POLITECNICO DI MILANO

Prova Finale (Progetto di Reti Logiche)

A.A. 2020/2021

Mauro Famà
CP: 10631287
Mat. 908861

Elia Fantini
CP: 10651951
Mat. 907960

Contenuti

1 Introduzione

1.1 Obiettivo del Progetto

L'obiettivo della prova finale è quello di implementare su FPGA un modulo in grado di leggere un'immagine di dimensione variabile (dimensione massima 128×128 pixel) da una memoria, elaborarla e infine scriverla in memoria. L'elaborazione consiste nell'applicazione di un algoritmo semplificato di equalizzazione ad un immagine in scala di grigi a 256 livelli.

1.2 Specifiche

L'immagine da elaborare dovrà essere letta da una memoria in cui sarà memorizzata sequenzialmente, pixel per pixel, in ordine di riga. Ogni pixel è rappresentato da un byte. A partire dall'indirizzo 0, i primi due byte dell'immagine definiscono rispettivamente la dimensione di colonna e la dimensione di riga; a partire dall'indirizzo 2, ogni byte corrisponderà ad un pixel dell'immagine, assumendo il valore del livello di grigio. L'immagine equalizzata dovrà essere memorizzata a partire dal primo indirizzo libero immediatamente dopo l'ultimo pixel dell'immagine originale, non riportando le dimensioni di colonna e riga. Nella seguente tabella è presente una schematizzazione della memoria.

Indirizzo	Valore
0	Numero-Colonne
1	Numero-Righe
2	Primo Pixel
...	...
1+(N-Col×N-Rig)	Ultimo Pixel
2+(N-Col×N-Rig)	Primo Pixel Eq.
...	...
1+(N-Col×N-Rig)×2	Ultimo pixel Eq.

1.3 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_data :     in std_logic_vector(7 downto 0);
        o_address :  out std_logic_vector(15 downto 0);
        o_done :     out std_logic;
        o_en :       out std_logic;
        o_we :       out std_logic;
        o_data :     out std_logic_vector (7 downto 0)
    );
end project_reti_logiche;
```

- `i_rst` è il segnale di `reset` che inizializza la macchina. Successivamente sarà pronta per ricevere il primo segnale di `i_start`, generato dal Test Bench. Per poter comunicare con la memoria, sia in lettura che in scrittura, `o_en` è il segnale di enable da dover alzare.
- `o_we` è il segnale di `write enable` da portare alto per la scrittura, mentre deve essere abbassato per la lettura.
- `o_address` è il segnale di uscita utilizzato per mandare alla memoria l'indirizzo su cui leggere/scrivere. Mentre `i_data` contiene i dati in arrivo dalla memoria in seguito ad una richiesta di lettura,
- `o_data` contiene i dati in uscita dal componente verso la memoria durante la fase di scrittura.
- `i_clk` è il segnale di `clock` in ingresso generato dal Test Bench e `o_done` è il segnale di uscita che comunica la fine dell'elaborazione.

2 Architettura

2.1 Scelte progettuali

Per implementare il modulo si è scelto di utilizzare una FSM (Finite State Machine). Per descrivere la FSM si è deciso di sfruttare tre processi: un processo `lambda` per la funzione di transizione, ovvero per gestire lo stato corrente e determinare lo stato successivo della macchina; un processo `delta` per la funzione di uscita, in cui si trova la logica di ogni stato della FSM e vengono eseguite le operazioni di manipolazione dei registri; uno `state register` per effettuare le transizioni tra stati, passando dallo stato corrente a quello successivo ad ogni fronte di salita del clock e gestendo eventuali segnali di reset asincrono.

2.2 Funzionamento

Quando il segnale i_start verrà portato alto, la macchina partirà con l'elaborazione. Inizialmente verranno letti i primi due byte della memoria, da cui ricaviamo il numero di colonne e di righe, quindi si procederà con il calcolo della differenza tra l'indirizzo di scrittura del pixel equalizzato e l'indirizzo del pixel originale nell'immagine in input, pari anche all'indirizzo di scrittura del primo pixel equalizzato meno i due byte di dimensione immagine. Successivamente verrà effettuata la ricerca del massimo e del minimo valore di grigio presenti nell'immagine in ingresso e ne verrà calcolata la differenza. Questo valore servirà per calcolare lo shift level opportuno per l'immagine. A questo punto verranno letti in ordine tutti i pixel dell'immagine: per ogni pixel verrà calcolato il nuovo valore equalizzato, se ne controllerà la correttezza e verrà scritto in memoria. Quando il processo sarà effettuato per tutti i pixel, verrà alzato il segnale o_done e, non appena i_start tornerà basso, la macchina si porterà in stato di reset per poter ricominciare la computazione di una nuova immagine.

2.3 Insieme degli stati

La macchina è costituita da 13 stati, di seguito una descrizione di ciascuno di essi.

2.3.1 INIT

Stato iniziale in cui la macchina attende di ricevere un segnale. Una volta ricevuto il segnale di START o di RESET non sarà più possibile tornare in questo stato.

2.3.2 RESET

La macchina viene portata in questo stato ogni qual volta venga alzato il segnale i_rst o alla fine di ogni computazione, con l'abbassarsi di i_start. Esso imposta i segnali ai valori di inizializzazione. La macchina rimane in questo stato fintantochè i_start non viene alzato.

2.3.3 START

Stato in cui o_en viene portato alto, abilitando la comunicazione con la memoria.

2.3.4 READ_N_COL

Stato in cui viene letto il byte che riporta il numero di colonne dell'immagine, memorizzandolo nel registro n_col.

2.3.5 CALCULATE_WRITE_ADDR

Stato in cui, leggendo il secondo byte della memoria e moltiplicandolo per n_col, si ottiene il numero di pixel dell' immagine salvandolo nel registro write_addr. L'effettivo indirizzo di scrittura dell'immagine equalizzata sarà write_addr + 2.

2.3.6 COMPARE_MAX_MIN

Stato in cui vengono ricavati il massimo e il minimo valore di grigio tra tutti i pixel dell'immagine, i due valori vengono memorizzati in due registri appositi, rispettivamente max_pixel_value e min_pixel_value. La macchina ritorna in questo stato fintantochè il counter che gestisce l'indirizzo corrente non diventa maggiore o uguale a write_addr, visitando quindi tutti i byte dell'immagine. Alla fine del calcolo l'indirizzo corrente corrisponderà al primo byte libero della memoria.

2.3.7 CALCULATE_DELTA_VALUE

In questo stato viene calcolata la differenza tra il valore più alto di grigio presente nell'immagine e quello più basso. Questo valore viene memorizzato nell'apposito registro delta_value.

2.3.8 CALCULATE_SHIFT_LEVEL

Per prima cosa in questo stato viene riportato l'indirizzo corrente, salvato nel registro curr_addr, alla posizione del primo pixel dell'immagine, in modo tale da partire con l'elaborazione di ogni pixel dallo stato successivo. In seguito, in base al delta value calcolato precedentemente, si ricava lo shift level da utilizzare nell'operazione di calcolo del nuovo valore del pixel equalizzato. Per ricavarlo sono stati utilizzati una serie di controlli condizionali annidati che, a seconda del valore di delta_value, assegnano al registro shift_level il livello appropriato per un' equalizzazione corretta dell'immagine in ingresso. Di seguito è riportata una tabella che associa ad ogni delta value possibile lo shift level corrispondente, utilizzata per formulare il costrutto condizionale presente nella logica di questo stato.

Delta Value	Shift Level
0	8
1	7
2	7
3	6
...	....
7	5
...	...
15	4
...	...
31	3
...	...
63	2
...	...
127	1
...	...
255	0

2.3.9 CALCULATE_NEW_PIXEL_VALUE

In questo stato viene letto il valore di un pixel dell'immagine originale, gli viene sottratto min_pixel_value, quindi viene eseguita un'operazione di shift a sinistra di shift_level bit. Il valore ricavato viene assegnato a new_pixel_value. Dopo questa operazione, viene incrementato l'indirizzo corrente fino all'indirizzo di memoria riservato al nuovo valore del pixel appena calcolato.

2.3.10 CHECK_NEW_PIXEL_VALUE

Questo stato verifica che il valore di new_pixel_value sia al massimo 255. Nel caso in cui fosse maggiore, viene portato a 255.

2.3.11 WRITE_NEW_PIXEL_VALUE

Viene portato alto il segnale o_we. Questo stato provvede a scrivere in memoria `new_pixel_value`.

2.3.12 DISABLE_WRITING

Viene portato basso il segnale o_we. L'indirizzo corrente viene portato al pixel dell'immagine successivo a quello appena elaborato e, in caso fosse l'ultimo, il prossimo stato viene posto uguale a DONE. In caso contrario si ritorna a CALCULATE_NEW_PIXEL_VALUE.

2.3.13 DONE

Viene portato basso il segnale o_en e alto il segnale o_done. Quest'ultimo indica la fine dell'elaborazione e della scrittura in memoria della nuova immagine equalizzata. Da questo stato si torna allo stato di RESET non appena i_start tornerà basso.

2.4 Diagramma della Macchina a Stati Finiti

In Fig. 1 è presente una rappresentazione grafica dell'automa a stati finiti. In ogni momento della computazione l'innalzamento del segnale i_rst comporta il ritorno allo stato di RESET, rendendolo dunque raggiungibile da ogni altro stato della FSM. Tale informazione non è stata riportata nel diagramma per motivi di chiarezza.

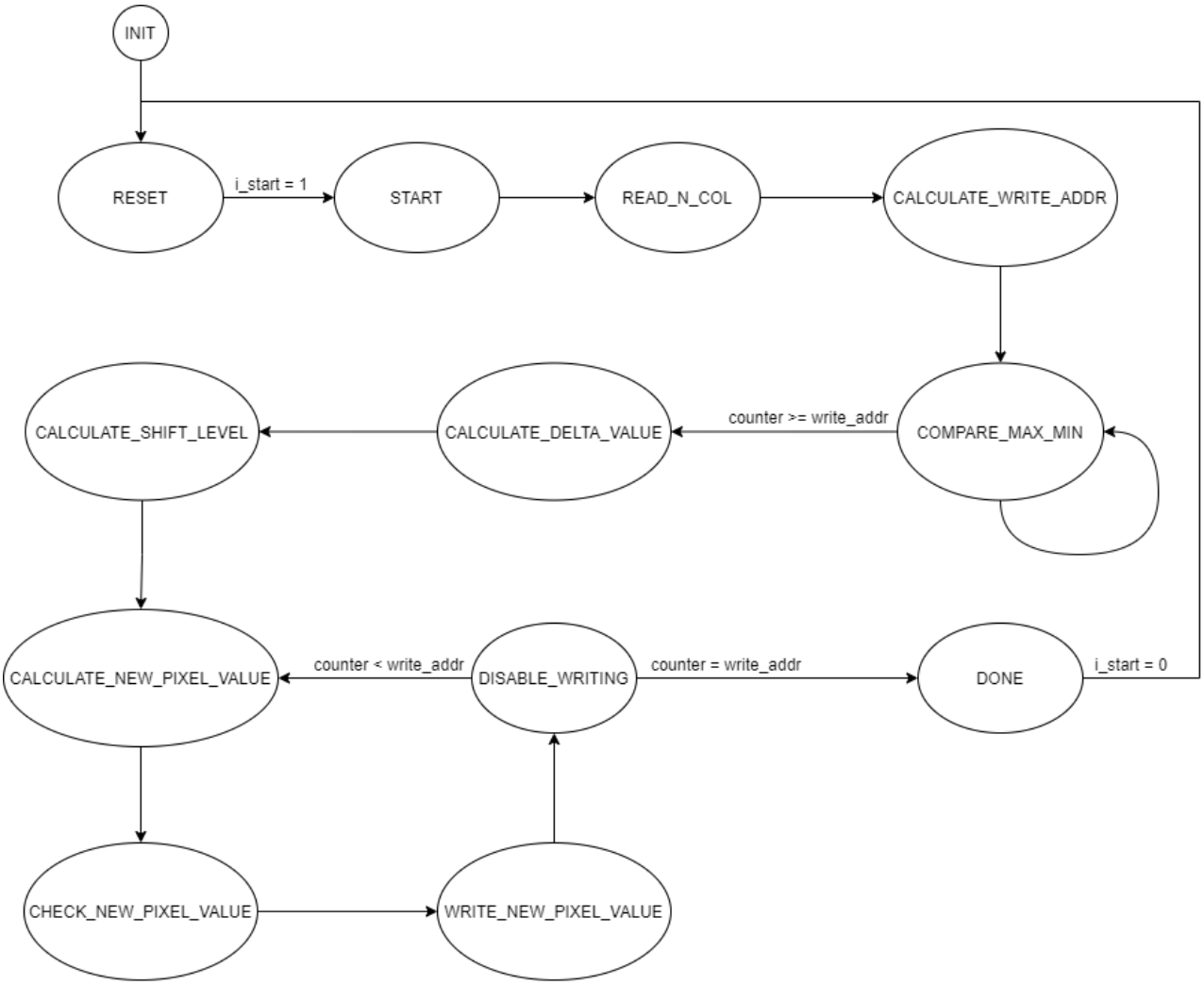

Figura 1: Rappresentazione grafica della macchina a stati finiti.

3 Risultati sperimentali

Per sintesi ed implementazione è stato simulato l'utilizzo di un'FPGA xc7a200tfbg484-1, del quale solo una minima parte è stata effettivamente utilizzata dal componente, come si può vedere in Fig. 2.

Resource	Utilization	Available	Utilization %
LUT	208	134600	0.15
FF	119	269200	0.04
IO	38	285	13.33

Figura 2: Utilization Report Summary

Secondo la specifica il componente deve avere un tempo di clock massimo di 100ns. A tale scopo è stato inserito un vincolo per i_clock di 100ns con duty cicle al 50%, ottenendo il timing report mostrato in Fig. 3 . Una volta eseguite sintesi ed implementazione è stato verificato che non avvenisse alcuna inferenza di latch e che nessun errore o avvertimento venisse sollevato dal programma.

Setup		**Hold**		**Pulse Width**	
Worst Negative Slack (WNS):	46,224 ns	Worst Hold Slack (WHS):	0,144 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	308	Total Number of Endpoints:	308	Total Number of Endpoints:	120

Figura 3: Timing Report Summary

4 Simulazioni

Il modulo è stato sottoposto a diversi test sviluppati per coprire ognuno un caso limite del problema proposto. In aggiunta, il modulo è stato sottoposto ad una serie di test bench casuali generati automaticamente, non riportati in questa sezione, ma utilizzati per verificare il corretto funzionamento del codice. Il componente passa con successo le simulazioni Behavioural e Post-Synthesis Functional di ognuno di essi e di seguito vengono riportati i risultati ottenuti nei più significativi test bench di corner cases e verifica di segnali:

4.1 Reset Asincrono

Il test verifica la corretta gestione di segnali di reset in momenti casuali durante la computazione. In Fig. 4 viene riportata la waveform generata dalla Post-Synthesis Simulation.

- Behavioral Simulation: 1495 ns
- Post-Synthesis Simulation: 1510100 ps

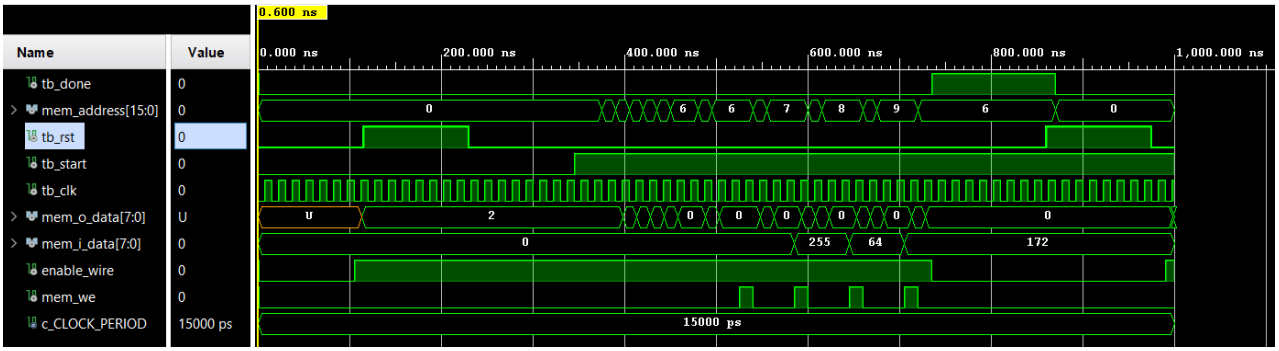

Figura 4: Waveform reset asincrono

4.2 Computazioni multiple

Il test verifica che il componente sia in grado di gestire correttamente più elaborazioni di immagini consecutivamente. In Fig. 5 viene riportata la waveform generata dalla Post-Synthesis Simulation.

- Behavioral Simulation: 3010 ns
- Post-Synthesis Simulation: 3035100 ps

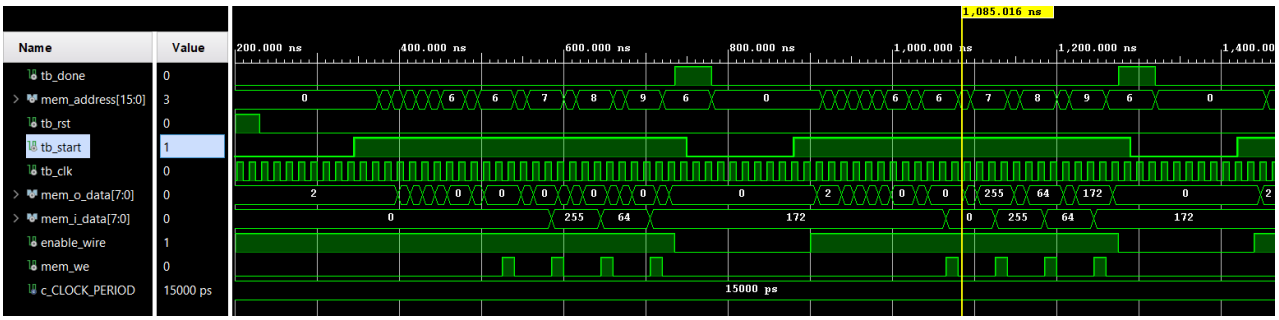

Figura 5: Waveform computazioni multiple

4.3 Zero pixel

Il test verifica che il componente sappia gestire il caso in cui l'immagine sia vuota, ovvero senza alcun pixel.

- Behavioral Simulation: 36053700 ps
- Post-Synthesis Simulation: 36054250100 ps

4.4 Un pixel

Il test verifica la corretta gestione di un'immagine composta da un solo pixel.

- Behavioral Simulation: 1285 ns
- Post-Synthesis Simulation: 1315100 ps

4.5 128x128 pixel

Il test verifica la corretta gestione di un'immagine composta dal massimo numero di pixel, 128x128. Per questa casistica in particolare sono stati creati diversi testbench. Di seguito vengono riportate le tempistiche di uno di essi.

- Behavioral Simulation: 1229365 ns
- Post-Synthesis Simulation: 1229380100 ps

4.6 Delta value minimo, pixel tutti 255

Il test verifica il comportamento del componente nel caso in cui il valore di tutti i pixel sia pari a 255.

- Behavioral Simulation: 1015 ns
- Post-Synthesis Simulation: 1030100 ps

4.7 Delta value minimo, pixel tutti 0

Il test verifica il comportamento del componente nel caso in cui il valore di tutti i pixel sia pari a 0.

- Behavioral Simulation: 1015 ns
- Post-Synthesis Simulation: 1030100 ps

4.8 Delta value massimo

Il test verifica il comportamento del componente nel caso in cui almeno un pixel abbia valore 0 ed un altro pixel abbia invece valore 255, portando delta_value ad assumere il suo valore massimo.

- Behavioral Simulation: 1015 ns
- Post-Synthesis Simulation: 1030100 ps

4.9 Pixel con potenze del due come valori

Il test verifica il comportamento del componente nel caso in cui i pixel assumano solo valori pari a potenze di due.

- Behavioral Simulation: 1240 ns
- Post-Synthesis Simulation: 1255100 ps

5 Conclusioni

Il codice è stato più volte modificato con lo scopo di diminuire il numero di stati della FSM e ottenere un Worst Negative Slack più contenuto, pur cercando di mantenere una corretta suddivisione delle funzioni eseguite da ciascuno stato. In particolare si è cercato di ottimizzare il cambiamento dell'indirizzo corrente nella memoria e conseguente lettura dei dati provenienti da essa, rispettando i tempi di propagazione necessari.
Per concludere, si ritiene che il componente rispetti le specifiche poste, rientrando largamente all'interno del massimo tempo di clock consentito. Non sollevando alcun tipo di errore e passando tutti i test a cui è stato sottoposto, si può affermare che il codice prodotto sia adeguato per l'applicazione dell'algoritmo di equalizzazione di immagini richiesto.