Reti Logiche

Prova Finale

Prof. Fabio Salice - Anno *2020/2021*

Chiara Fumelli (Codice Persona Matricola)
Leonardo Ferro (Codice Persona Matricola)

POLITECNICO
MILANO 1863

Indice

1 Introduzione 2

1.1 Scopo del Progetto
1.2 Specifiche Generali
1.3 Interfaccia del Componente
1.4 Dati e Descrizione della Memoria

2 Design 5

2.1 Stati della Macchina
2.2 Segnali Utilizzati
2.3 Scelte Progettuali

3 Risultati dei Test 8

4 Concluisoni 10

4.1 Risultati della Sintesi
4.2 Ottimizzazioni

POLITECNICO
MILANO 1863

Introduzione

1.1 Scopo del Progetto

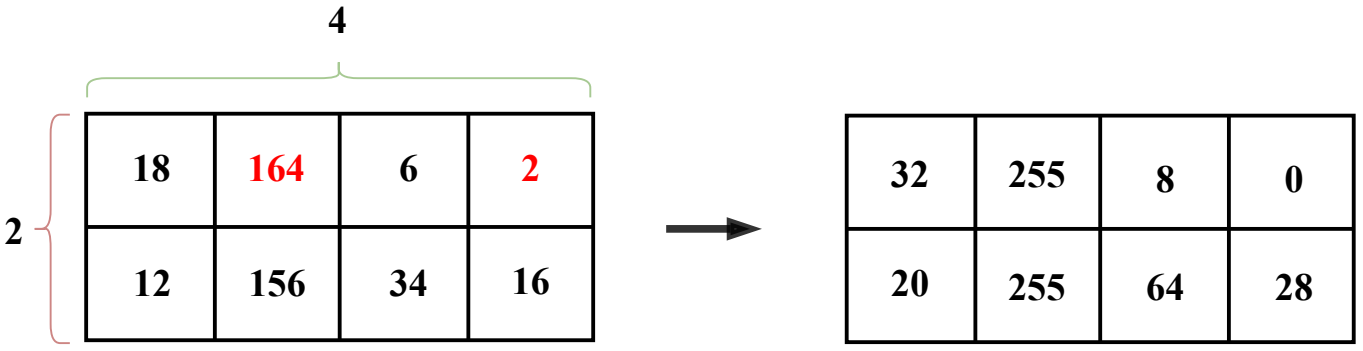
Data un'immagine con risoluzione variabile fino a 128x128 pixel, i quali possono riprodurre una scala di grigi di 256 tonalità, implementare un componente hardware descritto in *VHDL* per ricalibrare i livelli di contrasto. In ingresso verrà fornita una sequenza di pixel di un'immagine non equalizzata provenienti da una memoria, il componente dovrà essere in grado di equalizzare e scrivere in memoria i dati modificati.

1.2 Specifiche Generali

L'algoritmo di analisi dell'immagine procederà macroscopicamente seguendo questi passi:

1. Determino la risoluzione dell'immagine (il numero di pixel).
2. Scorrimento completo dell'immagine al fine di trovare i pixel con la tonalità di grigio più alta(*MAX_VALUE*) e quella più bassa(*MIN_VALUE*).
3. Determino *DELTA_VALUE*
4. *Determino SHIFT_LEVEL*
5. Scorro nuovamente l'immagine determinando il nuovo valore del pixel (compreso tra 0 - 255) e lo scrivo in memoria.

Esempio:



1) Risoluzione: 8px **2)** 164= Max, 2= Min **3)** DELTA_VALUE = 162 **4)** SHIFT_LEVEL = 1 **5)**Nuovi valori pixel

POLITECNICO MILANO 1863

1.3 Interfaccia del Componente

Il componente da sviluppare presenta la seguente interfaccia:

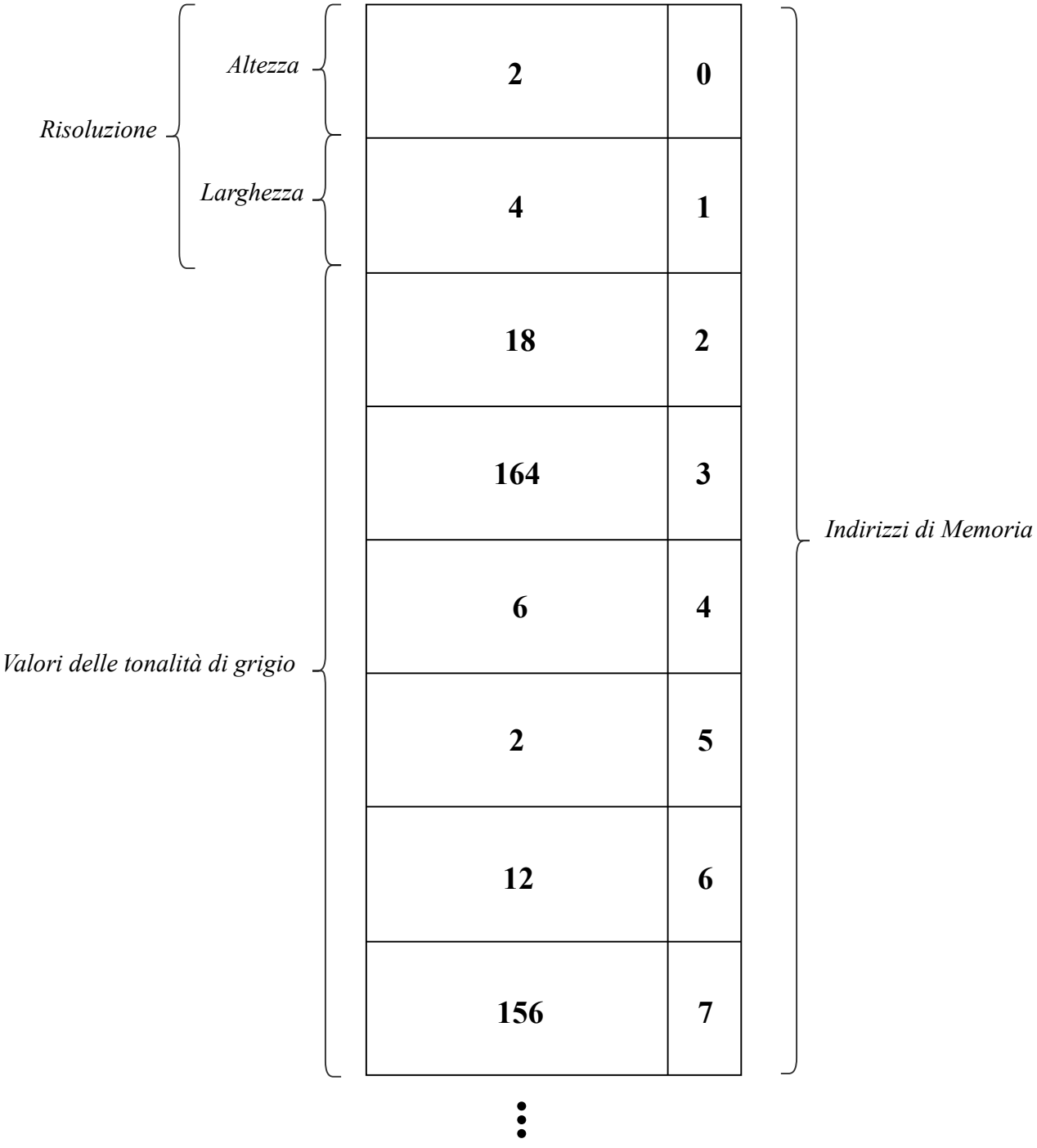
```
entity project_reti_logiche is
   port (
         i_clk : in std_logic;
         i_rst : in std_logic;
         i_start : in std_logic;
         i_data : in std_logic_vector(7 downto 0);
         o_address : out std_logic_vector(15 downto 0);
         o_done : out std_logic;
         o_en : out std_logic;
         o_we : out std_logic;
         o_data : out std_logic_vector (7 downto 0)
                  );

  end project_reti_logiche;
```

In particolare:

- **i_clk**: è il segnale di clock in ingresso che regola la frequenza di aggiornamento del sistema.
- **i_rst**: è il segnale in ingresso che ripristina ogni stato del sistema e lo predispone per acquisire una nuova immagine.
- **i_start**: è il segnale in ingresso che indica l'inizio dell'elaborazione dell'immagine.
- **i_data**: è il segnale-vettore in ingresso che fornisce informazioni sul valore del byte in memoria precedentemente richiesto.
- **o_address**: è il segnale-vettore in uscita che permette di interfacciarsi con la memoria e di richiedere l'indirizzo dei dati che verranno veicolati su i_data.
- **o_done**: è il segnale di uscita che notifica l'avvenuta elaborazione dell'immagine.
- **o_en**: è il segnale di uscita che abilita la possibilità di accedere alla memoria in lettura.
- **o_we**: è il segnale di uscita che notifica la possibilità di accedere alla memoria in scrittura.
- **o_data**: è il valore del dato che si vuole scrivere all'indirizzo di memoria precedentemente comunicato tramite o_address.

1.4 Gestione della Memoria

La stringa contente le informazioni dell'immagine è cosí composta: i primi due byte forniscono altezza e larghezza dell'immagine, i successivi byte rappresentano le tonalità di grigio dei singolo pixel.
Esempio:



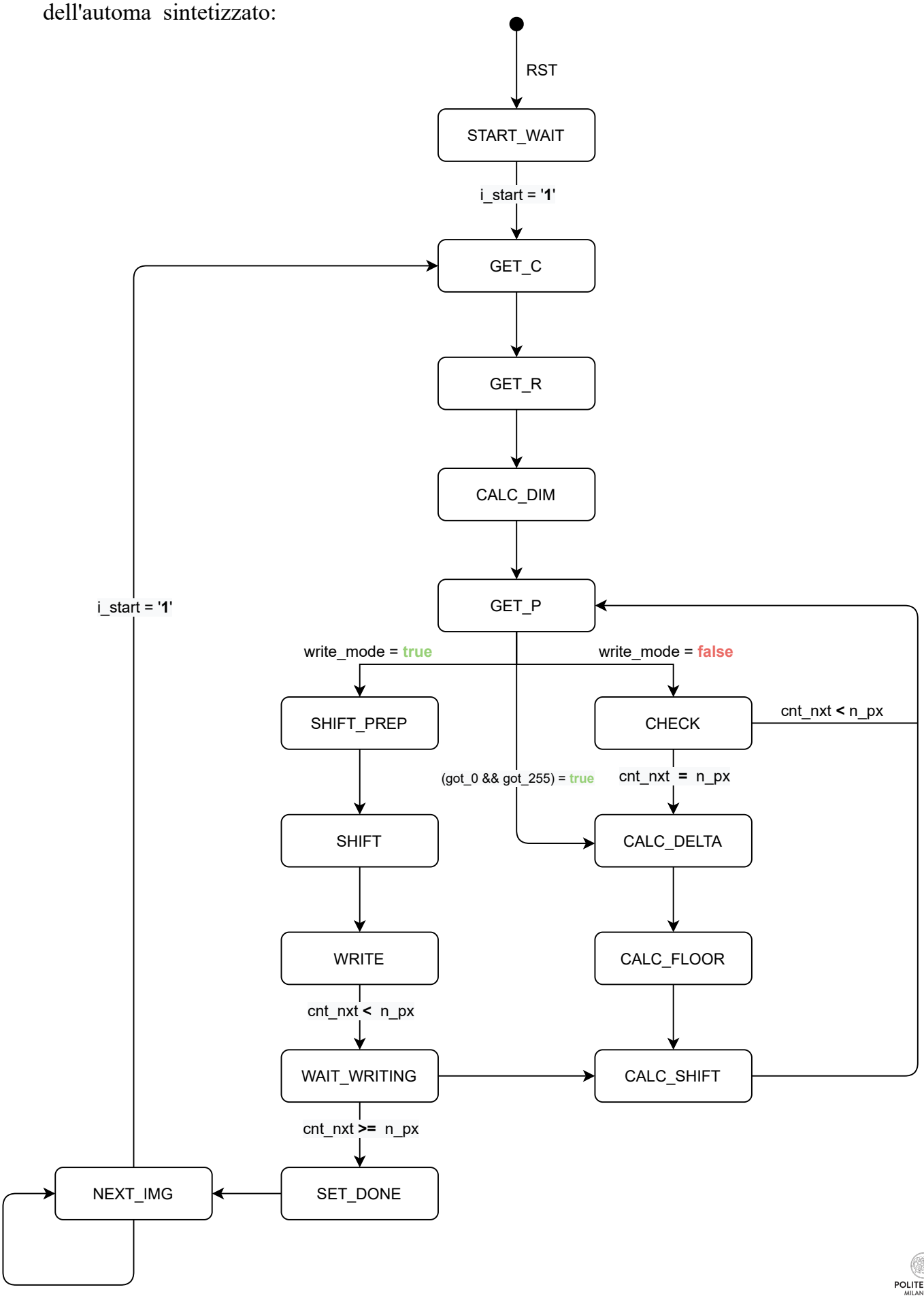
Design

2.1 Stati della Macchina

Di seguito una tabella sommaria con indicati gli stati della macchina:

1	**START_WAIT**	Stato in cui vengono inizializzati tutti i segnali a seguito di *i_start* = 1. Pronto ad agire dopo il segnale di reset
2	**GET_C**	Stato in cui si legge il primo valore della ram, corrispondente al valore dell'altezza dell'immagine
3	**GET_R**	Stato in cui si legge il secondo valore della ram, corrispondente al valore della lunghezza dell'immagine
4	**CALC_DIM**	Stato in cui vengono calcolati il numero di pixel (risoluzione) dell'immagine
5	**GET_P**	Stato in cui viene letto un singolo pixel dalla ram con i pixel di valore massimo e minimo correnti, ed eventualmente aggiorna i rispettivi valori
6	**CHECK**	Stato in cui il pixel appena letto viene confrontato ed eventualmente diventa il pixel_max o min
7	**CALC_DELTA**	Stato in cui viene calcolata la differenza tra max_px e min_px
8	**CALC_FLOOR**	Stato in cui viene calcolato floor con un controllo a soglia per trovarne il valore
9	**CALC_SHIFT**	Stato in cui viene calcolato il valore di shift_level. Aggiorna il contatore e l'indirizzo di uscita
10	**SHIFT_PREP**	Stato in cui viene sottratto px_min al pixel attuale e si prepara l'indirizzo di uscita su cui verrà poi memorizzato il pixel "ricalibrato"
11	**SHIFT**	Stato in cui il pixel viene fatto lo shift del byte che rappresenta il pixel
12	**WRITE**	Stato in cui viene scritto in memoria il nuovo valore del pixel "ricalibrato"
13	**WAIT_WRITING**	Stato necessario per la corretta sincronizzazione dei processi di scrittura
14	**SET_DONE**	Stato che determina la fine del processo ed alza il segnale di done
15	**NEXT_IMG**	Stato in cui la macchina si mette in attesa dell'immagine successiva

E una rappresentazione schematica dell'automa sintetizzato:



2.2 Segnali Utilizzati

Di seguito una tabella sommaria con indicati i segnali utilizzati:

1	state_type	**CURRENT_S**	Stato corrente dellla macchina
2	state_type	**NEXT_S**	Stato successivo della macchina
3	std_logic_vector (15 down to 0)	**ADDRESS_OUT**	Utilizzato per accedere alla memoria in fase di scrittura
4	std_logic_vector (15 down to 0)	**ADDRESS_NEXT**	Utilizzato per accedre alla memoria in fase di lettura
5	std_logic_vector (7 down to 0)	**max_px_value**	Valore del pixel con tonalità più alta tonalità di grigio
6	std_logic_vector (7 down to 0)	**min_px_value**	Valore del pixel con tonalità più bassa tonalità di grigio
7	std_logic_vector (7 down to 0)	**column_value**	Memorizza la lughezza dell'immagine letta da memoria
8	std_logic_vector (7 down to 0)	**row_value**	Memorizza l'altezza dell'immagine letta da memoria
9	std_logic_vector (7 down to 0)	**compare_px**	Segnale-vettore che viene utilizzato per memorizzare il valore del pixel corrente letto dalla memoria. Viene usato anche per determinare il pixel con tonalità massima e minima
10	std_logic_vector (7 down to 0)	**to_be_shifted_px**	Segnale-vettore contenente (compare_px - min_px_value) (secondo l'algoritmo)
11	std_logic_vector (7 down to 0)	**new_px_value**	Segnale-vettore che indica il nuovo valore del pixel che verrà memorizzato in memoria
12	integer range 0 to 255	**delta_value**	Valore di (max_px_value - min_px_value)
13	integer 0 to 8	**shift_level**	Valore di traslazione del pixel
14	integer 0 to 16384	**n_px**	Risoluzione dell'immagine
16	integer 0 to 16384	**cnt**	Contatore generico
17	integer 0 tp 16384	**cnt_nxt**	Contatore generico
18	integer 0 to 8	**floor**	Valore del logaritmo compreso tra 0 e 8
19	boolean	**write_mode**	False durante la lettura per trovare valori massimi e minimi dei pixel. True durante la lettura per lo shift dei pixel
20	boolean	**shift_value**	Valore per ottimizzare il carico computazionale
21	boolen	**got_0**	Segnala se all'interno della ram è stato letto un pixel con tonalità di grigio 0
22	boolean	**got_255**	Segnala se all'interno della ram c'è un pixel con tonalità di grigio 255

2.3 Scelte Progettuali

Data la presenza di segnali di START e RESET è stata scelta un'implementazione tramite macchina a stati finiti (FSM). La FSM descrive l'algoritmo in ogni sua fase:

- Inizializzazione di valori e variabili utilizzati
- Lettura della memoria
- Calcolo dei risultati
- Valutazioni opportune
- Scrittura in memoria
- Riposizionamento in attesa di una nuova immagine

Abbiamo scelto di utilizzare un solo Process sensibile al segnale di clock sul fronte di salita in grado di svolgere le seguenti operazioni:

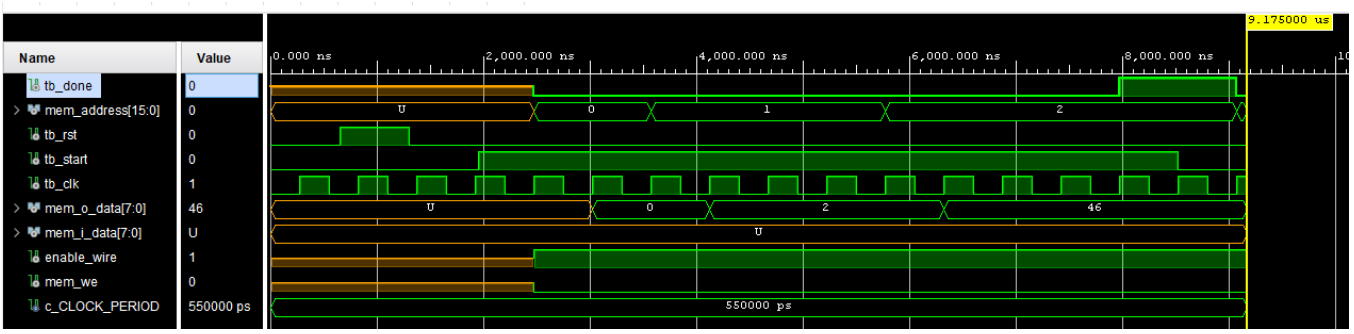
- Determinare gli stati da eseguire, assegnando ad ognuno un NEXT_S
- Controllare che il segnale di reset non sia alto in qualsiasi momento dell'elaborazione
- Fornire le istruzioni da svolgere per ogni stato della macchina

Risultati dei Test

Per verificare la robustezza del componente sintetizzato, abbiamo condotto diversi test con l'obiettivo di verificare la correttezza dell'esecuzione nei casi più comuni e collaudati, ma anche in casi più critici e caratteritici.
In totatale abbiamo individuato 4 casi meritevoli di particolari attenzioni, di seguito ne esponiamo una veloce panoramica:

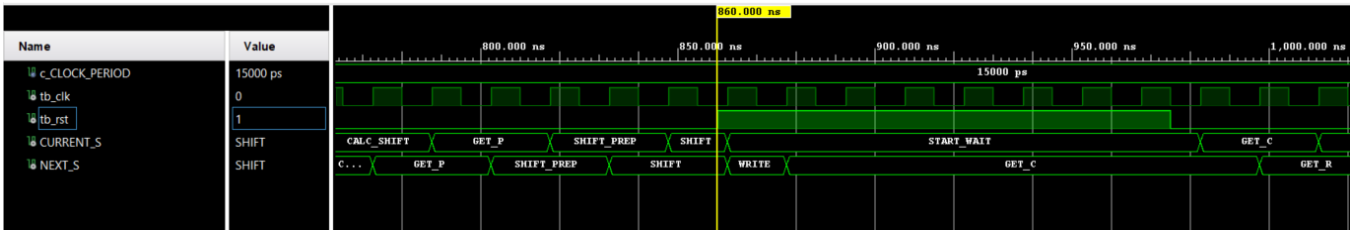
1. Immagine nulla (0x0 - 0xn - nx0)

Abbiamo dedicato accortezze particolari nel caso la memoria passi al componente un'immagine con risoluzione 0, l'immagine viene gestita e il segnale di DONE sale alto senza complicazioni.



2. Reset Asincrono

Un altro caso critico è il reset asincrono, ossia, quando il segnale di reset sale alto durante l'elaborazione di un'immagine. Il componente è sensibile a questa situazione ed è in grado di ripristinare la computazione in attesa di una nuova immagine.



Nella figura viene mostrato il segnale di reset alto nel mezzo di una generica elaborazione e la conseguente mutazione di Stato dell'automa.

3. *DELTA_VALUE = 255*

Nel caso in cui *DELTA_VALUE* raggiunga il valore di *255* il livello di contrasto dell'immagine rimarrà invariato. Questo è un caso particolare che viene si verifica ogni volta che i valori massimi e minimi dell'immagine sono rispettivamente 255 e 0. Il componente è grado di gestire questa particolarità.

4. *O*verflow di *TEMP_PIXEL*

Particolare attenzione è stata posta durante la fase di shift del byte. Nei casi in cui il nuovo valore del pixel avrebbe superato il valore di 255, viene applicata una soglia limite di 255.

Conclusioni

4.1 Risultati della Sintesi

Il componente sviluppato supera correttamente le simulazioni richieste: Behavioral, Post-Synthesis Functional.
I tempi di elaborazione di una generica immagine dipendono dal rispettivo numero di pixel, di seguito proponiamo i tempi di elaborazione di un immagini composte rispettivamente da 0 e 16384 pixel:

- Immagine con ***0*** pixel: *9175* ns *(Behavioral)* || *9175,1* ns *(Post-Synthesis)*
- immagine con ***16384*** pixel: *2974442,5* ns *(Behavioral)* || *2974442,6* ns *(Post-Synthesis)*

Proponiamo inoltre il numero di LUT e FF utilizzati:

- Numero di ***LUT*** in Post-Sythesis Functional: *256*
- Numero di ***Flip-Flop*** in Post-Sythesis Functional: *188*

4.2 Ottimizzazioni

Il progetto è stato condotto con riguardo nei confronti della flessibilità e del essenzialità. In un primo momento ci siamo concentrati sul riduerre il numero di stati e segnali, rimuovendo stati di attesa per la RAM. Sarebbe stata possibile un'ulteriore ottimizzazione accorpando vari stati, ma abbiamo scelto di mantenre l'attuale configurazione per avere maggiore suddivisione logica e funzionale della macchina.
In un secondo luogo abbiamo pensato ad ottimizzazioni mirate. La fase più onerosa dell'elaborazione, in seguito all'algoritmo scelto, è la lettura sequenziale dell'immagine. Perciò, nel caso in cui l'immagine presenti tonalità di grigio sia di valore 0 che 255, il componete smetterà di elaborare i pixel dell'immagine (in fase di lettura) ed assegnerà il valore di DELTA=255, saltando direttamente a scrivere in RAM i valori immutati, risparmiando tempo e risorse computazionali.