Relazione Prova Finale di Reti Logiche

Tiu Robert Andrei (10652209) & Giuseppe Urso (10628602)

A.A. 2020/2021

Indice

1 Introduzione

Lo scopo del progetto è realizzare un componente hardware per effettuare una equalizzazione (semplificata) dell'istogramma di un'immagine, mirata a ricalibrare il contrasto di immagini con valori di intensità molto vicini, in modo tale da distribuire uniformemente i valori dei pixel su tutta la scala.
Il componente, presa in ingresso una immagine in scala di grigi su 256 livelli e informazioni sulle sue dimensioni, dovrà restituire in output l'immagine equalizzata.

1.1 Specifica e Obiettivi

Il modulo comunica con l'esterno mediante una memoria RAM: i primi due byte della RAM vengono usati per rappresentare le dimensioni della immagine. Il byte in posizione 0 si riferisce al numero di colonne, mentre il byte in posizione 1 si riferisce al numero di righe.
La dimensione massima che può assumere l'immagine da elaborare è 128x128 pixel. Gli `N_PIXEL` byte successivi ai primi due sono i pixel dell'immagine, che il componente dovrà elaborare svolgendo le seguenti operazioni per ognuno di essi:

```
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)
```

`MAX_PIXEL_VALUE` e `MIN_PIXEL_VALUE`, sono il massimo e minimo valore dei pixel dell'immagine.
`CURRENT_PIXEL_VALUE` è il valore del pixel da trasformare.
L'operatore `<<` si riferisce allo shift logico in binario verso sinistra: ad esempio, se si ha la sequenza di bit `10011011`, eseguendo uno shift di 3 posizioni, si ottiene `11011000`.
`NEW_PIXEL_VALUE` è il valore del nuovo pixel.
I rimanenti byte della memoria RAM verranno infine usati per scrivere i pixel elaborati.

Ad esempio, presa la seguente immagine:

46	131
62	89

I cui valori dei pixel sono [46, 131, 62, 89], abbiamo che:

```
MAX_PIXEL = 131
MIN_PIXEL = 46
DELTA_VALUE = 85
SHIFT_LEVEL = 2
```

Per il primo pixel (46) avremo ad esempio che:

```
TEMP_PIXEL = 0
NEW_PIXEL = 0
```

Il valore equalizzato del primo pixel risulta quindi essere 0. Ripetendo queste operazioni per i pixel rimanenti, si ottiene la seguente immagine:

0	255
64	172

1.2 Interfaccia del componente

Il componente ha 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:

- il nome del modulo è project_reti_logiche
- `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.

1.3 Descrizione del contenuto della RAM

Le dimensioni dell'immagine, ciascuna di dimensione di 8 bit, sono memorizzate in una memoria con indirizzamento al byte partendo dalla posizione 0:

- il byte in posizione 0 si riferisce al numero di colonne `N_COL`.
- il byte in posizione 1 si riferisce al numero di righe `N_RIG`.

I pixel del'immagine originale, ciascuno di 8 bit, sono indirizzati in memoria partendo dalla posizione 2.

L'immagine equalizzata dovrà essere scritta in memoria immediatamente dopo l'immagine originale. Segue che anche i pixel dell'immagine equalizzata saranno ciascuno da 8 bit e sono indirizzati a partire dalla posizione `2+(N_COL*N_RIG)`.

0	00000010	Dimensioni immagine
1	00000010	
2	00101110	Bytes immagine originale
3	10000011	
4	00111110	
5	01011001	
6	00000000	Bytes immagine equalizzata
7	11111111	
8	01000000	
9	10101100	

Table 1: Esempio di una RAM per immagini 2x2

2 Architettura

2.1 Algoritmo e scelte progettuali

Data la limitata complessità del problema, è stato ritenuto opportuno implementare una soluzione monoprocesso mediante una macchina a stati finiti.
Quest'ultima si occuperà di tutta l'elaborazione: lettura dell'immagine dalla RAM, calcolo dei valori necessari per l'equalizzazione, ricalcolo dei nuovi valori dei pixel, scrittura della nuova immagine nella RAM. Oltre a ciò, la macchina si occupa anche della gestione del salvataggio dei valori degli stati nei registri, eseguendo le transizioni tra stati. È possibile visualizzare uno schema dell'automa nella sezione successiva.
L'algoritmo di elaborazione è sintetizzabile secondo questi passi:

1. Si legge il numero di righe e di colonne dalla RAM. Se uno dei due è 0, il processo termina.
2. Si calcola il primo indirizzo invalido, ovvero quello successivo all'indirizzo dell'ultimo pixel.
3. Si legge ogni pixel dell'immagine partendo dall'indirizzo 2, e per ognuno di essi si svolgono le seguenti operazioni:
   (a) Se è maggiore dell'attuale massimo, si aggiorna il valore di `MAX_PIXEL`.
   (b) Se è minore dell'attuale minimo, si aggiorna il valore di `MIN_PIXEL`.
   (c) Si incrementa l'indirizzo dell'attuale pixel: se è uguale al primo indirizzo invalido, si salta al punto 4.
4. Si calcola `DELTA_VALUE`.
5. Si calcola `SHIFT_LEVEL`.
6. Si legge per la seconda volta ogni pixel dell'immagine partendo dall'indirizzo 2, e per ognuno di essi si svolgono le seguenti operazioni:
   (a) Si calcola `TEMP_PIXEL`.
   (b) Si calcola `NEW_PIXEL`.
   (c) Si scrive il nuovo pixel in memoria, partendo dall'indirizzo `2+(N_COL*N_RIG)` in modo contiguo mantenendo l'ordine.
   (d) Si incrementa l'indirizzo dell'attuale pixel da leggere: se è uguale al primo indirizzo invalido in lettura, si salta al punto 7.

7. Si notifica che la elaborazione è finita.

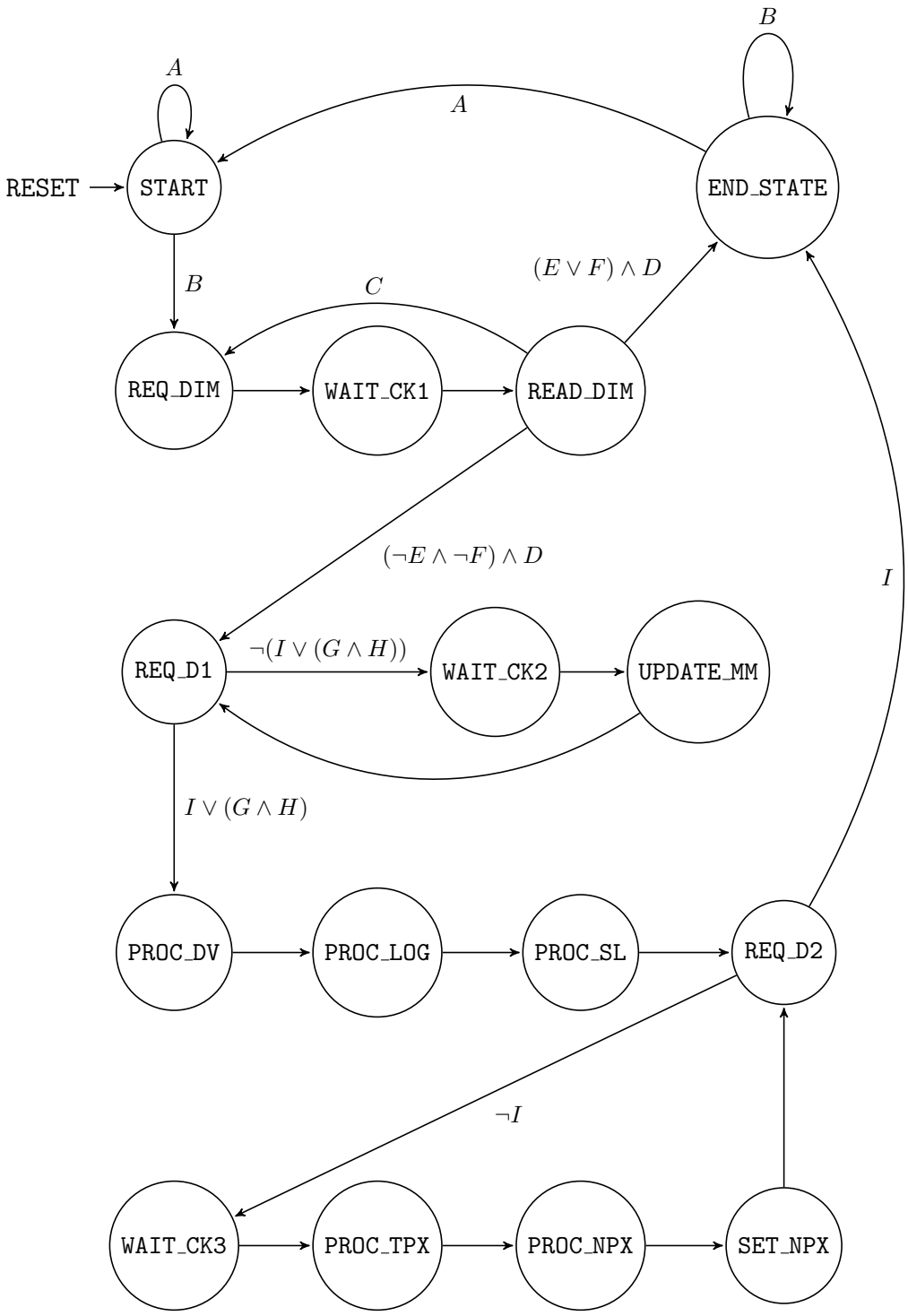
Si noti che l'algoritmo può eseguire elaborazioni multiple e può essere interrotto in qualsiasi momento da un segnale di reset asincrono, riportandolo al primo punto.

2.2 Macchina a stati del progetto

Di seguito si può vedere una figura della macchina a stati.
Per motivi di leggibilità del diagramma, sono state usate delle abbreviazioni per il nome degli stati e per le condizioni.

START_STATE:	START
REQ_DIM_STATE:	REQ_DIM
WAIT_CK1_STATE:	WAIT_CK1
READ_DIM_STATE:	READ_DIM
REQ_DATA1_STATE	REQ_D1
WAIT_CK2_STATE:	WAIT_CK2
UPDATE_MAX_MIN_STATE:	UPDATE_MM
PROC_DELTA_VALUE_STATE:	PROC_DV
PROC_LOG_STATE:	PROC_LOG
PROC_SHIFT_LEVEL_STATE:	PROC_SL
REQ_DATA2_STATE	REQ_D2
WAIT_CK3_STATE:	WAIT_CK3
PROC_TEMPPIXEL_STATE:	PROC_TPX
PROC_NEWPIXEL_STATE:	PROC_NPX
SET_NEWPIXEL_STATE:	SET_NPX

A: `i_start = '0'`
B: `i_start = '1'`
C: `request_type = '0'`
D: `request_type = '1'`
E: `num_colonne = '0'`
F: `num_righe = '0'`
G: `max_pixel = "11111111"`
H: `min_pixel = "00000000"`
I: `current_address = last_address`


RESET
START
A
A
B
END_STATE
B
C
$(E \vee F) \wedge D$
REQ_DIM
WAIT_CK1
READ_DIM
$(\neg E \wedge \neg F) \wedge D$
I
REQ_D1
$\neg(I \vee (G \wedge H))$
WAIT_CK2
UPDATE_MM
$I \vee (G \wedge H)$
PROC_DV
PROC_LOG
PROC_SL
REQ_D2
$\neg I$
WAIT_CK3
PROC_TPX
PROC_NPX
SET_NPX

2.3 Descrizione degli stati

2.3.1 START_STATE

Stato iniziale in cui, come da specifica, si attende che il segnale `i_start` venga impostato a '1' e si inizializzano tutti i segnali interni del componente. Si torna in questo stato se `i_rst` viene impostato a '1'.

2.3.2 REQ_DIM_STATE

Stato in cui si imposta il segnale `o_en` e `o_we` della RAM rispettivamente a '1' e a '0' per poter leggere la memoria, richiedendo poi di leggere un byte dalla RAM, corrispondente a una delle due dimensioni dell'immagine.

2.3.3 WAIT_CK1_STATE

Stato in cui si attende che la RAM risponda in seguito alla richiesta fatta nello stato `REQ_DIM_STATE`.

2.3.4 READ_DIM_STATE

Stato in cui:

1. si salva l'informazione delle dimensioni della immagine.
2. si effettua un controllo sulle dimensioni: se una delle due vale 0, si esegue una transizione direttamente verso `END_STATE`.
3. se le dimensioni dell'immagine sono 'regolari', si calcola il primo indirizzo invalido, utilizzato come indirizzo limite nelle condizioni dei successivi cicli di lettura.

Per effettuare queste operazioni, si usa un particolare segnale flag (`request_type`) per distinguere la lettura di numero di righe o numero di colonne e per determinare lo stato successivo.

2.3.5 REQ_DATA1_STATE

Stato in cui si abilita la RAM alla lettura, impostando `o_en` a '1' e `o_we` a '0', per poi richiedere un byte dalla memoria, corrispondente ad un pixel dell'immagine.
Si verifica se l'attuale indirizzo sia uguale al primo indirizzo invalido:

- In caso positivo, si esegue una transizione verso lo stato PROC_DELTA_VALUE_STATE.
- In caso negativo, si esegue una transizione verso WAIT_CK2_STATE.

2.3.6 WAIT_CK2_STATE

Stato in cui si attende che la RAM risponda in seguito alla richiesta fatta nello stato REQ_DATA1_STATE.

2.3.7 UPDATE_MAX_MIN_STATE

Stato in cui vengono fatti controlli condizionali per aggiornare, se necessario, i segnali max_pixel e min_pixel, corrispondenti al valore massimo e minimo dell'immagine.

2.3.8 PROC_DELTA_VALUE_STATE

Stato in cui viene effettuato il calcolo del DELTA_VALUE.

2.3.9 PROC_LOG_STATE

Stato in cui viene effettuato parzialmente il calcolo dello SHIFT_LEVEL.
Il calcolo del pavimento del logaritmo viene effettuato sfruttando il fatto che esso corrisponde sempre al numero binario costituito da un unico '1' nella stessa posizione del bit '1' più significativo del pixel, con tutti gli altri bit a '0'.

2.3.10 PROC_SHIFT_LEVEL_STATE

Stato in cui si completa il calcolo dello SHIFT_LEVEL.

2.3.11 REQ_DATA2_STATE

Stato in cui si abilita la RAM alla lettura, impostando o_en a '1' e o_we a '0', per poi richiedere un byte dalla memoria, corrispondente ad un pixel dell'immagine.
Si verifica poi se l'attuale indirizzo sia uguale al primo indirizzo invalido:

- In caso positivo, si esegue una transizione verso END_STATE.
- In caso negativo, si esegue una transizione verso WAIT_CK3_STATE.

2.3.12 WAIT_CK3_STATE

Stato in cui si attende che la RAM risponda in seguito alla richiesta fatta nello stato REQ_DATA2_STATE.

2.3.13 PROC_TEMPPIXEL_STATE

Stato in cui, una volta ottenuto il pixel da aggiornare, si effettua il calcolo di TEMP_PIXEL.

2.3.14 PROC_NEWPIXEL_STATE

Stato in cui si effettua il calcolo di NEW_PIXEL.

2.3.15 SET_NEWPIXEL_STATE

Stato in cui si abilita la RAM alla scrittura, impostando o_en a '1' e o_we a '1' e di conseguenza si scrive in memoria il nuovo pixel equalizzato nella sua posizione corrispondente.

2.3.16 END_STATE

Stato in cui si termina l'elaborazione, reimpostando a '0' i due segnali della RAM e portando a '1' il segnale o_done, notificando il termine dell'elaborazione. Se il segnale i_start viene portato a '0', si esegue una transizione verso START_STATE.

3 Testing e simulazioni

Per verificare che il componente funzioni correttamente, oltre alla testbench del docente, si è ritenuto opportuno effettuare due tipi di test:

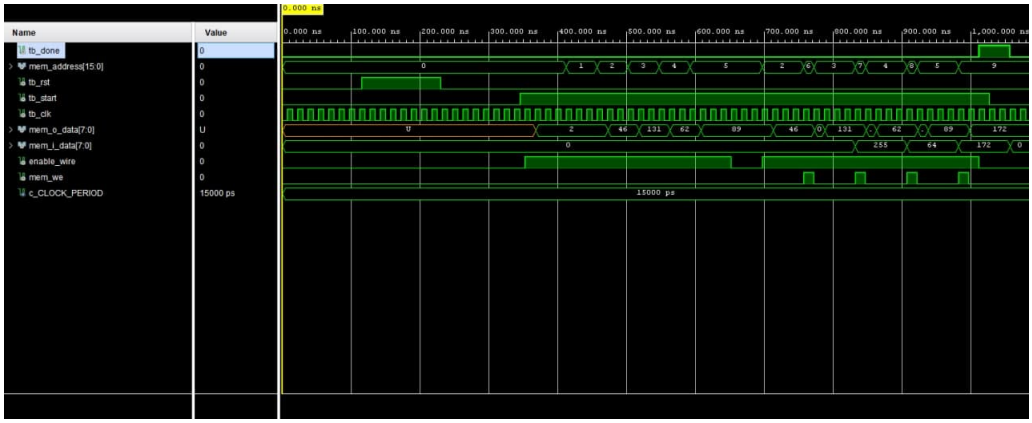
- Test scalabili per accertarsi che il componente funzioni correttamente anche nel caso di molteplici elaborazioni consecutive, per immagini di varia dimensione.

- Test mirati per accertarsi che non esistano casi limite problematici e che i segnali funzionino correttamente.

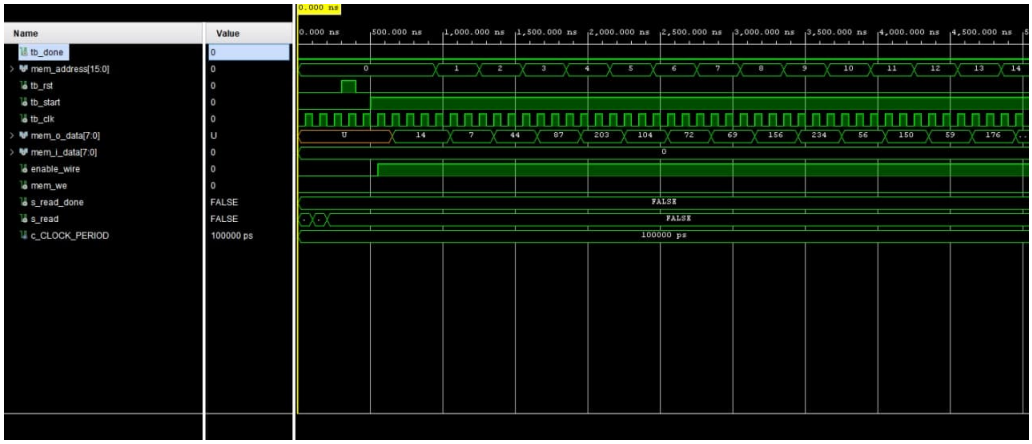
3.1 Test scalabili

Sono stati eseguiti i seguenti 3 test scalabili:

- Testbench fornita dal docente: costituita da una immagine 2x2.
  Di seguito si può vedere la waveform risultante:



- Testbench costituita da 10.000 immagini con numero di righe e di colonne comprese tra 1 e 16: mirata a verificare il corretto funzionamento del componente nel caso di un elevato numero di immagini.
  Di seguito si può vedere la waveform risultante:



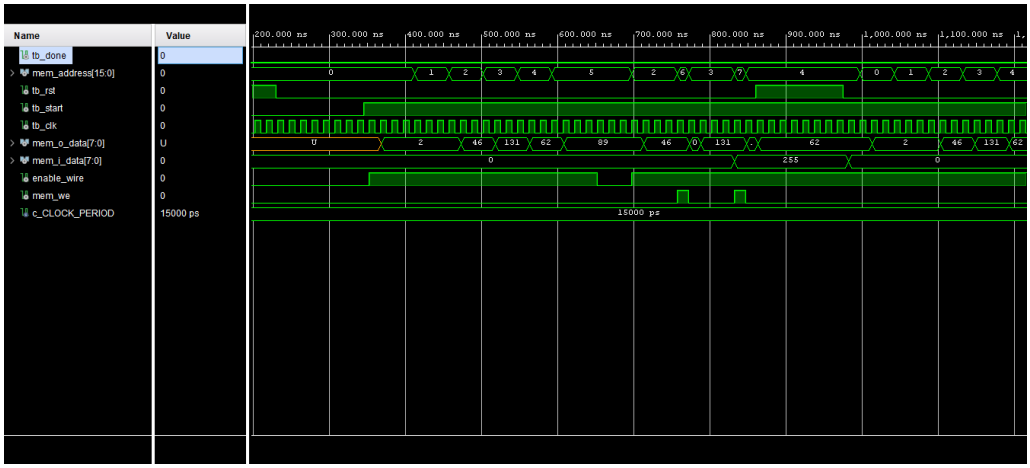
- Testbench costituita da 2.000 immagini con numero di righe e di colonne comprese tra 1 e 128: analoga alla precedente, ma mirata a stressare il componente nel caso di immagini di grande dimensione.

Attraverso questi test, si è verificato il funzionamento del componente con immagini con un ampio intervallo di valori sia di dimensioni che di pixel veri e propri.

3.2 Test mirati

I test mirati eseguiti sono stati i seguenti 7:

- Testbench con reset asincrono: come da specifica, il componente può ricevere un segnale di reset asincrono in qualsiasi momento della elaborazione, perciò si è ritenuto di vitale importanza verificare il corretto funzionamento del componente in questa casistica.
  Di seguito si può vedere la waveform risultante:



- Testbench con immagine 0xN: mirata a verificare che il componente si comporti correttamente nel caso in cui almeno una delle due dimensioni sia 0, terminando l'elaborazione lasciando la memoria invariata.
- Testbench con immagine 3x3 con valori di pixel tutti a '0'.
- Testbench con immagine 3x3 con valori di pixel tutti a '255'.
- Testbench con valori dei pixel crescenti: allo scopo di stressare l'aggiornamento del massimo.
- Testbench con valori dei pixel decrescenti: allo scopo di stressare l'aggiornamento del minimo.
- Testbench con valori dei pixel alternati: allo scopo di stressare l'aggiornamento del massimo e del minimo "in parallelo".

Attraverso i risultati derivanti da questo tipo di testing, si può affermare con ragionevole confidenza che il componente si comporti correttamente anche nei casi limite.

4 Conclusioni

I report di sintesi hanno inoltre prodotto la seguente Slice Logic Table:

Resource	Utilization	Available	Utilization %
LUT	210	134600	0.16
FF	145	269200	0.05
IO	38	285	13.33

La fase di testing ha guidato a uno sviluppo a iterazioni successive, introducendo diverse ottimizzazioni:

- riduzione del numero degli stati cambiando l'algoritmo di lettura delle dimensioni.
- migliorie prestazionali attraverso la rimozione di elaborazioni superflue.
  Difatti, se massimo e minimo vengono impostati rispettivamente a '255' e '0', si evita di analizzare i pixel rimanenti e si prosegue alla fase successiva dell'algoritmo.

Per ogni testbench, sono state eseguite simulazioni sia di tipo Behavioral, che di tipo Post-Synthesis (Functional & Timing), producendo sempre risultati corretti.

Il componente si comporta correttamente sia in casi di test generati casualmente che in test specifici mirati a identificare criticità in situazioni limite, dunque si può concludere che la progettazione è stata effettuata correttamente.