Relazione prova finale Reti Logiche

Vlad Cimpeanu: 10606922
Danilo Castiglia: 10685118

09/03/2021

POLITECNICO
MILANO 1863

1 Introduzione

Lo scopo di questo progetto è di programmare una FPGA mediante il linguaggio VHDL, al fine di **equalizzare** delle immagini date in ingresso. Queste ultime sono memorizzate in una memoria ram di tipo sincrono.
Le immagini sono in **scala di grigi** e ogni pixel assume valori da 0 a 255, dove 0 corrisponde al colore nero e 255 al colore bianco. La dimensione massima di ciascuna immagine è di 128 pixel in larghezza e 128 pixel in altezza, per un peso totale di 16KB.
Il modulo progettato, si aspetta di trovare in memoria nei primi due indirizzi la dimensione rispettivamente di larghezza e altezza, seguite da tutti i pixel dell'immagine, riga per riga. Data l'immagine in ingresso, il componente effettua alcune semplici operazioni e restituisce l'immagine aumentandone il contrasto.
La tecnica che viene utilizzata è l'**equalizzazione dell'istogramma**, che consiste nello spalmare i valori di intensità dell'immagine su tutto l'intervallo (da 0 a 255).
Di seguito è riportato un esempio di immagine che è stata realmente[1] equalizzata dal modulo sintetizzato.

(a) Immagine da equalizzare. (b) Immagine equalizzata.

Figure 1: Esempio di equalizzazione di un'immagine.

2 Architettura

2.1 Scelte progettuali

Per la realizzazione di questo progetto si è deciso di dividere **unità di elaborazione** e **unità di controllo**, facendo uso di un datapath e di una macchina a

[1] Per vedere in azione il modulo sintetizzato, è stato creato uno script in python che carica un'immagine nella ram del modulo FPGA e, una volta processata, renderizza l'immagine equalizzata presente in ram per poter osservare il risultato.

stati di Moore. Il datapath utilizza una rete combinatoria per il calcolo dei logaritmi, sommatori, sottrattori, comparatori, multiplexer e registri, con lo scopo di elaborare tutti i calcoli richiesti dalla macchina a stati.

2.2 Datapath

Row & Column counter

Il modulo Row & Column counter è stato ideato con lo scopo di riconoscere la fine dell'immagine da leggere che si trova memorizzata all'interno della RAM. Memorizzando le dimensioni di larghezza e altezza nei rispettivi registri e utilizzando dei contatori che tengono traccia delle righe e delle colonne per riga già lette è possibile parallelizzare la ricerca della fine dell'immagine con la lettura della stessa.
Così facendo si va a ricreare quello che in programmazione è definito come nested loop:

```
address = 2
for column in range(column_max):
    for row in range(row_max):
        address += 1
```

Listing 1: Nested loop in pseudocodice.

La scelta di adottare questa tecnica è stata dettata dall'intenzione di voler modellare un circuito efficiente. Nel caso in cui si fosse deciso di seguire una via più semplice, ovvero l'utilizzo di un moltiplicatore, si sarebbe ottenuto un aumento significativo del critical path diventando un collo di bottiglia per l'intero circuito.
Per migliorare l'efficienza del componente, si è scelto di verificare che le dimensioni dell'immagine da caricare siano diverse da 0 e in caso contrario di segnalare questa anomalia alla FSM che in seguito gestirà in maniera appropriata tale situazione. L'assenza dei comparatori (utilizzati a tale scopo) non preclude il funzionamento corretto del circuito, tuttavia nel caso pessimo si effettuano più di 2^{16} operazioni elementari inutili.

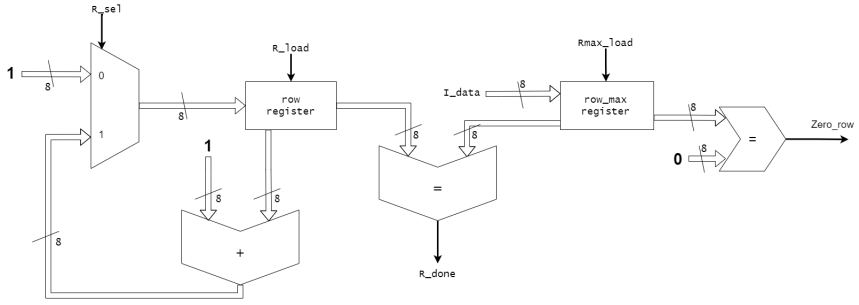

Figure 2: Rappresentazione schematica del contatore di righe.

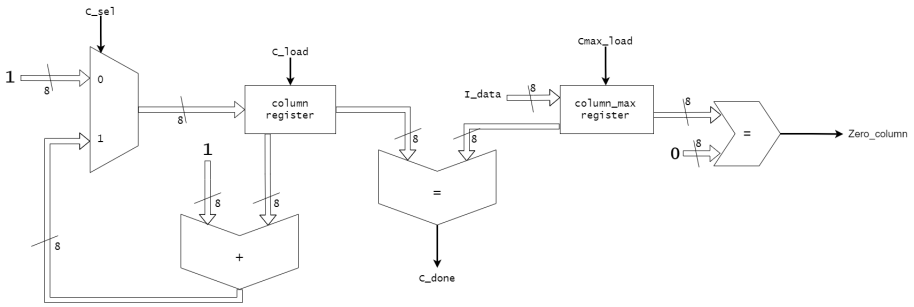

Figure 3: Rappresentazione schematica del contatore di colonne.

Address Counter

Il modulo Address Counter è stato sviluppato per richiedere gli indirizzi alla RAM, sia in fase di lettura che in fase di scrittura.
Come si può notare dalla Figura 4 il modulo è principalmente formato da due contatori e da un multiplexer a 4 ingressi. Il multiplexer seleziona quale valore dell'indirizzo o_address debba essere mandato in uscita:

- l'ingresso 0 restituisce il valore salvato nell'address register;
- l'ingresso 1 restituisce il valore salvato nell'address2 register;
- l'ingresso 2 restituisce l'indirizzo 0x00, utilizzato per richiedere la larghezza dell'immagine;
- l'ingresso 3 restituisce l'indirizzo 0x01, utilizzato per richiedere l'altezza dell'immagine.

In fase di lettura viene utilizzato solo address register dove viene memorizzato l'indirizzo del prossimo pixel da leggere, mentre nella fase di scrittura viene sfruttato anche address2 register.
Al termine della fase di lettura, address register contiene già il primo indirizzo dove scrivere il nuovo pixel, mentre address2 register viene inizializzato al valore dell'indirizzo contenente il primo pixel originale, infine viene salvato il valore di address register in un nuovo registro chiamato 'end register'. Durante la fase di scrittura i valori di address register e address2 register vengono incrementati in parallelo fino a quando address2 register contiene lo stesso valore di end register, in quanto esso indica che tutti i pixel originali sono già stati letti e quindi trasformati.

Max & Min calculator

Questo modulo permette di trovare i valori massimo e minimo dei pixel presenti nell'immagine.
Il funzionamento del modulo è semplice. Vengono impiegati due registri: max register e min register, nei quali alla fine del calcolo saranno presenti rispettivamente il valore massimo e minimo dei pixel dell'immagine.

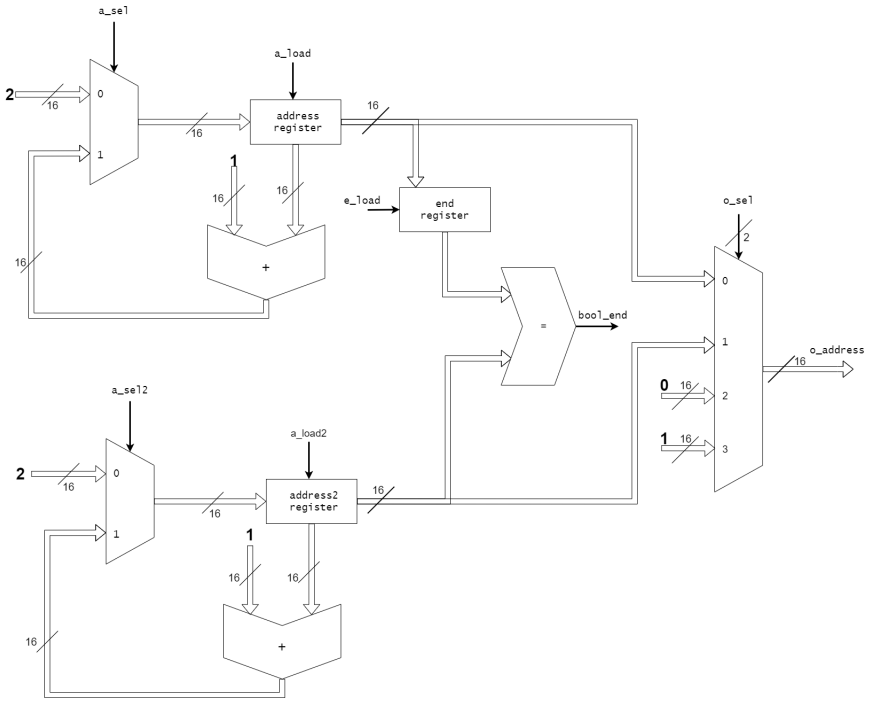

Figure 4: Rappresentazione schematica dell'Address Counter.

Inizialmente,(con l'ausilio del segnale "init") viene caricato in max register e min register il valore del primo pixel dell'immagine. Successivamente, per ogni altro pixel dell'immagine, lo si confronta con i valori contenuti nel registro max e min e, se il valore del pixel è maggiore del massimo (contenuto in max) o minore del minimo (contenuto in min), si aggiorna il valore nel registro interessato.
Il calcolo termina dopo aver letto e processato l'ultimo pixel dell'immagine.

New pixel value

Questo modulo è il cuore dell'algoritmo di equalizzazione di immagini, calcola le seguenti funzioni:

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)

Si presti particolare attenzione al fatto che DELTA_VALUE assume valori compresi tra 0 e 255, perciò bastano 8 bit per rappresentarlo. Al contrario, DELTA_VALUE + 1 necessiterà di 9 bit per essere correttamente rappresentato.

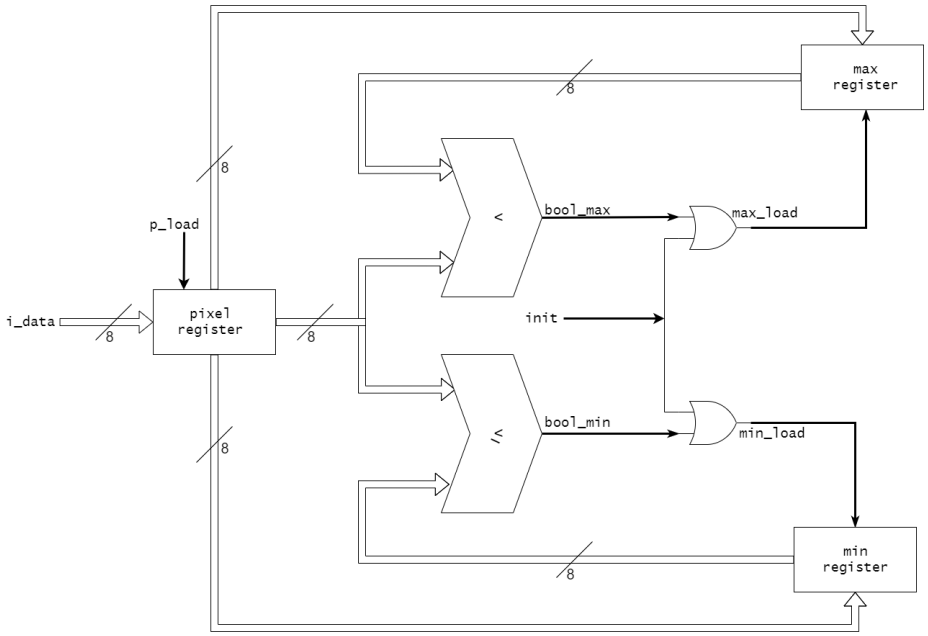

Figure 5: Rappresentazione schematica del modulo Max & Min.

L'uscita del logaritmo sarà di 4 bit, poichè i valori da rappresentare sono compresi tra 0 e 8.
La funzione FLOOR(LOG2()) è stata implementata in maniera combinatoria con controlli a soglia:

Intervallo	Logaritmo
$x \in [\ ;\ ]$	$log_2(x)$
$[1;1]$	0
$[2;3]$	1
$[4;7]$	2
$[8;15]$	3
$[16;31]$	4
$[32;63]$	5
$[64;127]$	6
$[128;255]$	7
$[256;256]$	8

Table 1: Logaritmo Combinatorio.

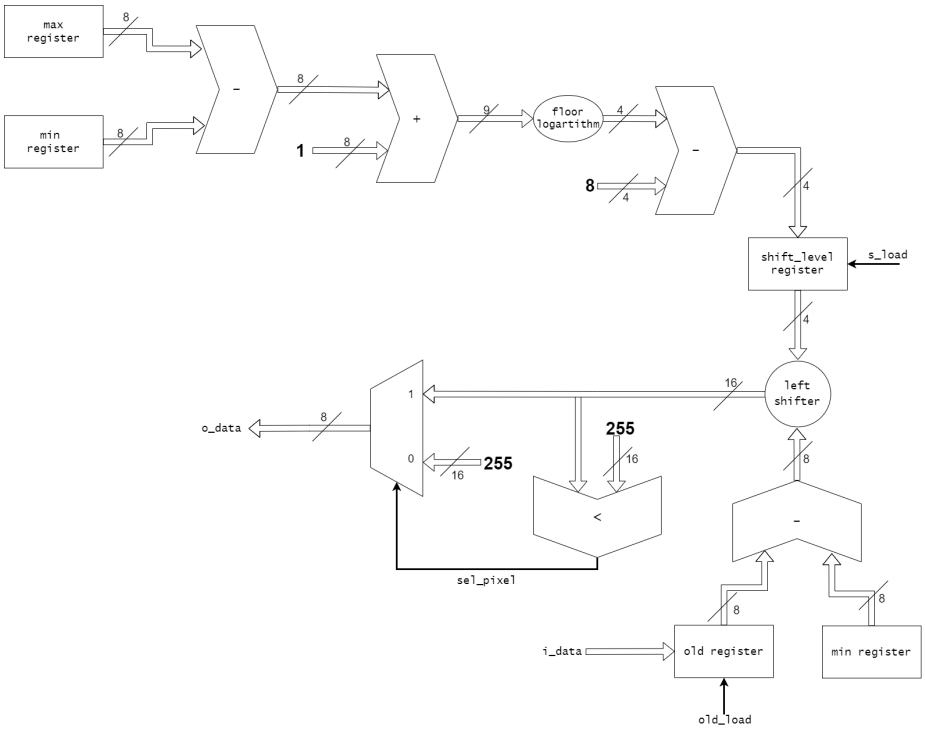

Figure 6: Rappresentazione schematica del modulo di calcolo funzioni.

2.3 Macchina a Stati

Per modellare l'unità di controllo si è deciso di utilizzare una macchina a stati di Moore (rappresentata in Figura 7), in quanto si è valutato che ciascuno stimolo del datapath proveniente dalla macchina a stati è univocamente determinato dallo stato della stessa.
La scelta di chiamare gli stati con nomi simbolici (S_i) piuttosto che utilizzare nomi significativi è stata dettata dal fatto che nella FSM modellata, uno stato non svolge singolarmente una funzione specifica ma è invece l'unione di più stati ad implementare una determinata funzionalità.

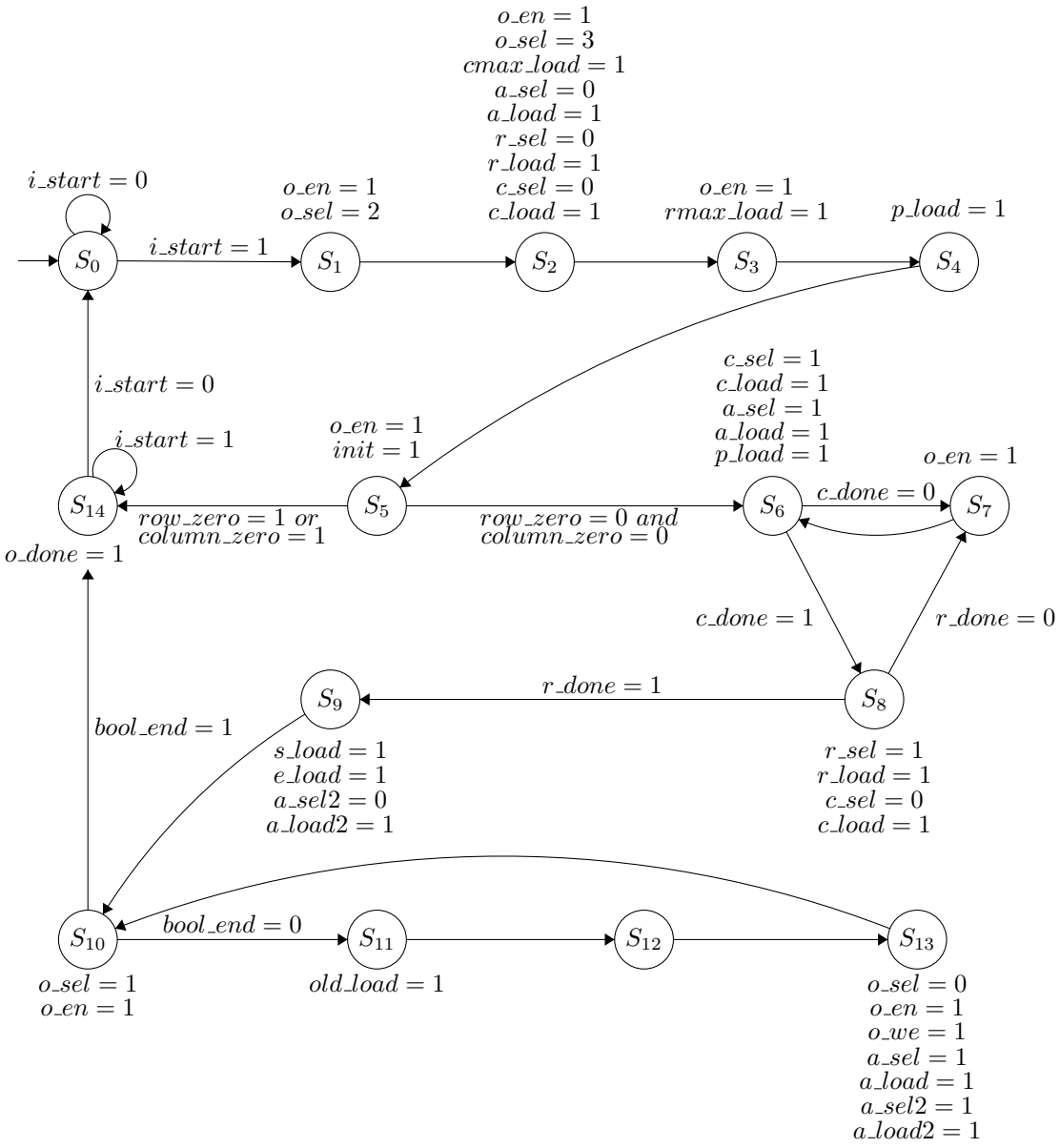

Figure 7: Diagramma della macchina a stati finiti.

S_0 È lo stato di Reset della macchina, quest'ultima, rimane in questo stato fino a quando non riceve il segnale i_start alto, che indica che può iniziare la conversione dell'immagine.

S_1 In questo stato la macchina richiede la lettura del valore in memoria all'indirizzo '0', ovvero il numero di colonne che costituiscono l'immagine.

S_2 Mentre la macchina si occupa di caricare nel registro column_max register la dimensione larghezza dell'immagine appena ottenuta dalla memoria, richiede il valore all'indirizzo '1', ovvero il numero di righe che compongono l'immagine. Inoltre, la macchina si occupa di inizializzare address register, row register e column register rispettivamente ai valori '2', '1' e '1', che serviranno a tenere traccia degli indirizzi letti.

S_3 In questo stato viene caricata la dimensione altezza dell'immagine all'interno del row_max register, infine viene richiesta la lettura del primo pixel.

S_4 La memoria restituisce il valore del primo pixel che viene memorizzato in pixel register.

S_5 La macchina alza il segnale init[2] che ha lo scopo di inizializzare i registri max register e min register con il valore del primo pixel dell'immagine: questi registri saranno aggiornati correttamente nel ciclo S_6, S_7, S_8. Infine, nel caso in cui i segnali zero_column o zero_row dovessero valere '1' (ovvero l'immagine ha dimensione nulla), la macchina si sposta nello stato S_{14}, altrimenti si sposta nello stato S_6.

Ciclo S_6, S_7, S_8 Questo ciclo costituisce la fase di lettura, che ha lo scopo di individuare e di memorizzare nei registri max register e min register rispettivamente il valore di intensità massima e minima dell'immagine.
Ad ogni iterazione il valore di address register viene aggiornato con l'indirizzo del pixel da richiedere alla RAM.

S_6 In questo stato viene salvato il valore del pixel richiesto in pixel register e vengono incrementati i valori all'interno di column register e address register. Al termine del ciclo di clock viene verificato che i contenuti di column register e column_max register siano uguali (c_done = '1'): in caso affermativo, si è appena conclusa la lettura di una riga dell'immagine, di conseguenza la macchina va nello stato S_8, altrimenti si sposta nello stato S_7. Essendo stato caricato un nuovo pixel, nel prossimo stato verranno aggiornati i valori di max register e min register.

[2]Il segnale init è essenziale nel caso di letture successive di immagini, infatti nel caso in cui si inizializzassero i registri max e min solo dopo il reset e, il massimo e il minimo dell'immagine precedente fossero rispettivamente maggiore e minore del massimo e del minimo dell'immagine successiva, verrebbero mantenuti i valori precedenti di massimo e minimo.

S_7 La macchina richiede il pixel in memoria all'indirizzo memorizzato in address register.

S_8 Viene resettato il column register a '1' per permettere la lettura di una nuova riga, mentre viene incrementato il valore di row register: se row register e row_max register contengono gli stessi valori (r_done = '1') allora l'immagine è stata letta completamente e la macchina va in S_9, altrimenti in S_8.

S_9 Al termine del ciclo appena descritto, address register contiene l'indirizzo del pixel immediatamente successivo all'ultimo pixel dell'immagine. A partire da questo indirizzo verrà salvata l'immagine equalizzata. Viene inizializzato il contenuto di address2 register con il valore '2', indirizzo del primo pixel dell'immagine da equalizzare.
Avendo precedentemente individuato il massimo e il minimo valore dei pixel, si può calcolare il valore di SHIFT_LEVEL e caricarlo nell'omonimo registro. Infine, viene caricato il valore di address register in end register, quest'ultimo sarà utile in seguito.

Ciclo S_{10}, S_{11}, S_{12}, S_{13} Ogni iterazione di questi 4 stati legge il valore di un pixel dell'immagine da equalizzare, lo processa e lo scrive in memoria all'indirizzo desiderato. Per gestire correttamente gli indirizzi sarà necessario utilizzare due registri:

- address: contiene l'indirizzo in cui verrà scritto il pixel dell'immagine equalizzata;
- address2: contiene l'indirizzo da cui verrà letto il pixel dell'immagine da equalizzare.

Questi due registri, sono stati inizializzati nello stato S_9, e verranno aggiornati (aumentando l'indirizzo di 1) in parallelo alla fine di ogni iterazione del ciclo. Occorrerà ciclare in questi 4 stati finché tutti i pixel da equalizzare non saranno stati letti, processati e scritti in memoria. Per effettuare questo controllo, sarà sufficiente confrontare l'indirizzo contenuto in address2 register con l'indirizzo finale dell'immagine, precedentemente salvato in end register.
Quando questi due indirizzi coincidono, la macchina esce dal ciclo e va nello stato S_{14}. Di seguito vengono descritti nel dettaglio questi 4 stati:

S_{10} La macchina richiede alla memoria il valore del pixel da equalizzare, fornendo come o_address il valore contenuto in address2 register.

S_{11} Il valore del pixel da equalizzare viene salvato in old register.

S_{12} In questo stato, è disponibile in uscita da old register il valore del pixel da processare: viene eseguito lo shift di SHIFT_LEVEL posizioni e il confronto con 255. Viene così calcolato NEW_PIXEL_VALUE, che è fornito in uscita attraverso o_data e potrà essere scritto in memoria nel ciclo di clock successivo.

S_{13} Il valore del pixel equalizzato calcolato nello stato precedente, viene scritto in memoria all'indirizzo contenuto in address register.
Infine vengono incrementati di 1 sia l'indirizzo contenuto in address register che quello contenuto in address2 register.

S_{14} La computazione è terminata e viene alzato il segnale o_done: la macchina rimane in questo stato finché i_start = '1', quando i_start commuta, procede nello stato S_0 ed è pronta per una nuova computazione.

3 Risultati Sperimentali

3.1 Report di sintesi

Dal report di sintesi emerge che il modulo è correttamente sintetizzabile e sono stati inferiti 131 Flip Flop e 0 Latch, mentre il Data Path Delay è di 5.361ns, rispettando il limite di 100ns del constraint di clock.

3.2 Simulazioni

Per verificare la correttezza del nostro modulo sono stati utilizzati dei testbench che si ritiene possano stimolare i corner case.
I testbench di seguito elencati sono stati utilizzati con lo scopo di verificare il corretto funzionamento del ciclo di lettura della FSM:

- Numero di righe e/o numero di colonne uguali a 1.
- Immagine di dimensione massima (128x128 pixel).

Per testare invece il modulo di calcolo del logaritmo si è scelto di utilizzare i seguenti tesbench:

- Immagine bianca (tutti i pixel valgono 255).
- Immagine nera (tutti i pixel valgono 0).
- Immagine monocromatica (tutti i pixel hanno lo stesso valore).

Sebbene un'immagine non possa avere dimensione nulla, per motivi di robustezza, si è scelto di testare comunque questo caso:

- Immagini anomale: Numero di righe e/o numero di colonne uguali a 0.

Per verificare che l'intero sistema si comporti correttamente nel caso di letture successive e nella gestione dei reset:

- Reset sincroni e asincroni durante l'esecuzione.
- Lettura sequenziale di più immagini.

Per verificare che il numero di iterazioni della macchina a stati sia corretto, controllando che anche l'ultimo pixel possa modificare max e min register:

- 0 o 255 come ultimo pixel.

4 Conclusioni

In conclusione l'architettura progettata, oltre a rispettare le specifiche fornite, risulta particolarmente veloce.
Il Critical Path limita il clock ad almeno 5.361ns, di conseguenza il limite imposto da specifica di 100ns è stato rispettato con un ampio margine: si può perciò ridurre notevolmente il periodo di clock, ad esempio a 15ns, ottenendo il tempo massimo di conversione di un'immagine (128x128) pari a 1.48ms.
Grazie alle scelte effettuate il sistema risulta inoltre scalabile, infatti adottando il modulo Row & Column counter invece che un moltiplicatore, al crescere delle dimensioni dell'immagine, la complessità temporale aumenta linearmente: con l'utilizzo di un moltiplicatore si andrebbe anche ad incrementare la durata del clock compromettendo la velocità del circuito.