Prova Finale

Progetto di Reti Logiche

Matteo Pierini

Anno accademico 2020/2021 - Prof. Gianluca Palermo
Politecnico di Milano

Indice

1 Introduzione

1.1 Obiettivo del progetto

L'obiettivo della prova è la progettazione di un componente hardware, descritto in VHDL e sintetizzabile, che implementi una versione semplificata dell'algoritmo di equalizzazione dell'istogramma delle immagini.

1.2 Riepilogo della specifica

Il modulo può equalizzare immagini in scala di grigi a 256 livelli, di dimensioni massime di 128x128 pixel. Al componente vengono forniti in memoria le dimensioni dell'immagine e, in sequenza, i valori dei pixel. Il componente implementa il seguente algoritmo per calcolare il valore equalizzato di ogni pixel dell'immagine:

```
DELTA_VALUE = MAX_PIXEL_VALUE - MIN_PIXEL_VALUE
SHIFT_LEVEL = 8 - FLOOR(LOG2(DELTA_VALUE + 1))
TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL
NEW_PIXEL_VALUE = MIN(255, TEMP_PIXEL)
```

dove `MAX_PIXEL_VALUE` e `MIN_PIXEL_VALUE` rappresentano rispettivamente i valori dei pixel più chiari e più scuri, `CURRENT_PIXEL_VALUE` il valore del pixel da trasformare e `NEW_PIXEL_VALUE` il valore del pixel aggiornato.
Infine, l'immagine equalizzata viene scritta in memoria a partire dalla posizione immediatamente successiva all'immagine originale.

CURRENT_PIXEL_VALUE		
71	226	36
236	108	148
220	201	88

MIN_PIXEL_VALUE:	36
MAX_PIXEL_VALUE:	236
DELTA_VALUE:	200
SHIFT_LEVEL:	1

TEMP_PIXEL		
70	380	0
400	144	224
368	330	104

NEW_PIXEL_VALUE		
70	255	0
255	144	224
255	3255	104

Tabella 1: Esempio con un'immagine 3x3

1.3 Memoria

La memoria RAM connessa al modulo deve essere indirizzata al byte, secondo lo schema che segue:

indirizzo	**valore**	
0	larghezza immagine	input
1	altezza immagine	
2	valore pixel 0 immagine originale	
3	valore pixel 1 immagine originale	
	. . .	
$n+1$	valore pixel n-1 immagine originale	
$n+2$	valore pixel 0 immagine equalizzata	output
$n+3$	valore pixel 1 immagine equalizzata	
	. . .	
$2n+1$	valore pixel n-1 immagine equalizzata	

Figura 1: Schema della configurazione della memoria

dove n è pari al numero di pixel dell'immagine (*larghezza* $\cdot$ *altezza*), quindi i pixel 0 e $n-1$ sono rispettivamente il primo e l'ultimo.

1.4 Interfaccia del componente

Segnali di ingresso:

- `i_clk`: segnale di clock
- `i_rst`: segnale di reset asincrono, per l'inizializzazione del modulo
- `i_start`: segnale di start, per dare inizio all'elaborazione
- `i_data` (8 bit): valore letto dalla memoria

Segnali di uscita:

- `o_address` (16 bit): indirizzo di lettura/scrittura da/in memoria
- `o_data` (8 bit): valore da scrivere in memoria
- `o_en`: segnale di enable (scrittura o lettura) della memoria
- `o_we`: segnale di enable di scrittura in memoria
- `o_done`: segnale di fine elaborazione

2 Architettura

Il modulo è composto da due processi che implementano una *macchina a stati finiti*. Il processo `STATE_REG` si occupa di rispondere ad un eventuale segnale di reset asincrono e di aggiornare i vari flip-flop con i valori successivi, rappresenta dunque il funzionamento della parte di memoria interna del componente. Il processo `STATE_FUNC` rappresenta invece lo stato e, come descritto successivamente, gestisce la computazione, tutte le uscite e i segnali interni.
Prima di iniziare l'elaborazione si attende che il segnale `i_start` assuma un valore logico alto. Al termine della stessa, l'uscita `o_done` viene portata a `1` e si attende che il segnale di start torni a `0` prima di poter iniziare l'elaborazione successiva.
Come verrà precisato più avanti, ai fini dell'ottimizzazione, gli indirizzi della memoria a cui è necessario fare accesso vengono scritti sull'uscita `o_address` quando la macchina si trova nello stato precedente a quello in cui il valore è effettivamente utilizzato. Con questa scelta si riesce a non introdurre degli stati che altrimenti avrebbero come unico scopo quello di attendere la risposta da parte della memoria.

L'algoritmo implementato consiste dei seguenti passaggi:

1. lettura delle dimensioni dell'immagine e calcolo del numero di pixel
2. iterazione su tutti i pixel per trovare i valori di minimo e di massimo
    a) lettura nuovo pixel
    b) confronto con minimo e massimo già trovati
    c) eventuale aggiornamento di minimo e massimo
3. iterazione su tutti i pixel per calcolare e scrivere in memoria il valore aggiornato
    a) lettura nuovo pixel
    b) calcolo valore aggiornato
    c) scittura pixel aggiornato

2.1 Macchina a stati finiti

In figura 2 una rappresentazione grafica della macchina a stati implementata dal componente, i cui stati sono dettagliati di seguito. Sono state indicate qualitativamente le transizioni che dipendono dagli ingressi di comando.

2.1.1 Stato `READY`

Il componente, una volta inizializzato, si trova in questo stato e vi rimane fino a quando viene alzato il segnale di `i_start`. Il segnale `o_address` assume già come valore il primo indirizzo della memoria, così da poter leggere, quando l'elaborazione viene avviata, la larghezza dell'immagine durante il prossimo periodo di clock.

2.1.2 Stato `READ_X_DIM`

Viene letto il valore della larghezza dell'immagine e salvato (al prossimo segnale di clock) nel registro `pixel_count`. Analogamente allo stato precedente, `o_address` assume il valore del secondo indirizzo di memoria, per poter leggere l'altezza dell'immagine.

2.1.3 Stato `READ_Y_DIM`

Viene letto il valore dell'altezza dell'immagine e moltiplicato per la larghezza precedentemente salvata, così che `pixel_count` sia pari al numero di pixel dell'immagine. Viene inizializzato il contatore `counter` in modo da poter iterare il prossimo stato per ogni pixel. `o_address` viene posto all'indirizzo della memoria che contiene il valore del primo pixel.

2.1.4 Stato `FIND_MIN_MAX`

Si attraversa questo stato per ogni pixel dell'immagine, grazie al contatore `counter` che, ad ogni iterazione, viene incrementato e confrontato con `pixel_count`.
Per ogni iterazione viene letto un nuovo pixel, il quale è confrontato con `min_pixel_value` e `max_pixel_value` che, eventualmente, vengono aggiornati. `o_address` viene posto all'indirizzo necessario per la lettura del pixel successivo, calcolato a partire dal valore di `counter`.
Esauriti tutti i pixel e ottenuti i valori di minimo e di massimo, `counter` e `o_address` vengono inizializzati per iniziare una nuova iterazione sull'intera immagine.

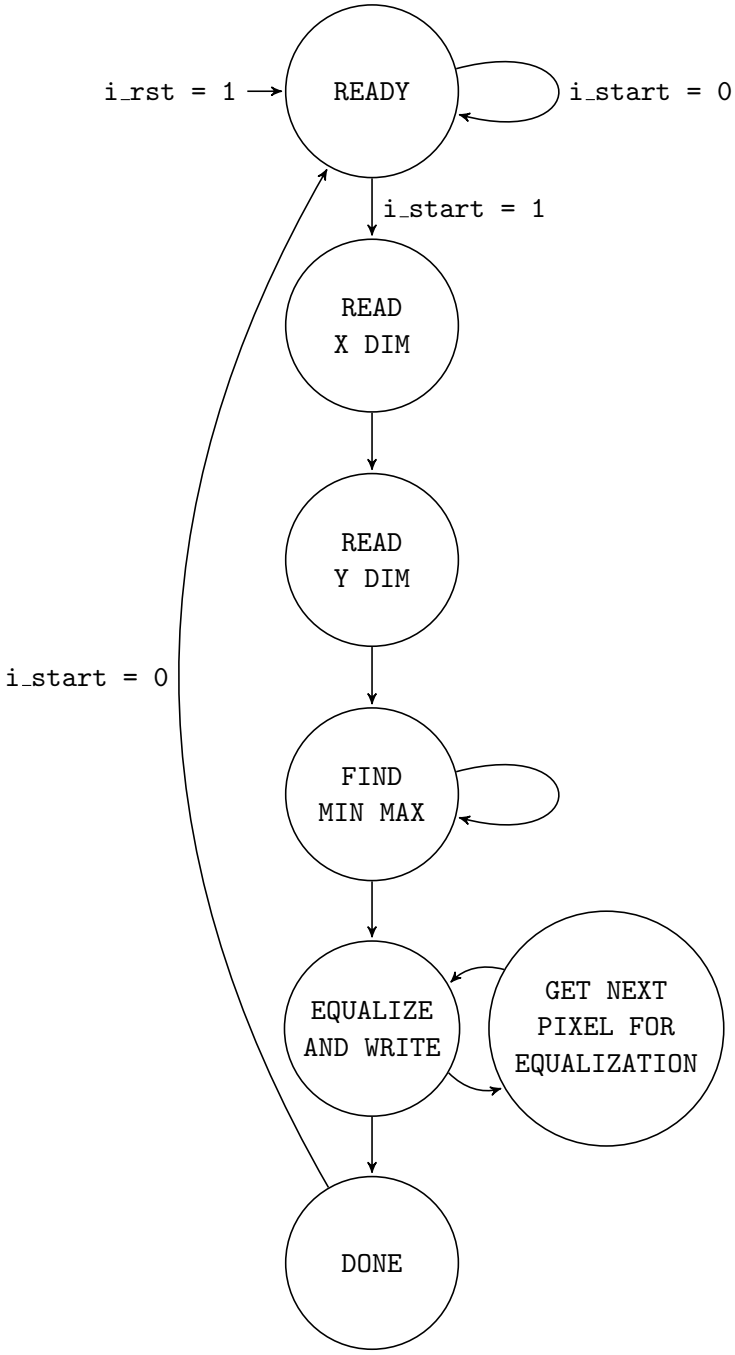

Figura 2: Grafo della macchina a stati

2.1.5 Stato EQUALIZE_AND_WRITE

Per ogni pixel dell'immagine si attraversa il ciclo di stati composto da EQUALIZE_AND_WRITE e GET_NEXT_PIXEL_FOR_EQUALIZATION.
In questo stato viene letto il valore del pixel originale, applicato l'algoritmo con i parametri acquisiti precedentemente, e preparate le uscite o_data, o_address, o_we per la scrittura del nuovo pixel, che avverrà effettivamente in memoria solo durante lo stato GET_NEXT_PIXEL_FOR_EQUALIZATION.
Quando sono stati scritti tutti i nuovi valori dei pixel si passa allo stato DONE.

2.1.6 Stato GET_NEXT_PIXEL_FOR_EQUALIZATION

Questo stato permette alla RAM di scrivere il valore aggiornato fornito precedentemente, mentre o_address e o_we vengono preparate per la lettura del pixel successivo e counter viene incrementato. Quindi si torna allo stato EQUALIZE_AND_WRITE.

2.1.7 Stato DONE

L'elaborazione è terminata, viene alzato il segnale o_done e disabilitata la memoria ponendo o_en a 0. Quando viene abbassato i_start, il modulo esce da questo stato e si sposta nello stato READY.

3 Risultati sperimentali

3.1 Sintesi

Il componente risulta correttamente sintetizzabile, producendo il seguente schema:

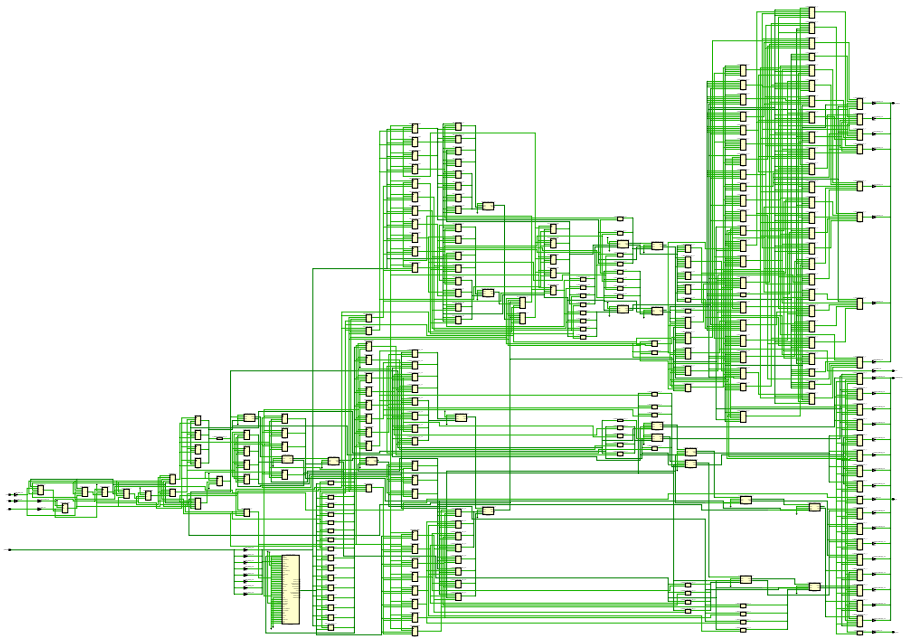

Figura 3: Schema del componente sintetizzato

3.2 Testing

Il corretto funzionamento è stato verificato tramite i seguenti test bench, sia in simulazione *behavioral* che in simulazione *post-synthesis*. Per ogni test è indicato anche il risultato atteso.

- immagine già equalizzata (almeno uno 0 e un 255 tra i pixel originali): immagine scritta uguale all'originale
- dimensioni dell'immagine nulle (immagine 0x0): nessuna scrittura in memoria
- dimensioni dell'immagine massime (immagine 128x128): equalizzazione corretta

- tutti i valori dei pixel uguali: `DELTA_VALUE` pari a 0, immagine di output composta da tutti pixel con valore 0
- valori dei pixel sempre crescenti/decrescenti (per sollecitare la ricerca del massimo/minimo): equalizzazione corretta
- reset asincrono: interruzione dell'elaborazione e ritorno immediato allo stato `READY`
- test bench casuali, anche composti da più immagini consecutive, di dimensioni variabili (prodotti da un generatore automatico scritto in C)

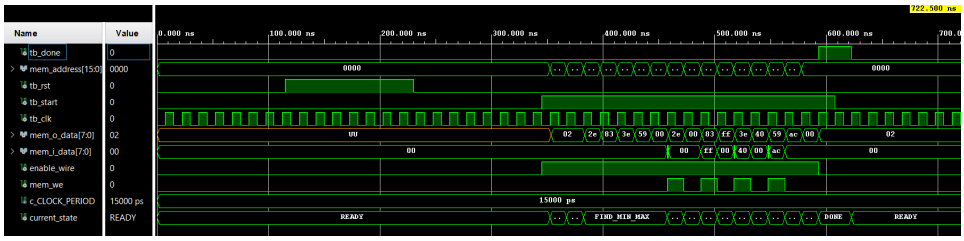

Figura 4: Esecuzione di un test bench di esempio, immagine 2x2 e periodo di clock di 15 ns.

4 Conclusioni

Il componente progettato supera correttamente i test indicati precedentemente e rispetta le specifiche richieste.

4.1 Prestazioni ottenute

Di seguito sono indicati i tempi di elaborazione nei due casi limite, entrambi eseguiti con un periodo di clock di 15 ns:

- immagine vuota (0x0): **120 ns**
- imagine 128x128: **737,4 µs**

In generale, il tempo necessario all'elaborazione dipende linearmente dalla dimensione dell'immagine. Più precisamente, sono necessari $8 + 3n$ cicli di clock, dove n rappresenta il numero di pixel dell'immagine.