Prova Finale Reti Logiche

Prof. William Fornaciari

Alice Cariboni – Matricola 909839

Anno Accademico 2020 – 2021

Indice

1 Introduzione

Scopo del progetto è l'implementazione di un modulo che elabora un'immagine e la riscrive in memoria equalizzandola, secondo una versione semplificata dell'algoritmo standard.

Le immagini sono rappresentate in memoria da valori compresi tra 0 e 255 che rappresentano il valore del singolo pixel, mentre i primi due byte della memoria indicano numero di righe e numero di colonne dell'immagine (ciascuno rappresentato su 8 bit).

Il valore dei nuovi pixel va calcolato e riscritto in memoria a partire dal byte successivo all'ultimo byte dell'immagine data secondo le seguenti formule:

DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE

SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE +1)))

TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL

NEW_PIXEL_VALUE = MIN(255 , TEMP_PIXEL)

Dove MAX_PIXEL_VALUE e MIN_PIXEL_VALUE sono rispettivamente il massimo e minimo valore che i pixel dell'immagine assumono e NEW_PIXEL_VALUE è il valore del nuovo pixel da scrivere in memoria.

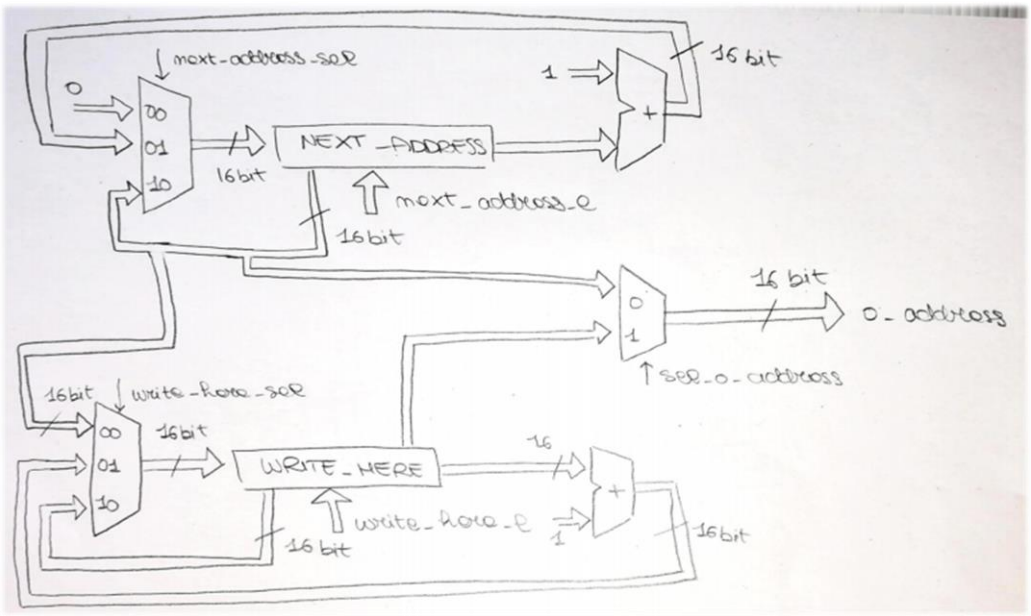
La memoria, quindi verrà letta due volte: una per trovare il massimo e minimo tra i valori dei pixel e una seconda volte per applicare le formule al valore del pixel e scrivere in memoria il nuovo valore.

2 Architettura

L'architettura è organizzata in due moduli, il datapath e la macchina a stati.

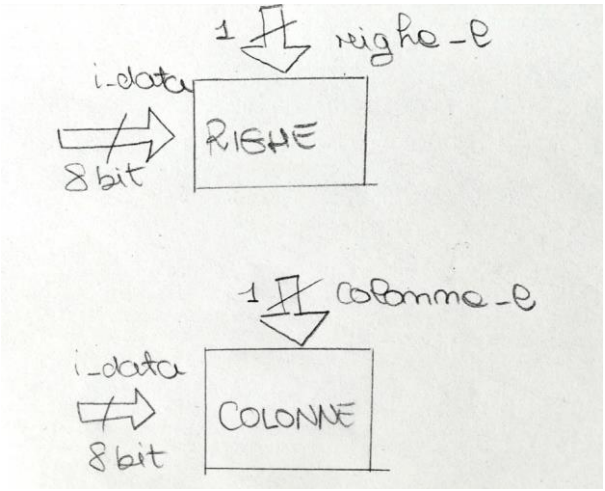
- **Datapath**
  Il componente datapath è realizzato da una collezione di process che controllano, tramite opportuni segnali, il caricamento dei valori nei rispettivi registri.

- **Registri next_address e write_here**



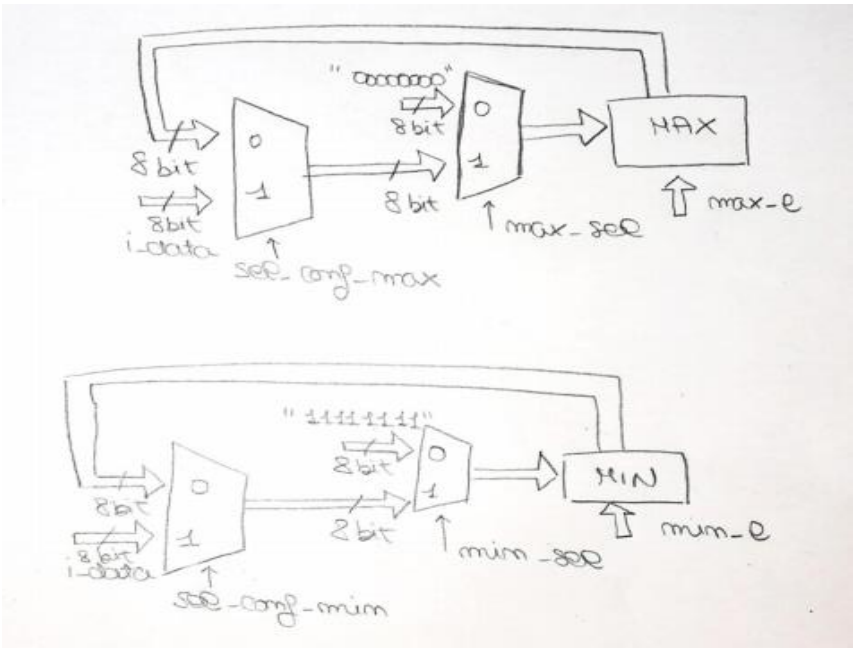
Questi due registri gestiscono i valori che o_address assume, next_address è l'indirizzo del prossimo valore da leggere, write_here è il prossimo indirizzo di memoria in cui bisogna scrivere ed è inizializzato al valore più grande assunto da next_address + 1. I due mux permettono di scegliere se inizializzare il registro, mantenerne il valore o incrementarlo di un'unità. Il mux in uscita, invece permette di scegliere il valore assunto da o_address.

- **Registri righe e colonne**



Sono i primi registri ad essere inizializzati durante la computazione e mantengono il valore di righe e colonne dell'immagine da elaborare. Con rispettivi mux per l'inizializzazione

- **Registri max e min**

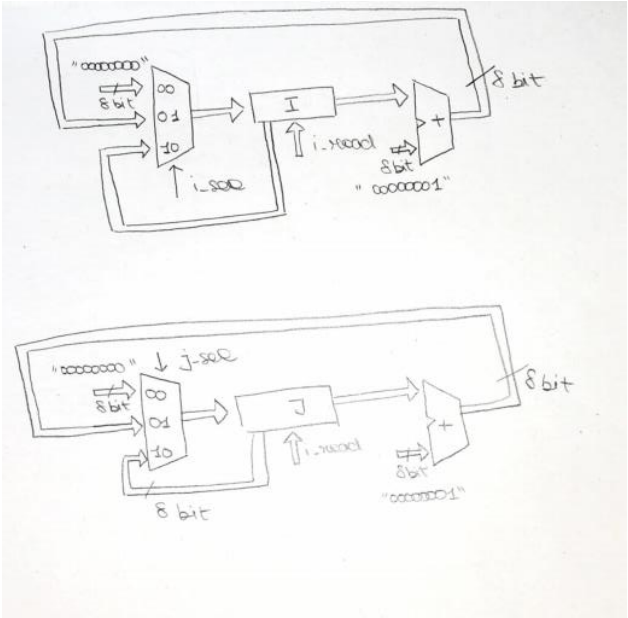


Questi registri servono prima per cercare poi per mantenere il valore del massimo e minimo pixel.

I primi mux, di confronto, servono per scegliere ad ogni ciclo di clock se caricare nei registri il valore del pixel preso dalla memoria o mantenere il valore nel registro.

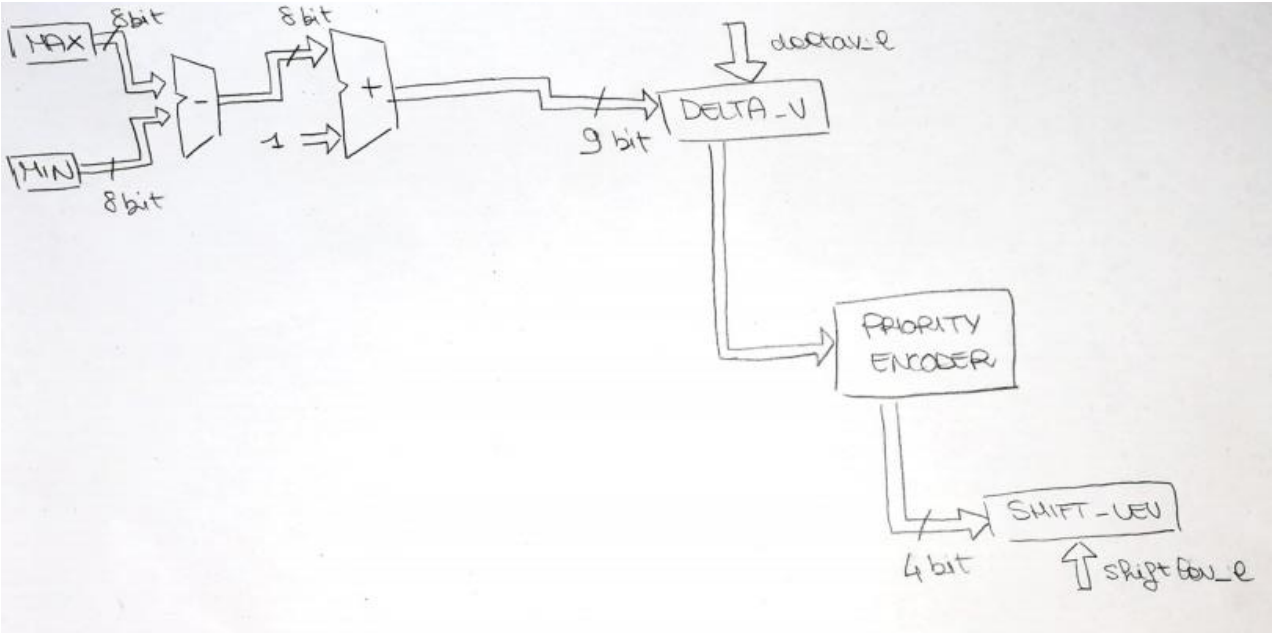
I secondi mux servono per inizializzare i registri a 0 nel caso di max , a 255 nel caso di min.

- **Registri i e j**



Fungono da contatori per righe e colonne, i mux permettono di inzializzare, incrementare o mantenere il valore dei registri.

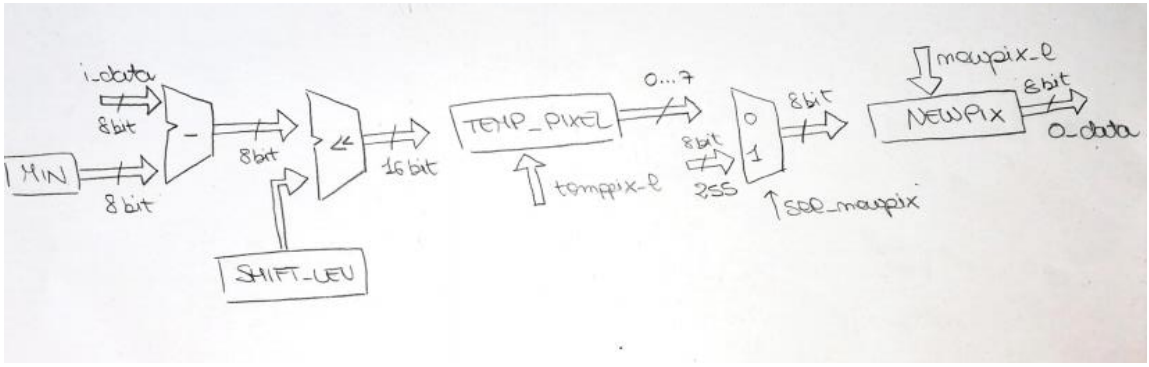
- **Registri shiftlev e deltav**



In deltav viene caricato il valore di max – min + 1(che sarà un valore tra 1 e 256 quindi rappresentabile su 9 bit). L'uscita del priority encoder corrisponde al risultato di questa formula

SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE +1)))) . E' stato realizzato con una catena di elsif in un process avente esclusivamente deltav nella sensitivity list.

- **Registri temp_pixel e new_pixel**



Il contenuto di questi registri corrisponde al risultato delle rispettive formule nella specifica.

Segnali

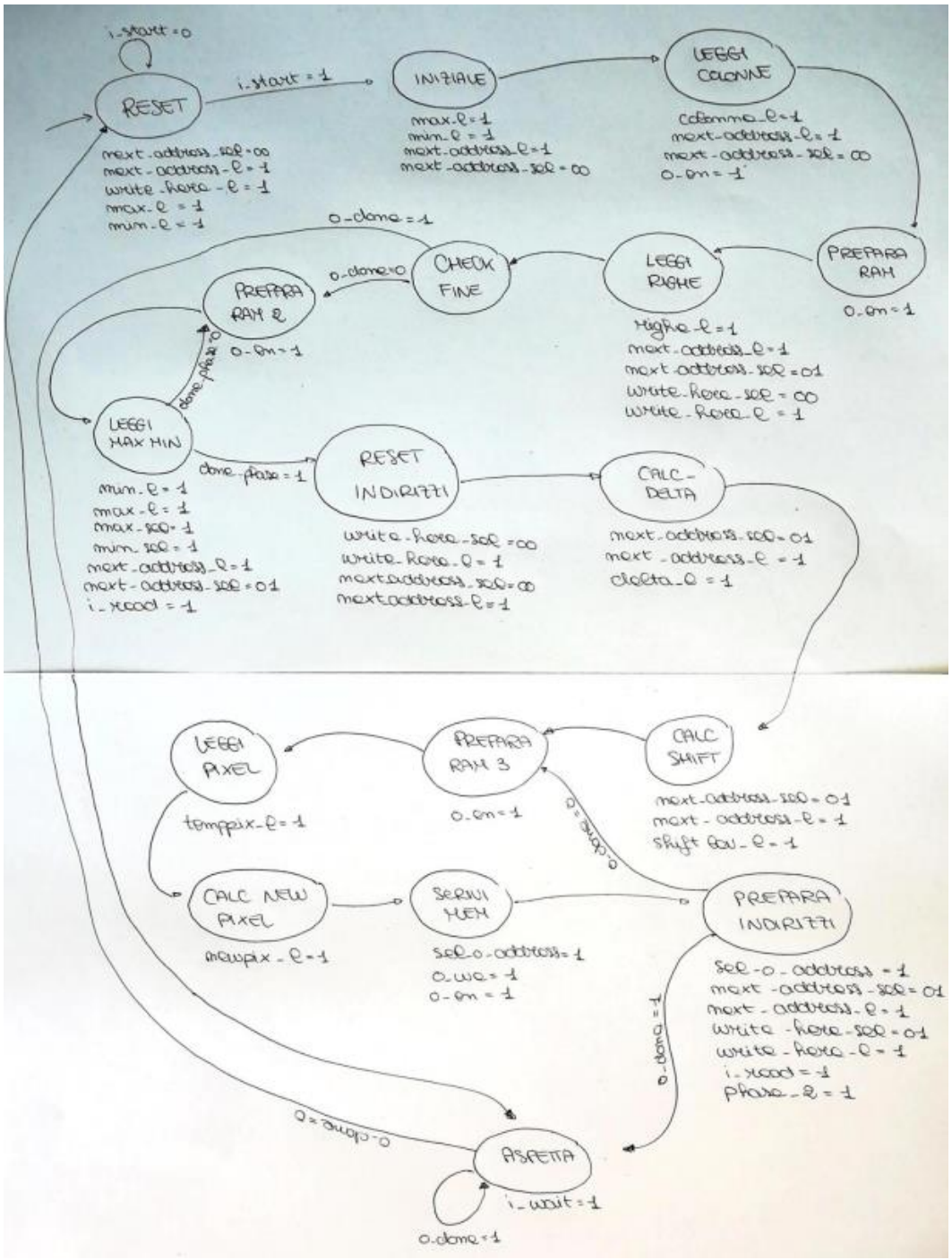
I segnali gestiti all'interno di questo modulo sono:

- o_done : viene messo a 1 a computazione finita e mantenuto a 1 finché i_start non viene abbassato, è gestito inoltre il caso particolare in cui righe o colonne abbiano valore 0.
- done_phase: viene messo a 1 quando finisce la fase di lettura e ricerca di massimo e minimo e quando termina la fase di scrittura in memoria dei nuovi pixel. Ovvero quando i registri i e j raggiungono contemporaneamente i valori (righe – 1) e (colonne – 1).
- Righe_sel e colonne_sel a 0 per inizializzare i registri a 1 per leggere da memoria il valore da inserire nei registri.
- Max_sel , min_sel per inizializzare i registri max e min.
- Sel_conf_max , sel_conf_min per selezionare il valore maggiore (o minore) tra quello letto dalla memoria e quello già presente nei registri max o min.
- I_sel e j_sel per controllare i rispettivi mux inizializzandoli, mantenendo il valore o incrementando di un'unità i valori di uscita.
- Sel_new_pix che pone il limite di 255 al valore del nuovo pixel.
- Next_address_sel e write_here_sel per controllare i rispettivi mux inizializzandoli, mantenendo il valore o incrementando di un'unità i valore di uscita.
- Sel_o_address per scegliere se caricare in o_address l'indirizzo per leggere o quello per scrivere.

- **Macchina a stati**

  Il componente macchina a stati realizza gli stati attraversati durante la computazione. E' realizzato da 3 process: uno che definisce lo stato prossimo in base allo stato corrente, uno che assegna valori ai segnali del datapath in base allo stato corrente e uno che ad ogni ciclo di clock assegna stato prossimo allo stato corrente e riporta la computazione allo stato di RESET in caso di segnale reset alto.

  La macchina a stati è rappresentata nel seguente diagramma.



- RESET

  Stato in cui si arriva dopo che il segnale di reset è stato portato alto o a computazione terminata, vengono inizializzati alcuni registri.

- INIZIALE

  Stato in cui si arriva dopo che il segnale di start è stato portato alto.

- LEGGI_COLONNE

  Viene portato a 1 il segnale di load per le colonne, al termine del ciclo di clock sarà presente nel registro colonne il dato letto da memoria.

- PREPARA_RAM

  Viene inviata alla RAM una nuova richiesta di lettura, questo stato è necessario in quanto il dato richiesto alla RAM sarà disponibile solo al ciclo di clock successivo.

- LEGGI_RIGHE

  Viene portato a 1 il segnale di load per le righe, al termine del ciclo di clock sarà presente nel registro colonne il dato letto da memoria.

- CHECK_FINE

  In questo stato viene controllato se righe o colonne hanno valore 0, quindi se o_done è stato portato a 1, in quel caso la computazione termina perché l'immagine non ha pixel da elaborare.

- PREPARA_RAM_2

  Ricomincia la lettura della memoria dal primo pixel dell'immagine.

- LEGGI_MAX_MIN

  Viene fatta una prima lettura di tutta l'immagine per trovare il massimo e minimo valore che i pixel assumono.

- RESET_INIDRIZZI

  Il registro next_address viene riportato all'inizio della memoria per poter rileggere i pixel e per ognuno calcolare il nuovo valore, mentre in write_here viene caricato il valore del primo indirizzo di memoria dopo l'ultimo pixel letto.

- CALC_DELTA

  In questo stato viene calcolato il valore di delta value.

- CALC_SHIFT

  In questo stato viene calcolato il valore di shift value.

- PREPARA_RAM_3

  Viene inviata la richiesta di lettura alla ram .

- LEGGI_PIXEL

  Viene letto il pixel e calcolato il suo valore temporaneo.

- CALC_NEW_PIXEL

  Viene calcolato il valore del nuovo pixel secondo le formule.

- SCRIVI_MEM

  Viene inviato alla memoria il valore del nuovo pixel con richiesta di scrittura.

- PREPARA_INDIRIZZI

  Gli indirizzi di lettura e scrittura vengono incrementati per leggere il pixel successivo, se il segnale o_done è portato alto la computazione termina.

- ASPETTA

  Stato in cui si arriva a computazione terminata in attesa che il segnale di start venga abbassato per poter abbassare o_done.

Segnali

I segnali gestiti all'interno di questo modulo, oltre a quelli per la gestione dei registri e multiplexer nel datapath sono:

- Stato_corrente e stato_prossimo che contengono rispettivamente lo stato corrente della computazione nella macchina a stati e lo stato in cui si troverà ne successivo ciclo di clock.
- I_read che viene portato alto ogni volta che si legge da memoria per poter incrementare i registri i e j.
- Phase_2 che viene portato alto al termine della scrittura in memoria di ogni pixel, serve al datapath per controllare se la computazione è finita e deve alzare il segnale di done.
- I_wait che viene tenuto alto nello stato ASPETTA, quindi finché non viene abbassato start.

3 TestBench

Per verificare il corretto funzionamento del componente dopo averlo testato con i testbench di esempio e altri numerosi testbench generati casualmente, ho definito alcuni testbench che spingessero la simulazione in casi limite.

1 Test vuoto
Questo test verifica che la simulazione termini non appena viene caricato nel registro un valore di riga o colonna nulli. Il componente è stato inoltre testato su più immagini vuote consecutive.

2 Test 1 pixel e tutti pixel uguali
Questi test verificano che il componente funzioni con immagini con massimo e minimo pixel uguali, questo fa si che lo shift_level sia uguale a 8, quindi tutti i nuovi pixel scritti in memoria a 0.

3 Test pixel con valori tra 0 e 255
Questo test verifica che il componente funzioni con immagini che generano uno shift_level uguale a 0, quindi i pixel non cambiano valore.

4 Test con immagine di massima dimensione
Questo test verifica che il componente funzioni con un'immagine da 128 x 128 pixel.

5 Test immagini multiple
Questo test verifica che la simulazione proceda correttamente anche su più immagini consecutive senza fornire il segnale di reset tra una e l'altra.

6 Test reset asincrono

Questo test verifica che la simulazione riprenda correttamente dopo un reset asincrono durante l'elaborazione di un'immagine.

7 Test con massimo e minimo in prima e ultima posizione
Questo test verifica che durante l'elaborazione i registri max e min contengano i giusti valori anche se questi sono agli estremi dell'immagine.

4 Sintesi

Il componente sintetizzato supera correttamente tutti i test specificati e quelli generati casualmente nelle 3 simulazioni: Behavioral, Post-Synthesis Functional e Post-Synthesis Timing. Il tempo di simulazione dipende direttamente dal numero di pixel dell'immagine da elaborare, quindi si avrà il tempo più corto con l'immagine da zero pixel e il tempo più lungo con l'immagine da 128 x 128 pixel.

• 4225 ns - tempo di simulazione (Behavioral) immagine con 0 pixel

• 1720892,5 ns - tempo di simulazione (Behavioral) immagine con 128 x 128 pixel

La sintesi vede l'impiego di 157 LUT e 133 Registri, non sono presenti inferred latch.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	157	0	134600	0.12
LUT as Logic	157	0	134600	0.12
LUT as Memory	0	0	46200	0.00
Slice Registers	133	0	269200	0.05
Register as Flip Flop	133	0	269200	0.05
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Il componente inoltre funziona con un constraint sul clock a 100 ns e con un Worst Negative Slack di 95.143 ns.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	95,143 ns	Worst Hold Slack (WHS):	0,155 ns	Worst Pulse Width Slack (WPWS):	4,500 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:	241	Total Number of Endpoints:	241	Total Number of Endpoints:	134

5 Conclusioni

Avevo inizialmente optato per un'implementazione mono processo che gestisse la computazione all'interno di un solo switch case, in uno stile più simile alla programmazione in C. La complessità di questa soluzione, però, aumentava con il numero di case e diventava quasi impossibile risalire agli statement che causavano i numerosi inferred latch.

Successivamente ho deciso di implementare il componente tramite un'architettura organizzata in due moduli (datapath e macchina a stati) con process che gestissero ogni registro e ogni aspetto della macchina a stati. Questa soluzione, dopo una prima progettazione, è risultata molto più semplice da implementare e ha reso molto più semplice rintracciare gli eventuali errori.

Una delle maggiori difficoltà nella realizzazione di questo progetto è stata abbandonare una logica di programmazione simile a C per imparare e applicare una logica più adatta alla risoluzione di questo tipo di problema.