Progetto Reti Logiche
A.A 2020/2021

A cura di:

Samuel Kala (Cod. Persona: 10584699)

Mattia Magliano (Cod. Persona: 10538658)

INDICE

1. RICHIESTE DI PROGETTO

1.1 Introduzione

La specifica della prova finale (progetto di reti logiche) 2020 è ispirata al ***metodo di equalizzazione*** di un'immagine.

Il metodo di equalizzazione dell'istogramma di una immagine è un metodo pensato per ricalibrare il contrasto di essa quando l'intervallo dei valori di intensità sono molto vicini.

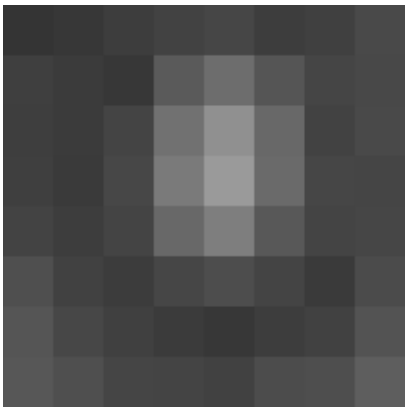


Immagine originale

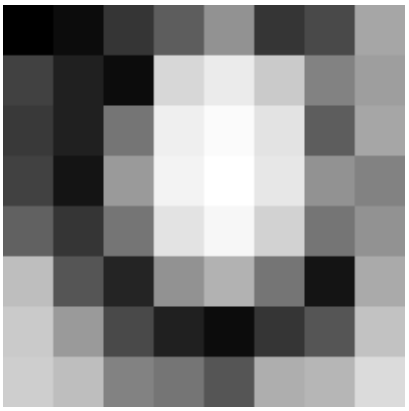


Immagine equalizzata

L'algoritmo di equalizzazione sarà applicato solo ad immagini in scala di grigi a 256 livelli e deve trasformare ogni suo pixel attraverso questo schema di calcoli:

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)

Schema n.1

MAX_PIXEL_VALUE e ***MIN_PIXEL_VALUE*** sono, rispettivamente, il massimo e il minimo valore dei pixel dell'immagine.

DELTA_VALUE è la differenza tra il massimo e il minimo valore di tutti i pixel dell'immagine.

SHIFT LEVEL è il valore dello shift logico verso sinistra, che dovrà essere applicato a CURRENT_PIXEL_VALUE – MIN_PIXEL_VALUE.

CURRENT_PIXEL_VALUE è il pixel corrente che si sta analizzando in memoria.

TEMP_PIXEL è il pixel temporaneo che viene calcolato.

NEW_PIXEL_VALUE è il valore del nuovo pixel restituito (uguale a *temp_pixel* se è minore di 255, altrimenti uguale a 255) che viene scritto in memoria.

1.2 Scopo

Lo scopo del progetto è di equalizzare tutti i pixel di un' immagine (dimensione massima 128x128) attraverso l'algoritmo di equalizzazione semplificato. In particolare, si deve implementare, nel linguaggio VHDL, un componente hardware che deve leggere l'immagine da una memoria in cui sono inseriti, sequenzialmente e riga per riga, tutti i pixel dell'immagine da elaborare.

L'immagine equalizzata deve essere scritta in memoria immediatamente dopo l'immagine originale.

1.3 Interfaccia

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:

- 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.

1.4 Memoria

La memoria presenta una struttura con lettura sequenziale e indirizzamento al byte partendo dalla posizione 0.

In particolare, la **dimensione dell'immagine**, ciascuna di dimensione di 8 bit, è definita dai primi 2 byte:

- il byte nell'indirizzo di memoria 0 corrisponde alla dimensione di colonna (COL)
- il byte nell'indirizzo di memoria 1 corrisponde alla dimensione di riga (RIG)

I **pixel dell'immagine**, ciascuno di dimensione di ***8 bit***, sono memorizzati, in byte contigui, partendo dall'indirizzo 2 della memoria.

I **pixel dell'immagine equalizzata**, anch'essi di ***8 bit*** ciascuno, sono memorizzati con indirizzamento al byte partendo dalla cella di memoria pari a 2 + (COL*RIG).

Esempio di una memoria in cui si memorizza e si restituisce un'immagine equalizzata di dimensione 2x2.

MEMORIA RAM

Indirizzo	Valori Pixel	
0	2	Dimensione Immagine
1	2	
2	46	Pixel immagine originale
3	131	
4	62	
5	89	
6	0	Pixel immagine equalizzata
7	255	
8	64	
9	172	

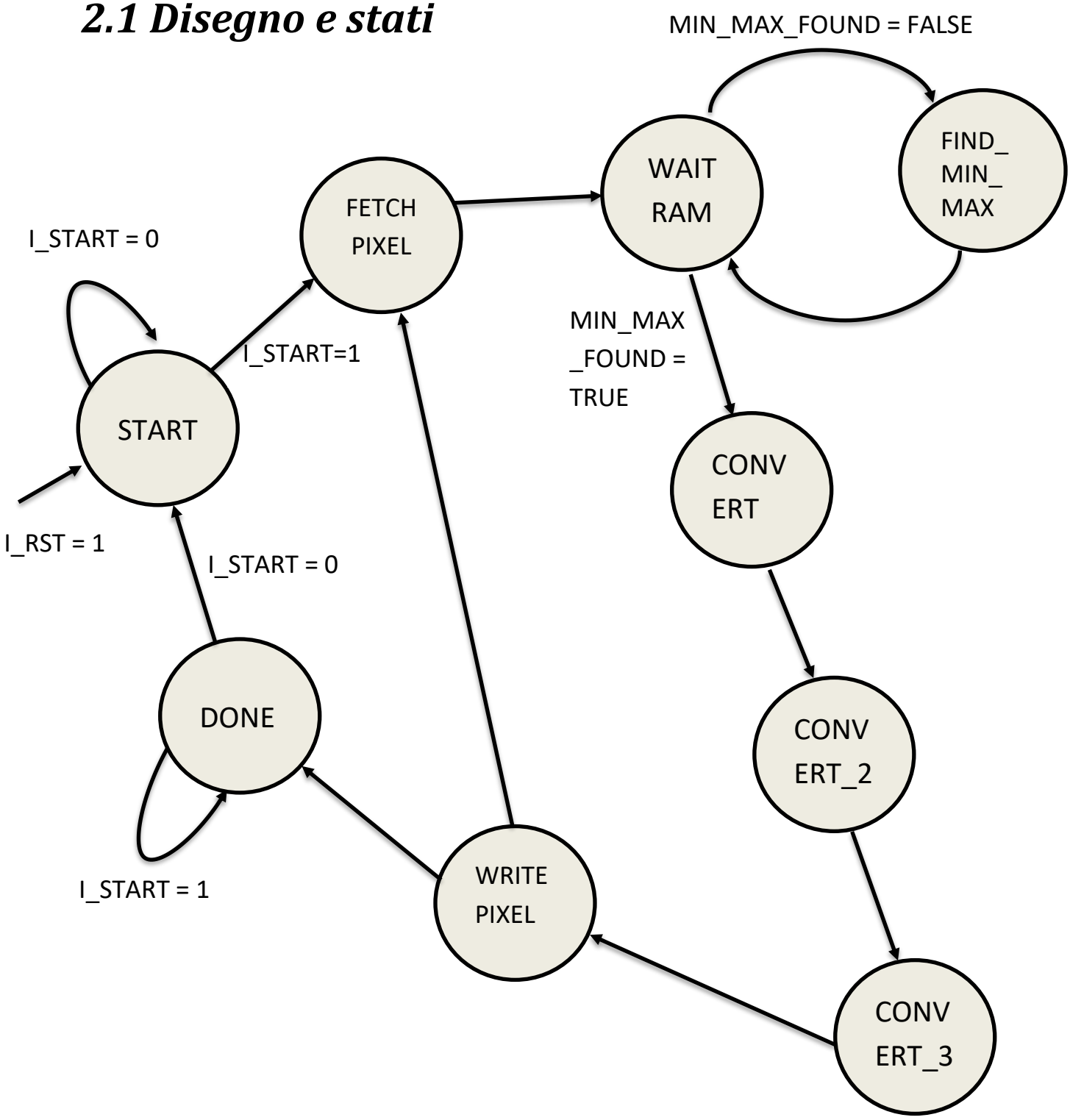
Dati Input: indirizzi 0–5

Dati Output: indirizzi 6–9

Figura n.1 : Esempio Memoria RAM

2. Macchina a stati

2.1 Disegno e stati



La macchina è composta dai ***9 stati*** seguenti:

- ***START*** :
  Stato iniziale nel quale si attende che *i_start* sia uguale a 1. Quando *i_rst* viene alzato a 1, in qualsiasi momento, si ritorna a questo stato.

- ***FETCH_PIXEL*** :
  Stato in cui viene richiesta la comunicazione con la memoria alzando il segnale *o_en.*
  Il flag *min_max_found,* presente in questo stato, permette di differenziare due casi:

  - se uguale a "*false*", nelle parti di codice seguente, si visita ogni indirizzo della RAM per trovare il minimo e il massimo valore di tutti i pixel.

  - se uguale a "*true*" si procede con il "prelievo" di un pixel e la sua successiva conversione.

  Lo stato successivo è WAIT_RAM.

- ***WAIT_RAM*** :
  Stato in cui si attende la lettura della RAM.
  Il prossimo stato sarà FIND_MIN_MAX se il valore del pixel minimo e del pixel massimo non sono stati trovati (flag *min_max_found* = *"false"*); altrimenti, lo stato successivo sarà CONVERT (flag *min_max_found* = "*true"*).

- ***FIND_MIN_MAX*** :
  Stato in cui vengono memorizzati il numero delle colonne, il numero di righe e i pixel con valore minimo e massimo negli appositi registri.
  Prima di passare allo stato WAIT_RAM, nel caso in cui il minimo e il massimo siano stati trovati, il flag *min_max_found* viene messo a "*true*", altrimenti rimane a "*false*".

- ***CONVERT*** :
  Stato in cui viene calcolato *shift_level*.
  Nel caso in cui *shift_level* sia stato calcolato (se il flag *shift_found* = "*true*") non viene ricalcolato.
  Lo stato successivo è CONVERT_2.

- ***CONVERT_2*** :
  Stato in cui viene calcolato *temp_pixel*.
  Lo stato successivo è CONVERT_3.

- ***CONVERT_3*** :
  Stato in cui viene calcolato *new_pixel*.
  Lo stato successivo è WRITE_PIXEL.

- ***WRITE_PIXEL*** :
  Stato in cui scriviamo sulla RAM il valore del pixel convertito.
  Se il pixel convertito è l'ultimo da scrivere allora si va nello stato DONE; altrimenti, si va nello stato FETCH_PIXEL per procedere con la conversione dei pixel rimanenti.

- ***DONE*** :
  Stato finale in cui il segnale *o_done* è posto uguale a 1. Si rimane in questo stato finché il segnale *i_start* non è posto uguale a 0. Quando ciò avviene si ritorna allo stato START e *o_done* viene posto uguale a 0.

2.2 Scelte progettuali

Si è deciso di implementare ***due process***: il primo per la gestione dei segnali di *i_clk* e *i_rst* e la rappresentazione dei registri di stato; il secondo per la gestione della parte combinatoria.

Per quanto riguarda la scelta e la disposizione degli stati, due sono i punti più critici: la corretta lettura in memoria e il calcolo del pixel da convertire. Il primo problema è stato risolto con l'introduzione di uno stato WAIT_RAM per garantire una lettura coerente della memoria. Infatti, per leggere da un determinato indirizzo della RAM servono ***due cicli di clock.*** Al termine del primo ciclo viene aggiornato l'indirizzo da cui leggere, al termine del secondo viene letto il dato corretto.

Per il secondo problema, si è proceduto suddividendo la fase di conversione del valore del pixel in tre stati. Questo perché, per fare certe operazioni, sono necessari degli elementi che siano stati calcolati in stati precedenti.

Per esempio, per calcolare *temp_pixel* nello stato CONVERT_2 serve il valore di *shift_level* che, per essere corretto, deve essere stato calcolato in uno stato precedente (CONVERT). Stessa cosa per quando riguarda *new_pixel* e *temp_pixel*. Infatti, per calcolare *new_pixel* in

CONVERT_3 serve che *temp_pixel* venga calcolato in uno stato precedente (CONVERT_2).

Altri due aspetti da considerare sono il calcolo di delta_value e temp_pixel. Per quanto riguarda *delta_value,* esso può assumere il valore massimo di 256 (nel codice l'operazione di (*max_pixel – min_pixel + 1*) è stata assegnata direttamente a *delta_value*) nel caso in cui i valori di *min_pixel* e di *max_pixel* siano rispettivamente 0 e 255. Un *delta_value* pari a 256 ***non può essere rappresentato su 8 bit***. Di conseguenza, si è scelto di rappresentarlo su ***9 bit***.

Inoltre, per il calcolo di *temp_pixel* in CONVERT_2 si è dovuto gestire il caso in cui, dopo lo shift, si ottiene un valore di *temp_pixel* maggiore di 255. Per stabilire il numero di bit necessari per rappresentare temp_pixel, si considera il caso limite in cui *shift_value* e *cur_pixel* sono rispettivamente 8 e 255. A tal proposito, *temp_pixel* è rappresentabile con 16 bit.

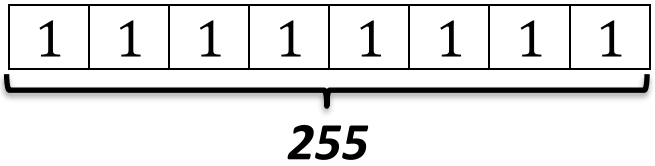


Figura n.2 : caso *cur_pixel* = 255

1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figura n.3 : caso *temp_pixel* massimo (dopo shift a sinistra di 8 bit di *cur_pixel* = 255)

A questo punto, si assegna a *new_pixel* il valore 255 (su 8 bit: “11111111”) se *temp_pixel* è maggiore di 255; altrimenti *new_pixel* prende il valore del numero rappresentato sugli 8 bit meno significativi di *temp_pixel*.

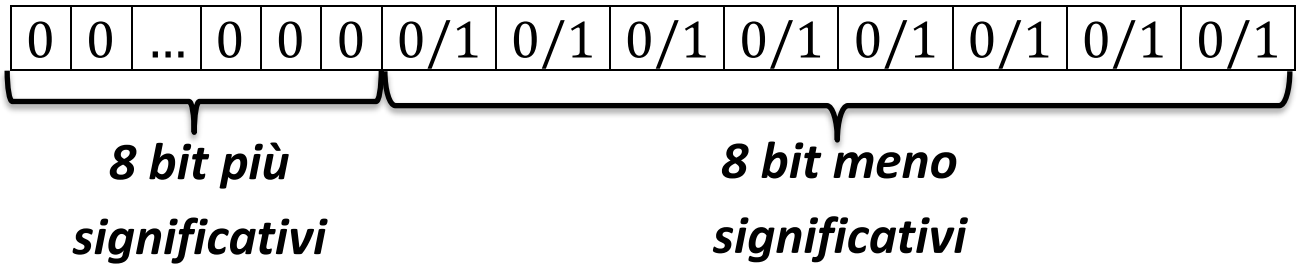


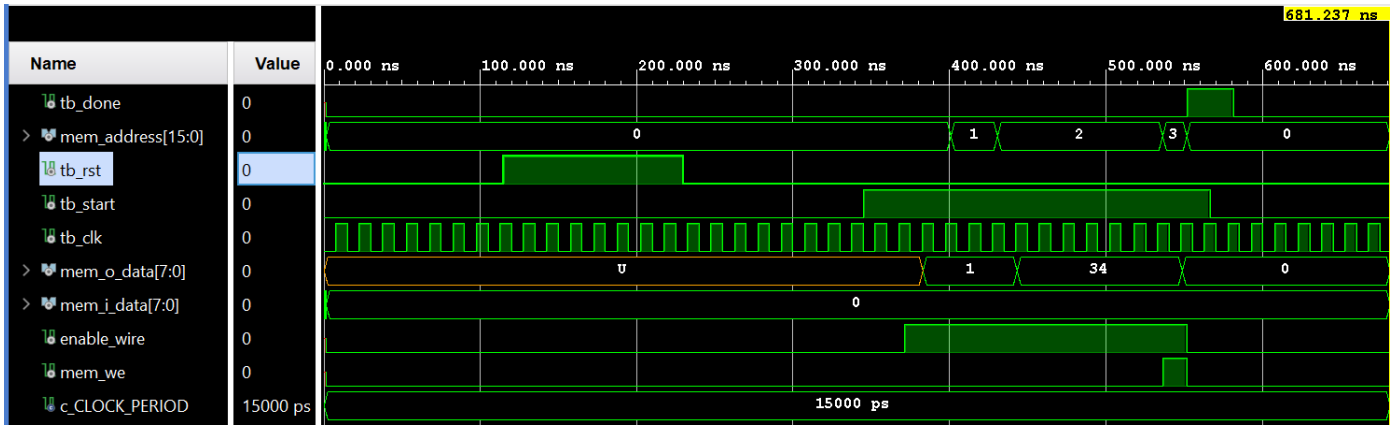
Figura n.4 : caso *temp_pixel* <=255

Infine, un’ ottimizzazione che si è pensato di implementare è quella di trovare in unico stato (FIND_MIN_MAX) il numero di righe, di colonne, il pixel minimo e il pixel massimo. Tutti questi valori sono utili nelle fasi successive di calcolo.

3. Test Bench

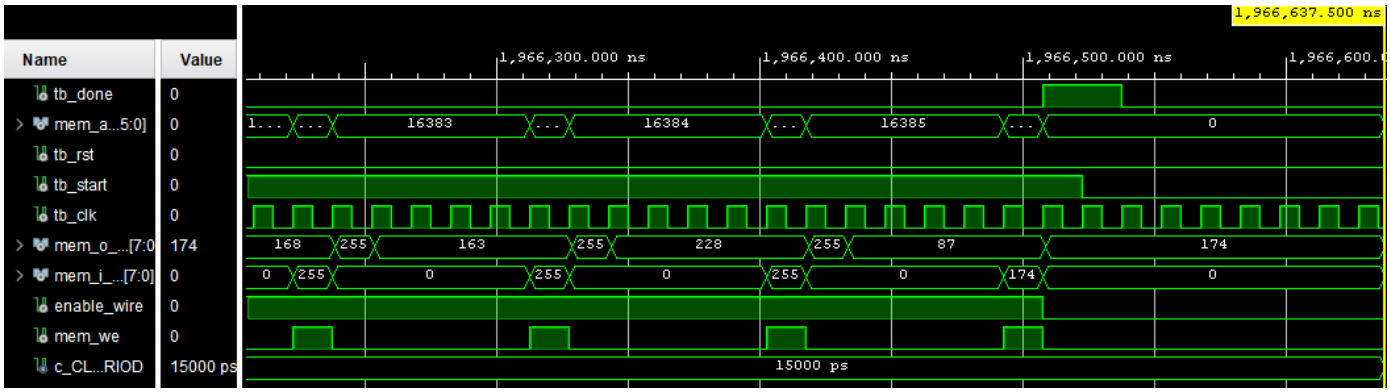
Sono stati creati dei Test Bench per testare il corretto funzionamento della macchina. In particolare, si sono effettuate prove che andassero a testare i casi limite in tre modalità di simulazione: Post-synthesis Functional, Post-synthesis Timing e Behavioural. I test effettuati sono i seguenti:

- Test con immagine da convertire formata da ***1 pixel***:



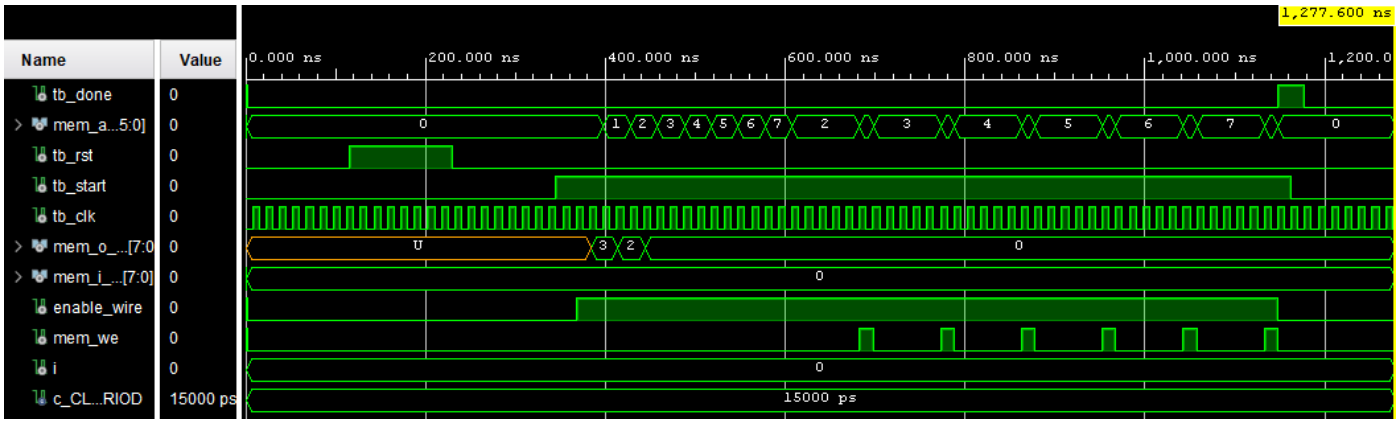
Test 1

- Test con immagine da convertire formata da ***128x128 pixel***:



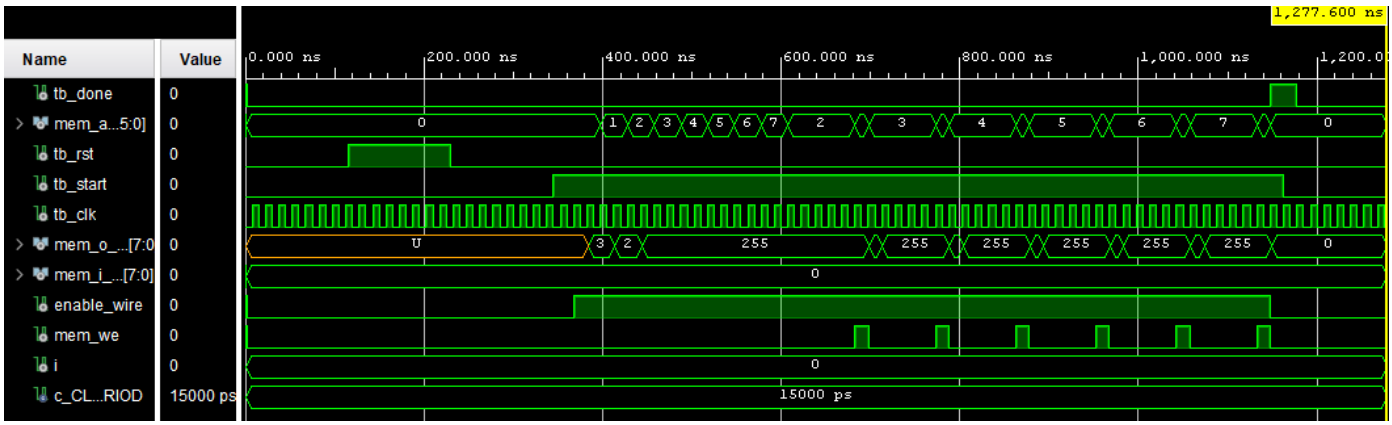
Test 2

- Test con immagine da convertire contenente solo ***pixel uguali a 0***:



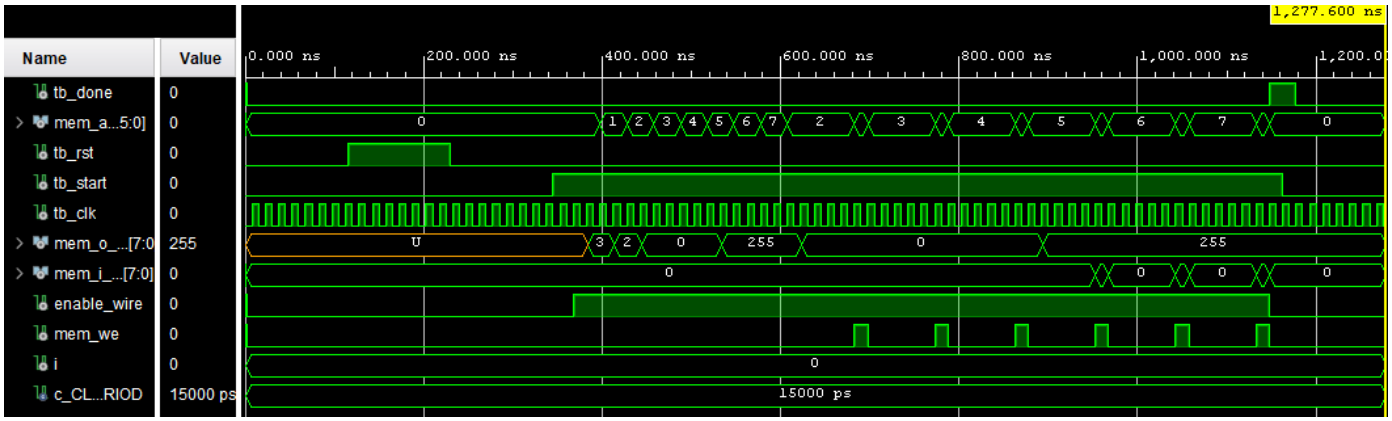
Test 3

- Test con immagine da convertire contenente solo ***pixel uguali a 255***:



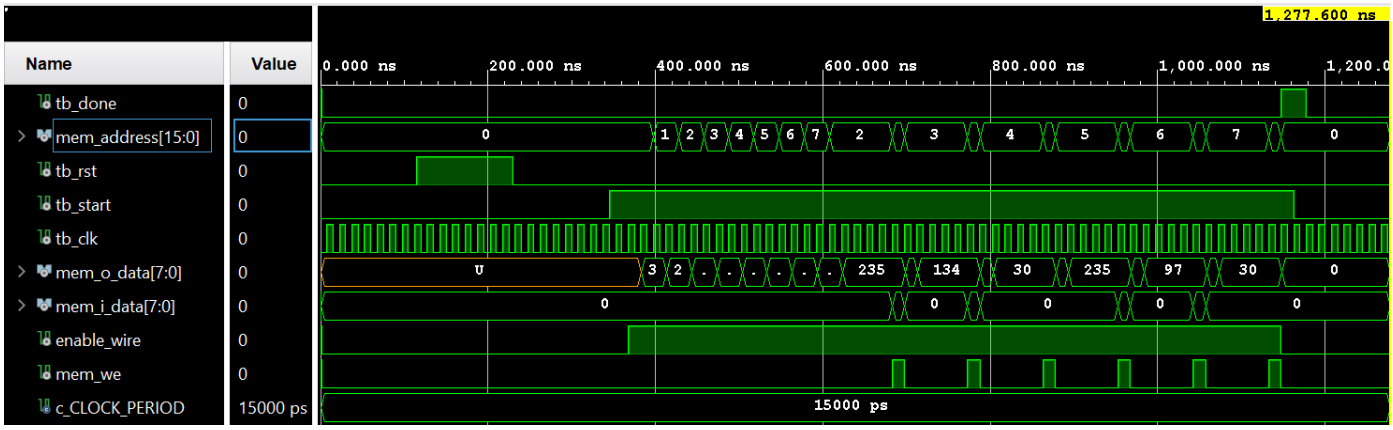
Test 4

- Test con immagine da convertire contenente ***pixel uguali a 0 o 255***:



Test 5

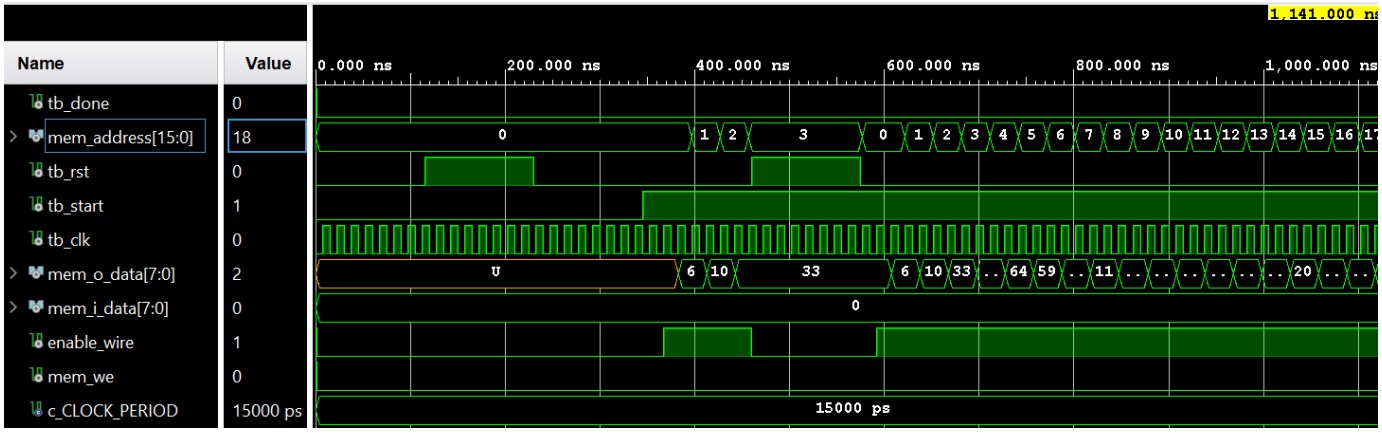
- Test con immagine da convertire contenente ***pixel massimo in prima posizione e pixel minimo in ultima posizione***:



Test 6

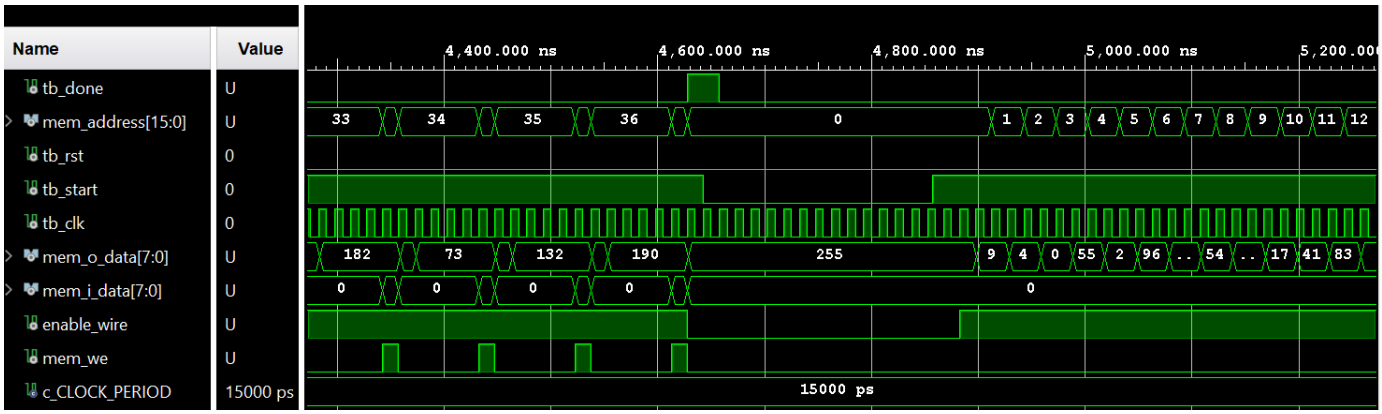
- Test con immagine da convertire contenente ***pixel minimo in prima posizione e pixel massimo in ultima posizione*** (simile a quello precedente).

- Test con ***reset asincrono*** (i_rst=1 durante l'esecuzione):



Test 7

- Test con ***più immagini da convertire sequenzialmente***:



Test 8

Inoltre, sono stati eseguiti ulteriori test casuali, creati tramite l'utilizzo di un apposito programma scritto in linguaggio C.

4. Conclusioni

4.1 Risultati

Dai report ottenuti tramite l'apposito tool di Vivado (IDE di sviluppo), si possono osservare alcuni parametri significativi. In particolare, il *Worst Negative Slack* (*WNS*), che quantifica sul tempo totale a disposizione qual è il tempo del path peggiore (espresso in *ns*).

WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF
							288	110
85.308	0.000	0.160	0.000	0.000	0.133	0	284	110

Report 1

Altri due parametri importanti riguardano i componenti che occupano fisicamente la superficie della FPGA:

- Lookup tables (LUTs)
- Registri

Come si può notare dalla tabella *Report 2*, le percentuali di utilizzo di entrambe le categorie di componenti è molto bassa. Questo è stato possibile grazie a scelte progettuali che cercassero di ottimizzare l'area di utilizzo della FPGA.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	284	0	133800	0.21
LUT as Logic	284	0	133800	0.21
LUT as Memory	0	0	46200	0.00
Slice Registers	110	0	267600	0.04
Register as Flip Flop	110	0	267600	0.04
Register as Latch	0	0	267600	0.00

Report 2

4.2 Commento finale

Dopo tutte le considerazioni fatte in precedenza si può affermare che il componente è stato sintetizzato correttamente.