PROVA FINALE DI RETI LOGICHE

Filippo Caliò (907675) - Cod.Persona: 10628126
Giovanni Caleffi (907455) - Cod.Persona: 10665233

Prof. William Fornaciari - AA: 2020/2021

Contents

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto è la realizzazione di un componente hardware, scritto in VHDL. Esso riceve in ingresso un'immagine in scala di grigi a 256 livelli e, dopo aver applicato un algoritmo di equalizzazione a ciascun pixel, scrive in output l'immagine equalizzata.
Di seguito è raffigurato un esempio di un'immagine 2x2 equalizzata (l'indirizzo dei dati in memoria verrà spiegato nel paragrafo 1.4).

0	1	2	3	4	5	6	7	8	9
2	**2**	46	131	62	89	0	255	64	172

Fig.1.1: Esempio: 2x2

1.2 Specifiche generali

L'algoritmo usato per l'equalizzazione delle immagini è una versione semplificata rispetto all'algoritmo standard. Esso può essere applicato solo a immagini in scala di grigi e per trasformare ogni pixel dell'immagine, esegue le seguenti operazioni:

$$\text{DELTA\_VALUE} = \text{MAX\_PIXEL\_VALUE} - \text{MIN\_PIXEL\_VALUE}$$
$$\text{SHIFT\_LEVEL} = (8 - \text{FLOOR}(\text{LOG2}(\text{DELTA\_VALUE} + 1)))$$
$$\text{TEMP\_PIXEL} = (\text{CURRENT\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE}) << \text{SHIFT\_LEVEL}$$
$$\text{NEW\_PIXEL\_VALUE} = \text{MIN}(255\ ,\ \text{TEMP\_PIXEL})$$

MAX_PIXEL_VALUE e MIN_PIXEL_VALUE rappresentano rispettivamente il massimo e il minimo valore dei pixel dell'immagine, CURRENT_PIXEL_VALUE rappresenta il valore del pixel da trasformare e NEW_PIXEL_VALUE rappresenta il valore del nuovo pixel in output.

Il componente hardware è inoltre progettato per poter codificare più immagini, una dopo l'altra. Prima di codificare l'immagine successiva, però, l'algoritmo di equalizzazione deve essere stato applicato prima a tutti i pixel dell'immagine precedente.

1.3 Interfaccia del componente

L'interfaccia del componente, così come presentata nelle specifiche, è la seguente:

```
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;
```

In particolare:

- `i_clk`: segnale di CLOCK in ingresso generato dal TestBench;
- `i_rst`: segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- `i_start`: segnale di START generato dal Test Bench;
- `i_data`: segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address`: segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done`: segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en`: segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we`: segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data`: segnale (vettore) di uscita dal componente verso la memoria.

1.4 Dati e descrizione memoria

Le dimensioni dell'immagine (max 128x128 pixel), ciascuna di dimensione di 8 bit, sono memorizzati in una memoria con indirizzamento al Byte:

- Nell'indirizzo 0 viene salvato il numero di colonne (N-COL) dell'immagine.
- Nell'indirizzo 1 viene salvato il numero di righe (N-RIG) dell'immagine.
- A partire dall'indirizzo 2 vengono memorizzati i pixel dell'immagine, ciascuno di 8 bit.
- A partire dall'indirizzo 2+(N-COL*N-RIG) vengono memorizzati i pixel dell'immagine equalizzata. Come nell'esempio di figura 1.1, i pixel equalizzati vengono salvati a partire dall'indirizzo 7.

N_COLONNE	Indirizzo 0
N_RIGHE	Indirizzo 1
PIXEL_1	Indirizzo 2
...	
PIXEL_N	
NEW_PIXEL_1	Indirizzo 2+(N-COL*N-RIG)
...	...
NEW_PIXEL_N	Indirizzo 1+2*(N-COL*N-RIG)

Fig.1.2: Rappresentazione indirizzi significativi della memoria

2 Design e scelte progettuali

La macchina a stati è composta da 18 stati ed è principalmente divisa in due macro-parti:

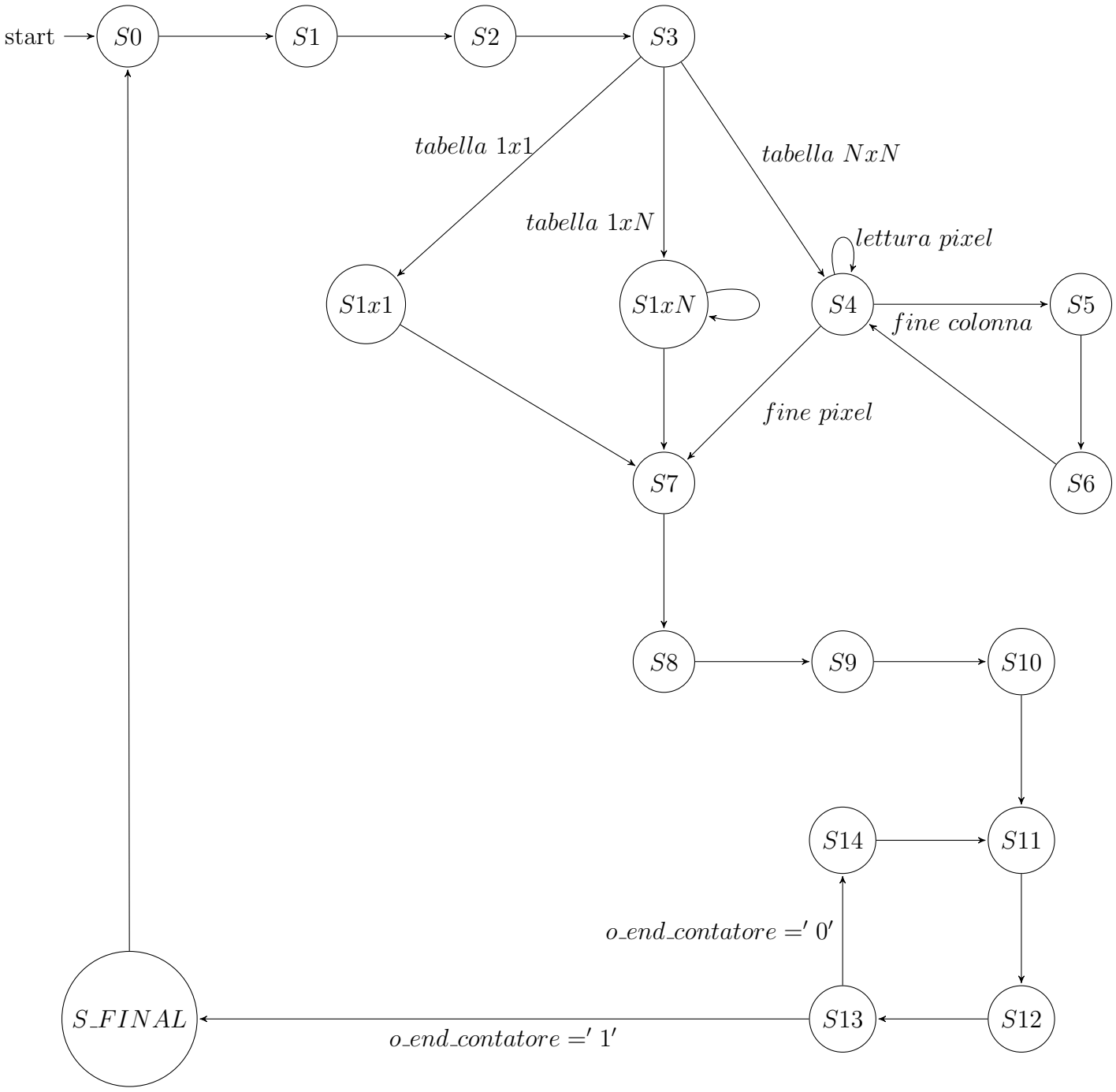
- Dallo stato S0 allo stato S8 viene eseguita la prima parte dell'algoritmo che verge alla lettura di tutti i pixel allo scopo di determinare il pixel con valore massimo, il pixel con valore minimo e conseguentemente il valore di `delta_value`.

I primi 3 stati sono dedicati alla lettura della dimensione della tabella, successivamente, in **S3** viene letto il primo valore del pixel e, in base al numero di colonne e righe, la macchina può andare in 2 stati eccezionali (**S1x1, S1xN**) che trattano casi particolari che il normale algoritmo non riesce a gestire (spiegazione più dettagliata nel paragrafo 2.3), oppure dallo stato **S3** si passa allo stato **S4** che, insieme agli stati **S5** e **S6**, gestiranno l'algoritmo per tabelle di dimensione NxN e Nx1.
In **S7** viene letto l'ultimo pixel dell'immagine. Una volta letti tutti i pixel e determinati `max_pixel_value` e `min_pixel_value`, la macchina passa allo stato **S8** dove viene calcolato e salvato il valore di `delta_value`.

- La seconda parte della macchina a stati (da S9 a S_FINAL) è dedicata alla determinazione dei valori equalizzati dei pixel originali e il loro caricamento in memoria.

  Una volta finito il primo ciclo grazie al quale ora si conoscono i valori di `max_pixel_value`, `min_pixel_value` e `delta_value` è possibile calcolare il valore di `shift_level` e, successivamente, per ogni pixel dell'immagine, determinare `temp_pixel` e `new_pixel_value` per poi caricare i nuovi valori in memoria.
  Lo stato **S9** è necessario per il corretto funzionamento della gestione di `o_address` (paragrafo 2.1).
  Nello stato **S10** viene salvato il valore di `shift_level` e viene letto il primo pixel da modificare.
  Gli stati **S11-S12-S13-S14** sono dedicati alla lettura, trasformazione e caricamento in memoria dei pixel dell'immagine.
  In **S13**, viene eseguito il ciclo tante volte quanto il numero di pixel presenti nell'immagine grazie al segnale o_end_contatore.
  Quando `o_end_contatore <= '1'`, la macchina passa in **S_FINAL**, in cui viene mandato `o_done <= '1'`. La macchina torna poi in **S0** pronta a leggere, se esiste, una nuova immagine.

Negli stati **S2** e **S3**, se si verifica che almeno uno dei due valori salvati in `o_colonneIn` e in `o_righeIn` è uguale a 0 (cioè, almeno una delle dimensioni della tabella è nulla), allora la macchina va direttamente in **S_FINAL**.
Per maggiore ordine e chiarezza nella lettura e scrittura del codice, il programma è stato diviso in 3 processi dediti ognuno a precisi compiti. Nei paragrafi 2.1, 2.2 e 2.3 viene spiegato nel dettaglio il compito di ogni processo e il ruolo di ogni stato nell'esecuzione di questo.


start
S0
S1
S2
S3
tabella 1x1
tabella 1xN
tabella NxN
lettura pixel
S1x1
S1xN
S4
S5
fine colonna
fine pixel
S7
S6
S8
S9
S10
S14
S11
o_end_contatore =' 0'
S_FINAL
S13
S12
o_end_contatore =' 1'

2.1 Gestione dell'o_address, dell'enable, dell'o_done e del caricamento di o_data:

Il valore di `o_address` viene gestito in maniera diversa tramite l'uso di `mux_definitivo` (Fig.2.2) che, in base al segnale `mux_definitivo_sel`, gli assegna il valore adatto:

- Fase di lettura (`mux_definitivo_sel = '0'`): l'indirizzo di memoria aumenta tramite il sommatore, raffigurato in alto nella Fig.2.1, per poter leggere tutti i pixel.
- Fase di scrittura (`mux_definitivo_sel = '1'`): l'indirizzo di memoria si alterna, partendo dall'indirizzo del primo pixel, per poter scrivere il NEW_PIXEL_VALUE nel giusto indirizzo. La fase di scrittura termina quando abbiamo letto e scritto in output tutti i pixel presenti.

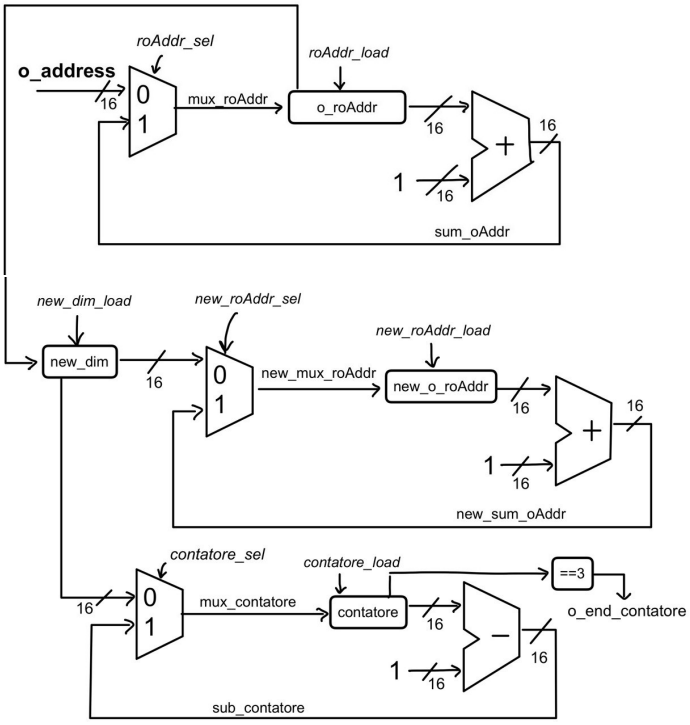


Fig.2.1

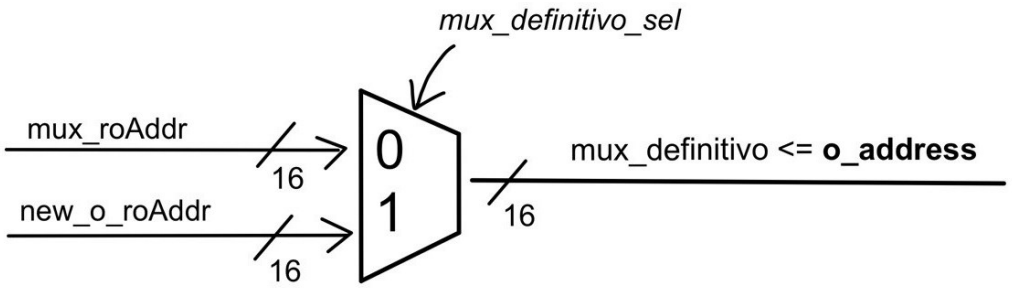


Fig.2.2

Nella prima macro-parte dell'algoritmo che va da S0 a S8, per la gestione dell'`o_address` viene mantenuto `mux_definitivo_sel <= '0'`.
Ciò implica che per i primi 9 stati, la macchina usa solo la parte superiore del datapath in figura 2.1 poichè sufficiente ad incrementare gli indirizzi linearmente.
Nella seconda parte, invece, dove è necessario passare da un indirizzo x, ad un indirizzo x + numero di pixel della tabella, per caricare il nuovo pixel equalizzato in memoria, viene utilizzato l'intero datapath alternando il valore di `o_address` tramite `mux_definitivo_sel`, che negli stati S9-S13-S_FINAL vale '0' mentre negli stati S10-S11-S12-S14 vale '1'.

- **S0**: caricamento nel registro `o_roAddr` del valore iniziale di `o_address` ("0000000000000000").
- **S1-S2-S3-S1xN-S4**: incremento il valore di `o_roAddr` per leggere tutti i valori in memoria.
- **S5-S1x1**: il valore dell'`o_address` smette di incrementare (necessario per il processo di gestione di righe e colonne).
- **S6**: ricomincia l'incremento di `o_address`.
- **S7**: caricamento nel registro `new_dim` dell'ultimo valore di `o_address` che indica quanti elementi sono stati letti in memoria nel primo ciclo. Reset dell'`o_address` e di `o_roAddr` al valore iniziale.
- **S8**: caricamento del valore del registro `new_dim` all'interno del registro `contatore`.

- **S9**: caricamento in `new_o_roAddr` del valore di `new_dim` e `o_roAddr` continua a incrementare.
- **S10**: l'`o_address` prende il valore `new_o_roAddr` che ora vale `new_dim+1` e smette di seguire `o_roAddr`. Nel frattempo il valore di `o_roAddr` continua a incrementare.
- **S11**: `new_o_roAddr` e `contatore` eseguono la stessa funzione dello stato precedente, tuttavia `o_roAddr` si ferma al valore che aveva in S10.
- **S12**: i 3 registri si comportano allo stesso modo di S11, ma in questo stato viene caricato in memoria il valore equalizzato di un pixel ponendo `o_we <= '1'`.
- **S13**: decremento il valore di `contatore` di 1, ricomincio a incrementare `o_roAddr` e `new_o_roAddr` facendo in modo che però `o_address` ora segua `o_roAddr`.
- **S14**: `o_roAddr` e `new_o_roAddr` non si incrementano più e ora `o_address` segue `new_o_roAddr`. Si ferma anche valore di `contatore`.
- **S_FINAL**: pongo `o_done <= '1'` e `o_en <= '0'` e la macchina termina.

2.2 Lettura numero dei pixel

Processo per la gestione del ciclo dedicato alla lettura di tutti i pixel tramite l'uso del numero di righe e colonne.
Il datapath (Fig.2.3) è costituito da due decrementatori, uno per le colonne e l'altro per le righe. Nella macchina a stati viene poi implementato come due cicli annidati allo scopo di eseguire la lettura dei pixel l'esatto numero di volte.

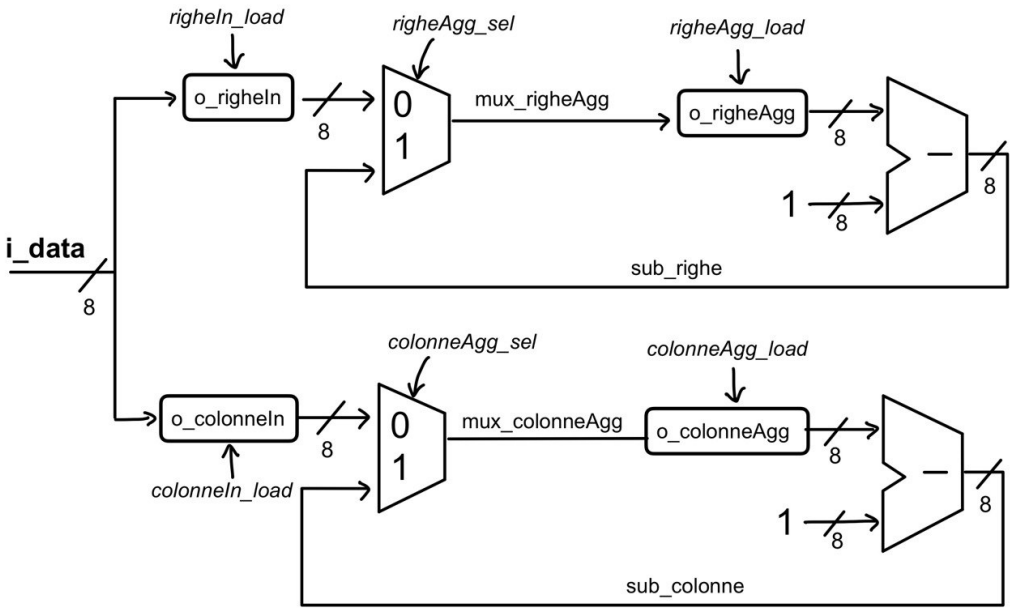


Fig.2.3

- **S1**: viene scritto il numero di colonne all'interno del registro `o_colonneIn` (registro che poi non verrà più modificato e utile per la gestione del secondo ciclo)
- **S2**: viene scritto il numero di righe all'interno del registro `o_righeIn` (registro che poi non verrà più modificato e utile per la gestione del secondo ciclo). Inoltre viene caricato nel registro `o_colonneAgg` il valore di `o_colonneIn` (registro che salva un valore e, quando necessario, decrementa il valore di 1).

- **S3**: viene caricato nel registro `o_righeAgg` il valore di `o_righeIn` (registro che salva un valore e, quando necessario, decrementa il valore di 1).
- **S1xN**: stato che decrementa di 1 il valore di `o_righeAgg` (tramite `sub_righe`), ponendo a 1 `righeAgg_sel`.
- **S4**: stato di loop che per ogni ciclo di clock decrementa di 1 il valore di `o_colonneAgg` (tramite `sub_colonne`), ponendo a 1 `colonneAgg_sel`.
- **S5**: stato che riporta il valore di `o_colonneAgg` al valore iniziale contenuto in `o_colonneIn` e nel frattempo decrementa di 1 il valore di `o_righeAgg` (tramite `sub_righe`), ponendo a 1 `righeAgg_sel`.
- **S6**: stato che riporta il valore di `o_colonneAgg` al valore iniziale contenuto in `o_colonneIn`.

Gli stati S0, S1x1, S7, S8, S9, S10, S11, S12, S13, S14, S_FINAL non vengono utilizzati all'interno di questo processo.

2.3 Calcolo max_pixel_value e min_pixel_value e applicazione algoritmo per calcolare new_pixel_value

Il datapath dedicato alla determinazione del pixel con valore massimo e minimo, del delta_value e dello shift_level è raffigurato in Fig.2.4. Mentre il datapath di Fig.2.5 descrive i componenti usati per l'assegnamento del nuovo valore del pixel.
Per trovare il valore del **pixel massimo**, si confronta il pixel in lettura (salvato in `o_pixelIn`) con il valore "0". Se maggiore, esso viene salvato nel registro `o_pixelMax`.
Per trovare il valore del **pixel minimo**, si confronta il pixel in lettura (salvato in `o_pixelIn`) con il valore "255". Se minore, esso viene salvato nel registro `o_pixelMin`.
Il `delta_value` è calcolato eseguendo la differenza fra il pixel massimo e minimo.
Per il valore di `o_floor`, vengono usati una serie di comparatori che, in base a un determinato range di valori, assegnano l'intero corrispondente (da 0 a 8).
Lo `shift_level` è calcolato sottraendo a 8 l'intero `o_floor`.

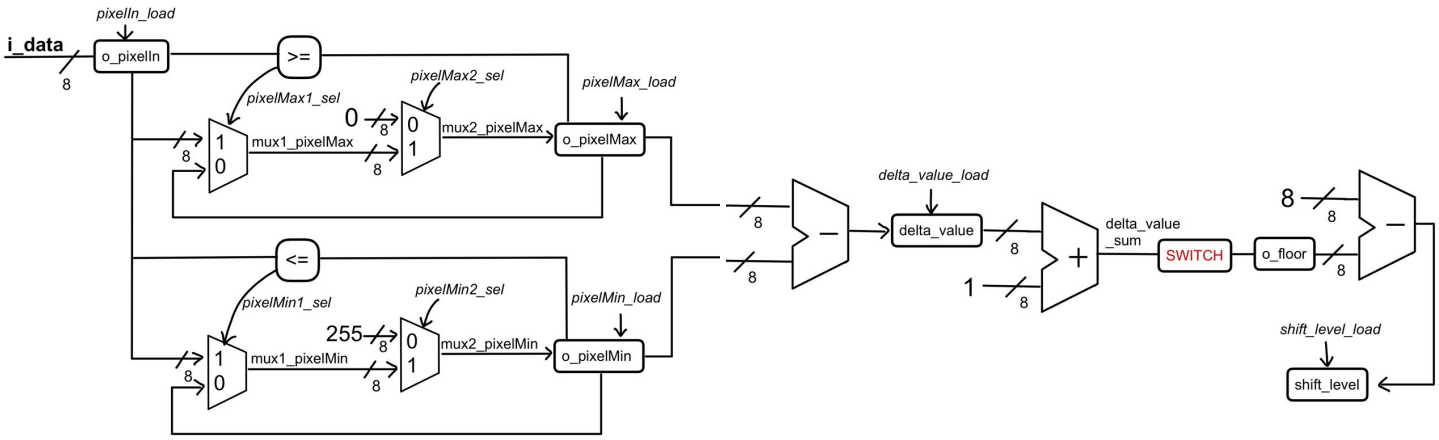


Fig.2.4

Una volta che siamo nella fase di scrittura, ci salviamo il valore del pixel da trasformare in `o_current_pixe_value` e ad esso sottraiamo il valore del pixel minimo, calcolato in precedenza.

Dopodichè, per shiftare il risultato della sottrazione (`sub_currentPixel`) del valore di `shift_level`, si è deciso di usare l'operatore concatenazione, come rappresentato di seguito.

```
if (shift_level = "0000") then
            shift_value <= "00000000" & sub_currentPixel;
```

Dopo aver calcolato `shift_value`, che ha dimensione 16 bit, il valore viene confrontato, tramite un comparatore, con l'intero 255.
Se minore, il valore del comparatore vale 1, assegnando a `o_data` i primi 8 bit di `shift_value`.
Se maggiore, a `o_data` viene assegnato l'intero 255.

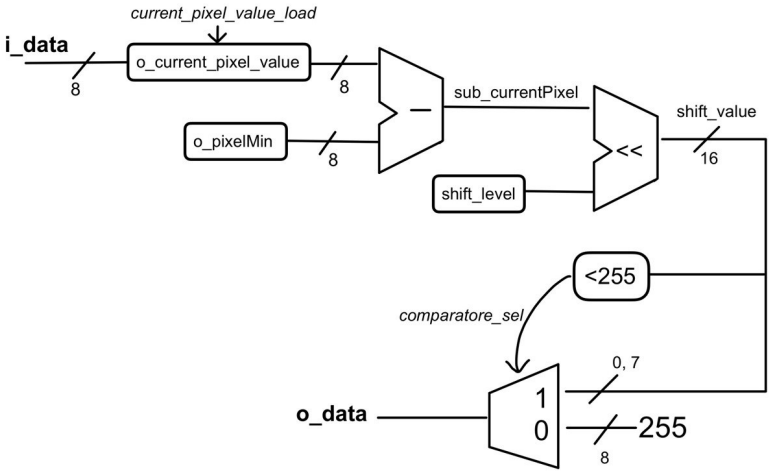


Fig.2.5

- **S3**: salva il valore del primo pixel in `o_pixelIn`, `o_pixelMax`, mentre in `o_pixelMin` viene caricato il valore 255.
- **S1x1**: stato di eccezione quando la tabella contiene un solo pixel, viene salvato il valore di quel pixel in `o_pixelMin`.

- **S4-S5-S6-S7-S1xN**: stati in cui vengono letti tutti i pixel di una colonna e viene verificato quale sia il pixel con valore massimo e minimo.
- **S8**: carica nel registro `delta_value` la differenza tra i valori finali di `o_pixelMax` e `o_pixelMin` e carica il valore di `i_data` in `o_pixelIn`.
- **S9**: carica il valore di `i_data` in `o_pixelIn`.
- **S10**: inserisco il primo valore della tabella nel registro `o_current_pixel_value` e salvo nel registro `shift_level`. Inizia il ciclo per la lettura e il caricamento in memoria di tutti i pixel equalizzati. la differenza tra 8 e il valore di `o_floor`.
- **S14**: stato che per ogni ciclo carica il valore di un pixel nel registro `o_current_pixel_value`.

Gli stati S0, S1, S2, S11, S12, S13, S_FINAL non vengono utilizzati all'interno di questo processo.

3 Risultati sperimentali

3.1 Report di sintesi

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	202	0	134600	0.15
LUT as Logic	202	0	134600	0.15
LUT as Memory	0	0	46200	0.00
Slice Registers	158	0	269200	0.06
Register as Flip Flop	158	0	269200	0.06
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Fig.3.1

3.2 Simulazioni

A seguire proponiamo 3 testbench per coprire tutti i percorsi possibili della macchina a stati:

3.2.1 Tabella 1x1: (min = max, delta_value = 0)

3.2.2 Tabella 1x4: (min = 0 , max = 255, delta_value = 255)

3.2.3 Tabella 2x3

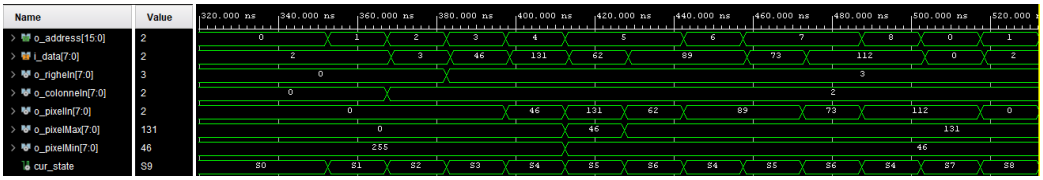


Fig.3.1

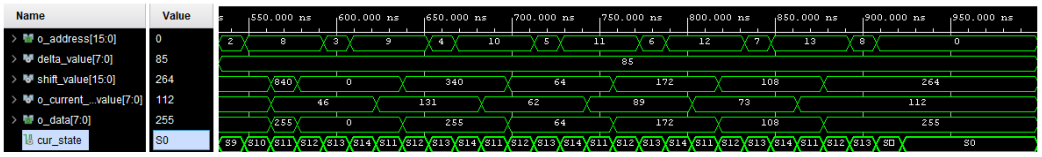


Fig.3.1

4 Conclusioni

Nel progettare il componente hardware, abbiamo prestato particolare attenzione nel rimuovere tutti i latch presenti.