POLITECNICO
MILANO 1863

Prova Finale di Reti Logiche

Equalizzatore dell'istogramma di una immagine

Giacomo Pizzamiglio (10620604)
Andrea Prisciantelli (10618568)

Ingegneria Informatica
Prof. Gianluca Palermo

AA 2020/21

Indice

1 Introduzione

1.1 Obiettivo

L'obiettivo del modulo sviluppato è la realizzazione di una versione semplificata del metodo di equalizzazione dell'istogramma di un'immagine. Tale modulo deve modificare le intensità dei pixel quando i valori sono vicini tra loro.

Il risultato quindi è quello di ricevere immagini con poco contrasto e di restituirle col contrasto equalizzato.

1.2 Specifiche generali

Algoritmo La versione semplificata dell'algoritmo di equalizzazione è applicata ad immagini di grandezza massima 128x128 pixel in scala di grigi a 256 livelli (8 bit per ogni pixel). Le variabili principali considerate sono le seguenti:

- $delta\_value = max\_pixel\_value - min\_pixel\_value$
- $shift\_level = (8 - floor(log_2(delta\_value + 1)$
- $temp\_pixel = (current\_pixel\_value - min\_pixel\_value) << shift\_level$
- $new\_pixel\_value = min(255, temp\_pixel)$

Da notare è il fatto che $shift\_level$ può assumere solo valori da 0 a 8 al decrescere del valore di $delta\_value$ da 255 a 0. Il fatto di applicare poi un shift a sinistra a $current\_pixel\_value - min\_pixel\_value$ di $shift\_level$ bit equivale quindi a moltiplicare per 2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8 in base al valore di $delta\_value$. Si può quindi utilizzare un controllo a soglia per determinare sia $shift\_level$ che il moltiplicatore.

Memoria Il modulo deve leggere l'immagine di partenza da una memoria sequenziale con indirizzamento al Byte.

I primi due Byte sono destinati alle dimensioni dell'immagine, rispettivamente alla dimensione di colonna n_col ed alla dimensione di riga n_rig. I seguenti Byte contigui invece sono destinati ciascuno ad un unico pixel.

Dopo aver eseguito l'equalizzazione dell'istogramma dell'immagine, i nuovi valori dei pixel devono essere scritti in memoria a partire dalla posizione $2 + (n\_col \times n\_rig)$, ovvero la prima cella libera dopo l'ultimo Byte dell'immagine originale. Data un'immagine che termina al Byte n, la sua copia equalizzata deve essere scritta nei Byte contigui a partire dalla posizione $n + 1$.

Byte_0	n_col
Byte_1	n_rig
Byte_2	$pixel_0$
Byte_3	$pixel_1$
...	...
Byte_n	$pixel_n$
Byte_{n+1}	new_pixel_0
Byte_{n+2}	new_pixel_1
...	...

Tabella 1: Organizzazione della memoria.

INDIRIZZO	0	1	2	3	4	5	6	7	8	9
VALORE	2	2	21	155	78	135	0	255	114	228

Figura 1: Esempio di equalizzazione di un immagine 2x2 pixel.

1.2.1 Interfaccia del modulo

Il componente da realizzare 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);
    );
```

I segnali in ingresso sono:

- i_clk, il segnale di clock;
- i_rst, il segnale di reset;

- i_start, che comanda l'inizio del processo di equalizzazione dei pixel di un'immagine;
- i_data, contenente il valore (in formato vettoriale a 8 bit) letto della signola cella di memoria.

I segnali di uscita sono:

- o_address, contenente l'indirizzo della cella di memoria da cui leggere od in cui scrivere un Byte (in formato vettoriale a 16 bit);
- o_done, che segnala la fine del processo di equalizzazione di un'immagine;
- o_en, per abilitare l'accesso alla memoria;
- o_we, per abilitare la scrittura in memoria;
- o _data, contenente il Byte (in formato vettoriale a 8 bit) inviato alla memoria.

Inoltre, il segnale o_done deve rimanere a 1 dopo la fine del processo di equalizzazione fino a che il segnale i_start non è riportato a 0.

2 Architettura

2.1 Algoritmo

L'algoritmo sviluppato per realizzare l'equalizzazione (mostrato in Figura 2) è organizzato in 3 fasi:

1. **SETUP**. Vengono lette dai primi due Byte della memoria le dimensioni dell'immagine, rispettivamente l'altezza n_col e la larghezza n_rig. Da queste è poi calcolato il numero totale di pixel $size = n\_col \times n\_rig$, che verrà utilizzato per contare i Byte dei pixel letti dalla memoria sequenziale.

2. **FASE 1**. I Byte dei pixel originali dell'immagine sono letti in ordine dal primo all'ultimo. Durante questo ciclo si stabilisce il minimo ed il massimo valore dei pixel, rispettivamente min_pixel_value e max_pixel_value. Infine si calcolano $delta\_value$ e $shift\_level$.

3. **FASE 2**. I Byte dei pixel originali sono nuovamente letti in ordine dal primo all'ultimo. Ad ogni lettura, dato il valore del Byte considerato $current\_pixel\_value$, si calcola $temp\_pixel\_value$. Infine, si trova il minimo tra il valore appena calcolato e 255; questo valore è inviato in scrittura alla prima cella libera di memoria dopo i Byte dell'immagine originale.

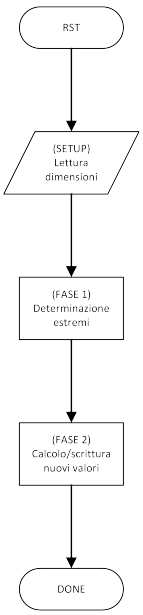
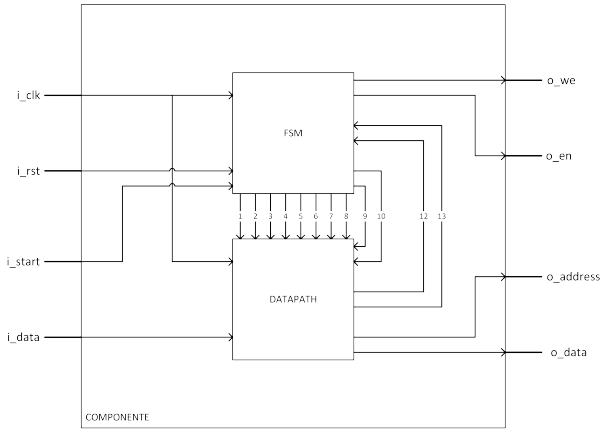


Figura 2: Diagramma di flusso dell'algoritmo.

2.2 Moduli

Il componente implementa due moduli, mostrati in figura 3: il *datapath*, responsabile della manipolazione dei dati, la macchina a stati, responsabile del coordinamento delle operazioni.



SEGNALI INTERNI	
1	reg_rst
2	phase_change
2	n_col_load
4	size_load
5	counter_rst
6	counter_en
7	address_sel
8	max_value_load
9	min_value_load
0	shift_mult_load
11	counter_end
12	phase_flag

Figura 3: Rappresentazioni in moduli del componente.

2.2.1 Il datapath

Lo scopo principale del datapath è il calcolo dei valori equalizzati dei pixel. Si occupa anche di effettuare le operazioni di calcolo e conteggio utili al controllo della memoria o per il ciclo operativo della macchina a stati. Nello specifico le operazioni svolte sono le seguenti:

- calcolo del numero di pixel dell'immagine.
- conteggio del numero di pixel letti,
- calcolo dell'indirizzo di memoria da cui leggere od in cui scrivere,
- determinazione del massimo e del minimo valore dei pixel,
- calcolo dello $shif\_level$ e del moltiplicatore ad esso corrispondente,

- calcolo dei nuovi valori dei pixel

Il modulo datapath riceve in ingresso *i_data* ed alcuni segnali di controllo per i suoi componenti e restituisce in uscita *o_data*, *o_address* ed alcuni segnali per monitorare il suo comportamento. Tra i segnali di controllo vi sono quelli di carico (*load*) e di reset dei registri e queli di selezione dei *multiplexer*; tra quelli in uscita vi è il segnale di fine (*end*) del conteggio dei pixel.

Il circuito è stato diviso in 5 parti principali, realizzate autonomamente, e poi unite:

- **circuito di setup**: riceve *n_col* ed *n_rig* e calcola il numero complessivo di pixel dell'immagine *size*.
- **circuito contatore dei pixel**: ad ogni ciclo di clock, incrementa di uno il valore contenuto in un registro per tenere conto delle operazioni effettuate sui pixel in memoria.
- **circuito calcolatore dell'indirizzo**: restituisce il segnale *o_address* da inviare alla memoria sulla base di un selettore e del valore di *size* e del contenuto del registro interno al contatore.
- **circuito di confronto degli estremi**: a sua volta ripartito in 3 sotto-circuiti. Di questi, due per trovare gli estremi, uno per *min_pixel_value* e uno per *max_pixel_value*. Il terzo serve per calcolare il moltiplicatore utilizzato nel calcolo dei *new_pixel_value* (si veda la sezione "Ottimizzazione" per ulteriori informazioni su questa parte).
- **circuito di calcolo dei nuovi valori**: riceve i valori dei pixel originali, ne calcola i nuovi e li invia in scrittura alla memoria tramite *o_data*.

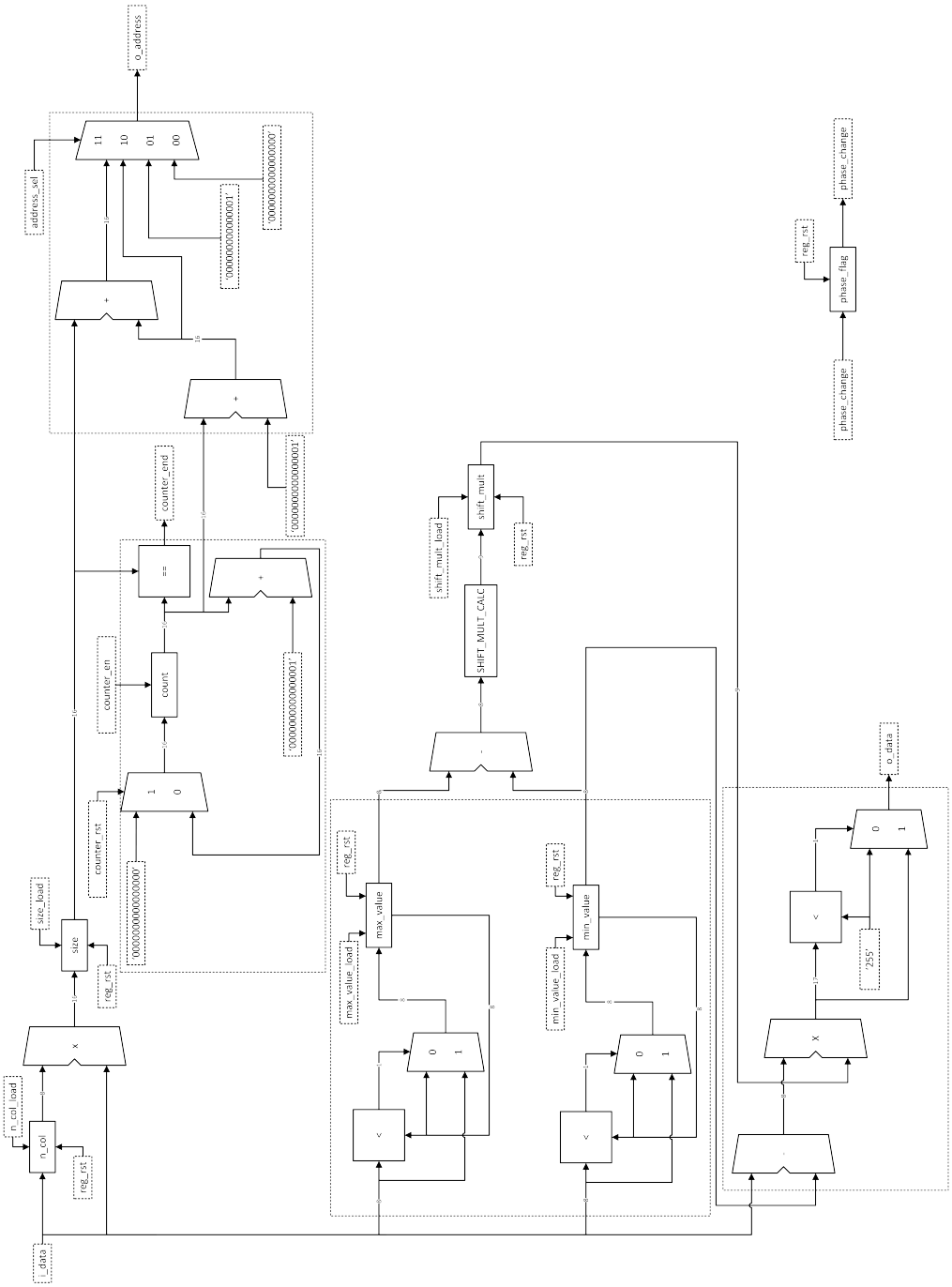


Figura 4: Diagramma del datapath.

2.2.2 La macchina a stati

La macchina a stati si interpone tra il datapath e la memoria per coordinare le operazioni.

In Figura 5 è mostrato lo schema della macchina a stati.

RST La macchina è in *idle* fintantoché il segnale di *start* non è posto a 1. Inoltre viene effettuato il *reset* dei registri.

REQ_N_COL La macchina imposta il datapath per richiedere alla memoria il numero di colonne dell'immagine.

READ_N_COL La macchina imposta il datapath per leggere e memorizzare il numero di colonne dell'immagine.

REQ_N_RIG La macchina imposta il datapath per richiedere alla memoria il numero di righe dell'immagine.

READ_N_RIG La macchina imposta il datapath per calcolare il numero di pixel totali dell'immagine: legge il numero di righe dell'immagine, lo moltiplica per il numero di colonne e memorizza il risultato.

COMP_EXT La macchina imposta il datapath per leggere il valore del pixel corrente dalla memoria e confrontarlo con il massimo e minimo valore tra quelli precedentemente letti.

CALC_SHIFT La macchina imposta il circuito per calcolare il moltiplicatore per effettuare lo shift a sinistra.

WRITE_NEW La macchina imposta il datapath per leggere il valore del pixel corrente, calcolare e scrivere in memoria il valore del nuovo pixel dell'immagine equalizzata.

DONE La macchina porta a 1 il segnale di *done* e resta in attesa che quello di start sia riportato a 0.

INIT_LOOP La macchina inizializza il circuito contatore per prepararsi alla lettura dei pixel dell'immagine.

REQ_PIXEL La macchina imposta il datapath per calcolare l'indirizzo del pixel corrente e richiedere il suo valore dalla memoria.

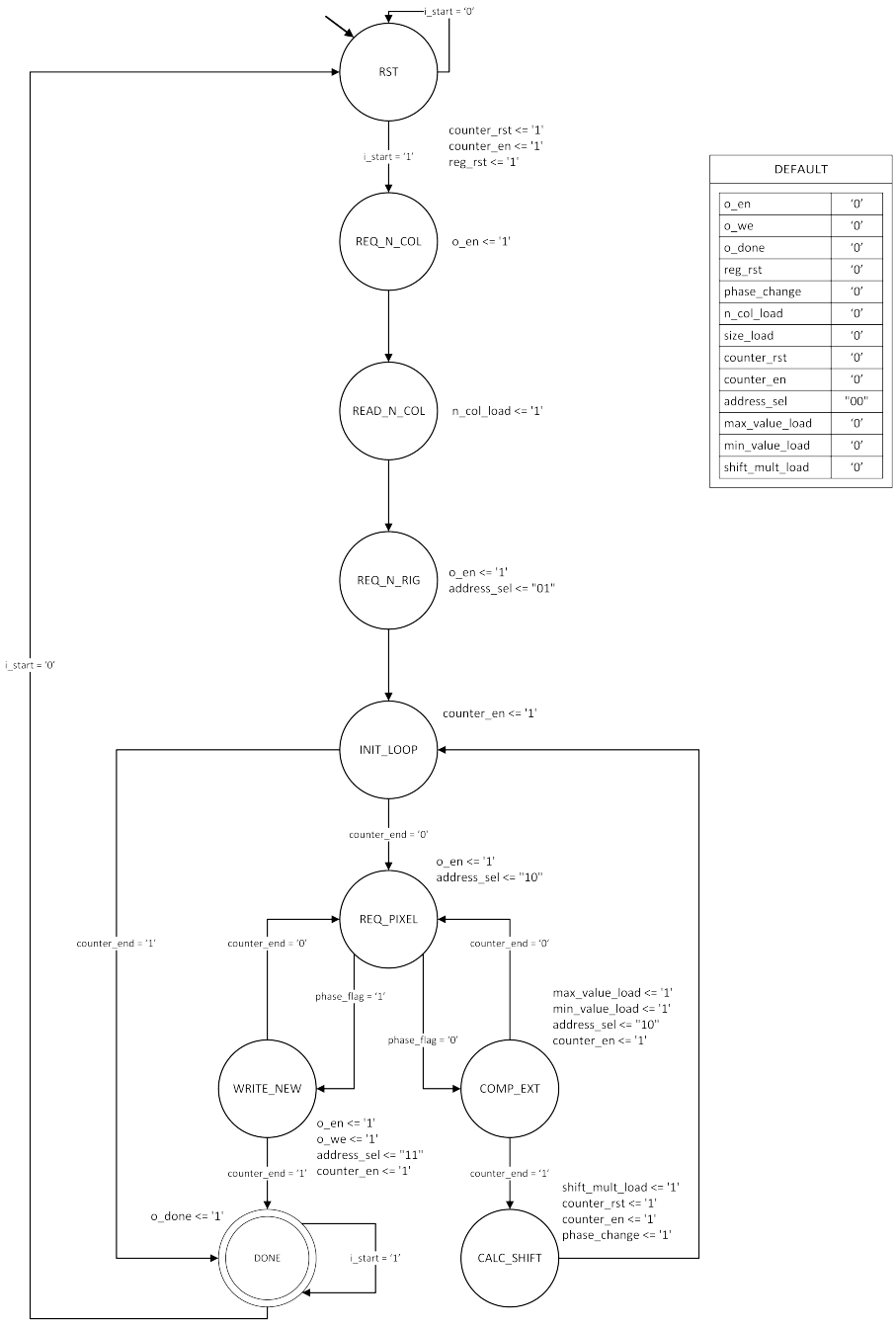


Figura 5: Diagramma della macchina a stati e valori di default dei segnali.

3 Risultati sperimentali

3.1 Test generali

Il componente è stato sottoposto a test di natura generica per verificare in primis il suo corretto funzionamento in condizioni normali.

3.1.1 Test fornito dal docente

E' stato effettuato un test tramite la *testbench* fornita dal docente. E' stato verificato il corretto funzionamento con un'immagine 2×2 contenente generici valori compresi in $[0; 255]$.

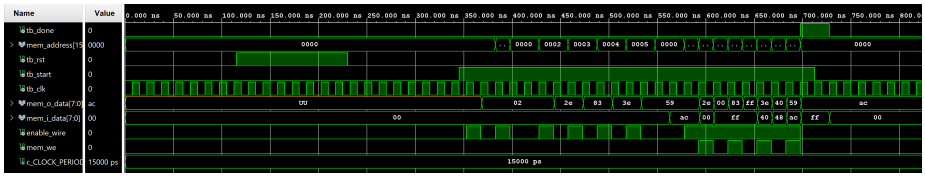


Figura 6: *Wave Window* del test fornito dal docente.

3.1.2 Test generici full-range

Per verificare il corretto funzionamento in condizioni di normale utilizzo, sono stati effettuati diversi test con immagini generiche di dimensioni non nulle e valori compresi in $[0; 255]$.

Infine, per assicurare il corretto comportamento del componente durante gli stati RST e DONE, sono state utilizzate serie di immagini generiche.

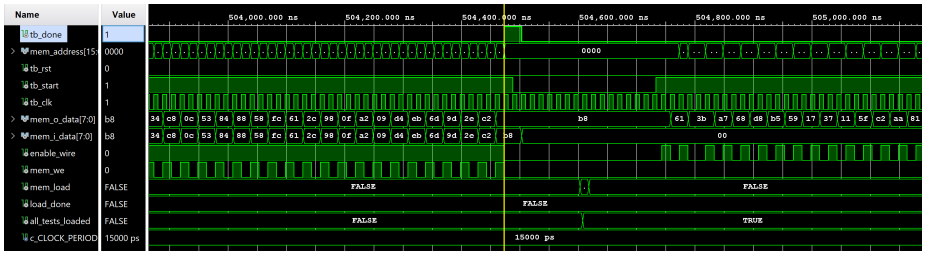


Figura 7: *Wave Window* del test con una serie di immagini generiche.

3.2 Test di copertura dei corner case

Dopo aver verificato il funzionamento in situazioni di normale utilizzo, il componente è stato sottoposto a test per verificare la copertura dei *corner case*. Per i test sono state utilizzate sia immagini singole che serie di immagini.

3.2.1 Immagine vuota

Per verificare che l'esecuzione termini immediatamente e senza creare situazioni inaspettate con valori indefiniti, sono stati effettuati test con immagini di dimensioni $0x0$, $0x128$, $128x0$ e serie di immagini "vuote".

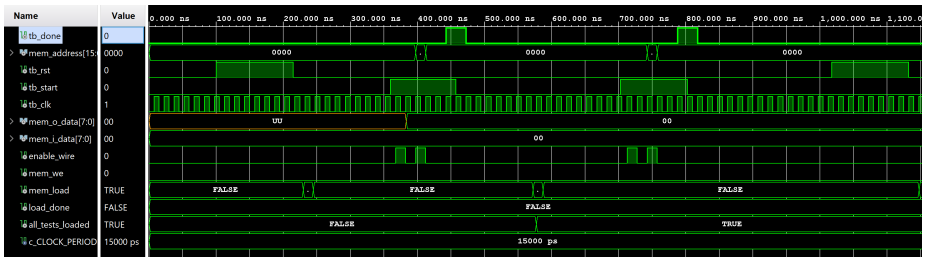


Figura 8: *Wave Window* del test con una serie di immagini "vuote".

3.2.2 Immagine di dimensione minima/massima

Obiettivo di questi test è verificare la corretta equalizzazione del contrasto e la corretta transizione tra stati, con immagini di dimensione minima (1×1) e massima (128×128).

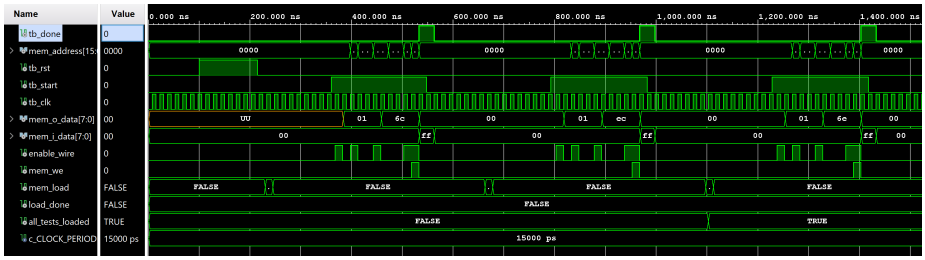


Figura 9: *Wave Window* del test con una serie di immagini 1×1.

3.2.3 Immagine bianca/nera

Obiettivo di questi test è verificare la corretta equalizzazione del contrasto ed il corretto calcolo dei valori intermedi, con immagini totalmente nere (tutti i pixel sono 0) e totalmente bianche (tutti i pixel sono 255).

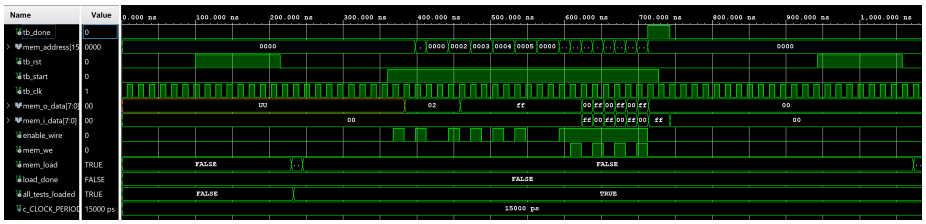


Figura 10: *Wave Window* del test con un'immagine bianca 2×2.

3.2.4 Reset asincrono

Lo scopo di questo test è verificare il corretto funzionamento in seguito alla ricezione di un segnale di reset prima della fine di un'equalizzazione ("reset asincrono").

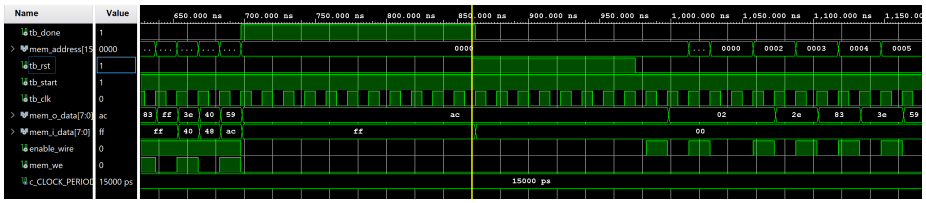


Figura 11: *Wave Window* del test con "reset asincrono"

3.3 Risultati della sintesi

Di seguito sono riportati le parti più rilevanti dei report *Timing* (Figura 12) e *Utilization* (Figura 13) della sintesi del componente.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	91,572 ns	Worst Hold Slack (WHS):	0,166 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:	212	Total Number of Endpoints:	212	Total Number of Endpoints:	78

All user specified timing constraints are met.

Figura 12: Report temporale post-sintesi.

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	241	0	134600	0.18
LUT as Logic	241	0	134600	0.18
LUT as Memory	0	0	46200	0.00
Slice Registers	77	0	269200	0.03
Register as Flip Flop	77	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Figura 13: Report *utilization* post-sintesi.

4 Conclusioni

Il componente supera correttamente tutti i test precedentemente descritti, sia in modalità *Behavioral* che in modalità *Post-Synthesis*. La robustezza è tale che il comportamento sia quello voluto anche nel caso dei *corner case*.

Considerando i casi limite delle dimensioni delle immagini, un periodo di clock abbiamo i seguenti tempi di esecuzione (calcolati dal segnale di start al segnale di done): $172,600\ ns$ con immagini 1×1; $983,152\ \mu s$ con immagini 128×128. Considerando immagini vuote (0×0) il tempo di esecuzione è $82,600\ ns$.

Dai report di sintesi risulta che il componente soddisfi i requisiti temporali con un *Worst Negative Slack* (WNS) di $91,572\ ns$ e non presenta alcun *latch*, perciò, la natura del componente è completamente combinatoria.

Nonostante sia possibile soddisfare i requisiti ottimizzando il numero di stati della macchina o riducendo la lunghezza del percorso critico (riducendo uno si rischierebbe di far aumentare l'altro), il risultato ottenuto è soddisfacente in quanto ritenuto un buon compromesso tra tempo di esecuzione totale e slack.