Progetto finale di Reti Logiche

Prof. Gianluca Palermo - Anno di corso 2020-21

Francesco Pastore - Codice persona: 10629332

Indice

1 Introduzione

Il progetto richiede l'implementazione in VHDL dell'algoritmo di equalizzazione dell'istogramma di un'immagine. Questo metodo di elaborazione permette di aumentare il contrasto di un'immagine andando a distribuire su tutto lo spetttro, in modo bilanciato, i valori di intensità precedentemente vicini. In particolare viene richiesta l'implementazione di una versione semplificata applicata solo ad immagini in scala di grigi (0-255) e grandi al massimo 128x128 pixel.

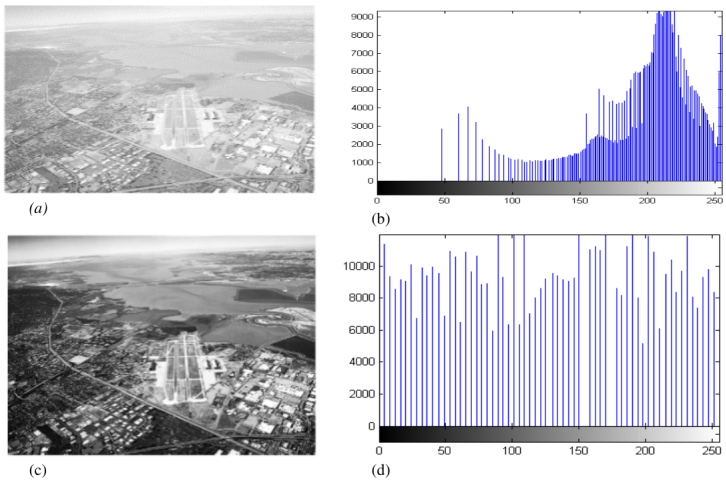

Figura 1: Esempio di equalizzazione dell'istogramma di un'immagine. [1]

1.1 Specifiche di progetto

L'interfaccia del componente è stata definita nella specifica con i relativi segnali di ingresso e di uscita. Oltre a questo, è stata definita la struttura della memoria e l'indirizzamento dei dati. All'indirizzo zero è possibile trovare il numero di colonne, seguito all'indirizzo uno da quello di righe. Dall'indirizzo due iniziano invece i valori dei singoli pixel dell'immagine fino alla posizione NUM_COLS * NUM_ROWS + 1. La scrittura dei pixel equalizzati deve avvenire invece dall'indirizzo immediatamente successivo all'ultimo pixel dell'immagine.

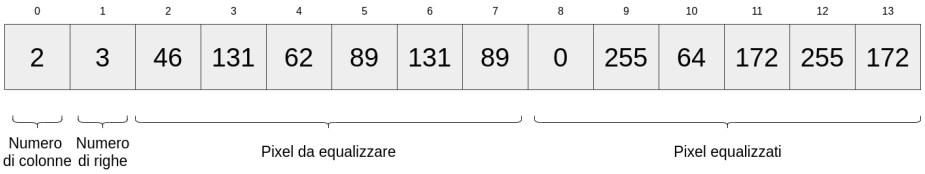

Figura 2: Dettagli della struttura della memoria

1.2 Algoritmo in breve

Di seguito una breve descrizione dei punti principali dell'algoritmo da implementare. Fare riferimento alle formule per il calcolo dei valori considerati.

1. Lettura del numero di colonne.
2. Lettura del numero di righe.
3. Verificare che l'immagine non sia vuota, altrimenti terminare l'esecuzione.
4. Lettura dei pixel dell'immagine cercando il valore minimo e massimo.
5. Calcolare il delta_value e il relativo shift_level.
6. Seconda lettura dell'immagine con equalizzazione dei pixel.
7. Scrittura in memoria dei nuovi valori dei pixel.

```
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)
```

Figura 3: Formule dell'algoritmo di equalizzazione fornite nella specifica.

1.3 Note di implementazione

Rispetto all'algoritmo descritto in precedenza, l'implementazione in VHDL richiede alcune modifiche.

- La lettura dalla memoria non è istantanea, ma necessita di un ciclo di clock di attesa dopo aver effettuato la richiesta. In particolare nel componente è stato implementato lo stato MEM_WAIT.
- L'assegnamento di valori ai segnali non è immediato, ma avviene al ciclo di clock successivo. Per questo motivo alcune operazioni vengono eseguite in stati successivi.
- Non è possibile assegnare ad un segnale esso stesso seppure con modifiche. Questo richiede, ad esempio per il contatore, di utilizzare un altro segnale come appoggio per effettuare l'incremento.

2 Architettura

2.1 Segnali utilizzati

2.1.1 num_cols, num_pixels, count e tmp_count

Per eseguire la lettura dell'immagine è necessario conoscere la dimensione, sfruttando il numero di colonne e di righe e un relativo contatore. In VHDL non essendo possibile assegnare un segnale a se stesso, è stato necessario aggiungere un contatore d'appoggio tmp_count. Il range di count, temp_count e num_pixels è pari a 0-16384 perchè ogni immagine può essere grande al massimo 128x128.

2.1.2 min_pixel_value e max_pixel_value

Questi segnali contengono, dopo la prima lettura dell'immagine, il valore del pixel massimo e minimo necessari per il calcolo delle varie formule di equalizzazione. Sono stati dichiarati di tipo INTEGER con range 0-255 perchè su di essi vengono effettuare diverse operazioni aritmetiche e logiche.

2.1.3 state_next, state_after_wait e state_after_read

Per le transizioni tra i vari stati sono stati defini alcuni segnali che contengono gli stati prossimi. Di specifico state_next contiene sempre lo stato immediatamente successivo, state_after_wait il seguente a MEM_WAIT mentre state_after_read lo stato successivo a READ_PIXEL.

2.1.4 shift_level e overflow_threshold

Dopo aver calcolato minimo e massimo è possibile determinare il valore del delta_value e del relativo shift_level. In particolare oltre a questo valore è stato aggiunta anche la soglia di overflow utile per controllare il valore da equalizzare prima di effettuare lo shift.

2.1.5 pixel_value e new_pixel_value

Entrambi questi segnali contengono valori di pixel dell'immagine e su di essi vengono effettuate diverse operazioni, per questo motivo sono di tipo INTEGER con range 0-255. In particolare pixel_value contiene il valore di ciascun pixel letto mentre il new_pixel_value serve come appoggio per il calcolo del pixel equalizzato.

```
SIGNAL min_pixel_value : INTEGER RANGE 0 TO 255;
SIGNAL max_pixel_value : INTEGER RANGE 0 TO 255;

SIGNAL num_cols : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL num_pixels : INTEGER RANGE 0 TO 13684;
SIGNAL count : INTEGER RANGE 0 TO 13684;
SIGNAL tmp_count : INTEGER RANGE 0 TO 13684;

SIGNAL pixel_value : INTEGER RANGE 0 TO 255;
SIGNAL new_pixel_value : INTEGER RANGE 0 TO 255;

SIGNAL shift_level : INTEGER RANGE 0 TO 7;
SIGNAL overflow_threshold : INTEGER RANGE 0 TO 255;

SIGNAL state_next : TYPE_STATE;      You, a day ago
SIGNAL state_after_wait : TYPE_STATE;
SIGNAL state_after_read : TYPE_STATE;
```

Figura 4: Dichiarazione dei segnali utilizzati

2.2 Descrizione degli stati

Il modulo è stato realizzato come una macchina a stati, in particolare comprende 14 stati descritti di seguito.

2.2.1 RESET

Lo stato di RESET è lo stato iniziale della macchina ed è l'unico raggiungibile da tutti gli altri. Quando il componente riceve un segnale di i_rst alto ferma l'esecuzione per poi ripartire dallo stato di reset. La macchina esce da questo stato solo con il segnale i_start alto.

2.2.2 READ_COLS_REQ

Nel primo byte della memoria è salvato il numero di colonne dell'immagine. Questo stato si occupa di effettuare la relativa richiesta di lettura. Essendo una lettura è necessario attendere che la memoria elabori la richiesta, per questo motivo lo stato successivo è MEM_WAIT.

2.2.3 READ_COLS

Dopo aver effettuato la richiesta di lettura nello stato READ_COLS_REQ in questo stato la macchina legge il numero di colonne passatole dalla memoria nel bus i_data.

2.2.4 READ_ROWS_REQ

Il secondo elemento in memoria dopo il numero di colonne è il numero di righe. Anche in questo caso è necessario effettuare la richiesta di lettura, aspettare un ciclo di clock nello stato MEM_WAIT e solo dopo leggere il valore richiesto.

2.2.5 READ_ROWS

Dopo aver effettuato la richiesta di lettura in READ_ROWS_REQ e aspettato per l'elaborazione da parte della memoria in MEM_WAIT in questo stato viene letto il numero di righe passato al componente tramite i_data. In questo caso viene calcolato direttamente il numero di pixel che costituiscono l'immagine moltiplicando il valore ricevuto per il numero di colonne richiesto in precedenza.

2.2.6 READ_PIXELS_START

In questo stato viene inizializzato il contatore e i segnali di minimo e massimo prima di effettuare la prima scansione dell'immagine. Il minimo viene settato a 255 che corrisponde al più alto valore possibile, il massimo invece a zero che rappresenta rispettivamente quello più basso. Viene controllato inoltre che non sia stata data un'immagine vuota, altrimenti si passa direttamente allo stato di DONE.

2.2.7 READ_PIXEL_REQ

Dopo aver calcolato il numero di pixel contenuti nell'immagine grazie al numero di colonne e di righe, è possibile leggerli scansionandola dall'inizio alla fine. In questo stato viene quindi settato l'indirizzo per la lettura del prossimo pixel che verrà poi letto nello stato READ_NEXT_PIXEL. Il contatore necessario per la lettura viene incrementato in questo stato sfruttando un segnale temporaneo d'appoggio.

2.2.8 READ_PIXEL

Dopo aver effettuato la richiesta di lettura e aspettato l'elaborazione da parte della memoria, in questo stato la macchina legge il pixel passatole nell'ingresso i_data. Viene inoltre salvato il valore del contatore, che era stato incrementato nello stato precedente sfruttando un segnale d'appoggio.

2.2.9 MEM_WAIT

La memoria richiede un ciclo di clock per l'elaborazione di una richiesta di lettura. Questo stato serve quindi come attesa dopo aver settato o_addr e o_en.

2.2.10 CHECK_MIN_MAX

La prima scansione serve per trovare i valori minimi e massimi dei pixel dell'immagine, in modo da poter poi effettuare l'equalizzazione. In questo stato viene quindi controllato ciascun pixel dopo la prima lettura e confrontato con i valori di massimo e minimo temporanei.

2.2.11 WRITE_START

Una volta effettuata la prima scansione e aver trovato quindi il massimo e il minimo, è possibile calcolare il delta_value dato dalla differenza dei due valori. Tramite uno switch e le relative soglie, viene determinato lo shift_level e il relativo overflow_threshold. Per poter effettuare la seconda scansione, in questo stato viene inizializzato nuovamente il contatore.

2.2.12 EQUALIZE_PIXEL

Per evitare di effettuare più volte lo stesso calcolo, in questo stato viene salvata nel segnale new_pixel_value la differenza tra il valore di ciascun pixel e il relativo minimo dopo ogni lettura.

2.2.13 WRITE_NEW_PIXEL

È in questo stato che la macchina scrive il valore del pixel equalizzato facendo attenzione ad effettuare lo shift solo quando non c'è overflow. A questo fine viene

sfruttato il valore di soglia definito nello stato WRITE_START sulla base del delta_value.

2.2.14 DONE

È lo stato finale in cui giunge la macchina al termine di un'esecuzione completa. Viene settato o_done alto e lo stato successivo è quello di RESET, in modo che il componente rimanga pronto per un'altra possibile esecuzione.

2.3 Diagramma degli stati

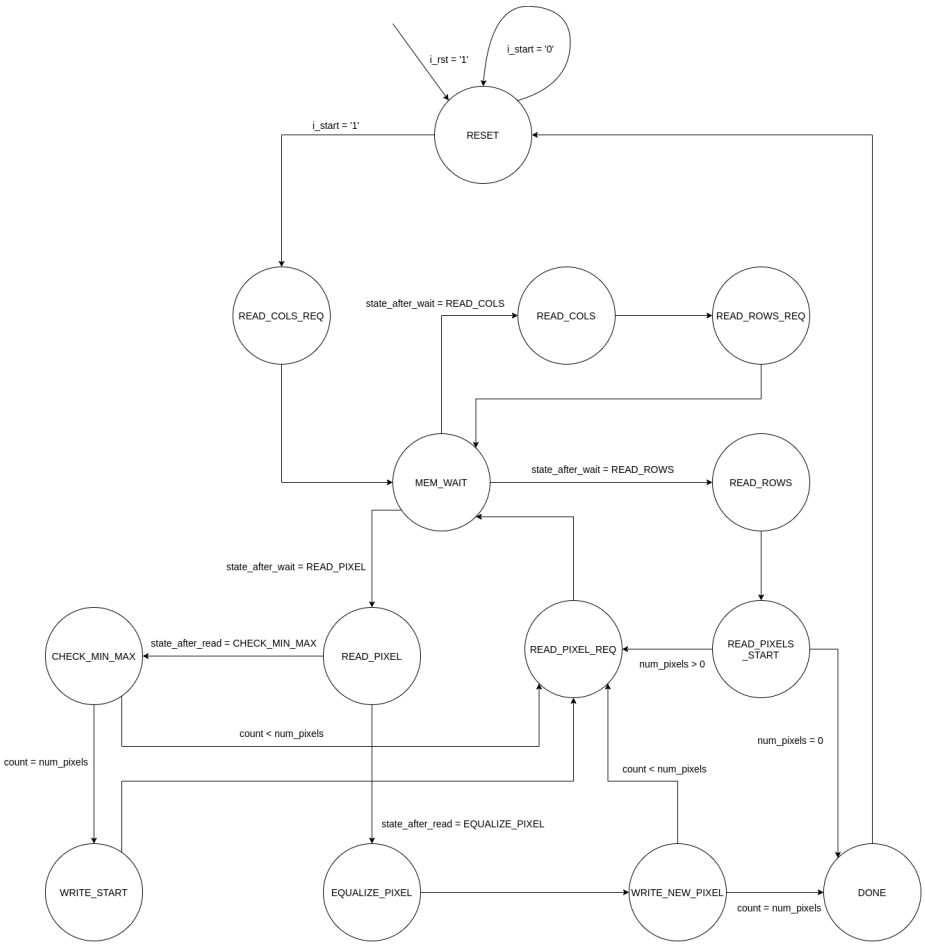

Figura 5: Diagramma della macchina a stati, in figura sono state specificate le condizioni di transizione principali.

3 Risultati sperimentali

3.1 Simulazioni significative

3.1.1 Simulazione standard

In questa simulazione è stato effettuato un test in una situazione standard. Da notare l'inizio dell'esecuzione solo dopo aver ricevuto il segnale di reset alto, poi abbassato e seguito da un segnale di start alto.

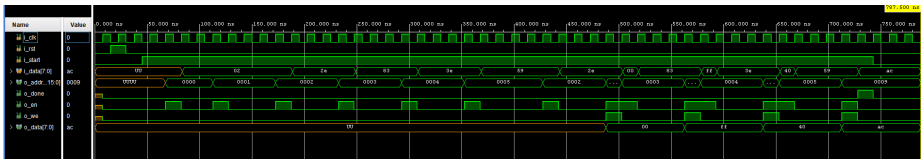

Figura 6: Simulazione standard

3.1.2 Simulazione con immagine vuota

Si può vedere come nel caso di un'immagine vuota, dopo aver letto il numero di colonne e quello di righe, la macchina termini immediatamente l'esecuzione. Da notare che il segnale di enable viene alzato soltanto due volte, appunto quelle necessarie per leggere solo le dimensioni.

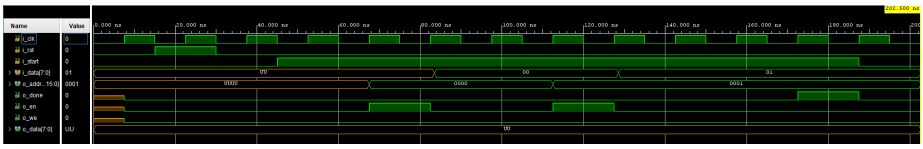

Figura 7: Simulazione con immagine vuota

3.1.3 Simulazione con segnale di start ritardato

Il componente prima di iniziare l'elaborazione deve attendere da specifica il segnale di start ad alto. È possibile vedere nell'immagine che esso rispetta correttamente l'attesa.

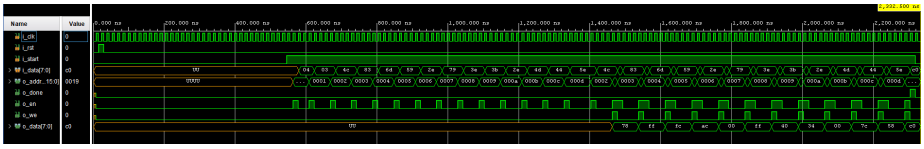

Figura 8: Simulazione con segnale di start ritardato

3.1.4 Simulazione con reset asincrono

Tra le specifiche di progetto è presente la richiesta che la macchina possa essere resettata in qualsiasi momento, per poi reiniziare una nuova elaborazione. Da notare quindi nell'immagine che dopo aver ricevuto il segnale di reset, la macchina termini l'esecuzione per poi riprenderla una volta letto un segnale di start alto.

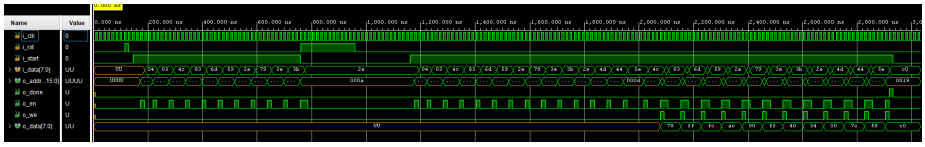

Figura 9: Simulazione con reset asincrono

3.1.5 Simulazione con immagini multiple

In questa simulazione si è testato l'elaborazione di immagini multiple controllando che la macchina rispetti correttamente il protocollo specificato. E possibile notare infatti che la macchina attende che il segnale di start venga alzato nuovamente prima di iniziare una nuova elaborazione. In particolare non è necessario un segnale di reset per far partire la nuova elaborazione.

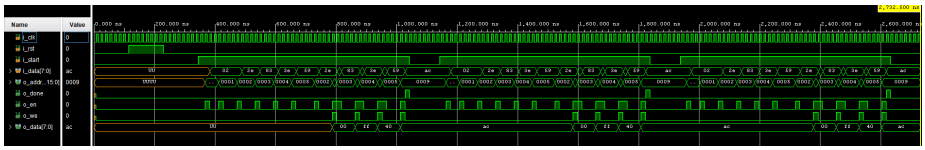

Figura 10: Simulazione con immagini multiple

3.1.6 Altre simulazioni

Oltre alle simulazioni riportate, sono stati scritti altri testbench per effettuare ulteriori controlli di correttezza.

- Simulazioni per controllare il calcolo di tutti i diversi delta_value e relativi shift_level, con verifica che venga gestito correttamente lo shift.
- Immagini con pixel tutti dello stesso valore, in questo caso il delta_value è pari a zero e anche i new_pixel_value saranno a loro volta nulli.
- Immagini contenenti pixel con valori già spalmati su tutto il range 0-255. In questo caso dovranno essere riscritti gli stessi identici valori dato che lo shift_level è pari a zero.

3.2 Report di sintesi

Di seguito è possibile visionare i principali risultati della sintesi del componente. Rispetto al report di utilizzo l'elevato numero di LUT e FF è dovuto principalmente alla moltiplicazione per il calcolo del numero di pixel e allo shift per l'equalizzazione dei pixel. Entrambe però sono operazioni fondamentali per la risoluzione dell'algoritmo e non sono stati trovati altri modi per ottimizzarle.

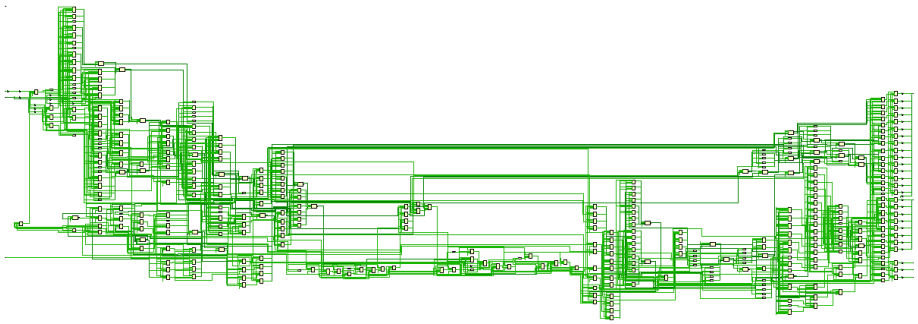

Figura 11: Schema del componente sintetizzato.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	93,856 ns	Worst Hold Slack (WHS):	0,148 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:	271	Total Number of Endpoints:	271	Total Number of Endpoints:	128

All user specified timing constraints are met.

Figura 12: Riassunto del report di timing

Resource	Utilization	Available	Utilization %
LUT	193	134600	0.14
FF	127	269200	0.05
IO	38	285	13.33

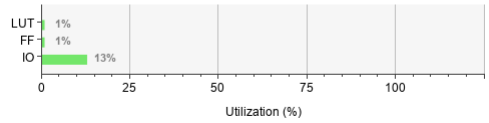

Figura 13: Riassunto del report di utilizzo

4 Conclusioni

Il componente passa correttamente i test mostrati sia in behavioural che in post sintesi rispettando, almeno per i casi proposti, i requisiti della specifica. Sono stati scritti anche altri testbench per verificare il funzionamento in situazioni diverse e nei rispettivi casi limite. Per evitare di effettuare lo shift su 16 bit e poi controllarne il risultato, è stata implementata una soglia di overflow. In questo modo basta eseguire un confronto su 8 bit prima di shiftare il valore di new_pixel_value. Le operazioni di moltiplicazione e il calcolo di shift sono quelle più onerose, ma non è stato possibile trovare soluzioni alternative. Maggiori ottimizzazioni si potrebbero quindi effettuare trovando metodi migliori e meno onerosi.

Riferimenti bibliografici

[1] Yogendra P. S. Maravi Omprakash Patel e Sanjeev Sharma. «Comparative study of histogram equalization based image enhancement techniques for brightness preservation and contrast enhancement». In: *Signal Image & Processing : An International Journal* 4.5 (2013). Cornell University Open Archive. URL: `https://arxiv.org/pdf/1311.4033.pdf`.