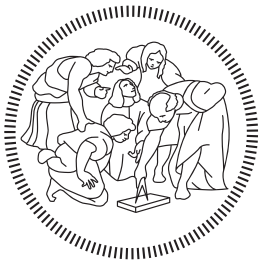

POLITECNICO
MILANO 1863

Prova finale

Progetto di reti logiche

Professore:
William Fornaciari

Relazione di:
Matteo Delton (10786485)
Gianluca Di Paola (10797584)

a.a. 2023/2024

Indice

1 Introduzione

Il progetto di reti logiche per l'anno accademico 2023/2024 consiste nella realizzazione di un modulo hardware nel linguaggio VHDL che si interfacci con una RAM, come mostrato in figura 1.

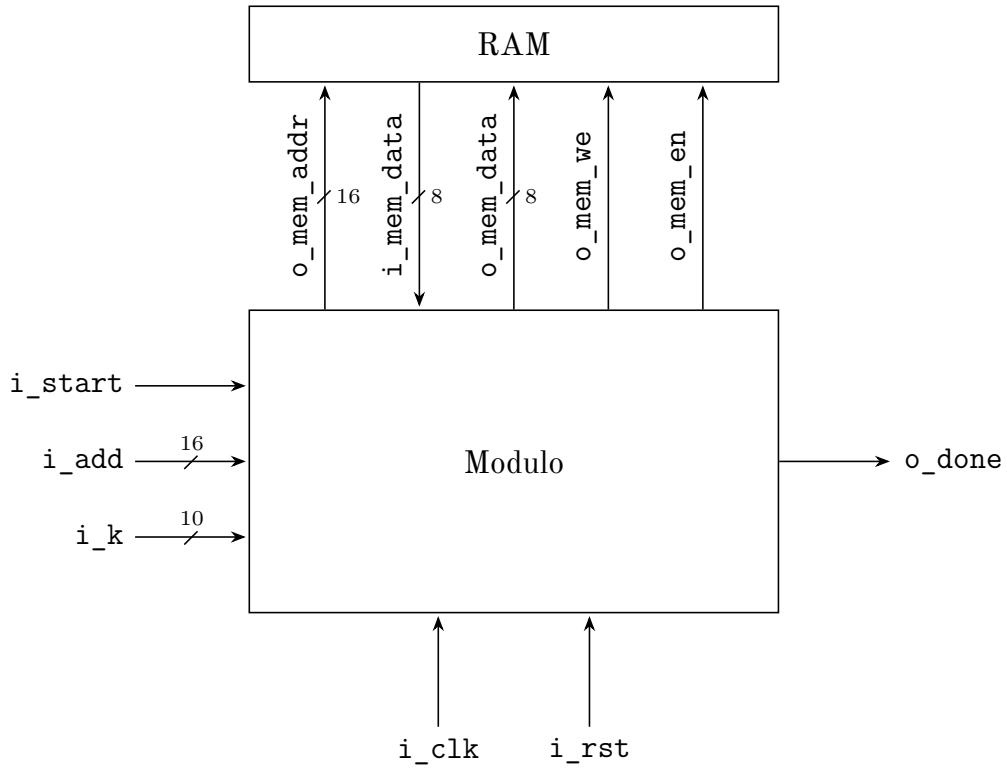

Figura 1: Rappresentazione di alto livello del modulo

Il modulo ha il compito di completare una sequenza a partire da un indirizzo di memoria ADD dato.

La sequenza è composta da K coppie di elementi di 8 bit ciascuno: il primo elemento identifica la "parola" W in ingresso, mentre il secondo sarà codificato nel suo "valore di credibilità" C.

Le parole hanno un valore compreso tra 0 e 255. Il valore 0 codifica l'informazione di "valore non specificato"; il modulo sostituisce tutte le parole di valore 0 con l'ultima parola letta di valore diverso da 0 (o con 0 se non ce ne sono mai state).

I valori di credibilità C hanno un valore compreso tra 0 e 31 e sono calcolati dal modulo come segue:

- se la parola ha un valore specificato (ovvero, è diversa da 0), il suo valore di credibilità sarà il massimo 31;
- se la parola *non* ha un valore specificato (ovvero, se è 0), il suo valore di credibilità sarà pari al valore di credibilità della parola precedente decrementato di 1, salvo se questo vale 0, nel qual caso non viene ulteriormente decrementato. Qualora non ci sia una parola precedente, il valore di credibilità è posto a 0.

Volendo ricavare una regola generale di funzionamento, noto l'indirizzo ADD della prima parola (fornito a inizio computazione), la parola i-esima W_i della sequenza si troverà all'indirizzo di memoria $\text{ADD} + 2(i-1)$ e il suo valore di credibilità C_i la seguirà all'indirizzo $\text{ADD} + 2(i-1) + 1$.

Il modulo deve essere in grado di riconoscere un segnale di reset `i_rst` asincrono, che può essere ricevuto in un qualsiasi momento. Ogni nuova computazione inizia con l'arrivo di un segnale alto `i_start`, che può essere ricevuto solo al termine di una elaborazione precedente e può (ma non deve necessariamente) essere preceduto da `i_rst` (lo è certamente alla prima computazione). Tale segnale `i_start` rimane alto mentre il modulo è attivo. Al termine dell'elaborazione, il modulo alza il segnale di `o_done`; quindi, non appena riconosce l'abbassamento di `i_start`, porta basso `o_done`.

2 Architettura

Per la sintesi del modulo, è stato usato il software Xilinx Vivado con una FPGA target Artix-7 xc7a200tfbg484-1. Abbiamo adottato il paradigma di progettazione *behavioral*.

Il componente è stato implementato attraverso una macchina a stati finiti (FSM), composta da tre processi principali (descritti in seguito) per rendere più atomica la struttura del modulo e facilitare la gestione complessiva. La FSM è formata da dieci stati, come mostrato in figura 2.

2.1 Stati della FSM

Di seguito una descrizione degli stati:

- `reset`: stato in cui si trova il modulo, a seguito del segnale di reset asincrono, in cui tutti i segnali sono posti a zero. Tale stato è raggiungibile da tutti gli altri stati in qualsiasi momento della computazione;
- `start`: stato in cui, letti la lunghezza della sequenza e il suo indirizzo iniziale, vengono inizializzati i contatori. La FSM entra in questo stato a seguito di `i_start` alto;
- `check`: stato in cui il modulo verifica se ci sono ancora parole da processare. Abbiamo deciso di inserirlo immediatamente dopo `start` per gestire il caso in cui la sequenza sia vuota (lunghezza `i_k` pari a zero)[1];
- `read_w`: stato in cui il modulo richiede la parola in lettura alla RAM;
- `wait_w`: stato in cui il modulo attende che la parola richiesta sia disponibile;
- `write_w`: stato in cui il modulo imposta il valore di credibilità e sovrascrive, se necessario, il valore di una parola non specificata. Viene anche fornito l'indirizzo della parola in questione;
- `write_c`: stato in cui il modulo scrive il valore di credibilità. Viene anche fornito l'indirizzo del valore di credibilità in questione;

[1] Anche se in una comunicazione è stato chiarito che la lunghezza della sequenza non può essere nulla, abbiamo deciso di gestire comunque questo caso per completezza.

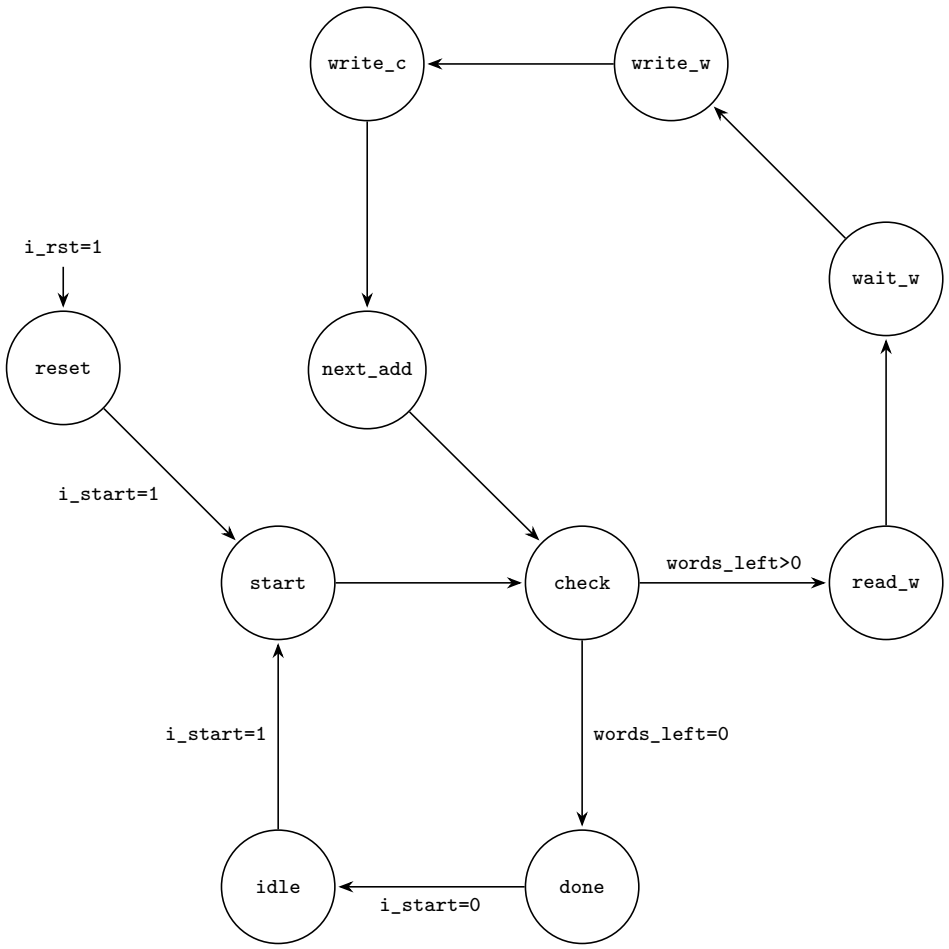

Figura 2: Diagramma degli stati della FSM

- `next_add`: stato in cui il modulo aggiorna l'indirizzo di memoria salvato e decrementa il contatore delle parole rimanenti;
- `done`: stato in cui il modulo segnala la fine della computazione, alzando `o_done`. La FSM entra in questo stato solo quando non ci sono altre parole da elaborare;
- `idle`: stato in cui il modulo abbassa `o_done` e resta in attesa di un nuovo segnale `i_start` alto per iniziare una nuova computazione (a quel punto la FSM va nello stato `start`). Si entra in questo stato solo quando viene abbassato `i_start`.

2.2 Processi

Il modulo è composto da tre processi:

- `state_transition`: questo processo si occupa di determinare, in base allo stato corrente e ai segnali di controllo, il prossimo stato della FSM.
- `output`: questo processo si occupa delle uscite del modulo verso la RAM. Gestisce le fasi di comunicazione con la memoria (tramite `o_mem_en` e `o_mem_we`), fornisce i valori di parola e credibilità elaborati completi di indirizzi (`o_mem_data` e `o_mem_addr`) e segnala la fine di una computazione alzando `o_done`;
- `registers`: implementa i registri interni del modulo, aggiornando i valori rispetto allo stato corrente (fase di computazione) e al caso specifico (non sempre i valori salvati vanno aggiornati).

Quindi, riassumendo, il componente sfrutta i tre processi per: gestire lo stato della FSM, gestire le uscite con la RAM, salvare e aggiornare i valori nei registri.

3 Risultati sperimentali

3.1 Sintesi

Il modulo è sintetizzabile correttamente. I risultati ottenuti attraverso l'utilizzo di Vivado sono stati analizzati per identificare le informazioni più significative e verificare il rispetto dei requisiti non funzionali.

In particolare, il modulo non presenta nessun *latch*, come si può vedere dalla figura 3.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	79	0	134600	0.06
LUT as Logic	79	0	134600	0.06
LUT as Memory	0	0	46200	0.00
Slice Registers	52	0	269200	0.02
Register as Flip Flop	52	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

Figura 3: Report componenti utilizzati

Inoltre, il requisito sul periodo di *clock* minimo di 20 ns è stato ampiamente rispettato, con uno *slack* di 16,204 ns, come mostrato in figura 4.

```
Timing Report

Slack (MET) :             16.204ns  (required time - arrival time)
  Source:                 current_address_reg[3]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns
  Destination:            current_address_reg[15]/D
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns
  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:        3.645ns  (logic 2.017ns (55.336%)  route 1.628ns (44.664%))
  Logic Levels:           5  (CARRY4=4 LUT5=1)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 22.100 - 20.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

Figura 4: Report di timing

Infine, in figura 5 è possibile notare come il consumo esergetico del modulo sia di soli 0,067 W.

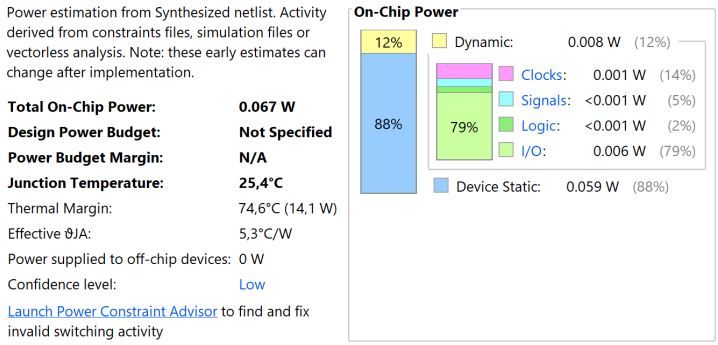

Figura 5: Report di consumo energetico

3.2 Simulazioni

Durante la fase di simulazione funzionale post-sintesi del progetto abbiamo testato il modulo in numerose situazioni. Partendo dal test di esempio fornito, la cui simulazione è mostrata in figura 6, abbiamo elaborato un generatore di test casuali; così facendo siamo stati in grado di testare sequenze a lunghezza e indirizzo di partenza variabili, nelle quali il valore di ogni elemento era puramente casuale.

Particolarmente degni di nota sono alcuni test – scritti ad hoc – sui casi limite, dei quali riportiamo qualche commento significativo.

Un caso di test di particolare importanza è quello in cui la sequenza è composta inizialmente da parole di valore diverso da zero, seguite da una sequenza di 33 o più zeri. Il componente gestisce correttamente questa possibilità, senza che i valori di credibilità escano mai dal loro dominio di definizione. Abbiamo verificato il corretto funzionamento del modulo anche in caso di sequenze composte da soli zeri.

Il modulo riesce a gestire anche più computazioni successive quando:

- le sequenze non sono separate da un segnale di reset;
- la prima viene interrotta bruscamente da tale segnale;
- la prima viene elaborata completamente e viene dato un segnale di reset ad anticipare la seconda.

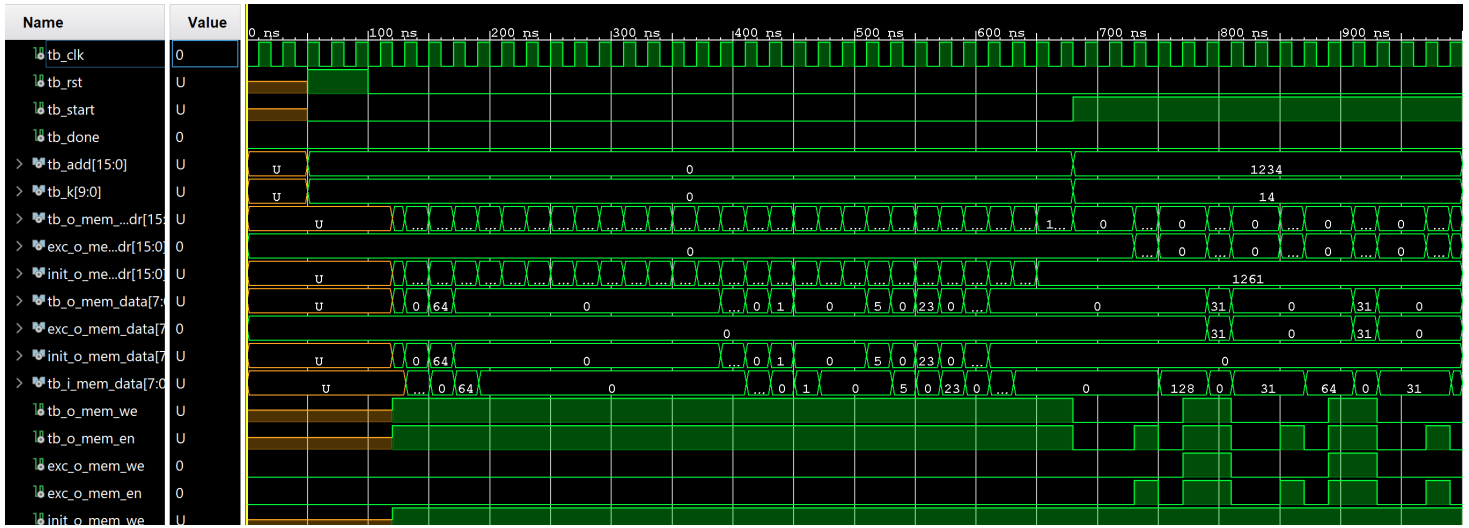

Figura 6: Simulazione con test di esempio

4 Conclusioni

Siamo complessivamente soddisfatti del nostro lavoro, avendo potuto constatare che il modulo hardware da noi elaborato si interfaccia correttamente con la RAM e si comporta come previsto in ogni caso di test, come accertato nelle simulazioni di pre- e post-sintesi.

Siamo contenti che il nostro componente sia sintetizzabile e che non sia necessario nessun *latch* a tal proposito, evidenziando una corretta gestione dei registri interni al componente. L'approccio *behavioral* da noi scelto si è rivelato essere affidabile e coinciso, rendendo chiara e lineare ogni analisi effettuabile sul codice.

Siamo altresì compiaciuti del basso consumo di energia del nostro componente, così come della sua efficacia (segnalata da un alto valore di *slack*) nello svolgere correttamente le operazioni.