PROVA FINALE DI RETI LOGICHE
A.A. 2020/2021

Componenti Gruppo:
Magliani Martina 10682333
Morelli Bruno 10654712

Professori Referenti:
Salice Fabio
Cassano Luca Maria

INDICE:

1. INTRODUZIONE

L'obiettivo della Prova Finale è implementare un componente hardware, descritto in VHDL, che prende in input un'immagine in scala di grigi e la restituisce con un contrasto incrementato secondo il metodo di equalizzazione dell'istogramma di un'immagine.
Il metodo di equalizzazione viene descritto tramite un algoritmo, composto da due fasi.
Nella prima fase il componente legge l'immagine in input, la analizza e calcola i valori per modificarla, quali:

DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE

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

La seconda fase applica la modifica a ogni singolo pixel nel seguente modo:

TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL

NEW_PIXEL_VALUE = MIN(255,TEMP_PIXEL).

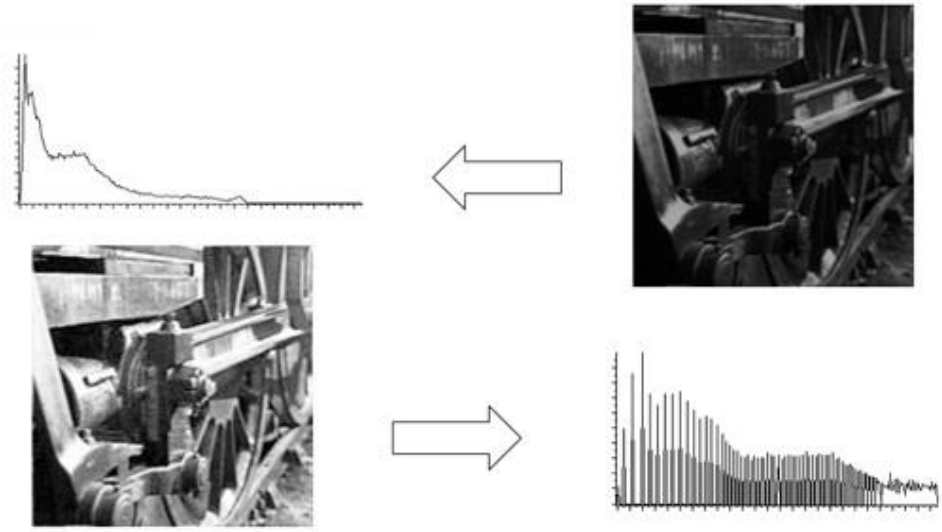

Figura 1. Esempio di equalizzazione di un'immagine.

Esempio esplicativo di un'immagine 2x2 pixel con indirizzi di memoria e relativi valori:

Immagine 2 x 2:	INDIRIZZO MEMORIA	VALORE	
	0	2	\\numero colonne
	1	2	\\numero righe
	2	46	\\primo byte immagine
	3	131	
	4	62	
	5	89	\\ultimo byte immagine
	6	0	\\primo byte immagine
	7	255	equalizzata
	8	64	
	9	172	\\ultimo byte immagine equalizzata

2. DESCRIZIONE IMPLEMENTAZIONE

2.1 ARCHITETTURA ESTERNA

Il componente comunica con l'esterno tramite i seguenti segnali:

1. *i_clk: in std_logic;* Segnale che gestisce il clock della macchina.
2. *i_rst: in std_logic;* Segnale di reset, utilizzato per posizionarsi nello stato di reset.
3. *i_start: in std_logic;* Segnale di start, avvia l'algoritmo. È alto fino alla fine.
4. *i_data: in std_logic_vector(7 downto 0);* Segnale che porta i valori letti dalla RAM alla FSM.
5. *o_address: out std_logic_vector(15 downto 0);* Segnale che indica l'indirizzo di memoria da cui leggere o in cui scrivere, in base alla configurazione dei segnali *o_en* e *o_we*.
6. *o_done: out std_logic;* Segnale di fine, viene alzato quando l'operazione finisce.
7. *o_en: out std_logic;* Segnale di enable, è alto quando si deve comunicare con la memoria.
8. *o_we: out std_logic;* Segnale di write, è alto quando si deve scrivere in memoria, per il giusto funzionamento di scrittura in memoria quando o_we è 1 anche o_en dovrà essere 1.
9. *o_data: out std_logic_vector (7 downto 0)*; Segnale che porta i dati da scrivere dalla FSM alla RAM.

La comunicazione con Test Bench e RAM avviene nel seguente modo:

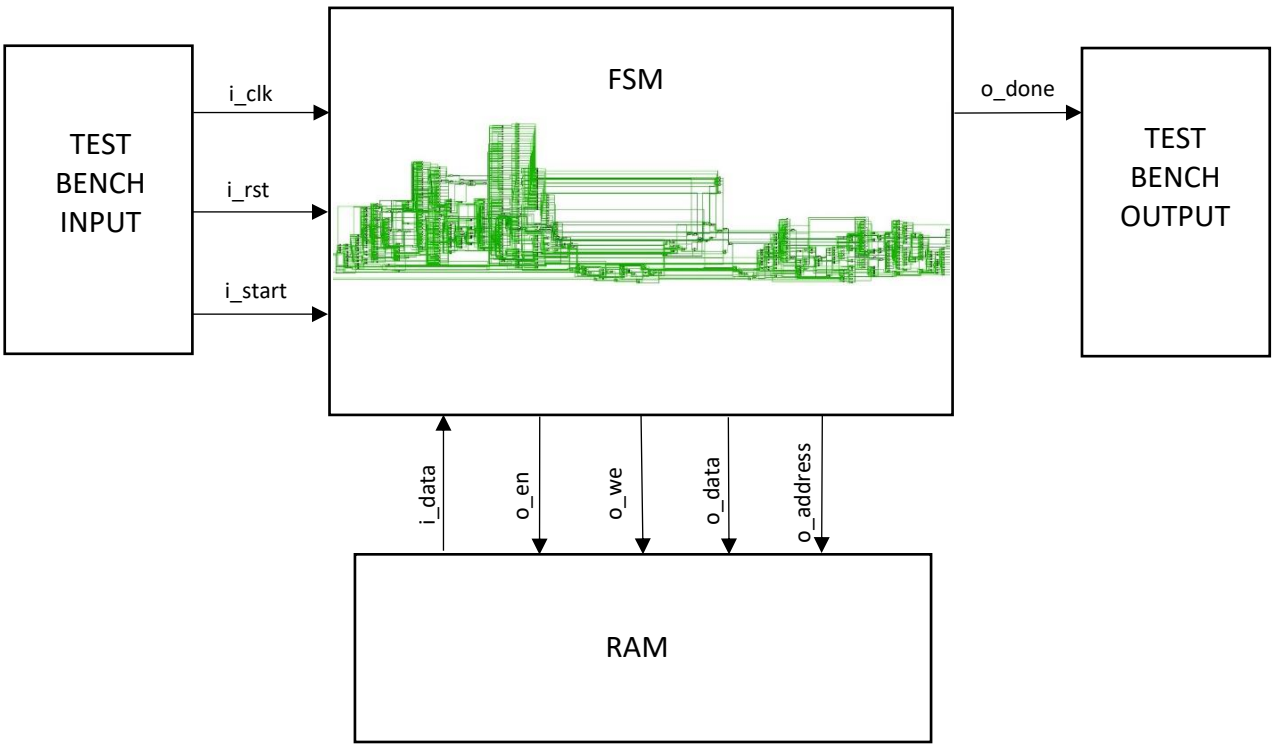

Figura 2. Architettura esterna.

2.2 ARCHITETTURA INTERNA

La soluzione è stata progettata su una Macchina a Stati Finiti (FSM), composta da un processo sequenziale (*reg_state*) e un processo combinatorio (*main_process*).
Il processo sequenziale si occupa della gestione del cambiamento dello stato corrente con il successivo ad ogni ciclo di clock, sul fronte di salita.
Il processo combinatorio determina la sequenza di stati da seguire e contiene le singole operazioni sugli stati.
I registri sono stati creati per la gestione degli indirizzi, da cui leggere e su cui scrivere in memoria, per salvare lo stato presente e futuro e, infine, per memorizzare le dimensioni dell'immagine, i valori dei pixel e i valori di modifica dell'immagine.

2.2.1 Descrizione Segnali Interni:

1. *current_state*: indica lo stato in cui si trova la FSM.
2. *next_state*: indica lo stato prossimo della FSM.
3. *o_address_cur*: è sempre uguale *ad o_address* e rende possibile lettura e salvataggio. Queste operazioni non sarebbero possibili perché *o_address,* essendo un segnale di output, può solo essere scritto ma non letto.
4. *o_address_next*: contiene *o_address_cur* incrementato di uno, serve per aggiornare *o_address.*
5. *o_address_r*: indica l'indirizzo da cui leggere il prossimo pixel durante le operazioni di modifica. È inizialmente settato a 2.
6. *o_address_w*: indica l'indirizzo dove scrivere il prossimo pixel della nuova immagine.
7. *n_col*: è il numero di colonne dell'immagine.
8. *n_righe*: è il numero di righe dell'immagine.
9. *min_pixel*: a fine algoritmo di calcolo di massimo e minimo conterrà il valore minimo tra i pixel dell'immagine. È inizialmente settato a 255 per garantire che il primo confronto vada a buon fine.
10. *max_pixel*: reciproco di *min_pixel*, conterrà il massimo valore tra i pixel dell'immagine ed è inizialmente settato a 0.
11. *cfr_pixel*: confronta tutti i pixel dell'immagine con *min_pixel* e *max_pixel* durante l'algoritmo di calcolo del minimo e del massimo.
12. *delta_value*: differenza tra *max_pixel* e *min_pixel*, usato per trovare lo *shift_level*.
13. *shift_level*: numero di bit di cui deve essere shiftato il pixel, a sinistra, per calcolare i nuovi pixel.
14. *current_pixel_value*: valore corrente di ogni pixel dell'immagine, viene shiftato per trovare il valore del pixel della nuova immagine equalizzata.
15. *new_pixel_value*: contiene il valore di *current_pixel_value* dopo lo shift a sinistra.

2.2.2 Grafo della Macchina a Stati Finiti:

Di seguito viene fornito il modello della FSM.

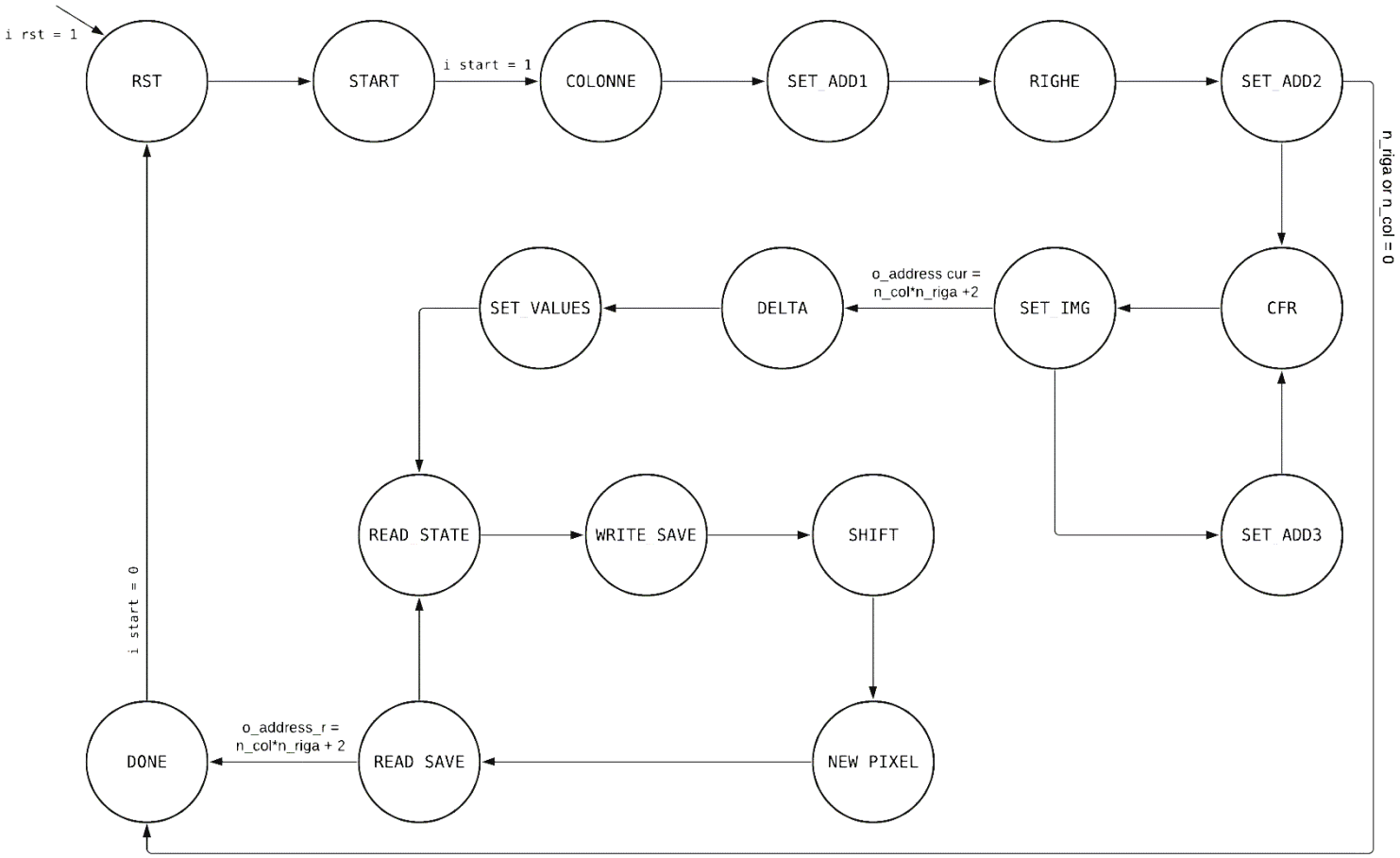

Figura 3. Descrizione modello Macchina a Stati Finiti.

3. DESCRIZIONE STATI

- **RST**: Stato di reset, in questo stato i segnali vengono portati al loro valore iniziale. Il segnale *i_rst* viene settato a 1 e si prosegue in START.
- **START**: Nello stato vengono settati il segnale di *enable* ad 1 e di *write* a 0, questa impostazione dei segnali rappresenta la configurazione di lettura. I due segnali vengono così definiti nello stato corrente al fine di essere pronti alla lettura della memoria nello stato successivo.
- **COLONNE**: Viene letto da memoria il primo byte del test bench in input, il quale contiene il numero di colonne dell'immagine da equalizzare, e viene salvato nel segnale *n_col*.
- **SET_ADD1**: L'indirizzo di lettura viene aumentato a 1 e i segnali di *enable* e *write* vengono riportati alla configurazione di lettura.
- **RIGHE**: Viene letto da memoria il secondo byte contenente il numero di righe dell'immagine data in input e salvato nel segnale *n_riga*.
- **SET_ADD2**: L'indirizzo di lettura viene aumentato a 1 e i segnali di *enable* e *write* vengono riportati nella configurazione di lettura. Successivamente si imposta l'indirizzo di lettura al terzo byte e l'indirizzo di scrittura della nuova immagine equalizzata a 2 + N_RIGA*N_COL. In questo stato viene controllato che il numero di righe e il numero di colonne sia diverso da zero; se uno dei due è nullo lo stato successivo è DONE, altrimenti l'algoritmo prosegue in CFR.
- **CFR**: Si legge ogni volta un pixel dell'immagine da memoria che sarà poi confrontato nel prossimo stato con il *min_pixel* e il *max_pixel*.
- **SET_IMG**: Viene svolto l'algoritmo per trovare il massimo e minimo pixel, confrontando l'attuale valore di *cfr_pixel* con gli attuali *min_pixel* e *max_pixel* e portando all'eventuale aggiornamento di questi al valore di *cfr_pixel*. Arrivati al confronto dell'ultimo pixel dell'immagine i valori massimo e minimo saranno definitivi e si potrà passare a calcolare il *delta_value*.
- **SET_ADD3**: Si incrementa il valore di *o_address* per leggere il prossimo *cfr_pixel* in CFR.
- **DELTA**: Si calcola il *delta_value* come differenza fra *max_pixel* e *min_pixel*.
- **SET_VALUES**: In base al valore di *delta_value* viene associato staticamente un valore di *shift_level*, secondo intervalli di valore predefiniti. Successivamente *o_address* viene riportato al terzo byte per iniziare daccapo la lettura da memoria dell'immagine.
- **READ_STATE**: Vengono letti nuovamente i pixel dell'immagine, per poi essere scritti in memoria con il valore modificato.
- **WRITE_SAVE**: *o_address* viene portato all'indirizzo di scrittura inizializzato in SET_ADD2 e successivamente incrementato in NEW_PIXEL.
- **SHIFT**: Ad ogni pixel letto si sottrae il valore di *min_pixel* e lo si shifta a sinistra di *shift_level* posizioni, dopo aver aggiunto 8 *leading zeros*.
- **NEW_PIXEL**: Viene scritto in memoria il minimo fra il pixel calcolato nello stato precedente e 255. I segnali di *enable* e *write* vengono portati alla configurazione di scrittura.

- **READ_SAVE**: in *o_address* salviamo l'indirizzo di lettura per il READ_STATE, se invece si è arrivati alla fine dell'immagine da equalizzare allora l'algoritmo è terminato e si arriva in DONE.
- **DONE**: Si setta il segnale *o_done* a 1 e nel caso in cui il segnale di *start* fosse 0 si ritornerebbe in RST.

4. RISULTATI SPERIMENTALI

Vengono riportati i dettagli principali del report di sintesi.

```
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
          2 Input   16 Bit       Adders := 1
          3 Input    8 Bit       Adders := 2
+---Registers :
                       5 Bit    Registers := 1
+---Muxes :
         18 Input   16 Bit        Muxes := 3
         18 Input    8 Bit        Muxes := 8
          2 Input    8 Bit        Muxes := 1
          2 Input    5 Bit        Muxes := 2
          2 Input    4 Bit        Muxes := 1
         18 Input    4 Bit        Muxes := 1
         18 Input    3 Bit        Muxes := 1
          2 Input    3 Bit        Muxes := 4
          2 Input    2 Bit        Muxes := 2
          5 Input    1 Bit        Muxes := 1
         18 Input    1 Bit        Muxes := 15
          2 Input    1 Bit        Muxes := 3
          8 Input    1 Bit        Muxes := 1
---------------------------------------------------------------------------------

Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |BUFG   |     1|
|2     |CARRY4 |    46|
|3     |LUT1   |     2|
|4     |LUT2   |   142|
|5     |LUT3   |    30|
|6     |LUT4   |    92|
|7     |LUT5   |    24|
|8     |LUT6   |   155|
|9     |FDCE   |     5|
|10    |LD     |   162|
|11    |IBUF   |    11|
|12    |OBUF   |    27|
+------+-------+------+
```

```
Report Instance Areas:
+------+---------+-------+------+
|      |Instance |Module |Cells |
+------+---------+-------+------+
|1     |top      |       |   697|
+------+---------+-------+------+

--------------------------------------------------------------------------------
Synthesis finished with 0 errors, 0 critical warnings and 17 warnings.
Synthesis Optimization Complete : Time (s): cpu = 00:00:47 ; elapsed = 00:01:49 .
Memory (MB): peak = 1589.199 ; gain = 880.977.
```

I warnings sono relativi agli 'inferring latch for variable', i quali mantengono i valori dei segnali, seguendo il comportamento dei latch di tipo D.

5. TEST-BENCH

5.1 Test 1 - Immagine normale

Il test consiste in un'immagine 2x2, questo è utile per comprendere il funzionamento l'algoritmo. Dalla *figura 4* si evince che il componente funziona attivamente dal tempo 350ns a 920ns, cioè da quando il segnale di *start* è alzato a quando viene abbassato dalla segnalazione della fine, con *o_done* a 1. Nella riga di *mem_o_data* sono scritti prima i valori del numero di colonne e righe (entrambi uguali a 2) e, successivamente, i valori dei pixel dell'immagine (46, 131, 62, 89). Questi sono letti due volte, la prima per il calcolo di minimo e massimo e la seconda per determinare i nuovi valori dell'immagine. I nuovi valori si trovano nella riga *mem_i_data* (0, 255, 64,172). Da notare la corrispondenza del valore alto di *mem_we* e la presenza di un nuovo valore nella suddetta riga, ciò non si nota per il valore 0 perché il segnale è inizializzato a 0. Osservando la riga degli indirizzi (*mem_address*) si nota un iniziale aumento lineare, corrispondente alla prima lettura dei pixel, seguita da una fase in cui l'indirizzo salta tra valori bassi e valori alti. Questi ultimi corrispondono agli indirizzi di lettura e scrittura che all'interno della macchina sono chiamati *o_address_r* e *o_address_w*. Per questo Test-Bench e per i successivi il tempo di clock utilizzato è di 15ns, o come segnato nella forma d'onda 15000 ps.

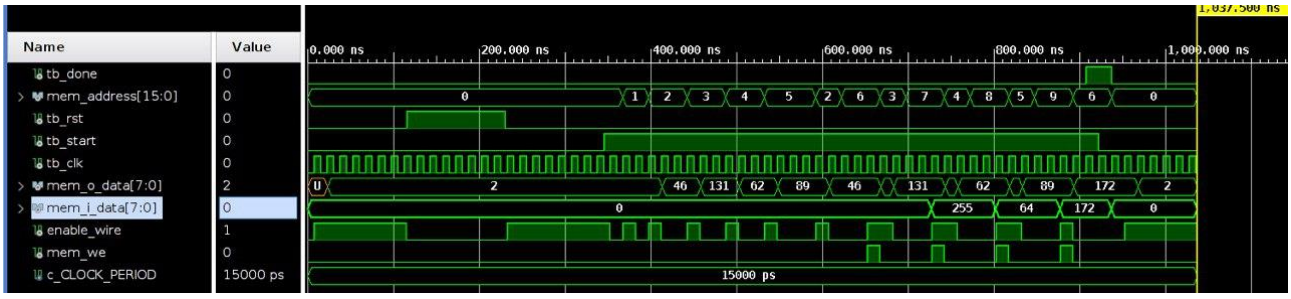

Figura 4. Forma d'onda del Test-Bench numero 1.

Time Behavioral Simulation: 1037,5 ns
Time Post-Synthesis Simulation: 1037,7 ns

5.2 Test 2 - 15 immagini consecutive

Lo scopo di questo test è quello di studiare la robustezza del componente con più immagini consecutive e, quindi, il giusto utilizzo del segnale *done* e la giusta risposta ai segnali *start* e *rst*.

Time Behavioral Simulation: 16802,3 ns
Time Post-Synthesis Simulation: 16802,5 ns

5.3 Test 3 - Reset Asincrono

Il test verifica che, se il segnale di reset è alzato in modo asincrono, la computazione non è compromessa, cioè la macchina ritorna correttamente nello stato iniziale RST.

Time Behavioral Simulation: 1682,5 ns
Time Post-Synthesis Simulation: 1682,7 ns

5.4 Test 4 - Immagine 128x128 pixel

Questa immagine rappresenta la dimensione massima di pixel dichiarata nella specifica.

Time Behavioral Simulation: 1966637,5 ns
Post-Synthesis Simulation: 1966637,7 ns

I successivi test studiano il comportamento del componente quando riceve come input immagini che rappresentano casi di frontiera. È particolarmente interessante il primo, cioè un'immagine di 0 pixel, perché testa l'abilità dell'algoritmo di finire subito e non sporcare la memoria.

5.5 Test 5 - Immagine con 0 pixel

Time Behavioral Simulation: 875 ns
Time Post-Synthesis Simulation: 875,2 ns

5.6 Test 6 - Immagine di tutti pixel '1'

Time Behavioral Simulation: 1517,5 ns
Time Post-Synthesis Simulation: 1517,7 ns

5.7 Test 7 - Immagine di tutti pixel '255'

Time Behavioral Simulation: 1517,5 ns
Time Post-Synthesis Simulation: 1517,7 ns

6. CONCLUSIONI

Partendo dai pixel di un'immagine in input si è arrivati a un'immagine equalizzata in output, secondo il metodo di equalizzazione dell'istogramma.
Come dimostrato dai test il componente è veloce e affidabile, infatti esso riesce ad equalizzare correttamente tutte le immagini che possono essere date in input, con dimensione massima 128x128 pixel.
Nella costruzione della FSM si è preferito avere stati semplici e funzionali piuttosto che avere stati con una logica complessa, perché ciò rende il codice facilmente modificabile.

Sono stati analizzati tutti gli aspetti progettuali del componente descritto in VHDL e sintetizzato usando XILINX VIVADO WEBPACK con Target FPGA: (xc7k70tfbv676-1).