# Prova finale di Reti Logiche

Lorenzo Romagnoni

a.a. 2020/2021

# Chapter 1

# Introduzione

L'obiettivo del progetto consiste nel design ed implementazione di un componente hardware che data un'immagine ne aumenti il contrasto visivo. Il compoenente deve essere implementato tramite VHDL, ed aumentare il contrasto utilizzando un algoritmo ispirato al metodo di equalizzazione dell'istogramma dell'immagine.

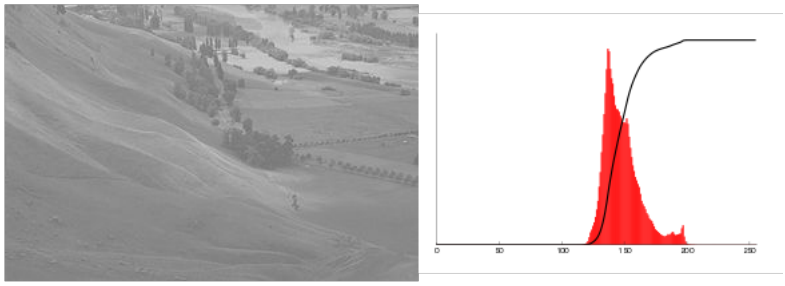


Figure 1.1: Immagine non equalizzata

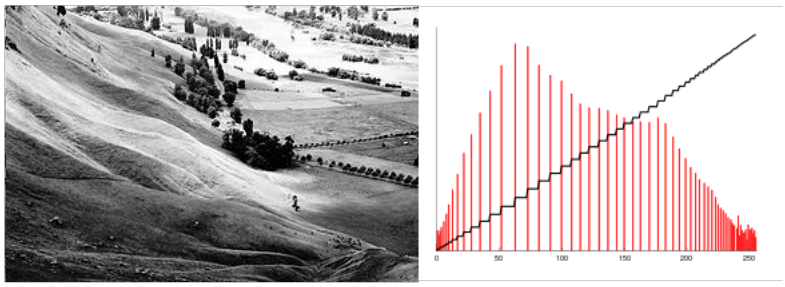



Figure 1.2: Immagine equalizzata

# Chapter 2

# Funzione del componente

Le immagini fornite in input saranno in scala di grigio a 256 livelli, avranno dimensione rettangolare, le cui dimensioni dei lati sono variabili tra 0px e 128px. Un esempio di equalizzazione è il seguente:

data un immagine 2px X 2px:


Figure 2.1: Immagine non equalizzata

l'output dopo aver calibrato il contrasto sarà il seguente:

$$f(46) = 0$$
$$f(131) = 255$$
$$f(62) = 64$$
$$f(89) = 172$$

La funzione f è definita come segue:

```
f(old_pixel):
delta_value = max_pixel_value - min_pixel_value;
shift = (8-floor(log2(delta_value + 1)))
temp = (old_pixel - min_pixel) << shift
```

```
return min(255, temp)
```

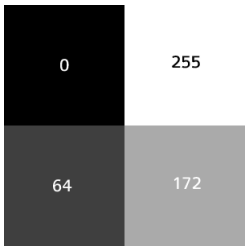



Figure 2.2: Immagine equalizzata

## 2.1 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_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 TestBench;
- `i_rst` è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- `i_start` è il segnale di START generato dal Test Bench;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;

- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

# Chapter 3

# Descrizione della memoria

Ogni pixel dell'immagine corrisponde ad un byte.
La dimensione è definita in 2 byte, ognuno rappresentante la dimensione di un lato dell'immagine. Il byte all'indirizzo 0 si riferisce alla dimensione di colonna, quello all'indirizzo 1 alla dimensione di riga.
I byte contigui dell'immagine sono memorizzati a partire dall'indirizzo 2. I pixel dell'immagine equalizzata sono memorizzati in memoria con indirizzamento al byte a partire dalla posizione 2 + n_righe * n_colonne.

Esempio memoria:

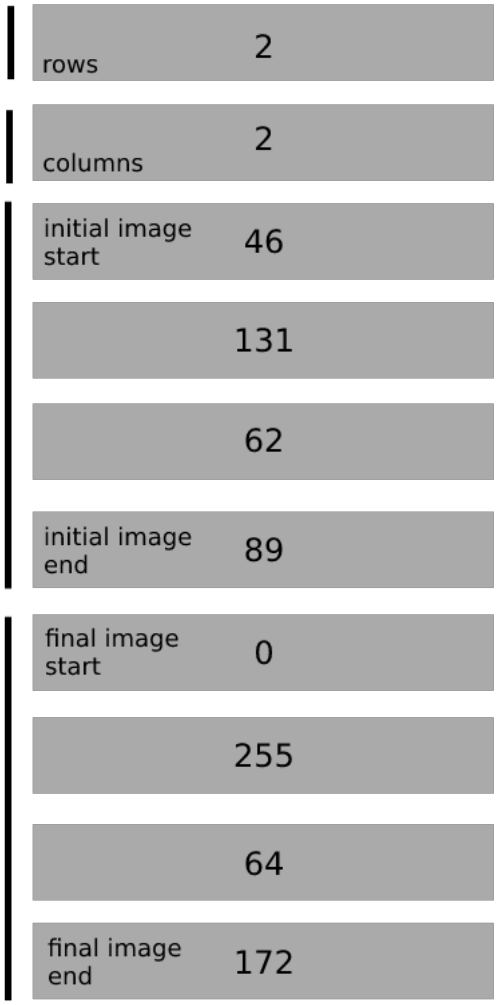



Figure 3.1: Esempio contenuto in memoria

# Chapter 4

# Architettura del componente

Considerando la presenza dei valori min_pixel e max_pixel all'interno della funzione di equalizzazione, il processo di elaborazione si divide nelle seguenti fasi:

1. lettura delle dimensioni dell'immagine
2. lettura delle immagini per il calcolo del massimo ed il minimo
3. lettura delle immagini per l'equalizzazione
4. scrittura dell'immagine equalizzata in memoria

Viste le diverse fasi per l'elaborazione dell'immagine, si è deciso di suddividere lo sviluppo in diversi componenti, ognuno dei quali rappresentabile come una macchina a stati.
Oltre a quelli spiegati in seguito, sono stati implementati due componenti aggiuntivi ausiliari, che svolgono rispettivamente la funzione di multiplexer e di porta logica OR.

- mem_scanner
- max_min_calculator
- equalizer
- finalizer

## 4.1 mem_scanner

Si occupa della lettura in memoria dell'immagine. Sono infatti necessarie almeno due letture di tutta l'immagine, la prima per il calcolo del massimo e minimo, la seconda per l'equalizzazione. Utilizzando un singolo componente

per questa funzionalità si è evitata duplicazione non necessaria.

Stati:

### 4.1.1 Stato `IDLE`

Lo stato iniziale del componente, in attesa del segnale `i_start`. Quando ricevuto resetta il puntatore usato per scorrere la memoria e passa allo stato `START`.
In caso di reset si passa in questo stato.

### 4.1.2 Stato `START`

Segnala alla memoria la sua attivazione.

### 4.1.3 Stato `WAIT_ROWS`

Attende la ricezione del numero di righe dalla memoria.

### 4.1.4 Stato `READ_ROWS`

Salva il numero di righe in un registro, ed incrementa il puntatore della memoria.

### 4.1.5 Stato `WAIT_COLUMNS`

Attende la ricezione del numero di colonne dalla memoria.

### 4.1.6 Stato `READ_COLUMNS`

Salva il numero di colonne in un registro, ed incrementa il puntatore della memoria.

### 4.1.7 Stato `CALCULATE_PIXELS`

Calcola il numero di pixel dell'immagine, il valore viene salvato in un registro.

### 4.1.8 Stato `WAIT_PIXELS`

Se ci sono pixel rimanenti, attende la ricezione dalla memoria, altrimenti passa allo stato `DONE`.

### 4.1.9 Stato `READ_PIXELS`

Legge il pixel dalla memoria.

### 4.1.10 Stato WAIT_ WRITE

Si posiziona in memoria per permettere una eventuale scrittura all'indirizzo corretto, e passa allo stato WAIT_PIXEL

### 4.1.11 Stato DONE

Segnala il termine della scansione e passa allo stato IDLE.

## 4.2 max_min_calculator

Calcola il massimo ed il minimo dei valori passati in ingresso.

Stati:

### 4.2.1 Stato IDLE

Il componente rimane in ascolto dei segnali in ingresso. Se viene segnalato un valore da leggere, in caso soddisfi la condizione, viene salvato come valore massimo o minimo nel corrispettivo registro.
Se viene segnalata la fine dello scanning, passa allo stato DONE.
In caso di reset passa in questo stato.

### 4.2.2 Stato DONE

Segnala in output i valori massimo e minimo calcolati.

## 4.3 equalizer

Effettua il calcolo relativo all'equalizzazione del pixel.

Stati:

### 4.3.1 Stato IDLE

Se segnalata l'inizio della scansione, passa allo stato START_SCAN e si prepara a ricevere la differenza tra massimo e minimo valore di pixel dal max_min_calculator.
In caso di reset passa in questo stato.

### 4.3.2 Stato START_SCAN

Segnala in uscita l'inizio della scansione e passa allo stato EQUALIZE_PIXEL

### 4.3.3 Stato `EQUALIZE_PIXEL`

Se segnalata la fine dello scanning passa allo stato `DONE`. Se è presente un valore da leggere, calcola e segnala in output il valore del pixel equalizzato.

### 4.3.4 Stato `DONE`

Segnala il termine dell'equalizzazione.

## 4.4 finalizer

Si occupa di soddisfare le specifiche che devono rispettare i segnali per segnalare il termine dell'esecuzione.

Stati:

### 4.4.1 Stato `IDLE`

Se riceve un segnale di `DONE`, passa allo stato `RESET_ALL`. In caso di reset si passa in questo stato.

### 4.4.2 Stato `RESET_ALL`

Manda un segnale di reset di tutti i componenti, e passa allo stato `WAIT_FOR_END`

### 4.4.3 Stato `WAIT_FOR_END`

Segnala il termine dell'esecuzione. Rimane in attesa di un segnale di start, quando lo riceve, passa allo stato `IDLE`.

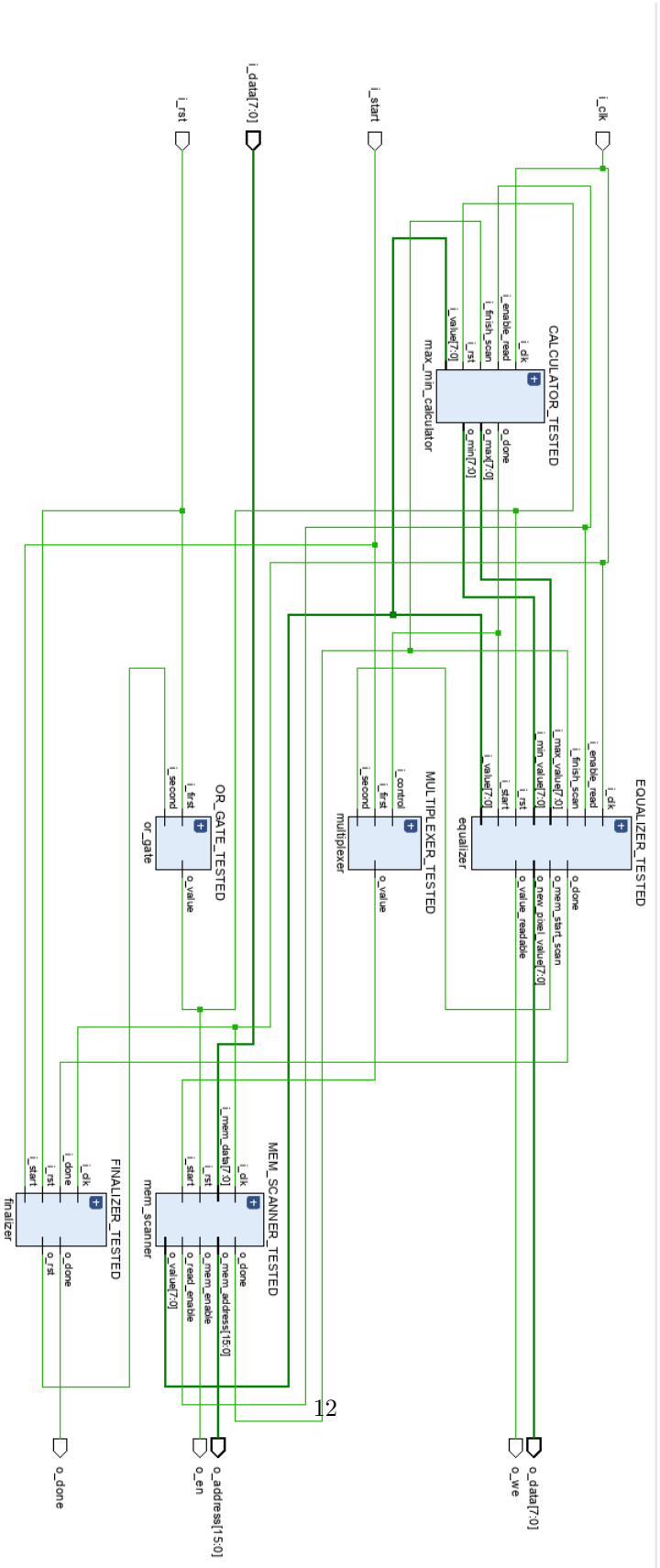



Figure 4.1: Visualizzazione schematica del componente

# Chapter 5

# Risultati dei test

Il testing del componente è stato effettuato in due stadi diversi.
Nel primo, quello di sviluppo, sono stati effettuati unit test per i singoli moduli, in modo da verificarne il loro funzionamento prima di integrarli tra di loro.
Una volta completati tutti, i test hanno avuto finalità di controllo del funzionamento del componente come insieme, e di verifica di funzionamento in casi limite.
Vista la specifica di progetto, i casi limite testati si suddividono in:

Condizioni particolari sulle dimensioni dell'immagine: $(0 \leq N \leq 128)$

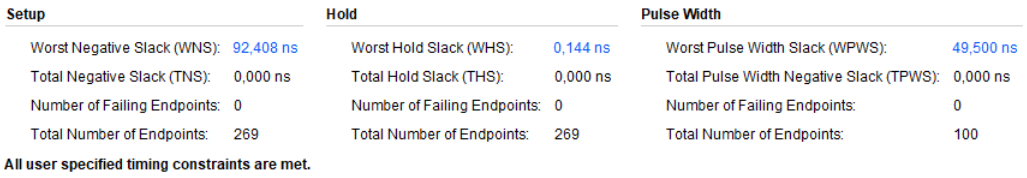
- 0x0
- 0xN
- Nx0
- 0x1
- 1x0
- 1x1
- 1xN
- Nx1
- 128xN
- Nx128
- 128x128

Condizioni particolari sul contenuto dell'immagine

- immagine bianca

- immagine nera
- immagine grigia

Oltre a questi test è stata effettuata un'arrività di smoke testing su dimensioni variabili NxN con reset asincrono del componente durante il processamento. Infine è stata verificata la condizione di funzionamento con periodo di clock pari a 100ns, rispettata come visibile nella seguente figura:



| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 92,408 ns | Worst Hold Slack (WHS): | 0,144 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: | 269 | Total Number of Endpoints: | 269 | Total Number of Endpoints: | 100 |

**All user specified timing constraints are met.**

Figure 5.1: Estratto del timing report di Vivado

Di seguito una simulazione di esempio d'equalizzazione di un immagine composta da 4 pixel.

Figure 5.2: Estratto simulazione - da 0ns a 600000ns

Figure 5.3: Estratto simulazione - da 400000ns a 950000ns

# Chapter 6

# Conclusioni

Il componente passa con successo tutti i test. L'utilizzo di diversi moduli che incapsulano i diversi compiti logici, aumenta l'efficienza dello sviluppo in caso di espansione o modifica del funzionamento del componente.

Si è fatta attenzione inoltre a mantenere la sintesi del componente priva di latch, in modo da garantire il corretto funzionamento del componente anche sintetizzato.