pari a 31 ogni qualvolta il valore della sequenza sia diverso da zero e, in caso contrario, verrà decrementato fino ad un minimo di zero.

Il componente da progettare ha 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;
```

La cui descrizione dettagliata può essere reperita nel file di specifica.

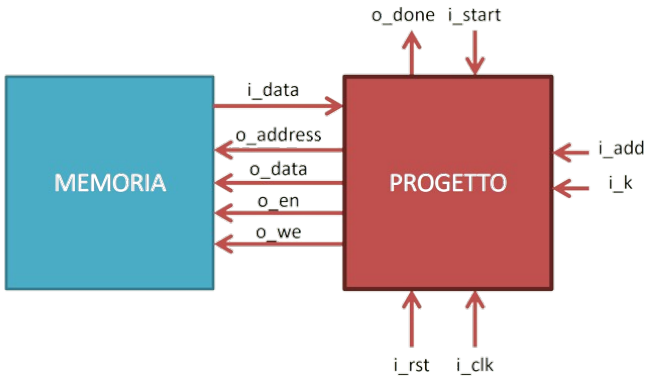

Figura 1: Segnali di ingresso e uscita del componente

Il componente mostrato in figura, presenta un segnale di clock (`i_clk`), unico per tutto il sistema, ed un segnale di reset (`i_rst`) asincrono, anch'esso unico, che reinizializza il componente. Inoltre, la progettazione prevede che all'istante iniziale, ovvero prima che il segnale di start assuma valore 1, venga sempre dato il comando di reset.

1.1 Note ulteriori di specifica

Il componente progettato seguirà il protocollo specificato di seguito in figura:

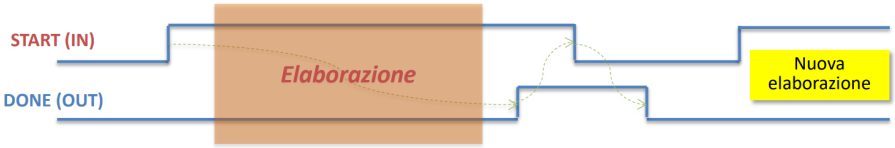

Figura 2: Protocollo di funzionamento

Un segnale di start (`i_start`), con associato l'indirizzo di partenza `i_add` e la lunghezza della sequenza `i_k`, determina la richiesta di codifica e rimane alto per tutto il tempo di elaborazione. Conclusa l'esecuzione, il componente imposta il segnale di `o_done` ad 1, fin tanto che il corrispettivo segnale di `i_start` è posto a 0. A quel punto, il componente ha il compito di impostare a 0 il segnale di `o_done`, in modo da mettersi nelle condizioni di poter elaborare una nuova sequenza di parole senza dover aspettare il reset del componente.

Nel caso in cui la sequenza inizi con un dato non specificato (pari a 0), il valore rimane inalterato e il corrispettivo valore di credibilità deve essere posto a 0. Tale funzionamento deve essere iterato fino alla lettura di un valore valido (diverso da 0), da cui l'esecuzione riprende ad essere standard, sopra specificata.

2 Architettura

A livello architetturale, il componente è stato progettato mediante l'utilizzo di un *Automa a Stati Finiti*, il cui funzionamento é presentato in figura

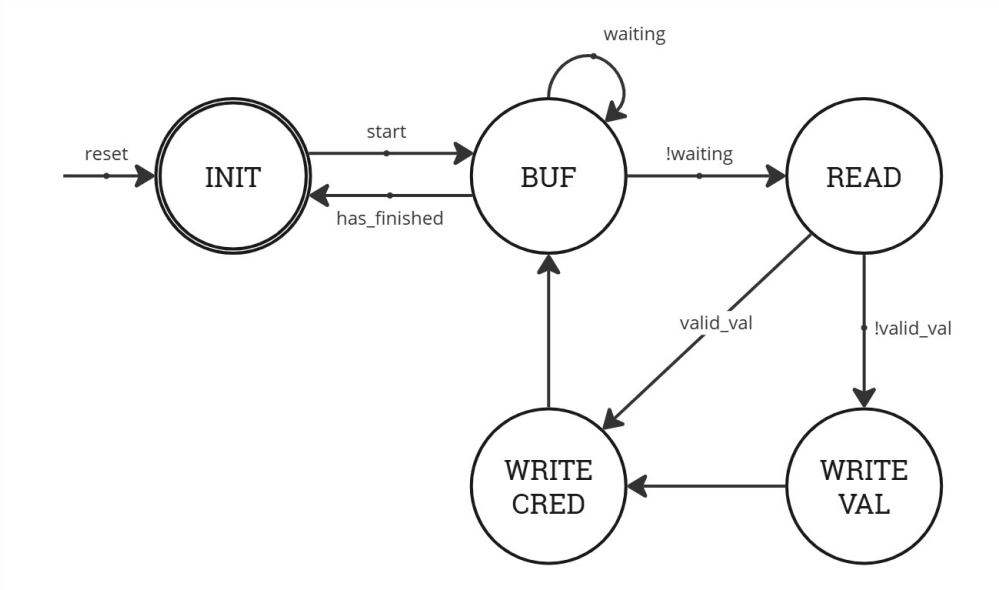

Figura 3: Macchina di Mealy utilizzata per implementare il componente

Tale FSA rappresenta una *macchina di Mealy* composta da 5 stati (`INIT`, `BUF`, `READ`, `WRITE_VAL` e `WRITE_CRED`). Per una maggiore semplicità implementativa, il componente è stato sviluppato utilizzando un solo *process*, sensibile ai segnali di `i_clk` e di `i_reset`. Così facendo, lo stato della macchina di Mealy viene aggiornato sul fronte di salita del ciclo di clock e il segnale di reset può essere ascoltato in maniera asincrona, in qualsiasi momento dell'esecuzione, senza dunque attendere il segnale di clock. Il funzionamento dei segnali utilizzati e degli stati implementati è spiegato nel dettaglio di seguito.

2.1 Segnali Ausiliari

Per implementare la logica del componente, sono stati utilizzati i seguenti segnali ausiliari:

```
-- stato corrente dell'FSA
signal current_state: status;

-- valore di credibility
signal credibility_val: std_logic_vector(7 downto 0);

-- ultimo valore valido
signal last_valid_val: std_logic_vector(7 downto 0);

-- posto a 1 se l'automa ha concluso l'esecuzione
signal has_finished: std_logic;

-- posto a 1 se l'automa deve attendere la stabilizzazione dei segnali
signal waiting: std_logic;

-- indirizzo corrente
signal current_addr: std_logic_vector(15 downto 0);
```

2.2 Stato `INIT`

Nello stato di `INIT`, il componente attende che il segnale di `i_start` assuma valore 1 per poter iniziare l'elaborazione della sequenza di ingresso. Una volta dato il segnale di start, il componente abilita la lettura della memoria (`o_mem_en = 1`) e imposta `o_mem_addr` all'indirizzo di memoria corrispondente alla prima parola da leggere, indicato in `i_add`. Durante l'attesa, il componente azzera il segnale di `o_done`, per segnalare la possibilità di lettura di una nuova sequenza.

La macchina di Mealy entra in questo stato nel ciclo di clock immediatamente successivo (in `rising edge`) al segnale di reset, dopo aver quindi azzerato tutte le uscite e i segnali interni, oppure nel momento in cui viene impostato il segnale `has_finished` ad 1, ovvero quando l'elaborazione della sequenza di ingresso è terminata.

Dallo stato di `INIT`, la macchina di Mealy transita nello stato di `READ` solo nella condizione in cui il segnale di `i_start` è posto ad 1 ed `has_finished` a 0, avviando così la computazione.

NB: Nello stato di `INIT`, azzero i valori dei segnali `last_valid_val` e `credibility_val` per poter gestire negli stati successivi il caso limite secondo cui il primo valore letto in ingresso sia pari a 0.

2.3 Stato `BUF`

Nello stato di `BUF`, il componente attende la stabilizzazione del segnale `i_mem_data`, reso instabile dal ritardo di `1ns`, il quale causa la perdita di un ciclo di clock prima che la lettura possa andare a buon fine. Inoltre, da specifica della memoria, ad `i_mem_data` viene assegnato l'ultimo valore scritto in memoria rimanendo stabile per un ciclo di clock. Si rende dunque necessario un segnale ausiliario `waiting`, il quale assume valore 1 dopo la scrittura del valore di credibilità, permettendo alla macchina di spendere un ulteriore ciclo di clock nello stato di buffer prima di una nuova lettura in memoria.

La macchina di Mealy, dallo stato di `INIT`, entra in questo stato dopo un ciclo di clock (in `rising edge`) quando il valore di start ha assunto valore 1 e `has_finished` 0.

Se la codifica delle `K` parole é terminata, ovvero quando il segnale `has_finished` assume valore 1, lo stato di `BUF` farà transitare la macchina nello stato di `INIT`, altrimenti, speso un ciclo di clock come spiegato sopra, farà transitare la macchina nello stato di `READ`.

2.4 Stato READ

Nello stato di READ, il componente legge dalla memoria il dato in ingresso all'indirizzo corrente (di parola), e verifica che questo sia specificato. Nel caso in cui lo fosse, imposta il segnale last_valid_val a tale parola, in modo che venga scritta in memoria qualora venga successivamente letto un dato non valido. Inoltre, imposta il contatore di credibilità a 31, l'indirizzo corrente a quello successivo e lo stato corrente a WRITE_CRED, in modo da scrivere il valore di credibilità nel corretto indirizzo di memoria. Al contrario, nel caso in cui il valore letto sia 0, passa direttamente allo stato WRITE_VAL, rimanendo nello stesso indirizzo di memoria.

La macchina di Mealy entra in questo stato senza alcuna condizione da soddisfare dopo aver atteso nello stato di BUFF.

Infine, come già detto, la macchina di Mealy transita nello stato di WRITE_CRED, nel caso in cui il valore appena letto dalla memoria sia diverso da 0, mentre transita nello stato di WRITE_VAL solo nel caso in cui il dato non è specificato.

2.5 Stato WRITE_VAL

Nello stato di WRITE_VAL, il componente abilita la scrittura in memoria, impostando o_mem_we ad 1. In seguito viene assegnato l'ultimo valore valido letto (last_valid_val) all'uscita o_mem_data, scrivendolo in memoria all'indirizzo specificato nello stato di INIT. Ogni qualvolta si entri in questo stato, il valore di credibilità viene decrementato, a meno che questo non abbia già raggiunto il minimo valore possibile (0), evitando possibili overflow. A questo punto assegno al segnale current_addr il valore dell'indirizzo di memoria successivo, così da preparare il componente e la memoria alla scrittura del valore di credibilità.

La macchina di Mealy entra in questo stato solamente nel caso in cui il dato letto precedentemente non era specificato, dunque nel ciclo di clock successivo ad una lettura in memoria, avvenuta nello stato di READ.

Infine, lo stato di WRITE_VAL permette alla macchina di transitare naturalmente nello stato di WRITE_CRED senza alcuna condizione di uscita.

2.6 Stato WRITE_CRED

Nello stato di WRITE_CRED, il componente abilita la scrittura in memoria, impostando o_mem_we ad 1, e scrive il valore di credibilità adeguato. A questo punto, se la macchina ha raggiunto l'ultimo indirizzo della sequenza, ovvero se il segnale current_addr ha assunto valore ADD + 2 * K - 1, assegno il valore 1 al segnale has_finished in modo da segnalare allo stato successivo (stato BUF) la conclusione dell'elaborazione. Infine, viene assegnato il valore 1 al segnale waiting così da preparare la macchina all'attesa di un ciclo di clock ulteriore per la corretta lettura del segnale i_mem_data come spiegato precedentemente.

La macchina di Mealy può transitare in questo stato in due casi specifici:

1. Dallo stato di READ, nel caso in cui il dato letto era valido.
2. Dallo stato di WRITE_VAL, nel caso in cui il dato letto era stato sovrascritto perché non valido.

Infine, lo stato di WRITE_CRED permette alla macchina di transitare naturalmente nello stato di BUF senza alcuna condizione di uscita.

3 Risultati Sperimentali

3.1 Sintesi

Il componente progettato è risultato idoneo alla sintesi su modulo hardware *Artix-7* `xc7a200tiffv1156-1L`. Per verificarne il corretto funzionamento è stato imposto come unico vincolo che il segnale di clock (`i_clk`) avesse una durata massima di `20ns`, come da specifica. Per farlo, è stato eseguito il seguente comando:

```
create_clock -period 20.000 -name clk -waveform {10.000 20.000} [get_ports i_clk]
```

Così facendo, sono stati ottenuti i seguenti risultati:

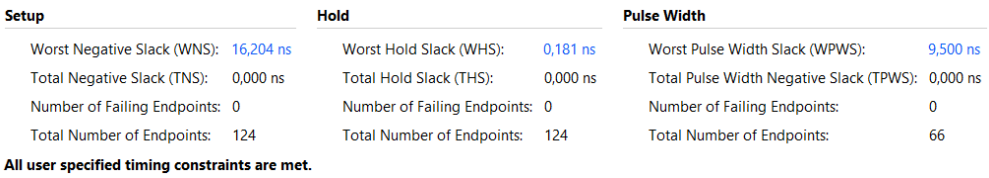

Figura 4: Risultati del Report Tool

Da come si può osservare in figura, il valore di *Worst Negative Slack (WNS)* è positivo: ciò significa che il componente sintetizzato soddisfa ampiamente le specifiche di tempo date.

3.2 TestBench

Il componente progettato è risultato idoneo a tutti i test effettuati, producendo una sequenza coerente con le specifiche date: in particolar modo, sono stati eseguiti con successo, oltre al test già presente nel testbench, 5 ulteriori test estrapolati dagli esempi del file di specifica e 2 test generati appositamente per verificare alcune condizioni limite. Da rimarcare il fatto che ognuno di tali scenari é stato simulato direttamente sul circuito sintetizzato dal software *Vivado*, con simulazioni di tipologia sia *functional* che *timing*.

Nello specifico sono stati analizzati i seguenti casi:

- **Reset Asincrono**: i test effettuati hanno mostrato una corretta risposta al segnale di reset, nonostante questo venga dato in maniera asincrona. Il modulo viene reinizializzato completamente e riportato allo stato di `INIT`, pronto all'elaborazione di una nuova sequenza di dati.
- **Più sequenze elaborate in successione**: i test effettuati hanno mostrato un corretto funzionamento nell'elaborazione di più sequenze di dati successive, rispettando quindi il protocollo indicato nelle specifiche. Il segnale `o_done` è riportato a 0 nel fronte di salita del ciclo di clock immediatamente successivo all'evento di azzeramento del segnale `i_start`.
- **Sequenza composta da soli dati non specificati**: i test effettuati hanno mostrato un corretto funzionamento nell'elaborazione di una sequenza di parole non specificate (poste a 0), mostrando in output la sequenza attesa, composta da soli zeri.

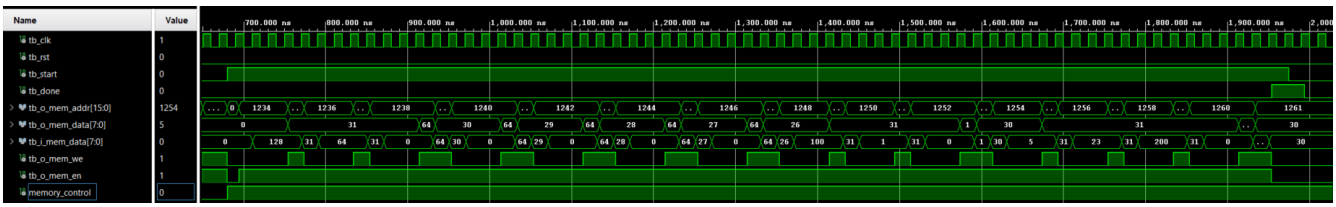

Figura 5: Simulazione post sintesi del componente

4 Conclusioni

Il componente è stato progettato cercando di mantenere un'architettura concisa, ma al contempo il più leggibile possibile, in modo da semplificare il processo di codifica in linguaggio VHDL. Questo ha portato allo sviluppo di un FSA composto da 5 stati, nonostante 3 di questi (`READ`, `WRITE_VAL` e `WRITE_CRED`) possano essere collassati in un unico stato, ipoteticamente chiamato `COMPUTING`, che ha il compito sia di leggere che di scrivere i dati secondo specifica. Una tale modifica del componente, però, porta ad un aumento significativo di segnali interni per la verifica delle condizioni di entrata e di uscita dal nuovo stato, rendendo così più complessa e meno intuitiva la sua implementazione. Di conseguenza si è optato per la realizzazione dell'FSA presentato. Applicando comunque le modifiche, si otterrebbe il seguente automa:

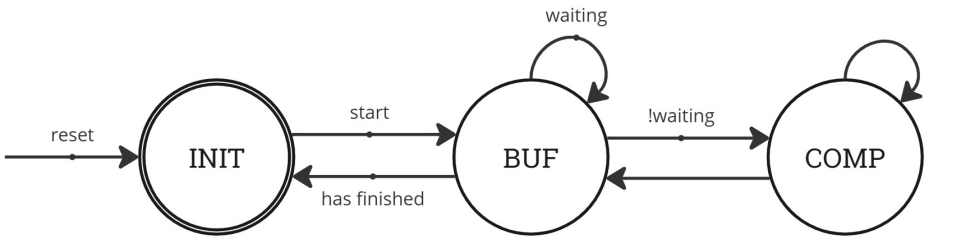

Figura 6: Possibile implementazione del componente hardware con un FSA alternativo

Un'ultima ottimizzazione che si potrebbe apportare al componente sviluppato riguarda il caso limite, per cui la sequenza in input ha inizio con un dato non specificato. In tal caso, è possibile saltare immediatamente all'indirizzo due volte successivo a quello corrente (`current_addr + 2`), in quanto, sia la parola letta, che il corrispettivo valore di credibilità, saranno già posti a 0, rendendo dunque superfluo seguire il flusso di esecuzione standard. Così facendo, si risparmierebbero 3 cicli di clock, ma aumenterebbe la complessità interna del componente hardware.