Progetto finale di Reti Logiche

Flavio La Manna – Matricola n. 910293

Anno Accademico 2020/2021

Indice

1 Introduzione

1.1 Scopo del progetto

Il contrasto di un'immagine è la differenza (o il rapporto) tra il pixel con valore più alto e il pixel con valore più basso (tra il punto più luminoso e il punto più scuro). Se si aumenta il contrasto, i valori più luminosi tendono al valore massimo mentre i valori più scuri tendono al valore minimo. Un'immagine si dice "contrastata" se esiste una netta differenza tra toni luminosi e scuri con la scarsa presenza di toni intermedi. Un'immagine poco contrastata è detta "morbida" perché mancano i neri. Un'immagine "troppo contrastata" ha toni chiari che tendono al bianco e toni scuri che tendono al nero.

Se per esempio il cielo è nuvoloso, aumentare il contrasto serve a diminuire i grigi ed esaltare le zone di luce e ombra. Se invece il cielo è molto soleggiato, diminuire il contrasto serve a diminuire la luce accesa e le ombre nette.

Lo scopo del progetto è implementare una versione semplificata del metodo di equalizzazione dell'istogramma di un'immagine. Questo metodo incrementa il contrasto delle immagini ridistribuendo i valori di intensità dei pixel sull'intero intervallo di intensità. È pensato soprattutto per le immagini con valori di intensità molto vicini in quanto i valori più frequenti possono essere "spalmati" lungo tutto l'intervallo.

Immagine morbida.

Immagine contrastata.

Immagine troppo contrastata.

1.2 Specifiche generali

L'algoritmo da sviluppare è una versione semplificata del metodo di equalizzazione. L'algoritmo è applicato ad immagini 128x128 pixel con un intervallo di valori [0-255]. Ogni valore è memorizzato in memoria su 8 bit.

L'idea è trovare massimo e minimo valore dei pixel (MAX_PIXEL_VALUE e MIN_PIXEL_VALUE). Conoscendo la differenza (DELTA_VALUE) fra i due valori, si calcola un numero (chiamato SHIFT_LEVEL) attraverso una formula (semplificata rispetto al metodo di equalizzazione reale). Successivamente per ogni pixel si sottrae MIN_PIXEL_VALUE trovato in precedenza e si applica al risultato uno scorrimento a sinistra, chiamato anche "shift left". Lo shift (indicato con il simbolo '<<') è fatto per un numero di bit pari a SHIFT_LEVEL ed equivale a moltiplicare il numero per 2^(SHIFT_LEVEL). Si ottengono così i valori dei pixel corrispondenti dell'immagine equalizzata (se qualcuno dovesse essere maggiore di 255, verrebbe posto a 255).

Esempio: immagine 4x4 pixel

10	23	103	57
26	139	204	42
47	3	29	175
189	123	66	133

DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE = 204 – 3 = 201

SHIFT_LEVEL = (8 – FLOOR (LOG2 (DELTA_VALUE + 1) = 1

Per ogni pixel:

TEMP_PIXEL = (CURRENT_PIXEL_VALUE – MIN_PIXEL_VALUE) << SHIFT_LEVEL

Il pixel corrispondente dell'immagine equalizzata è:

NEW_PIXEL_VALUE = MIN (255, TEMP_PIXEL)

Ad esempio, considerando il pixel con valore 189:

CURRENT_PIXEL_VALUE – MIN_PIXEL_VALUE = 189 – 3 = 186, in binario: 10111010

TEMP_PIXEL = 186 << 1 = 372, in binario: 101110100

NEW_PIXEL_VALUE = MIN (255, 372) = 255

e il ragionamento andrà ripetuto per tutti gli altri pixel.

1.3 Dati e descrizione della memoria

L'immagine è memorizzata in una memoria RAM con indirizzamento al byte. Ogni dato è rappresentato su 8 bit.

Le dimensioni dell'immagine sono definite da due byte memorizzati all'indirizzo 0 (numero di colonne) e all'indirizzo 1 (numero di righe). La dimensione massima di un'immagine è quindi 128x128 pixel.

I valori dei pixel dell'immagine sono memorizzati in byte a partire dall'indirizzo 2 e in quelli successivi. Il byte memorizzato all'indirizzo 2 indica il valore del primo pixel nella prima riga dell'immagine e così via.

I valori dei pixel dell'immagine equalizzata dovranno essere memorizzati negli indirizzi immediatamente successivi a quelli dell'immagine originale: il primo indirizzo utile è:
2 + (num. colonne * num. righe).

Esempio: immagine 2x2

Indirizzo	Valore	
0	2	--Numero righe
1	2	--Numero colonne
2	46	--Primo pixel nella prima riga
3	131	
4	62	
5	89	
6	0	--Primo pixel nella prima riga dell'immagine equalizzata
7	255	
8	64	
9	172	

1.4 Note ulteriori sulla specifica

Il modulo da sviluppare deve poter codificare più immagini sapendo che la nuova immagine è scritta in memoria solo dopo che la precedente è stata equalizzata.

Inizialmente tutto il circuito è resettato tramite un apposito segnale di RESET esterno. Fin quando il segnale di START è basso, il modulo non deve svolgere nulla. Quando il segnale di START è alzato dall'esterno, il circuito inizia la sua elaborazione leggendo l'immagine e scrivendo l'immagine equalizzata in memoria. Fin quando il circuito elabora, il segnale di START è sempre alto. Quando il modulo termina ovvero scrive in memoria l'ultimo pixel dell'immagine equalizzata, deve sollevare il segnale DONE. Successivamente il segnale START è abbassato dall'esterno. Se il segnale START è basso, il modulo deve abbassare anche il segnale DONE. A questo punto il segnale START potrebbe essere alzato dall'esterno per far svolgere al circuito l'equalizzazione di un'altra immagine.

1.5 Interfaccia del componente

Il componente da descrivere deve avere la seguente interfaccia:

```
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;
```

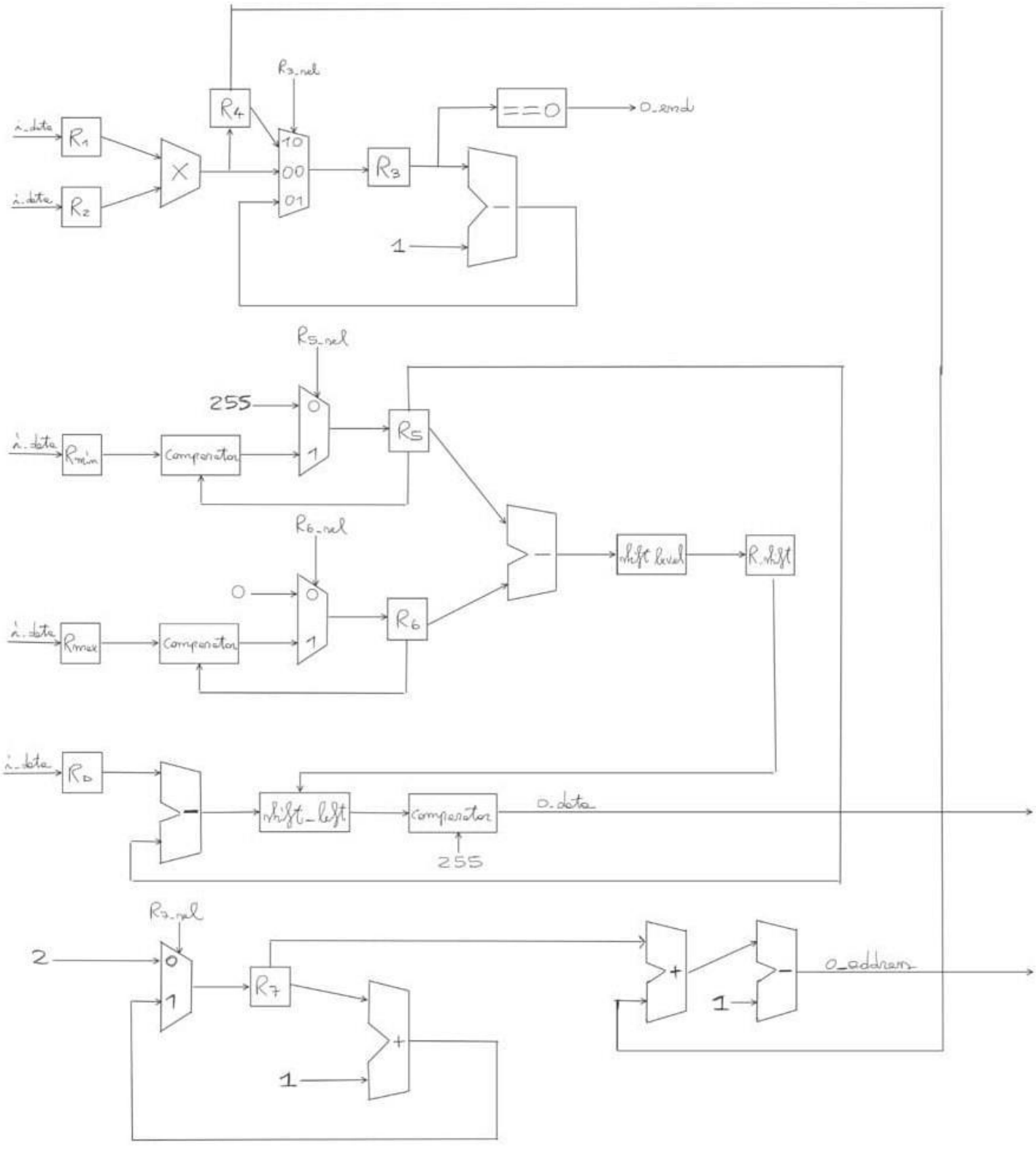
In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal test bench;
- i_rst è il segnale di RESET in ingresso che inizializza la macchina per ricevere il primo segnale di START;
- i_start è il segnale di START in ingresso generato dal test bench;
- i_data è il segnale in ingresso che indica il valore letto dalla memoria;
- o_address è il segnale in uscita che indica l'indirizzo della memoria da scrivere o leggere;
- o_done è il segnale in uscita che comunica la fine dell'elaborazione;
- o_en è il segnale di ENABLE in uscita da mandare alla memoria per poter leggere o scrivere;
- o_we è il segnale di WRITE ENABLE in uscita da mandare alla memoria per scrivere (o_we = 1). Invece per leggere dalla memoria deve essere posto a 0;
- o_data è il segnale in uscita che indica il valore da scrivere in memoria;

2 Architettura

Il modulo è stato implementato definendo una macchina a stati finiti con datapath che verranno illustrati qui di seguito.

2.1 Datapath



Il datapath può essere concettualmente diviso in quattro aree:

1. I registri R1 e R2 ricevono dalla memoria le dimensioni dell'immagine e, tramite un moltiplicatore, si calcola il numero di pixel che viene registrato in R3 e R4. Il registro R3 serve a contare il numero di pixel rimanenti da leggere in memoria: quando arriva a zero, viene alzato il segnale o_end (che servirà alla macchina a stati finiti per passare ad un altro stato). Il registro R4 conserva l'informazione riguardo il numero di pixel dell'immagine. Quando R3 raggiunge zero, tramite il multiplexer, viene salvato in R3 il valore di R4 per poter rileggere tutti i pixel una seconda volta.
2. I registri R5 e R6 sono inizializzati a 255 e 0, tramite il multiplexer. I pixel dell'immagine che vengono letti consecutivamente, sono salvati nei registri Rmin e Rmax. Un modulo apposito confronta il contenuto di Rmin con quello di R5 e il contenuto di Rmax con quello di R6; nel primo caso porta in uscita il minore mentre nel secondo caso porta in uscita il maggiore. Così facendo in R5 è salvato il pixel minimo mentre in R6 il pixel massimo. Tramite un sottrattore, si calcola la differenza tra i due valori e attraverso un altro modulo, si ottiene lo SHIFT_LEVEL che viene salvato nel registro Rshift.
   Visto che (8 – FLOOR (LOG2 (DELTA_VALUE + 1) è sempre un numero intero, il calcolo di SHIFT_LEVEL può essere fatto a soglie conoscendo DELTA_VALUE ovvero la differenza tra pixel massimo e minimo:

DELTA_VALUE	SHIFT_LEVEL
0	8
1-2	7
[3;6]	6
[7;14]	5
[15;30]	4
[31;62]	3
[63;126]	2
[127;254]	1
255	0

3. Una volta letti tutti i pixel, trovato il massimo e il minimo e calcolato lo SHIFT_LEVEL, si rileggono tutti i pixel per calcolare i corrispondenti dell'immagine equalizzata. I pixel letti consecutivamente sono salvati nel registro Rd. Tramite un sottrattore, si calcola la differenza tra il pixel letto e il pixel minimo (salvato nel registro R5). Al risultato viene applicato lo "shift left" per un numero pari a SHIFT_LEVEL (salvato nel registro Rshift). Successivamente un modulo confronta il valore ottenuto con 255 e porta in uscita il minore cioè il segnale o_data (il valore del pixel dell'immagine equalizzata da scrivere in memoria).

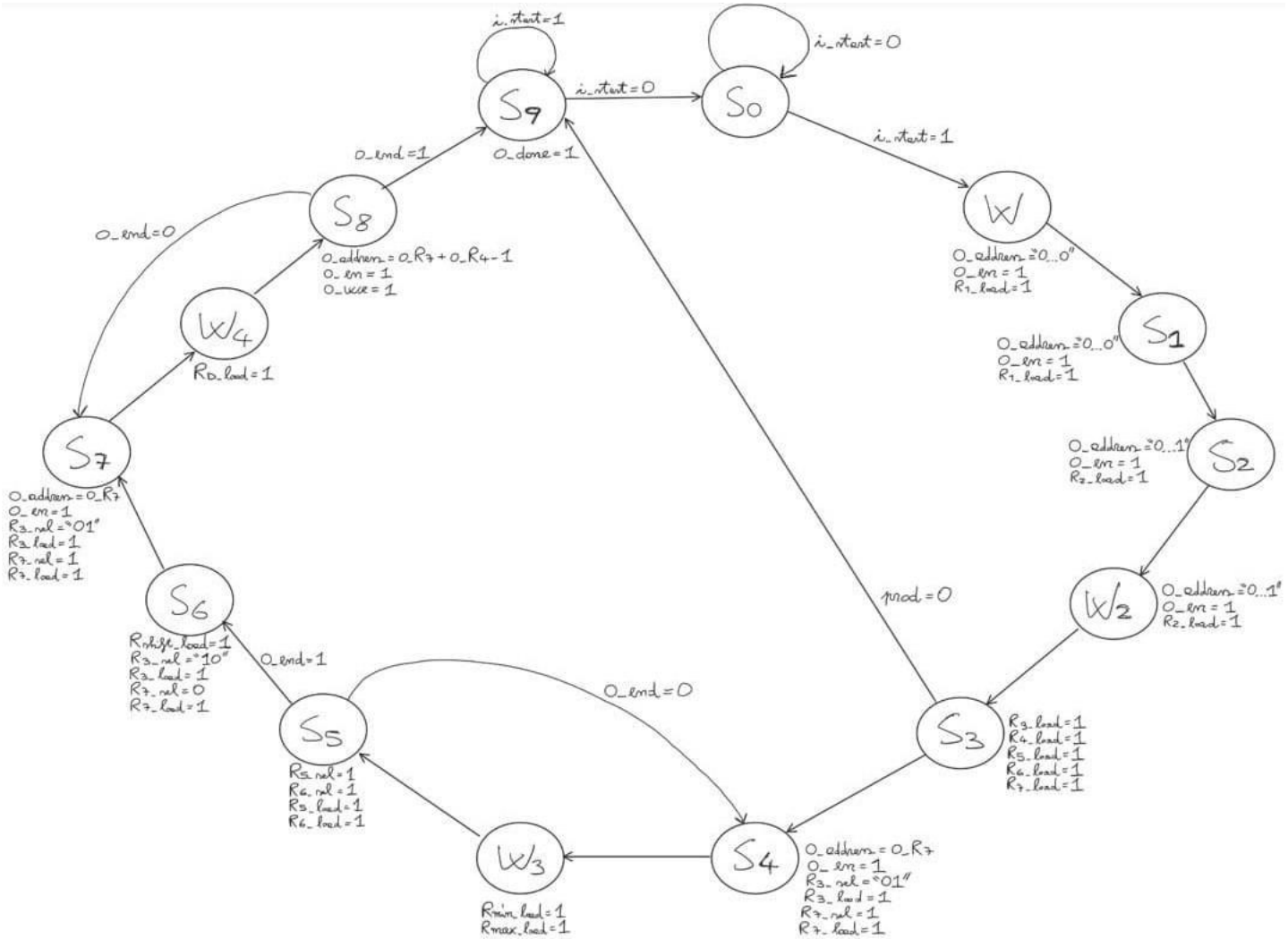
4. Il registro R7 è inizializzato a 2 tramite un multiplexer ed il suo valore è incrementato di 1 tramite un sommatore. Il registro è utilizzato per calcolare il segnale o_address, ovvero l'indirizzo di memoria dove scrivere il segnale o_data. È inizializzato a 2 perché è sempre l'indirizzo di memoria del primo pixel da leggere ed è incrementato per leggere tutti gli altri pixel negli indirizzi consecutivi.

2.2 Macchina a stati finiti

La macchina a stati finiti è stata implementata come una macchina di Moore (le uscite dipendono solo dallo stato corrente).

I segnali da controllare sono: i segnali di load di ogni registro (se è alto allora il registro salva il valore in ingresso), i quattro segnali che controllano i multiplexer e i segnali o_address, o_en, o_we e o_done prima specificati.

Tutti i segnali hanno come valore di default zero; se in uno stato un segnale non è specificato, allora è sottointeso che assuma il valore di default.



L'automa possiede 14 stati:

S0: Stato iniziale raggiunto, sincronicamente o asincronicamente, quando il segnale i_rst è alto. Si rimane in questo stato fin quando il segnale i_start non è alzato dall'esterno.

W: Si richiede di leggere il primo indirizzo di memoria.

S1: Il valore letto dal primo indirizzo di memoria è salvato nel registro R1.

S2: Si richiede di leggere il secondo indirizzo di memoria.

W2: Il valore letto dal secondo indirizzo di memoria è salvato nel registro R2.

S3: I registri R3, R4, R5, R6 e R7 sono inizializzati come descritto nel datapath. Se il prodotto delle dimensioni dell'immagine è zero, ovvero l'immagine non possiede pixel, lo stato successivo è S9.

S4: Si richiede di leggere il successivo indirizzo di memoria, il valore nel registro R3 è decrementato di un'unità e il valore nel registro R7 è incrementato di un'unità. L'indirizzo di memoria da leggere (o_address) è dato dal valore in uscita dal registro R7.

W3: Il valore letto dall'indirizzo di memoria è salvato nei registri Rmin e Rmax.

S5: Se il valore nel registro Rmin è minore di quello nel registro R5, allora è salvato nel registro R5. Se il valore nel registro Rmax è maggiore di quello nel registro R6, allora è salvato nel registro R6. Se il segnale o_end = 1, ovvero non rimangono altri pixel da leggere, lo stato successivo è S6, altrimenti è S4.

S6: Conoscendo pixel massimo e minimo, lo SHIFT_LEVEL è calcolato come descritto nel datapath e salvato nel registro Rshift. I registri R3 e R7 sono reinizializzati.

S7: Si richiede di leggere il successivo indirizzo di memoria, il valore nel registro R3 è decrementato di un'unità e il valore nel registro R7 è incrementato di un'unità. L'indirizzo di memoria da leggere è dato dal valore in uscita dal registro R7.

W4: Il valore letto dall'indirizzo di memoria è salvato nel registro Rd.

S8: Conoscendo il pixel minimo e lo SHIFT_LEVEL, o_data è calcolato come descritto nel datapath e scritto all'indirizzo di memoria del segnale o_address. Questo segnale è calcolato conoscendo il valore in uscita dal registro R7 (decrementato di un'unità perché era stato incrementato nello stato S7) e il numero di pixel dell'immagine (salvato nel registro R4): sapendo che il pixel è stato letto all'indirizzo di memoria x e che ci sono y pixel, il pixel corrispondente dell'immagine equalizzata deve essere scritto all'indirizzo di memoria x + y. Se il segnale o_end = 1, ovvero non rimangono altri pixel da leggere, lo stato successivo è S9, altrimenti è S7.

S9: Il segnale o_done è portato a 1 per segnalare il termine della computazione. Si rimane in questo stato fin quando il segnale i_start non è abbassato dall'esterno come da specifica.

2.3 Scelte progettuali

Si è scelto di implementare il prodotto tramite l'operatore * messo a disposizione dal linguaggio VHDL per i seguenti motivi: semplicità e chiarezza del codice, inoltre essendo entrambi i termini valori a 8 bit (quindi al massimo 255) non c'è rischio di avere una critica diminuzione delle prestazioni. Ovviamente per un caso più complesso sarebbe occorsa una scelta differente e più scalabile.

3 Risultati sperimentali

Attraverso i report di sintesi forniti da Vivado: "Utilization – Synth Design" e "synthesis_report", si riportano le seguenti informazioni ritenute più significative:

- Sono state utilizzate 264 LUT (0.2% del totale disponibili) e 104 FF (0.04% del totale disponibili). Sono assenti latch.

Utilization **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	264	134600	0.20
FF	104	269200	0.04
IO	38	285	13.33
BUFG	1	32	3.13

- Lo "slack", ovvero il tempo durante il quale tutti i segnali rimangono stabili in attesa del prossimo fronte di salita del clock è: 91.516ns.
  Il "datapath delay", ovvero il tempo necessario affinché tutte le porte logiche commutino al fronte di salita del clock è: 8.333ns.
  Visto che lo "slack" è maggiore del "datapath delay" il circuito potrebbe lavorare ad una frequenza di clock maggiore evitando tempi di attesa inutili.
  La frequenza di clock massima è: 0.12ns (1/(datapath delay)).

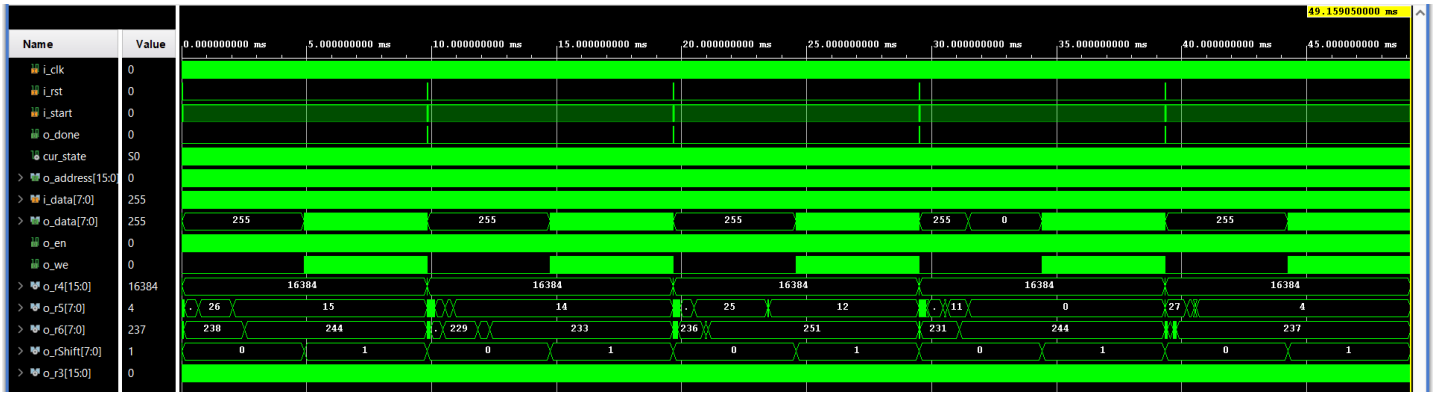
```
Timing Report

Slack (MET) :             91.516ns  (required time - arrival time)
  Source:                 o_r2_reg[3]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Destination:            cur_state_reg[3]/D
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@50.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        8.333ns  (logic 3.333ns (39.998%)  route 5.000ns (60.002%))
  Logic Levels:           10  (CARRY4=4 LUT4=2 LUT5=1 LUT6=3)
  Clock Path Skew:        -0.145ns (DCD - SCD + CPR)
    Destination Clock Delay (DCD):    2.100ns = ( 102.100 - 100.000 )
    Source Clock Delay      (SCD):    2.424ns
    Clock Pessimism Removal (CPR):    0.178ns
  Clock Uncertainty:      0.035ns  ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
    Total System Jitter     (TSJ):    0.071ns
    Total Input Jitter      (TIJ):    0.000ns
    Discrete Jitter          (DJ):    0.000ns
    Phase Error              (PE):    0.000ns
```

4 Simulazioni

Per verificare il funzionamento corretto del componente, sono stati definiti dei test bench che coprissero il maggior numero di casi possibili. Qui di seguito sono descritti quelli più significativi.

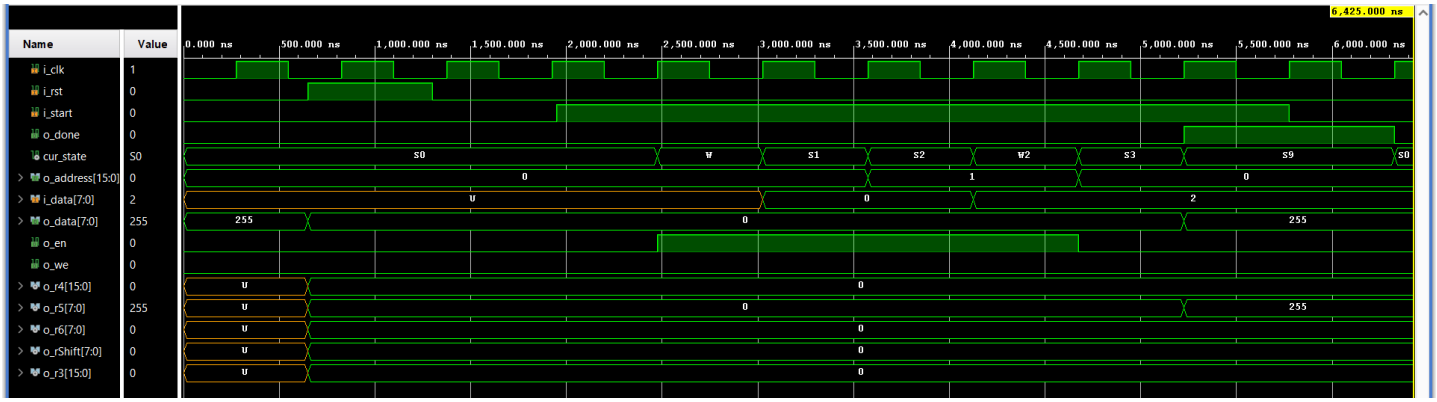
4.1 Cinque immagini consecutive 128x128

Tramite un generatore di test bench fornito dal docente si è potuto controllare la correttezza dell'algoritmo su immagini di massima dimensione e consecutive tra loro. In questo test bench sono state fornite cinque immagini 128x128.



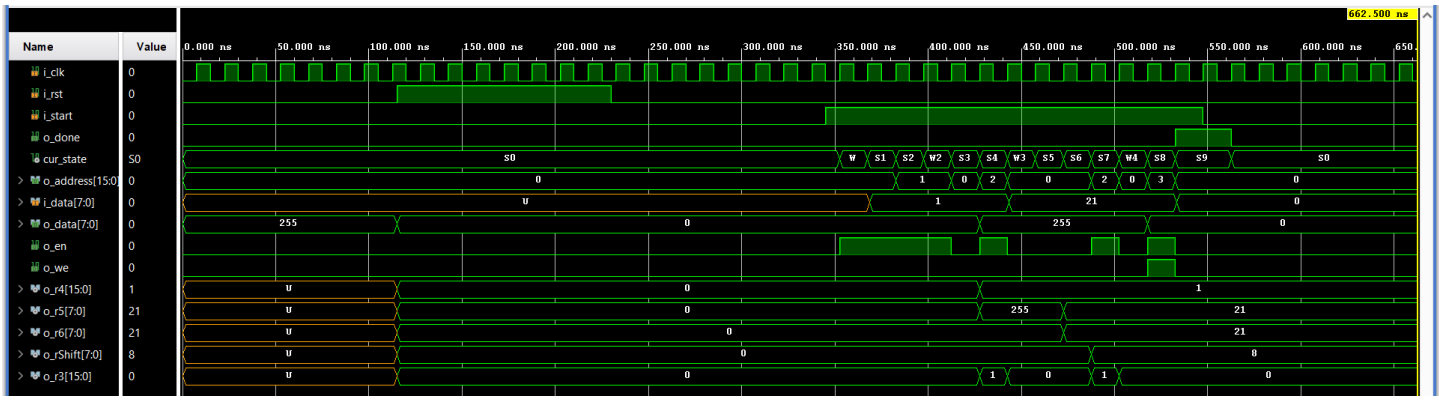
4.2 Immagine con zero pixel

Se il prodotto delle dimensioni dell'immagine è nullo ovvero l'immagine non ha pixel, l'automa non deve scrivere niente in memoria. Il test bench verifica che la macchina si sposti direttamente allo stato S9 dopo aver trovato che il prodotto delle dimensioni vale zero.



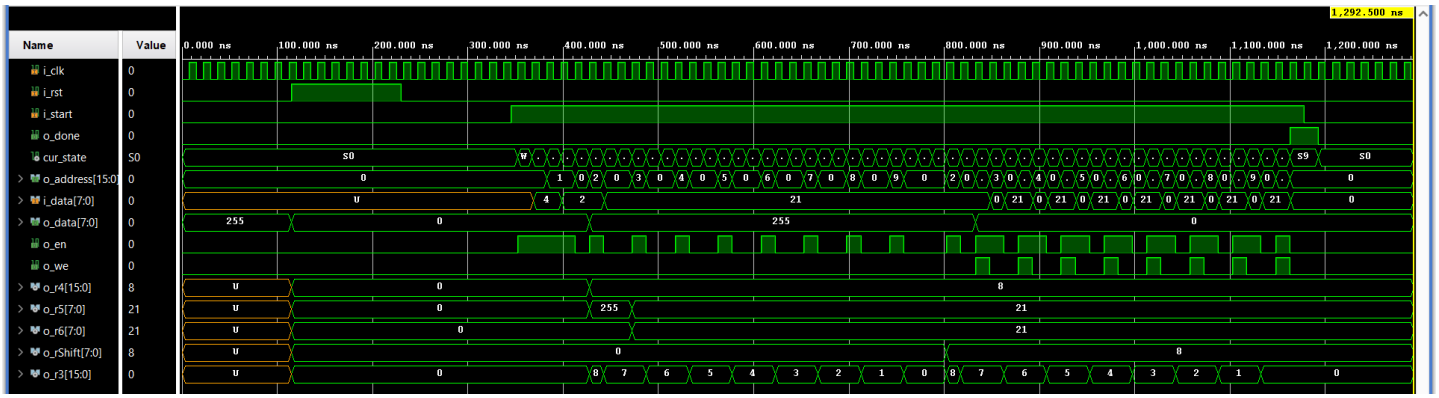
4.3 Immagine 1x1

Se l'immagine possiede un solo pixel, valore massimo e minimo sono uguali e quindi SHIFT_LEVEL vale 8. Indipendentemente però dal suo valore, il pixel dell'immagine equalizzata vale zero. Il test bench verifica che la macchina non torni nello stato S4 e S7 una seconda volta e che scriva in memoria il valore zero.



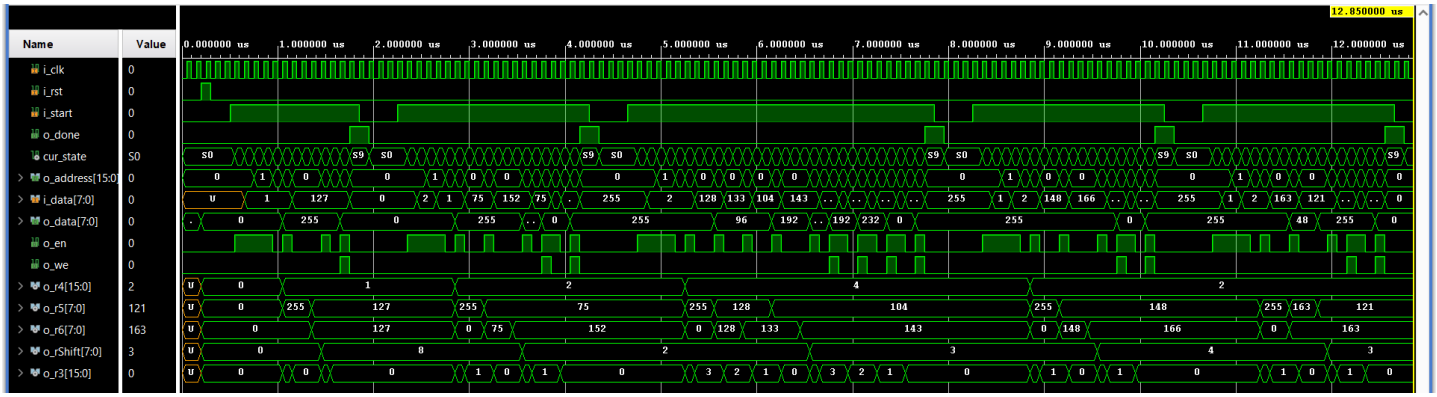
4.4 Immagine con pixel tutti uguali

Se l'immagine possiede pixel uguali, valore massimo e minimo sono uguali e quindi SHIFT_LEVEL vale 8. Indipendentemente però dal suo valore, tutti i pixel dell'immagine equalizzata valgono zero. Il test bench verifica che la macchina scriva solo valori pari a zero in memoria.



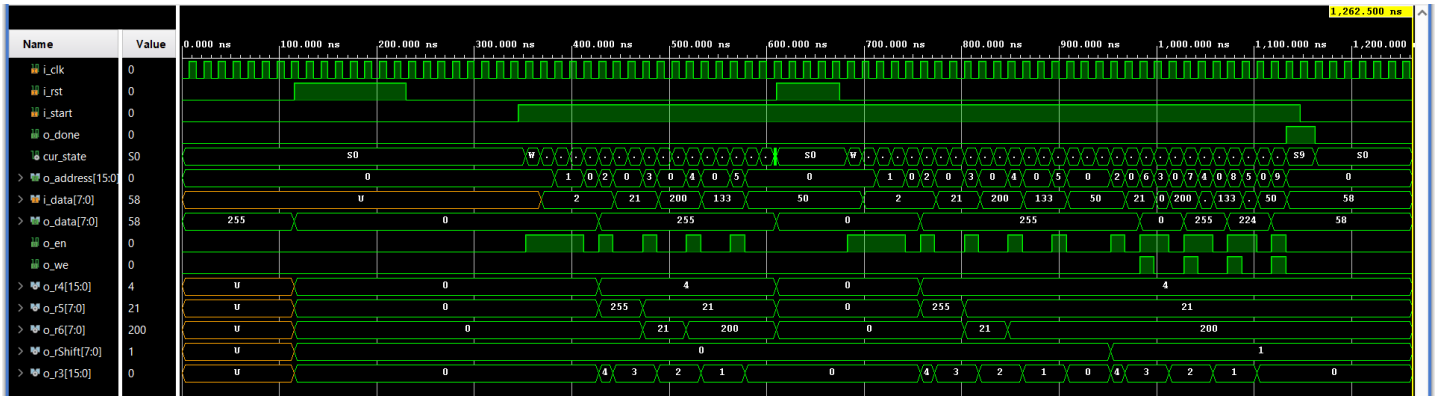
4.5 Immagini consecutive

Il test bench verifica che il componente sia capace di equalizzare più immagini consecutive come assegnato da specifica. Nel test sono presenti cinque immagini, ma ovviamente il funzionamento rimane invariato anche in presenza di un diverso numero di immagini.



4.6 Reset asincrono

Il test bench verifica che il componente, ricevuto il segnale i_rst alto, torni allo stato iniziale S0 e reinizializzi tutti i registri.



5 Conclusione

Tutti i test sono stati verificati sia in Behavioral Simulation che in Post-Synthesis Functional Simulation e non emergono errori.

Il progetto è stato molto interessante da sviluppare sia per quanto riguarda l'aspetto pratico che teorico. Dal punto di vista pratico ha permesso di conoscere, imparare e approfondire un nuovo linguaggio di programmazione (vhdl) e il software Vivado (ovviamente in una parte delle sue possibili funzionalità). Dal punto di vista teorico è stata una sfida molto stimolante creare un datapath e un automa a stati finiti che interagissero tra di loro in modo da rispettare la specifica assegnata e svolgere una determinata funzione. In particolare, la suddivisione in datapath e macchina a stati finiti ha reso molto agevole scrivere il codice che non ha necessitato di troppo debugging. Si è potuto applicare sotto un altro punto di vista ciò che è stato studiato nel corso di Reti Logiche e verificare il funzionamento di segnali visti solo teoricamente. Anche se in maniera semplificata, è stato interessante capire cosa fosse il contrasto di un'immagine e quale algoritmo ci fosse dietro. Infine, la scrittura di questa relazione è stata la prima occasione per confrontarsi con la descrizione formale di una specifica, con la sua idea risolutiva, e per adeguarsi all'utilizzo di un linguaggio tecnico e scientifico. La relazione è stata scritta nel tentativo di essere precisa e chiara in tutte le sue sezioni, cercando anche di essere comprensibile per coloro che non avessero mai seguito un corso di Reti Logiche.