POLITECNICO
MILANO 1863

PROVA FINALE

(PROGETTO DI RETI LOGICHE)

Stefano Fumagalli – Codice Persona: 10628587

Lara Ferro – Codice Persona: 10622035

Corso di Reti Logiche

Anno accademico 2020-2021

Professore: Fabio Salice

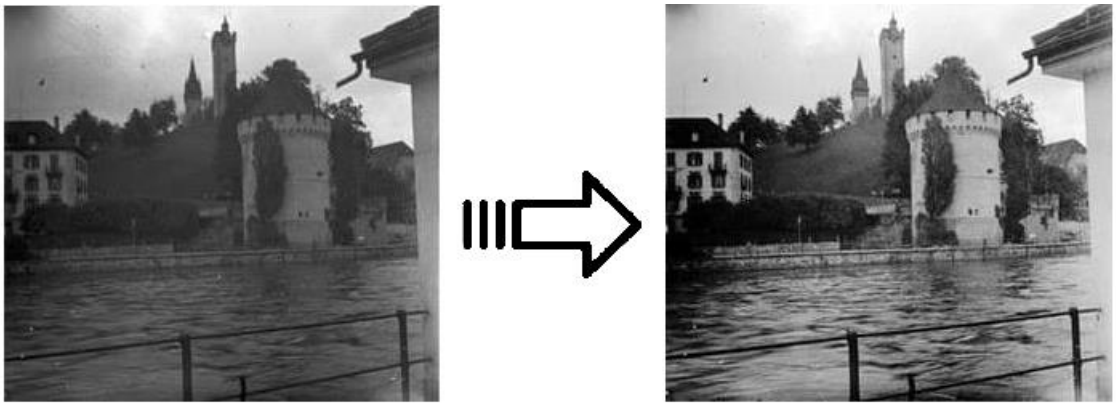
Indice:

1. Introduzione

1.1 Scopo del progetto

L'obiettivo del progetto consiste nel descrivere in VHDL e sintetizzare il componente hardware che, presi in ingresso la dimensione di una foto in scala di grigi a 256 livelli e il valore di ciascun pixel, applica l'algoritmo di equalizzazione e restituisce il valore dei pixel aggiornati.

Il processo di equalizzazione permette di incrementare il contrasto di un'immagine aumentando il suo intervallo di valori. Si fornisce di seguito un esempio:



1.2 Interfaccia del componente

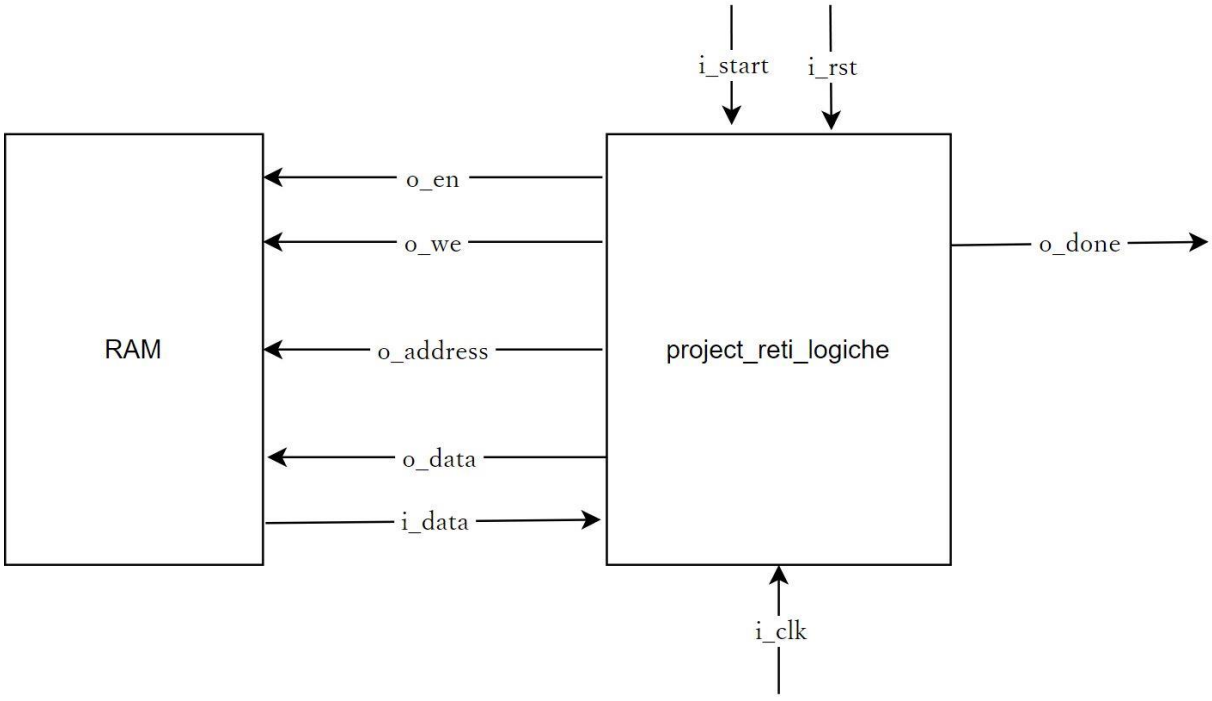
L'interfaccia del componente da descrivere è la seguente:

```
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;
```

Nello specifico:

Segnale	Descrizione
i_clk	Segnale di clock fornito in ingresso dal test bench
i_rst	Segnale di reset per preparare la macchina a ricevere la prima immagine
i_start	Segnale di start per iniziare ad eseguire le operazioni
i_data	Segnale di input che contiene i dati letti dalla memoria
o_address	Segnale di output che specifica l'indirizzo della memoria
o_done	Segnale di output per avvisare della fine dell'elaborazione
o_en	Segnale di enable per poter comunicare con la memoria
o_we	Segnale di write enable che indica l'operazione da svolgere in memoria: o_we = 0 ⇒ lettura o_we = 1 ⇒ scrittura
o_data	Segnale di uscita che contiene i dati inviati alla memoria

Si fornisce inoltre uno schema che illustra l'interazione tra la RAM ed il modulo implementato attraverso i segnali precedentemente descritti.



1.3 Descrizione della memoria

La memoria è indirizzata al Byte e contiene i dati riguardanti l'immagine da equalizzare, con i pixel memorizzati sequenzialmente riga per riga. In particolare, i primi 2 Byte in memoria indicano la dimensione dell'immagine e i successivi i valori dei singoli pixel, seguiti dai valori aggiornati una volta applicato l'algoritmo di equalizzazione.

Numero colonne	Indirizzo 0
Numero righe	Indirizzo 1
Valore primo pixel	Indirizzo 2
…	…
Valore ultimo pixel	Indirizzo n+1
Valore primo pixel equalizzato	Indirizzo n+2
…	…
Valore ultimo pixel equalizzato	Indirizzo 2n+1

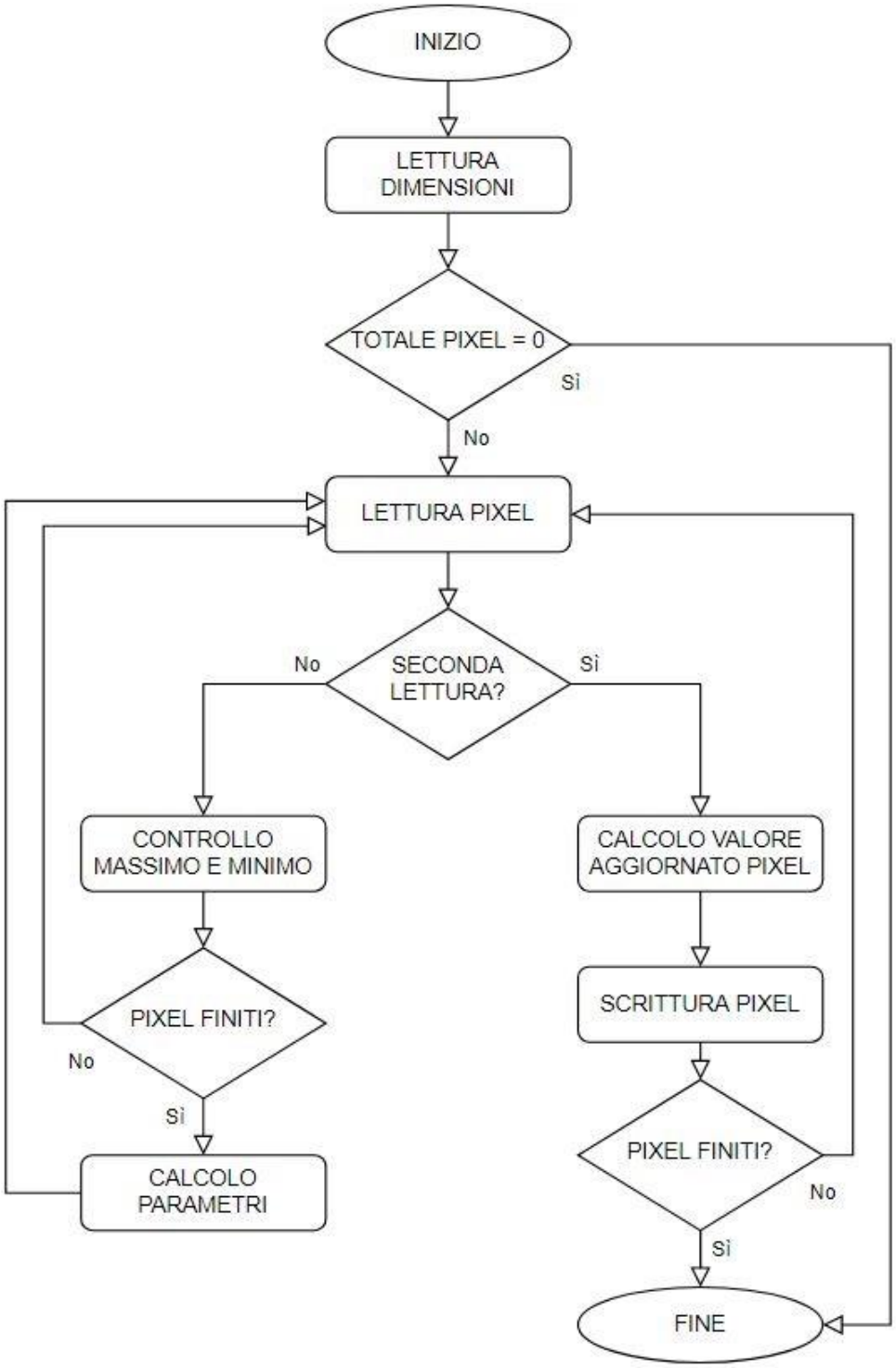
Dove n = # Colonne * # Righe = # Totale Pixel

2. Architettura

2.1 Algoritmo

L'algoritmo utilizzato, raffigurato nel diagramma di flusso sottostante, si sviluppa secondo i seguenti passaggi:

- Inizializzazione delle variabili e successiva lettura delle dimensioni dell'immagine, con controllo per verificare se l'immagine è nulla;
- Primo ciclo nel quale si effettua la lettura dei pixel che compongono l'immagine, individuando tra questi il valore massimo e il valore minimo;
- Calcolo dei parametri delta_value e shift_level necessari al processo di equalizzazione;
- Secondo ciclo nel quale, per ogni pixel, viene aggiornato e memorizzato il suo valore seguendo le regole di incremento del contrasto, terminando poi l'algoritmo.

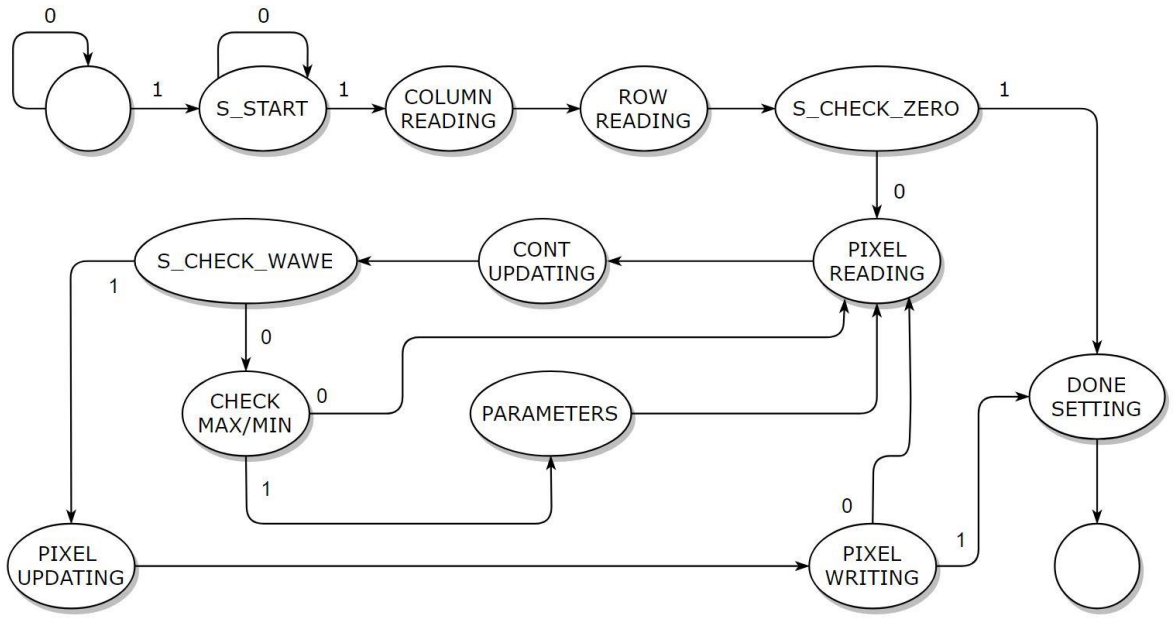


Si è deciso di sviluppare il componente richiesto come una macchina a stati finiti seguendo l'algoritmo descritto.

2.2 Macchina a stati finiti

Una prima versione della MSF che esegue la conversione di un'immagine è così strutturata:

- *Stato iniziale:* attesa ricevimento del segnale i_rst = 1;
- *S_START:* attesa del segnale i_start = 1 per avviare la lettura dell'immagine;
- *COLUMN READING:* lettura della cella in memoria contenente il numero di colonne;
- *ROW READING:* lettura della cella in memoria contenente il numero di righe;
- *S_CHECK_ZERO:* stato in cui si controlla se l'immagine è composta da zero pixel;
- *PIXEL READING:* lettura della cella di memoria contenente il valore di un pixel;
- *COUNT UPDATING:* incremento del contatore utilizzato per tenere traccia degli indirizzi di memoria e del numero totale di pixel letti;
- *S_CHECK_WAVE:* stato in cui si controlla se si è nel secondo ciclo di lettura;
- *CHECK MAX/MIN:* stato in cui si aggiornano i valori di massimo/minimo e in cui si controlla se tutti i pixel sono stati letti;
- *PARAMETERS:* stato in cui si calcolano delta_value e shift_level;
- *PIXEL UPDATING:* stato in cui si determina il nuovo valore di ogni pixel;
- *PIXEL WRITING:* scrittura in memoria del valore aggiornato e verifica che tutti i pixel sono stati letti;
- *DONE SETTING:* stato in cui si porta o_done a 1 e si attende che il segnale i_start venga riportato a 0 per riportare a sua volta o_done a 0;
- *Stato finale:* terminazione della conversione dell'immagine.



2.2.1 Vincoli strutturali

A causa dei vincoli strutturali imposti da VHDL la macchina a stati finiti precedentemente descritta non è sintetizzabile, perciò si sono apportate le seguenti modifiche:

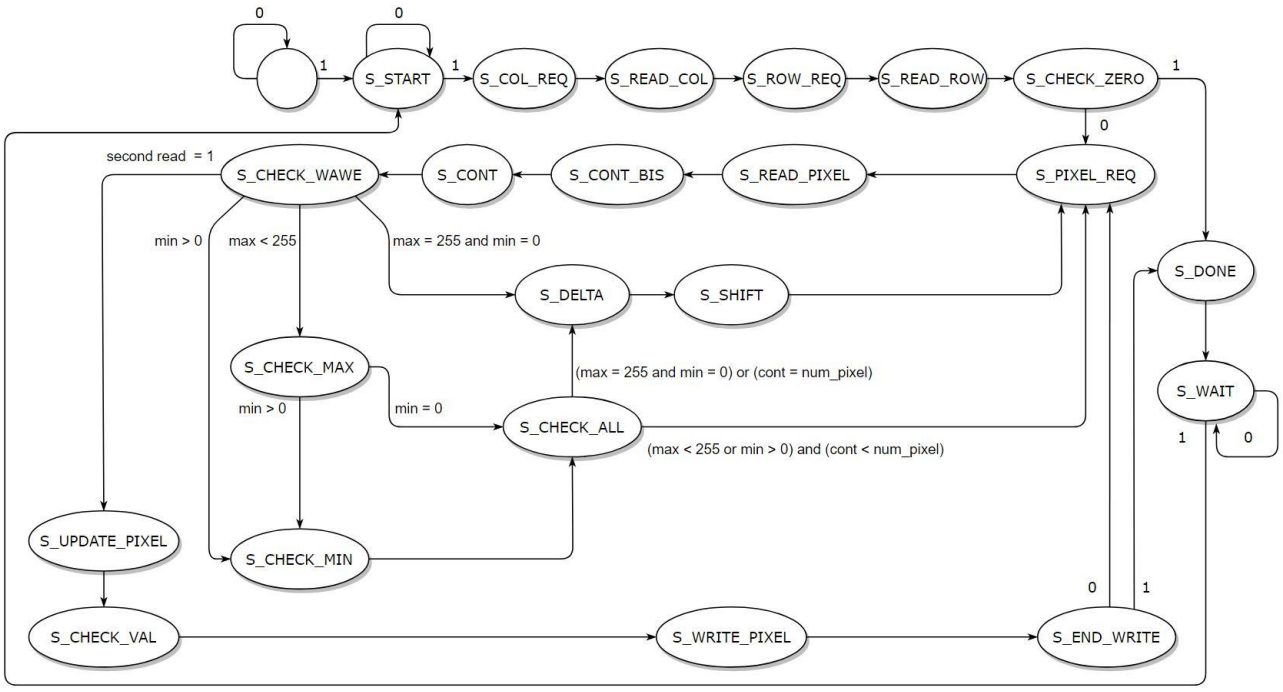
- Gli stati che effettuano la lettura, quali *COLUMN READING*, *ROW READING* e *PIXEL READING*, sono stati suddivisi in due stati. Il primo per la richiesta alla memoria attraverso il segnale o_en = 1, o_we = 0 e per settare o_address; il secondo per l'effettiva lettura del valore tramite i_data;
- Lo stato che effettua la scrittura, *PIXEL WRITING*, è stato anch'esso separato: il primo stato effettua la scrittura su memoria tramite o_en = 1, o_we = 1, o_address, o_data e il secondo stato per terminare l'interazione con la memoria e controllare lo stato futuro della macchina.

2.2.2 Ottimizzazioni

Le ottimizzazioni riguardano lo stato *CHECK MAX/MIN*. Si è deciso di divederlo in due stati: uno per controllare il valore massimo e uno per controllare il valore minimo. Così facendo la macchina visita questi stati solo quando max_pixel_value e min_pixel_value sono rispettivamente minore di 255 e maggiore di 0 (corrispondenti agli estremi dei valori assumibili dai pixel). Nel caso in cui si trovino max_pixel_value = 255 e contemporaneamente min_pixel_value = 0, la macchina ignora i pixel successivi andando direttamente al calcolo dei parametri, velocizzando l'esecuzione.

2.2.3 Progetto finale

Per un corretto funzionamento, i comandi degli stati *CONT UPDATING*, *PARAMETERS*, *PIXEL UPDATING* e *DONE SETTING* sono stati suddivisi in più passaggi. In questo modo aumenta la leggibilità del codice e il corretto aggiornamento dei segnali. La macchina risultante dopo tali modifiche è riportata di seguito:



NB: Ad ogni ciclo di clock il modulo controlla se i_rst è alto. In tal caso la macchina viene riportata allo stato S_START dove si pone in attesa di ricevere i_start = 1 per iniziare nuovamente il processo di equalizzazione. È perciò sottinteso, per ogni stato, un arco uscente diretto a S_START.

A questo punto si è deciso di implementare il componente secondo la specifica, utilizzando un approccio monoprocesso. Si è giunti a questa scelta ai fini di mantenere il codice semplice, di facile manutenzione e di conservare la sequenzialità dell'algoritmo.

3. Risultati sperimentali

3.1 Sintesi

Analizzando i reports di sintesi si sono individuate diverse informazioni riguardanti l'implementazione della macchina:

- Nel Report Utilization si trovano le informazioni riguardanti il numero di LUT e registri utilizzati:

Risorsa	Utilizzo	Disponibilità	Utilizzo in %
LUT	477	134600	0.35%
FF	199	269200	0.07%

Si nota inoltre che tutti i registri utilizzati risultano essere dei Flip Flop. Non vi è quindi la presenza di Latches che avrebbero potuto creare problemi nei test post sintesi.

- Nel Report Timing si trovano le informazioni riguardanti le tempistiche del clock, in particolare si è individuato che lo Slack Time è di 92.044 ns e il Data Path Delay è di 7.838 ns, inferiore al clock minimo che vale 100 ns. Possiamo inoltre calcolare la massima frequenza alla quale il componente può funzionare:

$$f_{MAX} = \frac{1}{7.838ns} \cong 127.6\ MHz$$

4. Simulazioni

4.1 Test bench della specifica

Il primo test effettuato è stato quello fornito insieme alla specifica del progetto. Esso prevede un'immagine di dimensione pari a 4 pixel, con 2 colonne e 2 righe. La macchina inizia leggendo i valori dei pixel e ricercando massimo e minimo (punto ① nella foto della Behavioral Simulation), successivamente calcola i valori delta_value e shift_level. Infine, dopo una seconda lettura e un controllo sul minimo tra il valore ottenuto e 255, scrive il valore aggiornato in memoria (punto ② nella foto della Behavioral Simulation). Dopo tali passaggi, il test controlla che i valori siano stati inseriti correttamente attraverso degli asserts. Se superati, la simulazione si conclude positivamente.

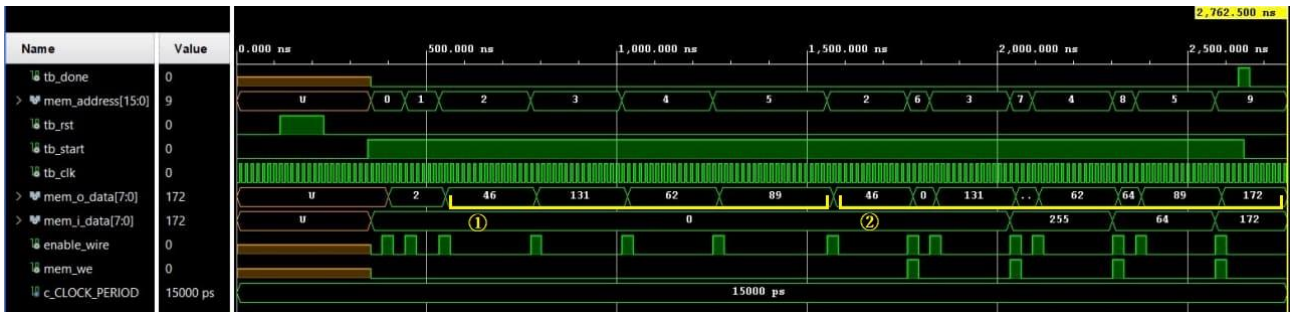
Si riportano di seguito i risultati attesi:

Parametro	Valore
delta_value = max_pixel_value – min_pixel_value	85
shift_level = $(8 - FLOOR\ (\log_2(\text{delta\_value} + 1)))$	2

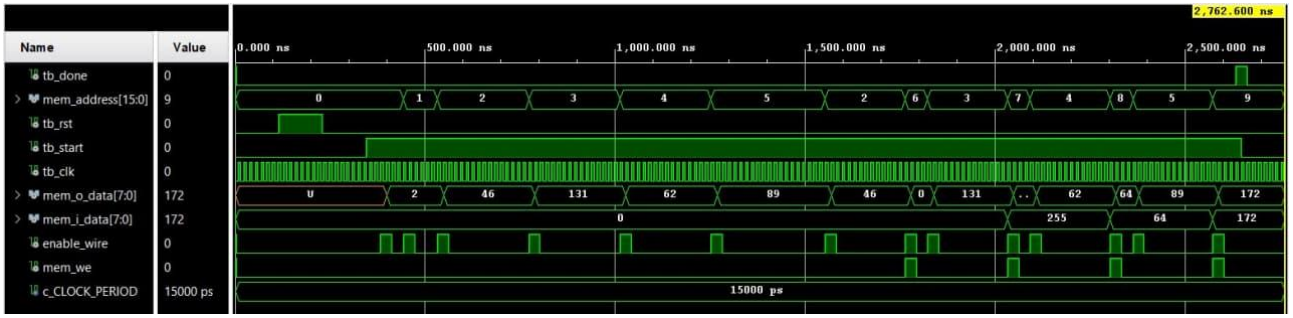
Indirizzo	Contenuto	Descrizione
0	2	Numero colonne
1	2	Numero righe
2	46	Valore primo pixel
3	131	Valore secondo pixel
4	62	Valore terzo pixel
5	89	Valore quarto pixel
6	0	Valore primo pixel equalizzato
7	255	Valore secondo pixel equalizzato
8	64	Valore terzo pixel equalizzato
9	172	Valore quarto pixel equalizzato

Sono state svolte sia la Behavioral Simulation e la Post-Synthesis Functional Simulation come richiesto. I risultati ottenuti sono riportati di seguito:

- Behavioral Simulation: la macchina supera correttamente il test con un tempo pari a 2,763 μs.



- Post-Synthesis Functional Simulation: la macchina supera correttamente il test con lo stesso tempo; si può notare che i segnali che prima venivano inizializzati come Unknown ora hanno un valore iniziale che rimane però ininfluente ai fini del test.



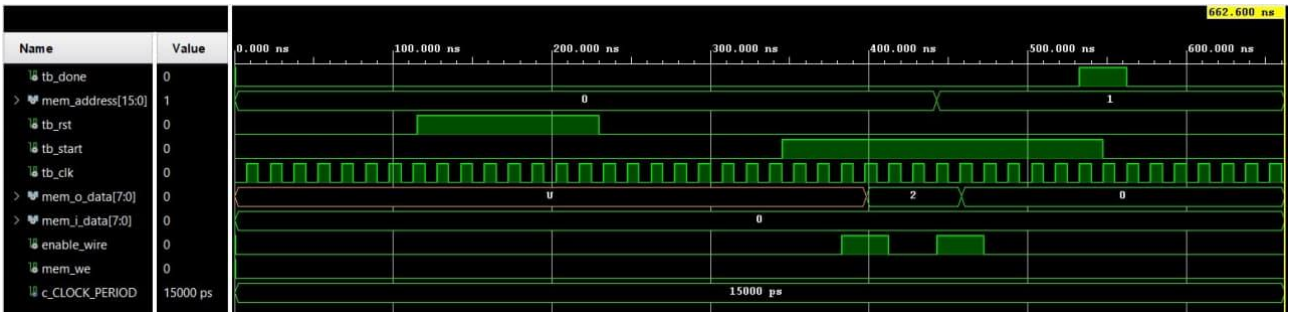
4.2 Test bench casi limite

Sono stati svolti altri test sia con la Behavioral Simulation che con la Post-Synthesis Functional Simulation per controllare il comportamento del componente nei casi limite. Sono di seguito elencati i più significativi:

4.2.1 Immagine con 0 pixel

In questo test si controlla il funzionamento del modulo quando viene fornita un'immagine con dimensione pari a 0 pixel. Il risultato ottenuto rispecchia le aspettative in quanto non appena si riscontra questa condizione il procedimento termina portandosi nello stato S_DONE lasciando inalterata la memoria. Tale test rappresenta il caso più veloce che si possa trovare e viene completato in soli 0.663 μs.

- La Post-Synthesis Functional Simulation ha prodotto il seguente andamento:



4.2.2 Immagine con 16384 pixel

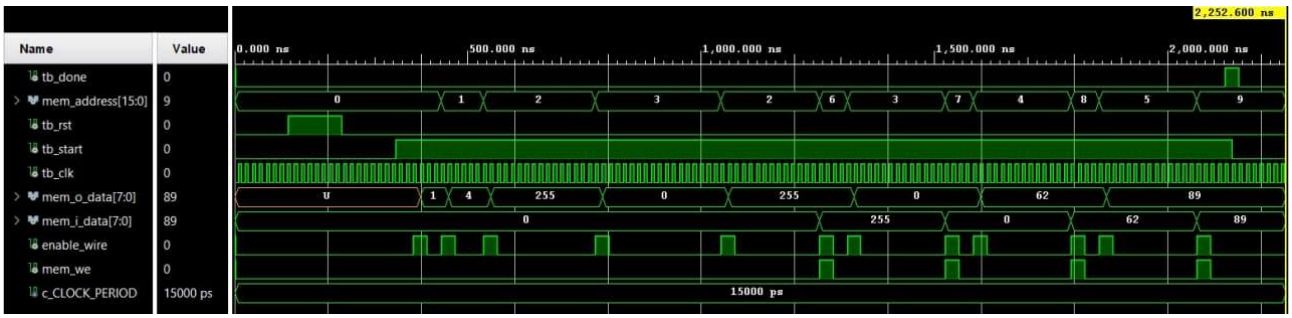
Nel successivo test si è controllato il funzionamento del modulo con in ingresso un'immagine da 16384 pixel, ovvero la massima dimensione possibile corrispondente a 128 colonne e 128 righe. La macchina lavora coerentemente con le specifiche, terminando l'esecuzione in un tempo di 29876.85 μs.

- Non si riporta il grafico della simulazione perché di dimensioni eccessive.

4.2.3 Immagine con MAX e MIN estremi

In questo test si è sottoposto a controllo il componente quando l'immagine ricevuta in ingresso ha pixel massimo pari a 255 e minimo a 0. Grazie all'ottimizzazione, ci si aspetta che il primo ciclo di lettura termini non appena vengono individuati i pixel con valori estremi. Per rendere evidente il miglioramento si è fatto uso di un'immagine della stessa dimensione del test fornito dalla specifica. Il risultato si ottiene dopo 2.253 μs, riportando una riduzione del tempo di 0,51 μs, pari a un miglioramento del 18,46%.

- L'andamento della Post-Synthesis Functional Simulation è il seguente:



4.2.1 Più immagini con estremi di soglia

L'ultimo insieme di casi limite individuato riguarda il controllo delle soglie possibili per il delta_value e l'acquisizione multipla di immagini. Nel test si susseguono 16 immagini caratterizzate da un delta_value rispettivamente pari a 0, 1, 2, 3, 6, 7, 14, 15, 30, 31, 62, 63, 126, 127, 254 e 255, ovvero i valori per cui il logaritmo cambia FLOOR. Il tempo di esecuzione di questo test è 200.45 μs.

- Non si riporta il grafico della simulazione perché di dimensioni eccessive.

4.3 Ulteriori test

Sono stati effettuati, inoltre, più di 10 mila test generati randomicamente da un programma in C++ che hanno portato tutti ad un esito positivo.

Infine, si è sottoposto il codice a dei test che prevedevano i_rst = 1 anche nel mezzo del processo di equalizzazione. Anche questi hanno avuto esito positivo.

5. Conclusioni

Il componente realizzato ha dimostrato di essere in grado di superare tutti i test eseguiti. Per completezza, tutti i test sono stati simulati anche in Post-Synthesis Timing Simulation e in Post-Implementation sia Functional che Timing, che sono risultati anch'essi corretti. Le tempistiche rispettano l'unico vincolo fornito dalla specifica riguardante il tempo minimo di clock pari a 100 ns.

Si ritiene quindi di aver descritto un componente hardware conforme alle specifiche in grado di rispondere correttamente ad ogni tipo di richiesta fornitagli.