Prova Finale (Progetto di Reti Logiche)

Prof. Fabio Salice – Anno Accademico 2020/2021

Claudio Galimberti (Codice Persona 10610720 – Matricola 911834)
Pietro Marco Gallo (Codice Persona 10665739 – Matricola 910579)

Indice

1 Introduzione

1.1 Descrizione generale

La Prova Finale (Progetto di Reti Logiche) 2020/2021 è finalizzata all'implementazione di un circuito hardware in una FPGA, che abbia lo scopo di ricalibrare e incrementare il contrasto di un'immagine in input quando i suoi valori d'intensità sono contenuti in un intervallo molto ristretto. Come risultato, i pixel dell'immagine sono distribuiti su tutto l'intervallo di intensità, con valori da 0 a 255. Per questo scopo, viene sfruttato l'algoritmo di equalizzazione di un'immagine, del quale il circuito hardware implementa una versione semplificata rispetto a quella originale. Le immagini coinvolte sono solo quelle in scala di grigi a 256 livelli.

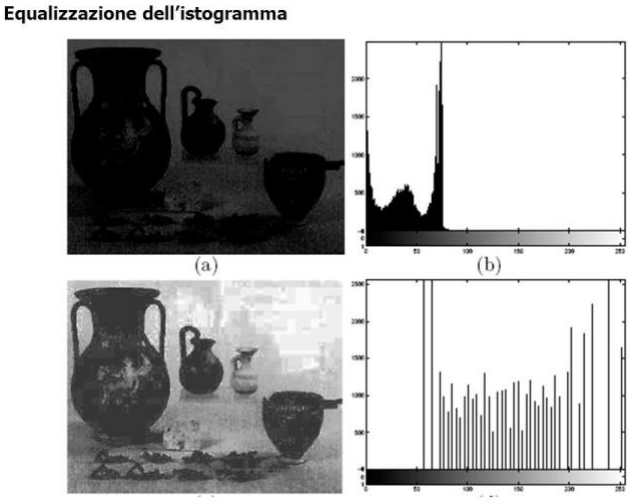


Figura 1 – figure di immagini equalizzate e non, con relativi istogrammi (sorgente: https://slideplayer.it/slide/596329/)

1.2 Descrizione della memoria

L'immagine è contenuta sequenzialmente in una memoria RAM con indirizzamento al byte e con una capienza massima di 65535 byte. Ogni pixel è memorizzato all'interno di un byte a partire dall'indirizzo 2. Nei primi due indirizzi sono memorizzati rispettivamente l'indice di colonna (indirizzo 0) e quello di riga (indirizzo 1) che definiscono la dimensione dell'immagine. Ogni immagine ha una dimensione massima di 128x128 pixel e viene letta riga per riga. L'immagine equalizzata viene salvata sequenzialmente a partire dall'indirizzo di memoria contiguo a quello in cui è memorizzato l'ultimo pixel dell'immagine non equalizzata.
È possibile equalizzare più immagini nell'intero processo, purché siano elaborate una alla volta. In caso di più di un'immagine da equalizzare, ciascuna risulta salvata in una memoria RAM diversa.

Indice di colonna (n_col)	Indirizzo 0
Indice di riga (n_rig)	Indirizzo 1
Pixel 1 non equalizzato	Indirizzo 2
Pixel 2 non equalizzato	Indirizzo 3
…	…
Pixel n_col * n_rig non equalizzato	Indirizzo n_col * n_rig + 1
Pixel 1 equalizzato	Indirizzo n_col * n_rig + 2
Pixel 2 equalizzato	Indirizzo n_col * n_rig + 3
…	…
Pixel n_col * n_rig equalizzato	Indirizzo 2 * n_col * n_rig + 1

Figura 2 – rappresentazione della memoria RAM utilizzata nel processo di equalizzazione

1.3 Specifiche del progetto

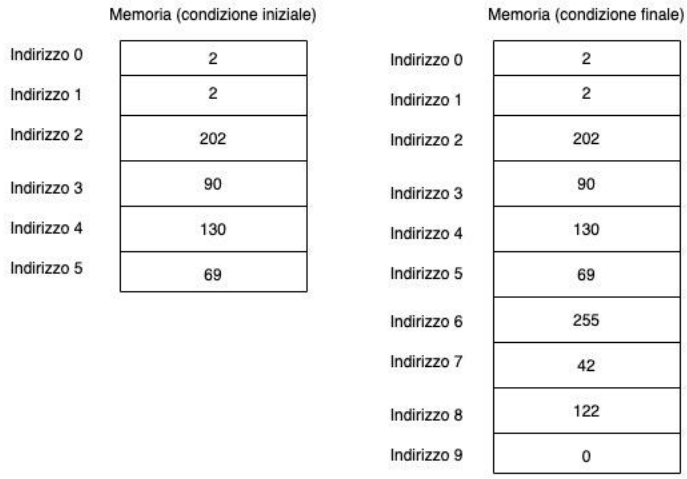
Ogni pixel dell'immagine non equalizzata viene sottoposto al seguente processo:

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)

Immagine in input:

$$\begin{pmatrix} 202 & 90 \\ 130 & 69 \end{pmatrix}$$

Figura 3 – esempio del processo di equalizzazione



1.4 Interfaccia del componente

Il componente 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;
```

- i_clk è il segnale di clock in ingresso generato dal test bench;
- i_rst è il segnale di reset che inizializza la macchina perché riceva il primo segnale di start;
- i_start è il segnale di start generato dal test bench;
- i_data è il segnale che arriva dalla memoria a seguito di una richiesta di lettura;
- o_address è il segnale 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 mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di write enable da mandare alto (= 1) alla memoria per poterci scrivere. Per leggere da memoria esso deve essere basso (= 0);
- o_data è il segnale in uscita dal componente verso la RAM.

Il segnale i_start viene posto a 1 dal test bench dopo un ciclo di clock nel quale è stato alto il segnale i_rst. L'innalzamento del segnale i_start comporta l'inizio della lettura in memoria. La fine dell'equalizzazione di un'immagine è scandita dall'innalzamento del segnale o_done da parte del circuito, al quale il test bench risponde con l'abbassamento del segnale i_start. Dopo un ciclo di clock, il circuito abbassa anche il segnale o_done ed è pronto a leggere una nuova immagine quando i_start viene nuovamente alzato dal test bench. Il segnale i_rst è asincrono e può essere posto a 1 non solo all'inizio della computazione della prima immagine, ma anche durante l'elaborazione di un'immagine qualunque.

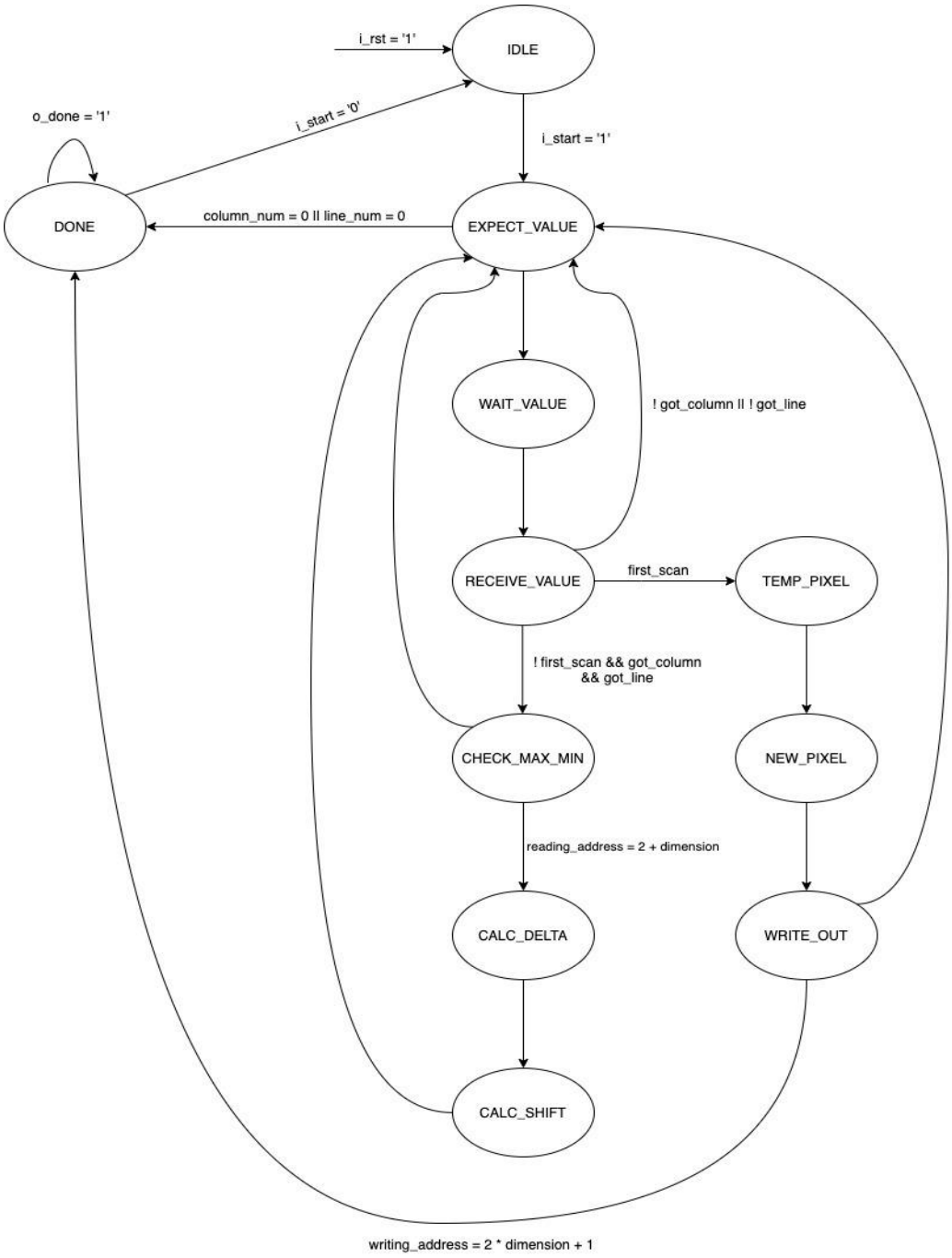


Figura 4 – descrizione della FSM(D) del componente[1]

[1] In caso di due transizioni a partire da uno stesso stato (p.es. le due transizioni da CHECK_MAX_MIN), si può osservare che su una è specificata una condizione e sull'altra no. Si segue la seconda transizione se la condizione indicata sulla prima non è soddisfatta.

2 Architettura

Il componente è stato interamente descritto all'interno di un singolo modulo mediante astrazione algoritmica. A tale scopo è stata definita una macchina a stati finiti di Moore deterministica con undici stati. La macchina legge anzitutto gli indici di colonna e riga dalla memoria, per calcolare la dimensione dell'immagine in ingresso. Successivamente, vengono letti sequenzialmente tutti i pixel dell'immagine, se ce ne sono (cfr. caso limite 1, capitolo 3.1.1), per definire qual è il valore massimo e quello minimo, di modo da calcolare DELTA_VALUE e SHIFT_LEVEL. In seguito, i pixel vengono nuovamente letti, sottoposti a modifica e scritti in memoria uno alla volta.
L'algoritmo della macchina è stato implementato con l'utilizzo di due process: il primo descrive la parte sequenziale, che assegna ai valori dei segnali e dello stato correnti i valori presi durante il precedente ciclo di clock dalla parte combinatoria; il secondo process descrive la parte combinatoria, che definisce il comportamento della macchina all'interno di ogni singolo stato.

2.1 Segnali interni

Il componente è corredato da una serie di segnali interni, ciascuno con una specifica funzione:

- **current_state** (state_type): definisce lo stato corrente della macchina;
- **first_scan** (boolean): indica se l'immagine è stata completamente letta per la prima volta;
- **got_column, got_line** (boolean): indicano se il valore rispettivamente di indice di colonna e di riga è stato letto dalla memoria;
- **multiplication** (boolean): indica se è già stata calcolata la dimensione dell'immagine;
- **writing_time** (boolean): indica se il valore del pixel è già stato processato per essere scritto in memoria;
- **reading_address** (std_logic_vector(15 downto 0)): è il segnale in cui viene salvato l'indirizzo corrente di lettura dalla memoria;
- **writing_address** (std_logic_vector(15 downto 0)): è il segnale in cui viene salvato l'indirizzo corrente di scrittura nella memoria;
- **dimension** (std_logic_vector(15 downto 0)): è il segnale in cui viene memorizzata la dimensione dell'immagine come prodotto dell'indice di colonna per l'indice di riga;
- **temp_value** (std_logic_vector(15 downto 0)): è il segnale in cui viene salvato TEMP_PIXEL (cfr. capitolo 1.3);
- **new_pixel_value** (std_logic_vector(7 downto 0)): è il segnale cui viene attribuito il valore effettivo del pixel equalizzato da salvare in memoria (NEW_PIXEL_VALUE, cfr. capitolo 1.3);
- **column_num, line_num** (std_logic_vector(7 downto 0)): sono i segnali in cui vengono memorizzati rispettivamente l'indice di colonna e l'indice di riga dell'immagine non equalizzata;
- **current_pixel** (integer range 0 to 255): è il segnale in cui viene memorizzato il valore del pixel non equalizzato letto dalla memoria (CURRENT_PIXEL_VALUE, cfr. capitolo 1.3);
- **min_value, max_value** (integer range 0 to 255): sono i segnali in cui vengono salvati rispettivamente i valori minimo e massimo dell'insieme dei pixel non equalizzati (rispettivamente MIN__PIXEL_VALUE e MAX_PIXEL_VALUE, cfr. capitolo 1.3);
- **delta_value** (integer range 0 to 255): è il segnale in cui viene salvata la differenza max_value – min_value (DELTA_VALUE, cfr. capitolo 1.3);
- **shift** (integer range 0 to 8): è il segnale in cui è memorizzato SHIFT_LEVEL (cfr. capitolo 1.3).

Tutti questi segnali definiscono nel loro insieme lo stato corrente della macchina. A ciascuno di essi è associato un corrispettivo segnale stato prossimo (per esempio, next_state per current_state, reading_address_next per reading_address, e via dicendo), che definisce il valore che sarà assunto dal segnale corrente nel successivo ciclo di clock. Tutti questi segnali associati, dunque, definiscono lo stato prossimo della macchina.

2.2 Stati della macchina

2.2.1 Stato IDLE

È lo stato iniziale nel quale si attende l'innalzamento di i_start a 1 per l'inizio della computazione. È anche lo stato cui viene inizializzata la macchina quando i_rst viene asincronicamente posto a 1.

2.2.2 Stato EXPECT_VALUE

È lo stato in cui viene definito l'indirizzo di memoria da cui leggere il valore in ingresso, sia esso l'indice di colonna o di riga, oppure il pixel corrente. Nel caso in cui l'indice di colonna o quello di riga risultasse pari a 0, allora in memoria non è salvata nessuna immagine, per cui la macchina può già terminare la computazione passando direttamente allo stato finale.

2.2.3 Stato WAIT_VALUE

È lo stato in cui la macchina attende il valore letto all'indirizzo di memoria definito nello stato EXPECT_VALUE.

2.2.4 Stato RECEIVE_VALUE

È lo stato in cui la macchina riceve in ingresso il valore corrente dalla memoria. Una serie di segnali booleani interni (got_column, got_line e first_scan) definisce come interpretare il valore ricevuto, che può essere:

- L'indice di colonna o l'indice di riga, a seconda che il segnale got_column o got_line sia ancora false;
- Un pixel dell'immagine non equalizzata. In questo caso, solo durante la lettura del primo pixel per la prima volta (ovvero quando multiplication è false) viene calcolato il valore della dimensione dell'immagine (prodotto di indice di riga e indice di colonna). Il risultato viene attribuito al segnale interno dimension (vettore di 16 bit).

2.2.5 Stato CHECK_MAX_MIN

È lo stato in cui la macchina controlla se il pixel letto dalla memoria è il massimo o il minimo dell'insieme di pixel dell'immagine.

2.2.6 Stato CALC_DELTA

È lo stato in cui, una volta letta tutta l'immagine per la prima volta, viene calcolata la differenza tra i valori massimo e minimo dei pixel dell'immagine.

2.2.7 Stato CALC_SHIFT

È lo stato nel quale viene calcolato shift_level, ovvero il numero di bit coinvolti nello shift logico a sinistra di temp_pixel. Sono implementati in questo stato controlli a soglia per realizzare la funzione a valori discreti floor( ).

2.2.8 Stato TEMP_PIXEL

È lo stato nel quale viene calcolato temp_pixel come shift logico a sinistra, per un numero di bit pari a shift_level, della differenza tra il pixel corrente letto dalla memoria per la seconda volta e il pixel minimo dell'immagine. Il risultato sarà il pixel corrispondente dell'immagine equalizzata, se rientra nell'intervallo [0, 255].

2.2.9 Stato NEW_PIXEL

È lo stato in cui viene calcolato il pixel effettivo dell'immagine equalizzata come minimo tra 255 e temp_pixel. La scelta di questo calcolo del minimo serve per mantenere l'intervallo della scala di 256 grigi anche nell'immagine equalizzata.

2.2.10 Stato WRITE_OUT

È lo stato in cui viene assegnato l'indirizzo di memoria in cui scrivere il pixel dell'immagine equalizzata.

2.2.11 Stato DONE

È lo stato in cui la prima volta il segnale o_done viene posto a 1, la seconda volta si attende che la memoria abbassi i_start, come conseguenza dell'innalzamento di o_done da parte del componente. Quando ciò avviene, la macchina viene riportata allo stato iniziale IDLE per incominciare l'elaborazione di una nuova immagine. Non è richiesta l'attesa dell'innalzamento del segnale i_rst per transitare da una computazione a un'altra.

3 Test Bench

Il progetto è stato sottoposto a una serie di test bench, simulandone il comportamento prima (Behavioral) e dopo la sintesi (Post-Synthesis Functional[2]). I test bench hanno coinvolto numerosi casi generali e casi limite di comportamento della macchina.
I casi limite reputati di maggior interesse si suddividono in base a una specifica peculiarità:

- Casi limite in termini di dimensione dell'immagine in ingresso;
- Casi limite in termini di valori dei pixel dell'immagine in ingresso.

3.1 Casi limite

3.1.1 Dimensione dell'immagine

1. **Immagine nulla:** in questo caso non è salvata nessuna immagine nella RAM e questo è identificato dal fatto che uno dei due indici (di colonna o di riga) è pari a 0. Quando il componente legge un indice nullo, passa immediatamente allo stato finale.

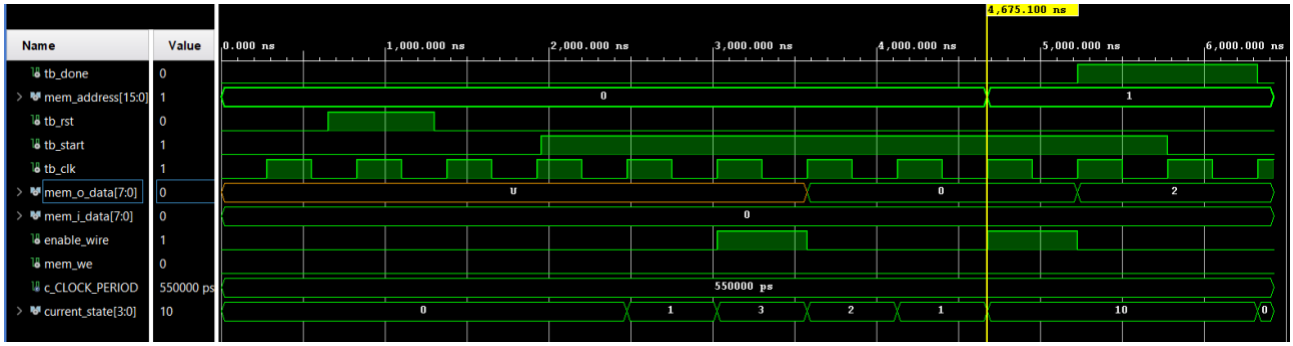


Figura 5 – immagine nulla con indice di colonna pari a 0

[2] La sintesi di Vivado codifica gli undici stati della FSM mediante codifica binaria naturale. Per una maggiore comodità di lettura, si è scelto di mostrare gli stati con i loro corrispettivi numeri interi decimali: 0 = IDLE, 1 = EXPECT_VALUE, 2 = RECEIVE_VALUE, 3 = WAIT_VALUE, 4 = CHECK_MAX_MIN, 5 = CALC_DELTA, 6 = CALC_SHIFT, 7 = TEMP_PIXEL, 8 = NEW_PIXEL, 9 = WRITE_OUT, 10 = DONE.

2. **Immagine 128x128:** l'immagine in memoria ha la massima dimensione possibile secondo la specifica di progetto. A livello di elaborazione di una singola immagine, questo è il caso di test che impiega il maggior tempo.

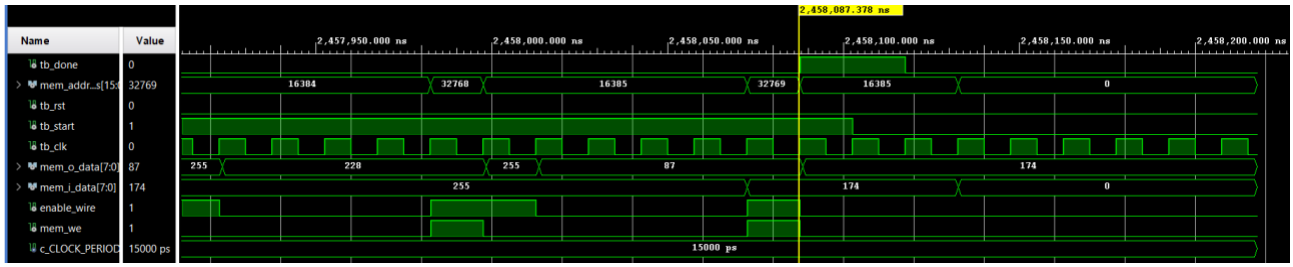


Figura 6 – immagine 128x128 (momento di fine elaborazione)

3. **Immagine 1x1:** l'immagine in memoria ha un solo pixel. In questo caso pixel massimo e minimo coincidono, quindi DELTA_VALUE è sempre 0 e lo shift è sempre 8. Siccome CURRENT_PIXEL_VALUE – MIN_VALUE è nullo, il pixel equalizzato avrà sempre valore nullo.

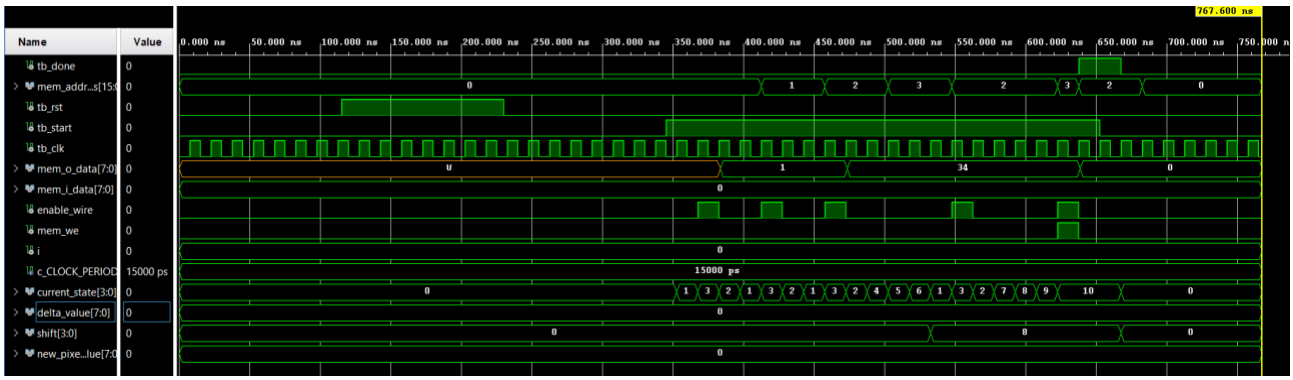


Figura 7 – immagine 1x1 (elaborazione completa)

3.1.2 Valori di pixel

1. **Immagine a tinta unita nera (tutti i pixel a 0):** in questo caso l'immagine non subisce modifiche di contrasto, in quanto l'uguaglianza tra tutti i pixel determina una differenza nulla tra massimo e minimo pixel, con conseguente shift_level pari a 8 e un'immagine equalizzata con tutti i pixel pari a 0.

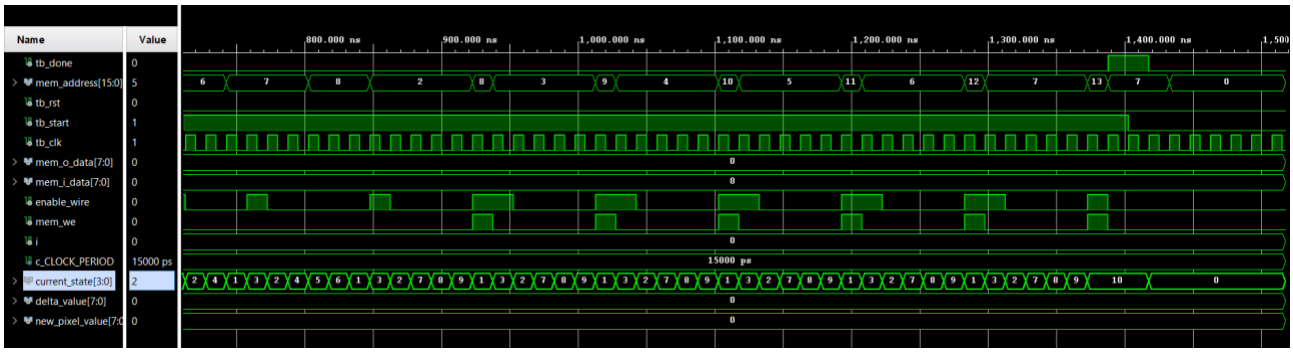


Figura 8 – immagine con pixel a 0 (fine elaborazione)

2. **Immagine a tinta unita bianca (tutti i pixel a 255):** in questo caso l'immagine subisce la massima modifica di contrasto, diventando un'immagine a tinta unita nera. Questo è dovuto sempre al fatto che i pixel sono tutti uguali, quindi DELTA_VALUE = 0 e SHIFT_LEVEL = 8.

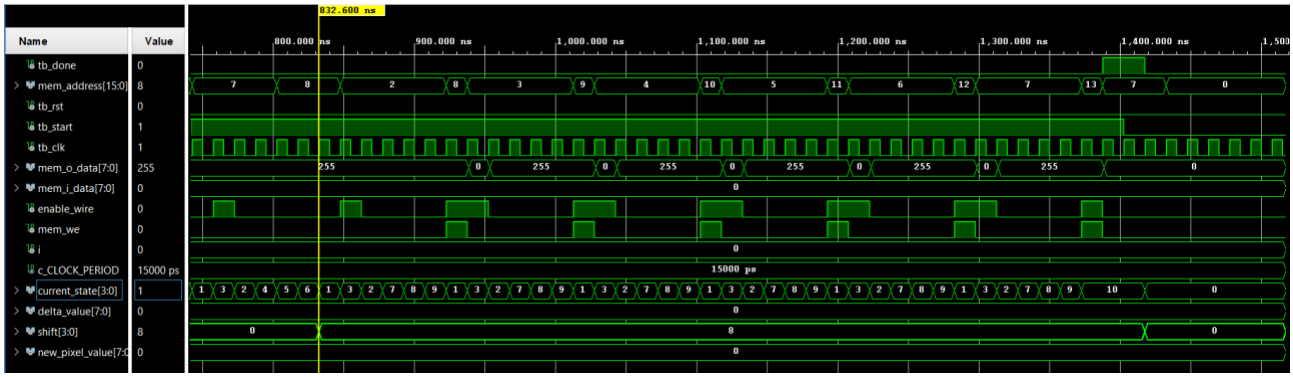


Figura 9 – immagine con pixel a 255 (fine elaborazione)

3. **Immagine a tinta unita (pixel tutti uguali):** è la generalizzazione dei due casi limite precedenti. Il risultato è sempre un'immagine a tinta unita nera. E questo vale per ogni valore assumibile dai pixel, purché essi siano tutti uguali, proprio in virtù del fatto che la differenza tra massimo e minimo pixel è nulla, quindi il generico pixel sarà sempre shiftato di 8 a sinistra e gli 8 bit meno significativi saranno tutti 0.

3.2 Trigger di reset asincrono

Se il segnale di reset viene alzato asincronicamente, la macchina interrompe l'elaborazione dell'immagine, qualunque sia il punto in cui è arrivata; si riporta allo stato IDLE, inizializzando tutti i segnali al loro valore iniziale; e ricomincia dall'inizio l'elaborazione della stessa immagine.
Il trigger del reset può avvenire sia prima dell'elaborazione della prima immagine, come indicato dalla specifica di progetto, sia in qualsiasi momento della computazione.

1. Reset in fase di lettura

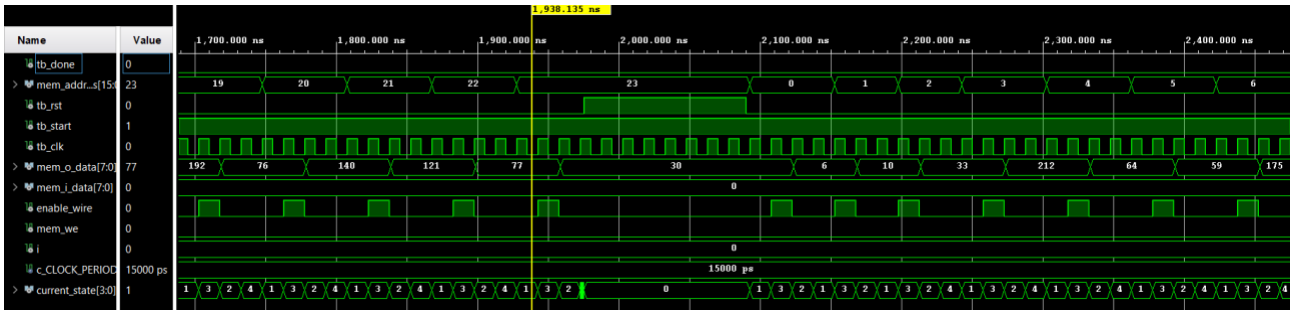


Figura 10 – reset asincrono durante la lettura[3] del ventiduesimo pixel (mem_address = reading_address = 23)

[3] Al momento esatto dell'innalzamento del segnale i_rst, current_state ha valore 4, cioè lo stato corrente è CHECK_MAX_MIN. A causa della brevissima durata dello stato, non è chiaramente visibile nell'immagine riportata.

2. Reset in fase di scrittura

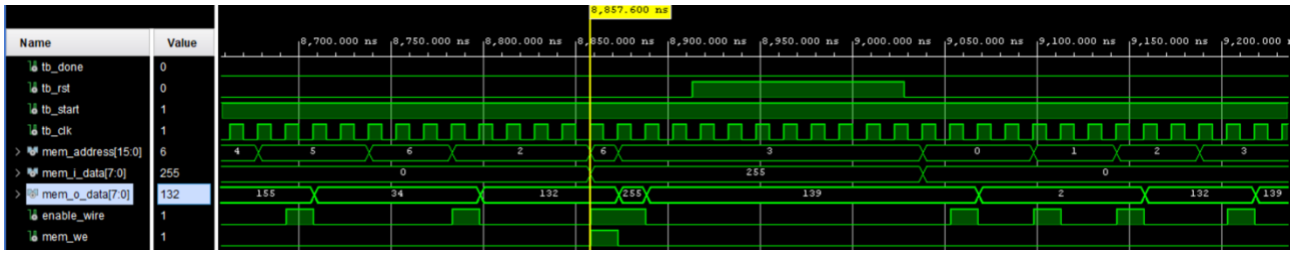


Figura 11 – reset asincrono alla fase di scrittura del secondo pixel equalizzato

3. Reset con segnale o_done = '1' (stato DONE)

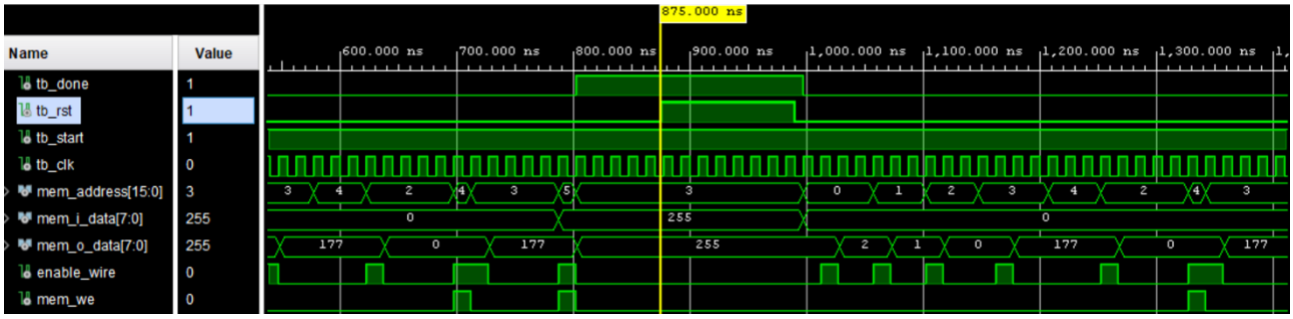


Figura 12 – reset asincrono esattamente prima che i_start sia abbassato

3.3 Più immagini in sequenza

Il componente è stato progettato per elaborare anche più immagini in sequenza. La transizione da una computazione a un'altra prevede la reciproca interazione tra i segnali i_start e o_done in questo modo: alla fine della computazione, il segnale o_done viene alzato la prima volta in cui la macchina entra nello stato DONE. Viene passato al test bench, il quale innesca l'abbassamento del segnale i_start. Il componente risponde a questa modifica abbassando il segnale o_done, mettendosi in attesa di un successivo innalzamento del segnale i_start da parte del test bench stesso.

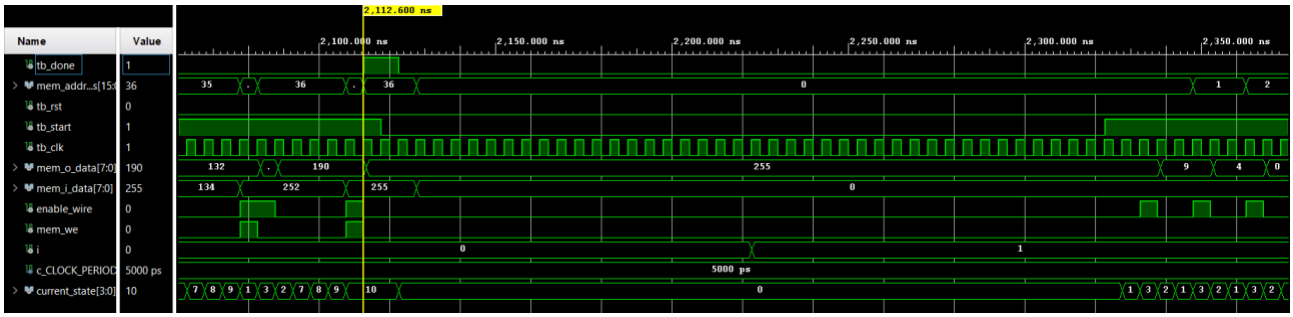


Figura 13 – transizione da un'immagine a un'altra

3.4 Altri test notevoli e considerazioni aggiuntive

Utilizzando un programma in linguaggio C, sono stati generati una serie di test casuali per verificare il buon funzionamento del progetto. Il 100% di essi è stato superato dal componente. Tra questi casi di test si rimarcano due tipologie particolari:

- Immagini nulle e non nulle in sequenza;
- Valore massimo o minimo come ultimo pixel dell'immagine.

4 Report e Conclusioni

Analizzando report_utilization, dopo la fase di sintesi logica effettuata dal software Vivado, risulta che il componente occupa uno spazio inferiore all'1% della FPGA in cui è inserito, la quale è di tipo xc7a200tfbg484-1. L'architettura del componente inferita da Vivado contiene 252 slice look-up tables e 160 flip-flop tipo D. Insieme ad essi sono presenti 38 input, dei quali uno è bufferizzato: il clock (i_clk).

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	252	0	134600	0.19
LUT as Logic	252	0	134600	0.19
LUT as Memory	0	0	46200	0.00
Slice Registers	160	0	269200	0.06
Register as Flip Flop	160	0	269200	0.06
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

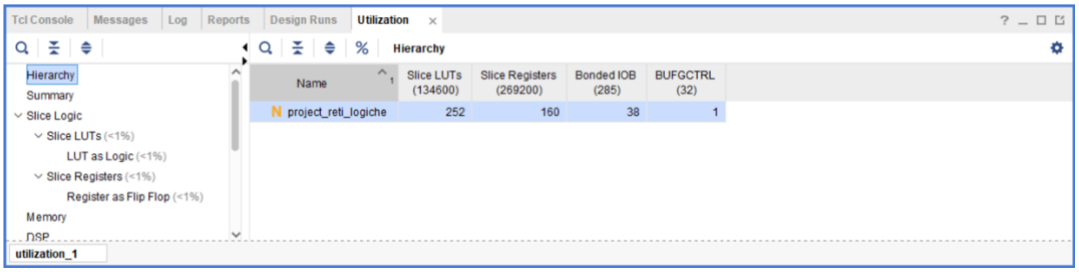


Figura 14 – componenti architetturali (report_utilization)

Come richiesto da specifica, è stato imposto un vincolo di tempo, cioè che il componente funzioni con un periodo del clock di almeno 100 ns. Analizzando report_timing e report_timing_summary, il componente impiega 6,928 ns per far commutare tutte le porte logiche, lasciando un intervallo di tempo massimo (WNS) di 92,921 ns tra quel momento e l'inizio di un nuovo ciclo di clock. Il componente, pertanto, può funzionare fino a una frequenza massima del clock di circa 144,34 MHz.

```
Path Group:             i_clk
Path Type:              Setup (Max at Slow Process Corner)
Requirement:            100.000ns  (i_clk rise@100.000ns - i_clk rise@0.000ns)
Data Path Delay:        6.928ns  (logic 3.077ns (44.414%)  route 3.851ns (55.586%))
```

Figura 15 – Report Timing

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	92,921 ns	Worst Hold Slack (WHS):	0,144 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:	292	Total Number of Endpoints:	292	Total Number of Endpoints:	161

All user specified timing constraints are met.

Figura 16 – Report Timing Summary