Progetto di Reti Logiche

Prof. William Fornaciari - A.A. 2020/2021

Dario Del Gaizo - CP: 10633295

Davide Davanzo - CP: 10660974

Indice

1 Introduzione

1.1 Specifiche di progetto

Si prendano in considerazione immagini in *scala di grigi* definite come prodotto $colonne \cdot righe$ di pixel, con valore compreso tra 0 e 255. Obiettivo del progetto è la creazione di un componente hardware descritto in VHDL, il quale fornisca, presa un'immagine in ingresso da memoria, una nuova versione *equalizzata* secondo una versione semplificata dell'algoritmo standard.

Le azioni da svolgere sulla singola immagine per ottenere il risultato voluto sono le seguenti:

- Trovare i valori min (minimo) e max (massimo) tra i pixel dell'immagine;
- Calcolare $delta\_value = max - min$;
- Definire lo $shift\_level = 8 - \lfloor \log_2(delta\_value + 1) \rfloor$;
- Calcolare un pixel temporaneo $temp_{pixel} = current_{pixel} - \min << shift\_level$;
- Infine, scrivere il nuovo pixel come $\min(255, temp_{pixel})$.

1.2 Interfaccia del componente

Il componente da descrivere ha 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;
```

- 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 TestBench;

- 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.

1.3 Rappresentazione dei dati in memoria

Le dimensioni di colonne e righe, così come i valori dei singoli pixel, sono rappresentati con 8 bit in una memoria indirizzata al byte. Quindi, ogni cella di memoria rappresenta un singolo dato.

Il primo indirizzo (0) contiene il numero di colonne dell'immagine, il secondo (1) il numero di righe. Dall'indirizzo 2 iniziano i valori dei pixel.

Ecco la rappresentazione in memoria di un'immagine 2 x 2 seguita dalla versione equalizzata:

[0]	2	Numero di colonne
[1]	2	Numero di righe
...	4	1° pixel
	8	...
	77	
	254	
	0	1° pixel equalizzato
	8	...
	146	
	255	

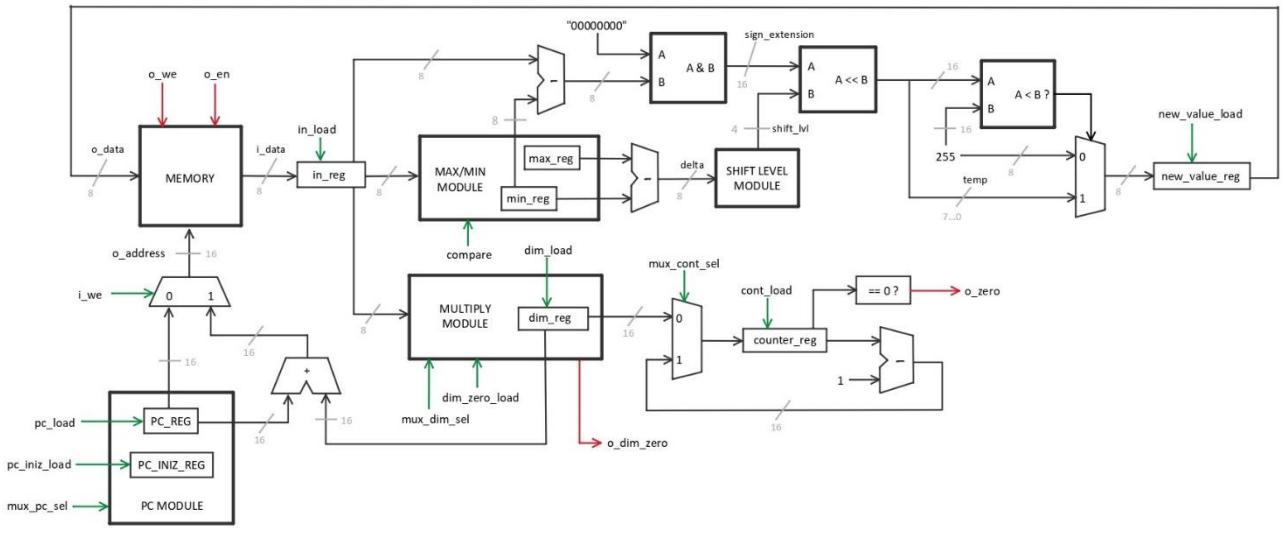
2 Datapath

2.1 Descrizione

Il Datapath rappresenta lo scheletro del progetto, il percorso che i dati compiono prima di ritornare alla memoria. È composto da vari moduli contenenti componenti logici e registri orchestrati da una macchina a stati finiti (FSM), che verrà trattata in seguito.

Questi moduli sono:

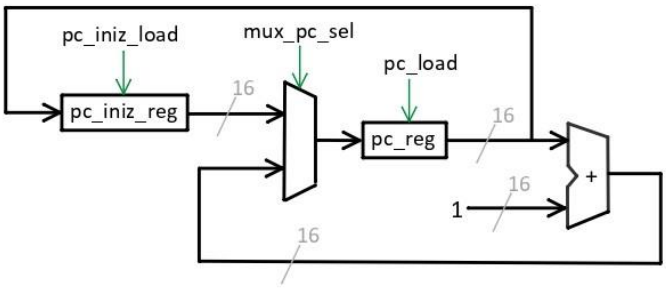
- *PC_module*, di gestione degli indirizzi;
- *multiply_module*, adibito al calcolo della dimensione;
- *max_min_module*, per individuare il valore massimo e minimo;
- *shift_level_module*, per il calcolo dello shift_level;
- varie componenti aggiuntive, come contatori per capire quando terminare la scansione dei pixel e multiplexer per alternare indirizzo di lettura e scrittura.



2.2 PC_module

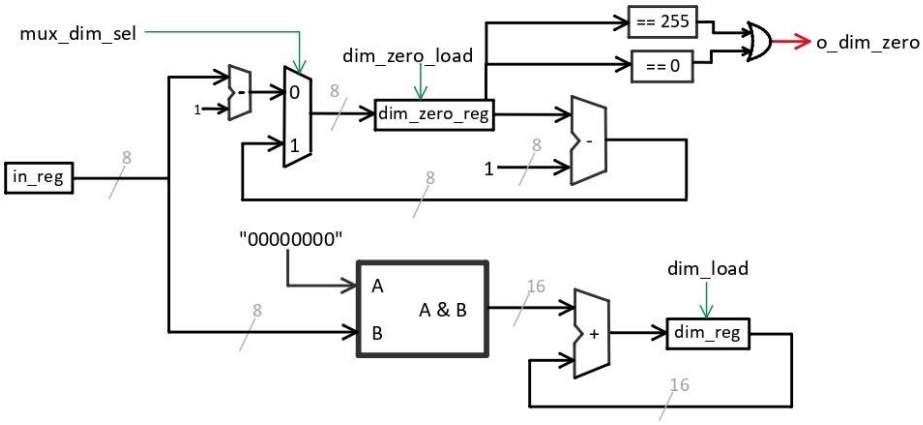
Il modulo è dedicato alla gestione degli indirizzi per la lettura e scrittura in memoria. All'inizio della computazione *pc_reg* e *pc_iniz_reg* vengono entrambi inizializzati a 0. Le dimensioni vengono lette e il *pc_reg* incrementato, finché arrivati alla fine dell'immagine con massimo e minimo calcolati, esso viene rimesso al valore di

pc_iniz_reg, dove era stato salvato dopo il calcolo della dimensione l'indirizzo del primo pixel. Per quanto concerne la corretta scrittura in memoria, un segnale *i_we* seleziona attraverso un mux la somma di *pc_reg* e della dimensione dell'immagine come indirizzo.



2.3 Multiply_module

Questo modulo è dedicato al calcolo della dimensione dell'immagine attraverso somme successive: il prodotto $n \cdot m$ viene calcolato come $\sum_{i=1}^{m} n$.



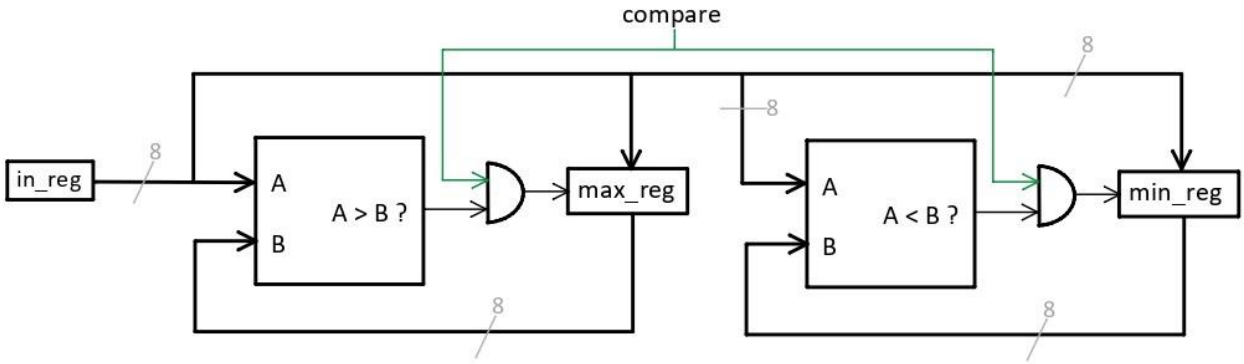
dim_zero_reg contiene la prima delle dimensioni, dopo una sottrazione iniziale, essa viene decrementata di uno tante volte quante la seconda dimensione viene sommata con sé stessa.

Nella parte superiore dell'immagine sono presenti due confronti uniti da un OR: il segnale *o_dim_zero* viene posto a '1' quando la prima dimensione arriva a zero, terminando il calcolo. Il caso di confronto con 255 è stato invece inserito per velocizzare la gestione di casi anomali come "dimensione = 0", che mandando il

registro in overflow causa l'attesa di un gran numero di cicli prima di concludere. In ogni caso la dimensione di pixel massima da considerare valida è di 128 x 128.

2.3 Max_min_module

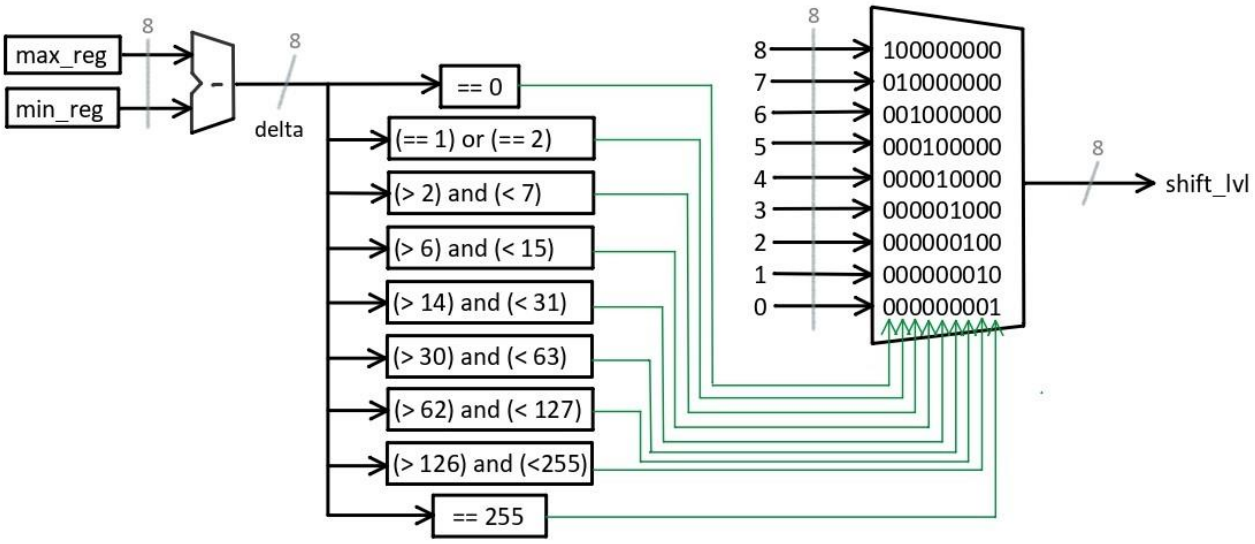
Il seguente modulo è dedicato al parsing dei pixel per trovare i valore minimo e il massimo. Il primo byte letto dopo le dimensioni assume entrambi i valori; successivamente ogni pixel viene confrontato con minimo e massimo e i registri aggiornati.



Nel caso del max, il segnale di load è un "and" logico tra il segnale *compare,* posto a '1' dalla macchina a stati e il controllo "maggiore del valore presente?". Terminato il confronto, il PC viene riportato all'indirizzo di memoria del primo pixel per iniziale l'equalizzazione.

2.4 Shift_level_module

Come sappiamo lo *shift_level* viene calcolato attraverso la formula $8 - \lfloor \log_2(delta\_value + 1) \rfloor$; questo consente di agire direttamente su *delta_value* incrementato di 1. Per il calcolo del logaritmo viene utilizzato un mux (con rappresentazione compatta) con valori da 0 a 8 in ingresso: in base alla soglia del valore in ingresso *shift_level* assume un determinato valore.

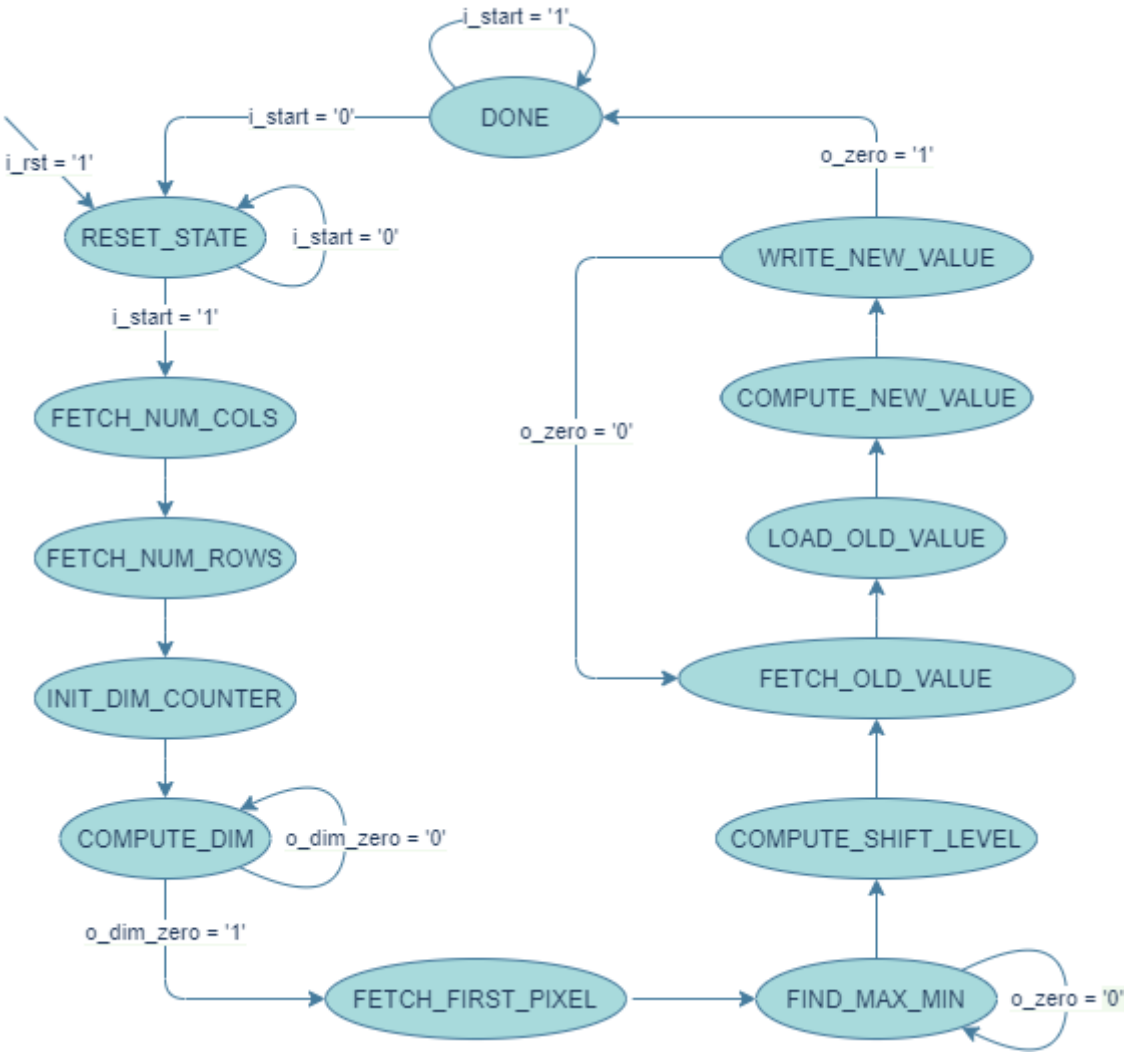


Infine, *new_value* viene salvato dopo il confronto del pixel equalizzato con 255. I segnali di *write_enable* si occuperanno di scegliere l'indirizzo corretto e scrivere il contenuto del registro in memoria.

3 FSM

3.1 Panoramica

Il Datapath rappresenta solo il percorso dei dati, ma non dà alcuna informazione sull'evoluzione temporale del circuito. Esso è infatti orchestrato da una macchina a stati finiti che si occupa della gestione dei segnali in ingresso, in uscita e dei segnali di *load* dei registri.



Questa è la rappresentazione grafica della FSM: i rami di uscita sono specificati nell'immagine solo se necessario.

Successivamente è riportata una tabella con una breve descrizione di ogni stato e la lista dei segnali che assumono un valore d'interesse per esso.

3.1 Stati

Reset	Stato in attesa del segnale di start, viene raggiunto con un reset	/
Fetch_num_cols	Lettura delle colonne dalla memoria e incremento il PC	o_en <= '1' pc_load <= '1'
Fetch_num_rows	Caricamento delle colonne, lettura delle righe e incremento del PC	o_en <= '1' in_load <= '1' pc_load <= '1'
Init_dim_counter	Caricamento righe, salvataggio di PC in PC_iniziale e caricamento delle colonne in dim_zero	in_load <= '1' pc_iniz_load <= '1' dim_zero_load <= '1'
Compute_dim	Stato di calcolo della dimensione, caricamento delle righe che vengono sommate per il numero di colonne	o_en <= '1' mux_dim_sel <= '1' dim_load <= '1' dim_zero_load <= '1'
Fetch_first_pixel	Lettura del primo pixel e caricamento del contatore con la dimensione	cont_load <= '1' o_en <= '1' in_load <= '1' pc_load <= '1'
Find_max_min	Ciclo per la ricerca del massimo e minimo. Scorrimento pixel, attivazione della comparazione, decremento del contatore	compare <= '1' o_en <= '1' in_load <= '1' pc_load <= '1' mux_cont_sel <= '1' cont_load <= '1'
Compute_shift_level	Calcolo del delta e caricamento PC con il valore PC_iniziale	mux_pc_sel <= '1' pc_load <= '1' cont_load <= '1'
Fetch_old_value	Lettura dal primo pixel e contatore posto al valore della dimensione	o_en <= '1' mux_cont_sel <= '1' cont_load <= '1'
Load_old_value	Caricamento del valore del pixel	in_load <= '1'

`Compute_new_value`	Caricamento new value	`new_value_load <= '1'`

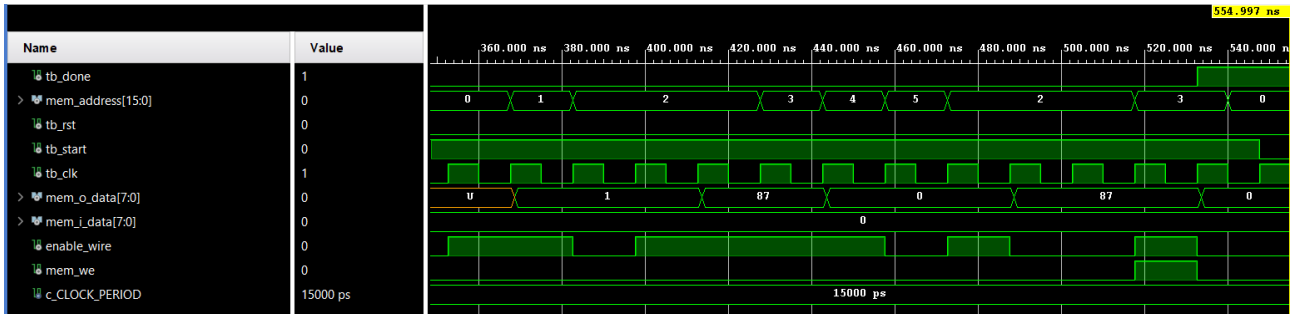
`Write_new_value`	Scrittura in memoria e incremento di PC. Si torna a Fetch_old_value se non si è all'ultimo pixel	`o_en <= '1'`
		`o_we <= '1'`
		`i_we <= '1'`
		`pc_load <= '1'`

`Done`	Stato di fine computazione	`o_done <= '1'` `i_done <= '1'`

4 Testing

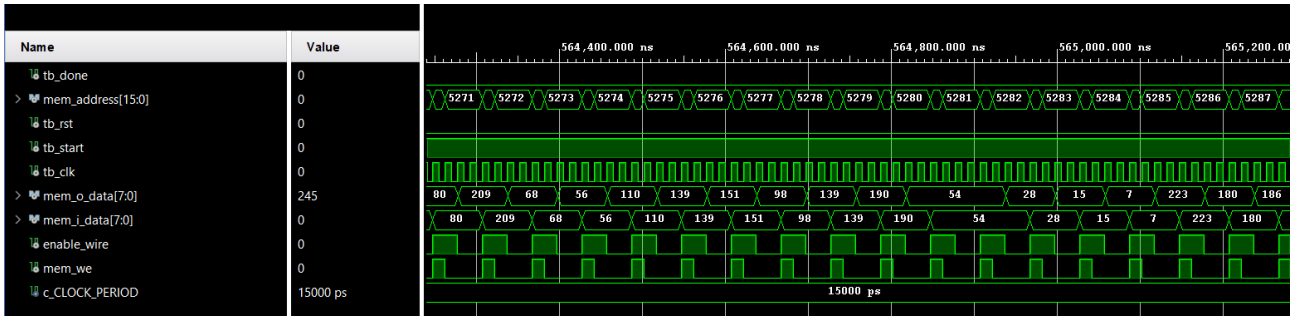
Il *testing* è stato effettuato esaustivamente per valutare il corretto funzionamento del circuito nella maggior parte dei casi. Questi ultimi derivano da una lettura critica delle specifiche di progetto, pertanto i test considerati possono avere forti correlazioni con i casi limite definiti tratti dalle richieste.
Ognuno dei seguenti test è stato simulato sia in Behavioural che in Post-Synthesis.

Un pixel: test effettuato per verificare il funzionamento con la più piccola dimensione possibile (diversa da 0).



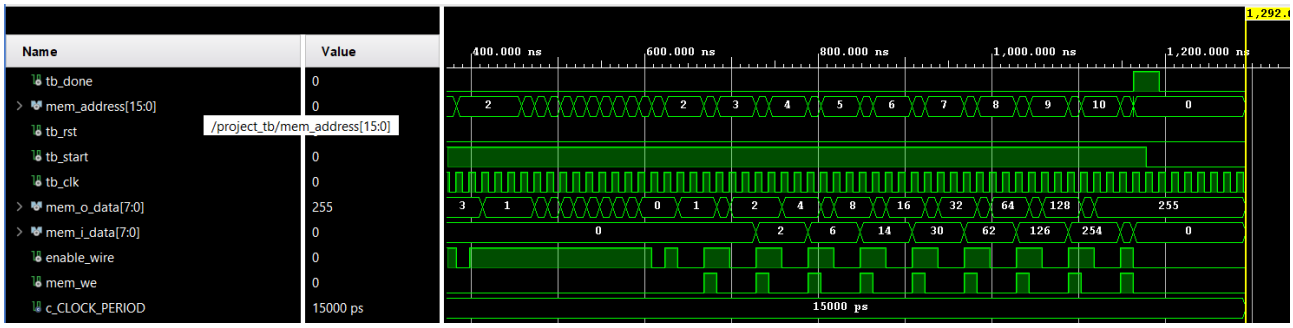
Simulazione Behavioural di Un pixel

Dimensione massima: test per verificare la correttezza del calcolo di immagini di una certa dimensione, in questo caso la massima prevista per questo circuito, 128 x 128.



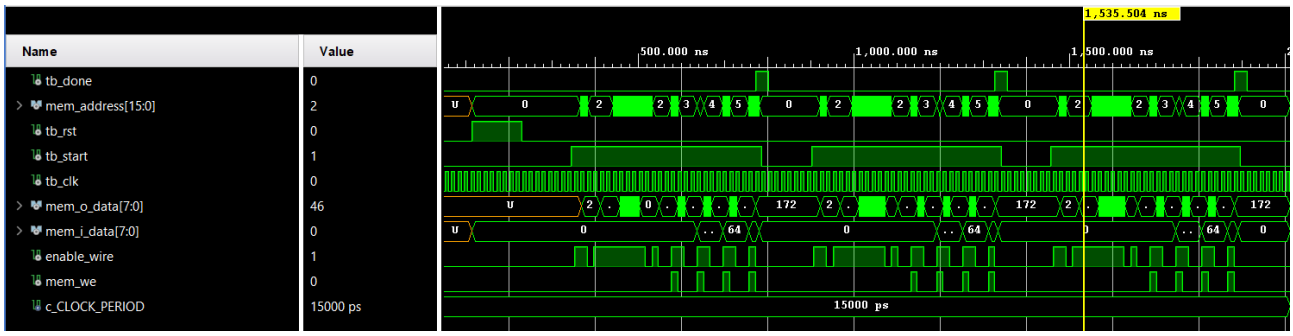
Simulazione Post-Synthesis di Dimensione massima

Incremento 2^n: questo test consiste nell'equalizzazione di un'immagine 9 x 9 contenente tutte le potenze di 2, per stressare al massimo il calcolo di *shift_value.*



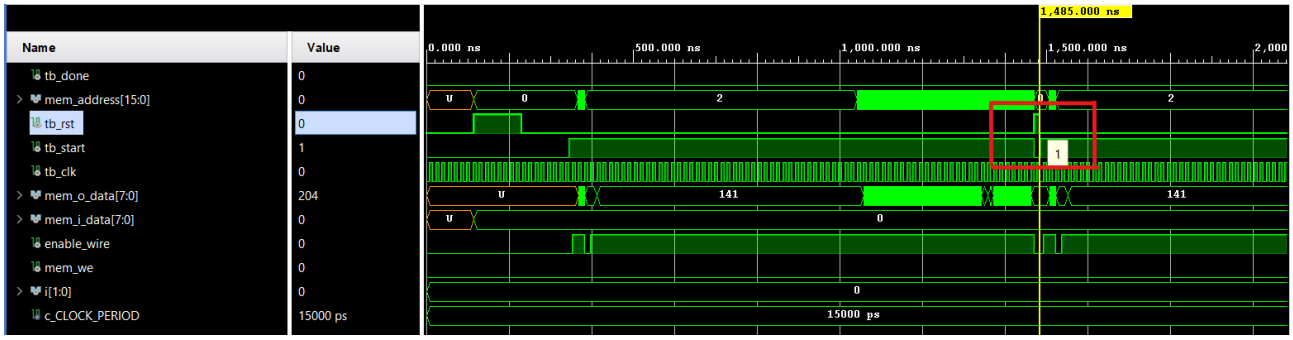
Simulazione Post-Synthesis di Incremento 2n

Tre immagini: il seguente test verifica la correttezza nell'equalizzazione di più immagini consecutive. Da notare come il segnale di *start* sia dato tre volte, seguito da tre segnali di *done.*



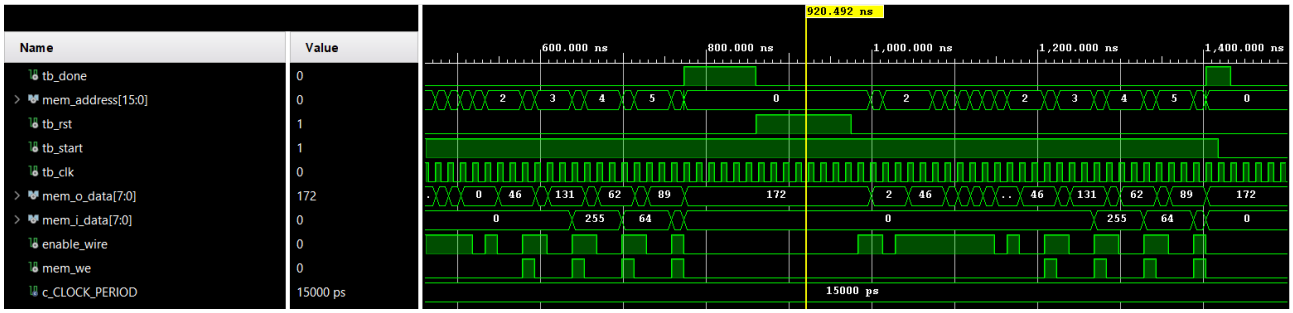
Simulazione Behavioural di Tre immagini

Tre consecutive e reset alla prima: test in linea di massima simile al precedente con tre immagini, stavolta però c'è un segnale di *reset* lanciato durante l'equalizzazione della prima immagine.



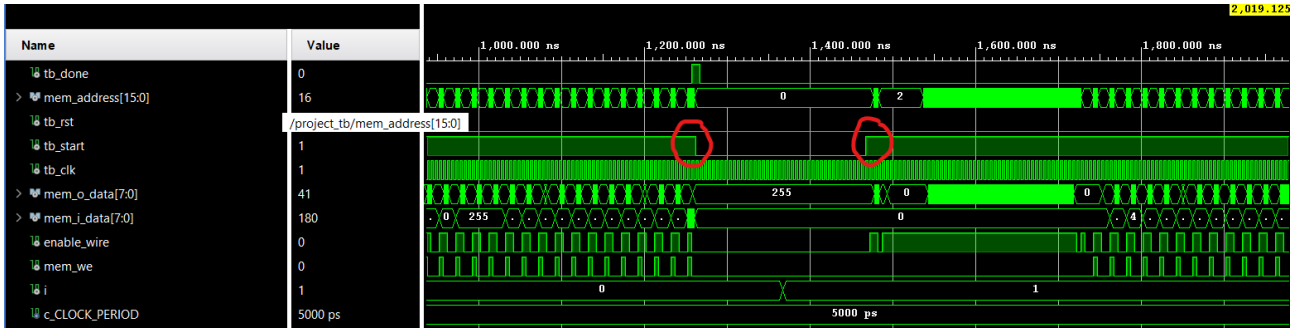
Simulazione Behavioural di Tre immagini con reset alla prima

Reset asincrono: verifica di risposta ad un reset asincrono.



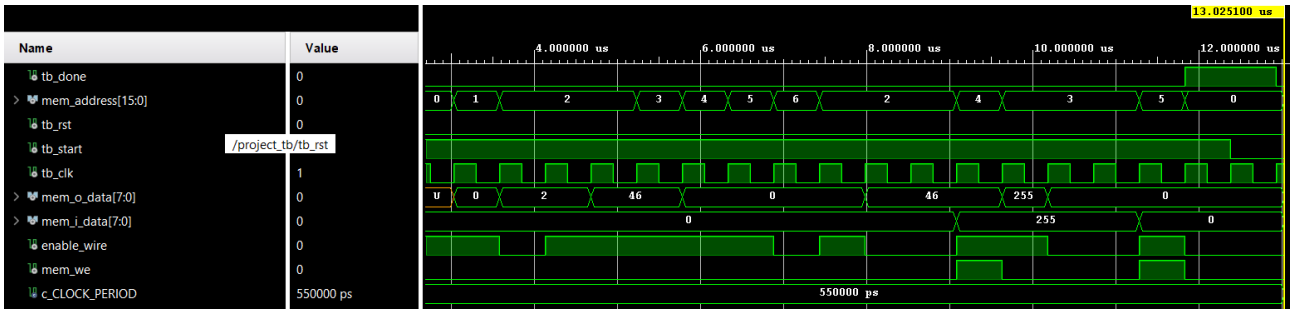
Simulazione Post-Synthesis di Reset asincrono

Multi start: test che stressa il funzionamento con più segnali di start lanciati.



Simulazione Behavioural di Multi start

Zero pixel: questo speciale test è stato effettuato per verificare il corretto funzionamento nel caso di un input errato, come può essere una dimensione uguale a zero. L'obiettivo del test consiste nel non vedere cambiamenti inaspettati nella memoria da cui si leggono i dati, garantendo ulteriore solidità al circuito.



Simulazione Post-Synthesis di Zero pixel

Ulteriori test: oltre ai più significativi qui riportati, sono stati effettuati altri test di *dimensione massima dei pixel*, nonché *1 x 1*, *reset*, *più immagini consecutive* e test che stressano le *equazioni di equalizzazione* (massimo e minimo alla fine, tutti i pixel a 1, tutti i pixel a 255..), per garantire la limitazione di comportamenti anomali. Tutti i test effettuati sono nella cartella di *simulazione* del progetto.

5 Conclusioni

5.1 Dati utili

Di seguito sono riportate informazioni utili come il *Design Timing Summary,* il *Clock Summary* e infine il numero di flip-flop e latch.

Si vuole ribadire che l'obiettivo del progetto era in ogni caso la creazione di un componente funzionante date specifiche e *constraints,* affidando gran parte del compito di ottimizzazione al tool di sintesi.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	94,636 ns	Worst Hold Slack (WHS):	0,148 ns	Worst Pulse Width Slack (WPWS):	49,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	342	Total Number of Endpoints:	342	Total Number of Endpoints:	122

All user specified timing constraints are met.

Design Timing Summary

Da specifica, il circuito deve funzionare con un clock di almeno 100 ns con duty-cycle al 50%.

Name	Waveform	Period (ns)	Frequency (MHz)
clock	{0.000 50.000}	100.000	10.000

Clock Summary

Non ci sono registri Latch e nessun latch-loop warning viene sollevato dal sistema.

Slice Registers	121	0	269200	0.04
Register as Flip Flop	121	0	269200	0.04
Register as Latch	0	0	269200	0.00

Registri: flip flop e latch

5.2 Ottimizzazioni

Le ottimizzazioni manuali sono state per la maggior parte svolte nella **riduzione del numero di stati** della FSM e nel dedicare un *process* ad ogni registro nel Datapath; così facendo gli stati "intermediari" sono stati quasi del tutto eliminati, ottenendo un alleggerimento del numero di flip-flop e delle tempistiche.