

|  |
| --- |
| Prova Finale  Progetto  Di  Reti Logiche  A.A 2020/2021 |
|  |
| Prof. William Fornaciari  Leonardo Dezi - 10588151  Roberto Donato - 10576279 |



# INDICE

1. Introduzione
   1. Scopo del progetto
   2. Interfaccia del componente
   3. Algoritmo utilizzato
   4. Memoria
   5. Esempio
2. Architettura
3. Risultati sperimentali
4. Conclusioni

INTRODUZIONE

* 1. Scopo del progetto

Il progetto prevede la realizzazione di un hardware in grado di equalizzare l’istogramma di un immagine in scala di grigi a 256 bit.

* 1. Interfaccia del componente

Il componente da descrivere possiede 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 deve essere 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. Algoritmo utilizzato

Il modulo implementato riceve i byte dell’immagine dalla memoria in cui è memorizzata leggendo i dati sequenzialmente riga per riga.

Il byte all'indirizzo 0 indica il numero di pixel per riga dell’immagine, mentre il byte all' indirizzo 1 indica il numero di pixel per colonna.

I valori degli n pixel dell'immagine sono scritti sequenzialmente dall' indirizzo “2” in poi, quindi fino all'indirizzo1+n, occupando un byte per ogni pixel.

La dimensione massima dell'immagine da equalizzare è di 128pixel X 128 pixel. L' algoritmo di equalizzazione dell'immagine, ispirato al metodo di equalizzazione dell'istogramma ma fortemente semplificato, trasforma ogni pixel dell'immagine nel modo seguente:

*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)*

In questo algoritmo MAX PIXEL VALUE e MIN PIXEL VALUE, rappresentano il massimo e minimo valore dei pixel dell'immagine, CURRENT PIXEL VALUE è il valore del pixel da trasformare, e NEW PIXEL VALUE si riferisce al valore del nuovo pixel.

La nuova immagine equalizzata verrà scritta in memoria immediatamente dopo l'immagine originale, con indirizzo del primo byte pari a 2+ (# colonne \* # righe)

* 1. Memoria

La memoria R.A.M. (Random Access Memory) prevede l’accesso diretto ad un qualsiasi indirizzo di memoria con lo stesso tempo di accesso. In questo progetto è stata utilizzata una memoria di tal tipo indirizzata a 216 byte.

Immagine che contiene testo

Descrizione generata automaticamente

* 1. Esempio

Qui di seguito viene riportato un esempio di contenuto della memoria al termine di una elaborazione al termine di un’elaborazione di un'immagine di dimensione 4x3. I valori che qui sono rappresentati in decimale sono memorizzati in memoria con codifica binaria su 8 bit senza segno.

*Immagine che contiene tavolo

Descrizione generata automaticamente*