PROVA FINALE (PROGETTO DI RETI LOGICHE) 2020 – 2021

Prof. William Fornaciari – Prof. Federico Terraneo

Alex Amati (Codice Persona: 10664910 – Matricola: 910780)

Politecnico di Milano

Indice

1 Introduzione

1.1 Scopo del progetto

Data un'immagine in scala di grigi lo scopo del progetto è quello di descrivere in VHDL un componente hardware per equalizzare l'istogramma dell'immagine.

1.2 Specifiche generali

Concettualmente l'immagine, in scala di grigi, sarà una matrice di valori ad 8 bit rappresentanti i pixel a 256 livelli. Un'immagine sarà composta al massimo da 128 x 128 pixel. Il componente riceverà in input il numero di righe e colonne in pixel dell'immagine e l'immagine stessa e dovrà fornire in output la stessa immagine equalizzata.

Per effettuare l'equalizzazione ogni pixel dell'immagine dovrà essere trasformato secondo il seguente algoritmo.

```
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` saranno rispettivamente il massimo e il minimo valore dei pixel dell'immagine da equalizzare, `CURRENT_PIXEL_VALUE` sarà il valore del pixel da trasformare, e `NEW_PIXEL_VALUE` sarà il nuovo valore del pixel.

1.3 Input/output e memoria

I dati sia di input che di output saranno scritti su una memoria di tipo RAM con indirizzamento al byte e bus indirizzi di 16 bit.

Agli indirizzi 0 ed 1 verranno forniti il numero di colonne e il numero di righe, seguiranno poi i pixel dell'immagine fino all'indirizzo `righe × colonne + 1`, dall'indirizzo successivo in poi dovranno poi essere scritti i pixel dell'immagine equalizzata in output.

Memoria:

Indirizzo	**Valore**
0	numero di colonne
1	numero di righe
2	primo pixel immagine originale
…	…
righe × colonne + 1	ultimo pixel immagine originale
righe × colonne + 2	primo pixel immagine equalizzata
…	…
2(righe × colonne) + 1	ultimo pixel immagine equalizzata

1.4 Interfaccia del componente

Il componente avrà 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;
```

Dove:

- `i_clk` sarà il segnale di clock in ingresso al componente;
- `i_rst` sarà il segnale di reset usato per inizializzare la macchina;
- `i_start` sarà il segnale utilizzato per avviare la computazione;
- `i_data` sarà il vettore contenente i dati letti dalla memoria;
- `o_address` sarà il vettore con cui il componente comunicherà l'indirizzo per leggere o scrivere la memoria;

- `o_done` sarà il segnale con cui il componente notificherà il termine della computazione;
- `o_en` sarà il segnale di MEMORY ENABLE che il componente porrà a 1 per comunicare con la memoria;
- `o_we` sarà il segnale di WRITE ENABLE che verrà posto dal componente a 1 per utilizzare la memoria in scrittura e a 0 per utilizzarla in lettura;
- `o_data` sarà il vettore contente i dati da scrivere in memoria.

1.5 Specifiche di funzionamento

All'avvio, prima dell'inizio della prima computazione, il componente dovrà essere resettato ad inizierà la computazione quando il segnale di start sarà posto a 1. La computazione procede con il prerequisito che il contenuto della memoria non verrà modificato da nessun altro nel mentre e che il segnale di start resti a 1 durante tutto il processo. Una volta terminato il processo di equalizzazione il segnale di done sarà posto ad 1 ed il componente aspetterà che il segnale di start tornerà a 0 prima di riportarsi nello stato iniziale da cui si potrà avviare un nuovo processo di equalizzazione senza dover necessariamente resettare il componente.

1.6 Esempi

Esempio di immagine non equalizzata (a sinistra) e la sua equalizzazione (a destra) - (da Wikipedia)

Segue un esempio del contenuto di ciò che si trova in memoria alla fine di un'elaborazione. Si noti che i valori dall'indirizzo 0 all'indirizzo 5 erano già scritti prima dell'elaborazione e fungono da input senza essere modificati. Alla fine dell'elaborazione il componente avrà scritto in memoria i valori dall'indirizzo 6 all'indirizzo 9.

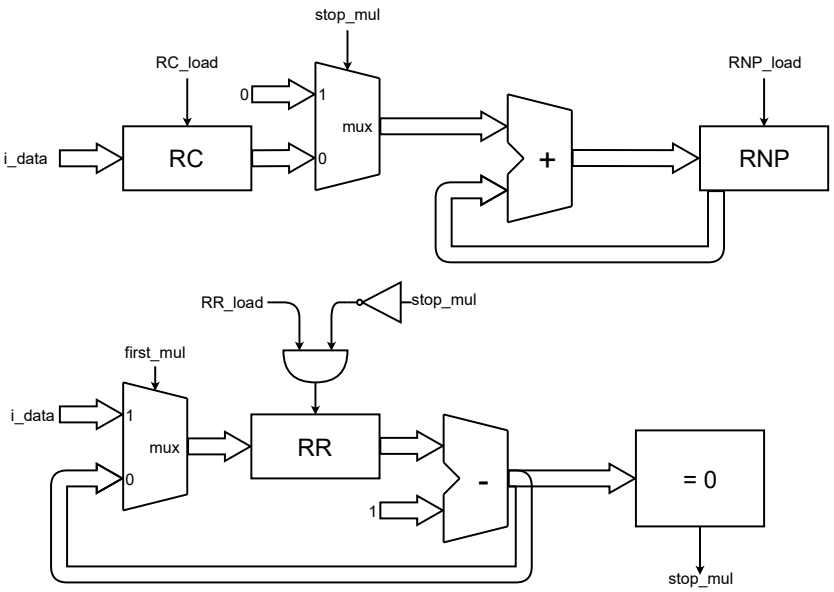
Indirizzo	**Valore**	
0	2	Numero di colonne
1	2	Numero di righe
2	46	Primo pixel originale
3	131	
4	62	
5	89	Ultimo pixel originale
6	0	Primo pixel equalizzato
7	255	
8	64	
9	172	Ultimo pixel equalizzato

2 Architettura

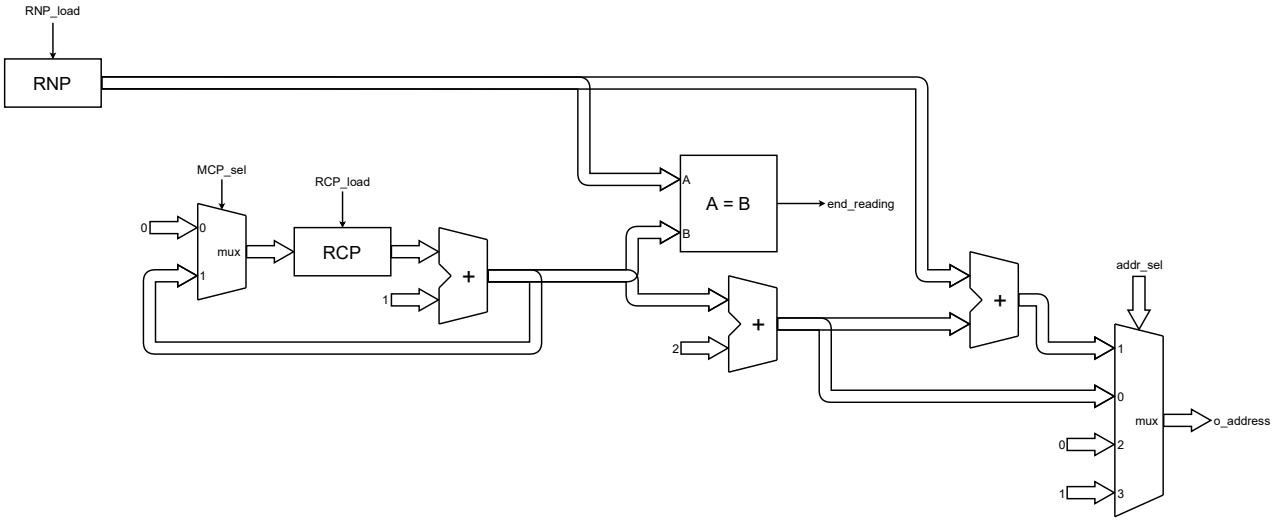
La descrizione VHDL del componente si compone di due moduli principali: un modulo data path che combina logica combinatoria e sequenziale contenente le unità di calcolo e i registri necessari all'elaborazione e un modulo di controllo, basato su una macchina a stati, per coordinare e gestire il comportamento del data path.

2.1 Data Path

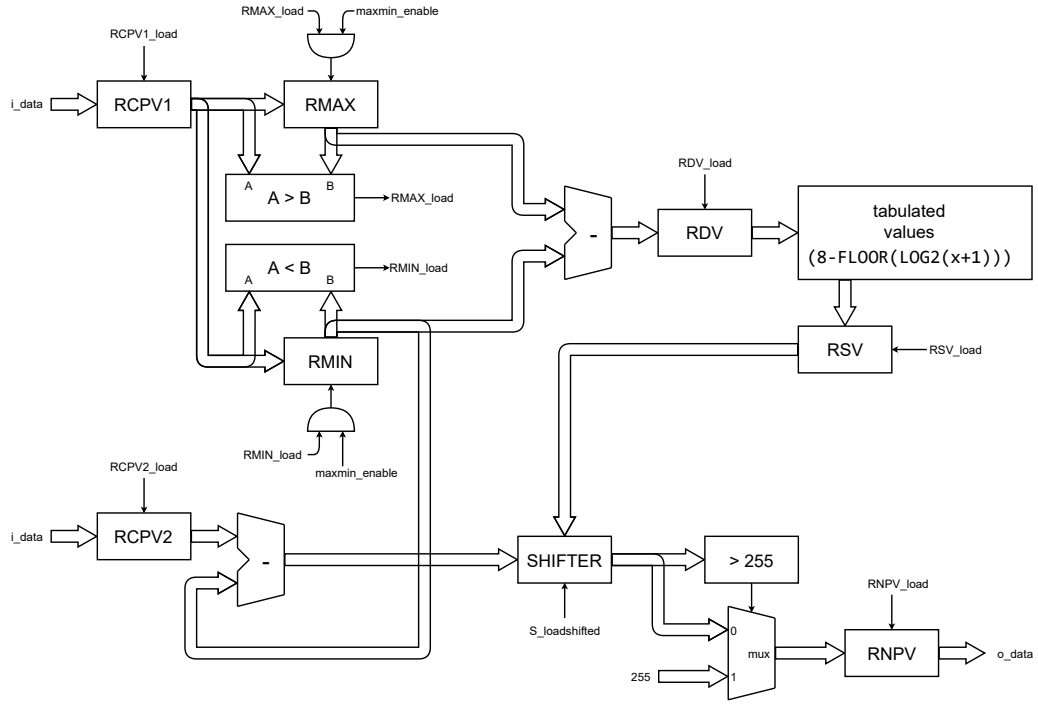
Per semplificare la descrizione VHDL del data path sono stati descritti diversi componenti di supporto come registri, multiplexer, sommatori, sottrattori et cetera. In questo modo la descrizione vera e propria del data path segue un approccio strutturale specificando quanti e quali componenti sono stati usati e come e con che segnali sono stati collegati tra loro.



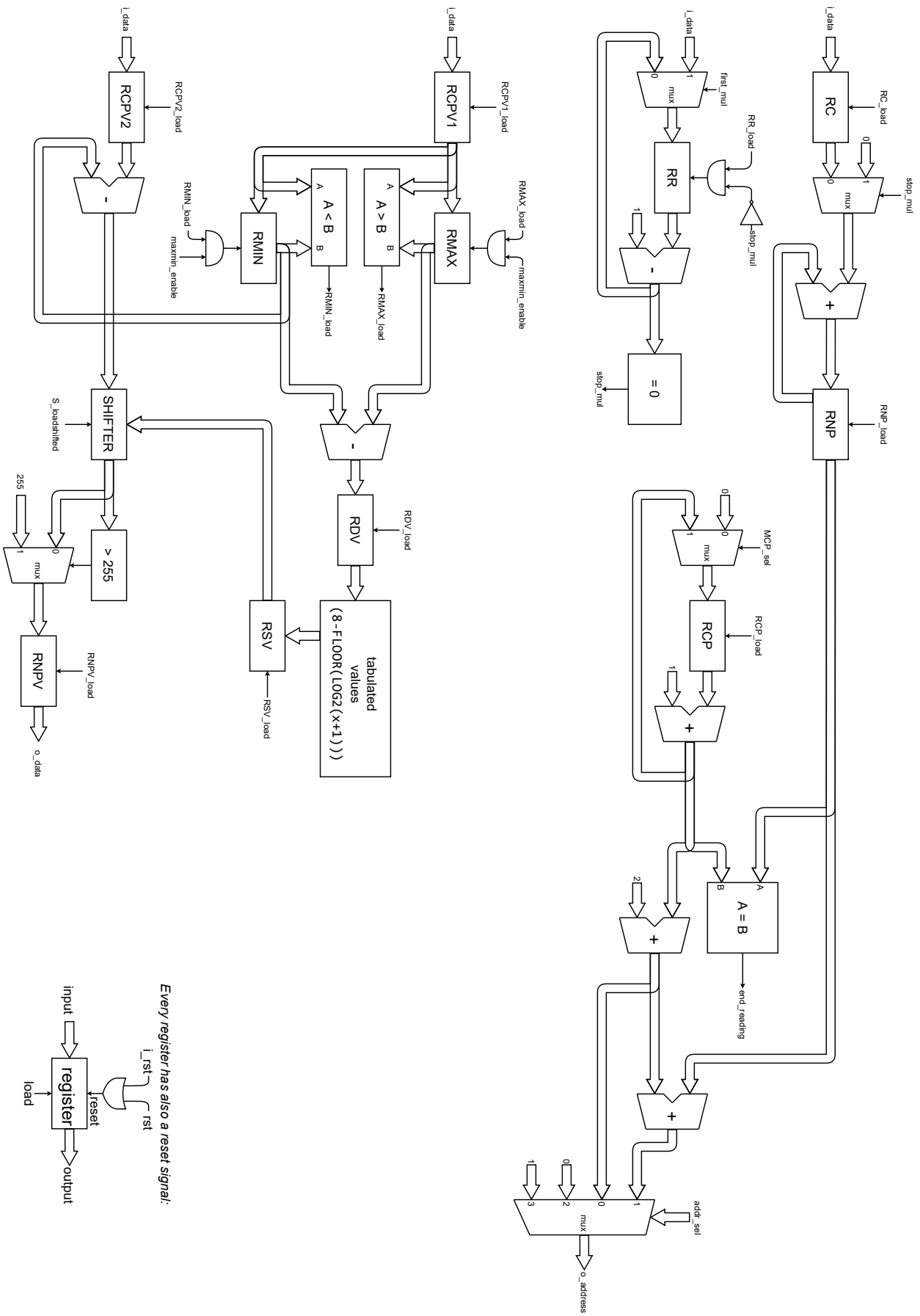
La prima parte del data path calcola il numero di pixel da cui è composta l'immagine. In RC verrà letto il numero di colonne e in RR il numero di righe, RNP partirà da zero. La computazione procede iterando la somma di RC a RNP e la sottrazione di 1 a RR finché RR conterrà il valore 1, a quel punto la moltiplicazione termina e in RNP sarà contenuto il numero totale di pixel dell'immagine. Per eseguire la moltiplicazione si è volutamente evitato l'utilizzo dell'operatore * di VHDL per non forzare il tool di sintesi ad eseguirla all'interno di un solo ciclo di clock (rischiando di non entrare più nel limite richiesto di 100 ns). Non essendoci limiti di tempo, o di cicli di clock, richiesti per l'esecuzione si è scelto di non utilizzare algoritmi di calcolo più sofisticati, anche se più efficienti (come quello di Booth), per non complicare troppo il data path.



La parte successiva serve per il calcolo degli indirizzi. Il segnale addr_sel controlla un multiplexer che può selezionare in uscita su o_address il valore 0 per la lettura del numero di colonne, il valore 1 per la lettura del numero di righe e altri due valori: uno per la lettura dei pixel dell'immagine da elaborare e uno per la scrittura dell'immagine equalizzata. RCP parte da zero e venendo incrementato man mano di 1 permette di generare gli indirizzi da 2 a RNP+1 necessari per la lettura all'ingresso 0 del mux, e gli indirizzi da RNP+2 a 2×RNP+1 necessari per la scrittura all'ingresso 1 del mux.



Infine, l'ultima parte del data path effettua i calcoli veri e propri. Ad una prima iterazione la parte superiore calcola il minimo, il massimo e il valore di shift da usare. Successivamente ad un'altra iterazione la parte inferiore calcola il nuovo valore per ogni pixel e lo salva in memoria.


i_data
RC
RC_load
stop_mul
mux
+
RNP
RNP_load
RR
RR_load
first_mul
-
= 0
stop_mul
MCP_sel
RCP
RCP_load
A = B
end_reading
addr_sel
o_address
RCPV1
RCPV1_load
RCPV2
RCPV2_load
RMIN
RMAX
RMIN_load
RMAX_load
maxmin_enable
A < B
A > B
SHIFTER
S_loadshifted
255
> 255
RNPV
RNPV_load
o_data
RDV
RDV_load
RSV
RSV_load
tabulated values
(8-FLOOR(LOG2(x+1)))
Every register has also a reset signal:
input
load
register
i_rst
rst
reset
output

2.2 Unità di Controllo

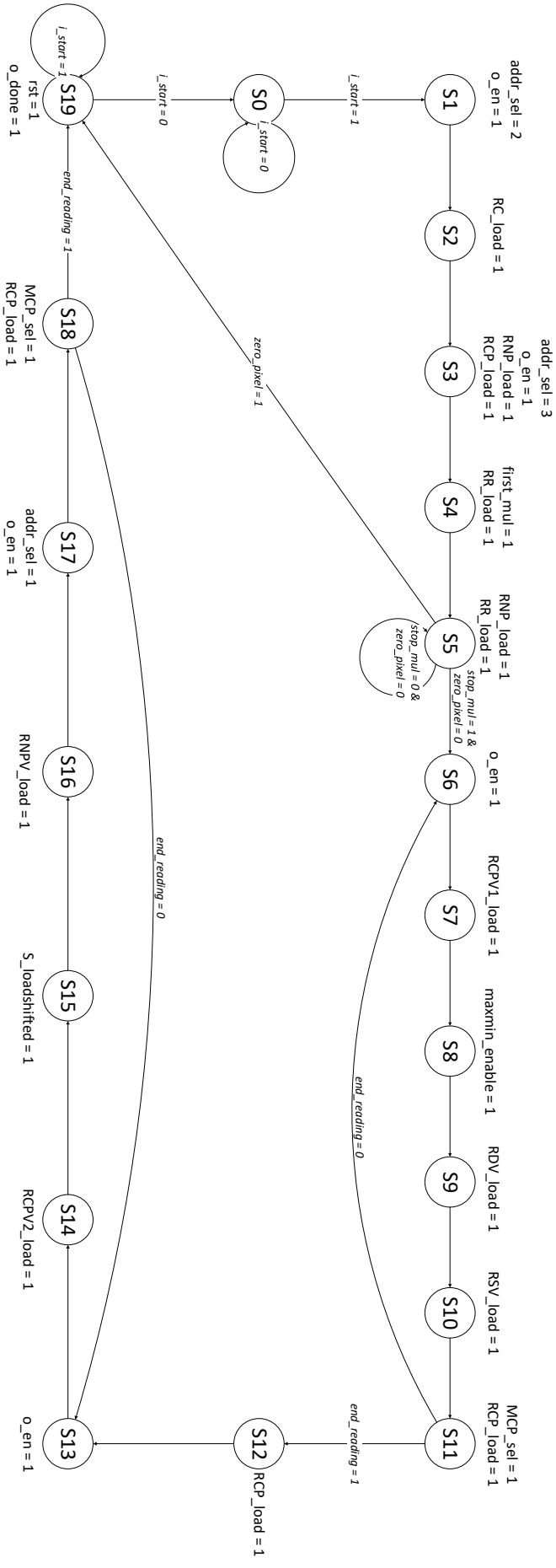
L'unità di controllo gestisce i segnali da e verso l'esterno e i segnali di controllo del data path per garantirne il comportamento corretto. È modellizzata come una macchina a stati finiti di Moore a 20 stati.

Gli stati della macchina sono:

- S0: è lo stato iniziale in cui la macchina attende il segnale di start;
- S1: la macchina richiede alla memoria il numero di colonne;
- S2: la macchina legge il numero di colonne dalla memoria;
- S3: la macchina richiede alla memoria il numero di righe;
- S4: la macchina legge il numero di righe dalla memoria;
- S5: la macchina resta in questo stato finché il calcolo del numero totale di pixel viene completato (se il numero di pixel è zero la macchina si porta direttamente allo stato finale S19);
- S6: la macchina chiede alla memoria il valore del primo pixel o del pixel successivo;
- S7: la macchina legge dalla memoria il valore del pixel richiesto;
- S8: la macchina attende l'eventuale aggiornamento del valore minimo o massimo;
- S9: la macchina aggiorna il delta value;
- S10: la macchina aggiorna il valore di shift;
- S11: la macchina controlla se è stato letto l'ultimo pixel e in tal caso termina l'iterazione altrimenti torna allo stato S6;
- S12: la macchina si prepara per una nuova iterazione;
- S13: la macchina chiede alla memoria il valore del primo pixel o del pixel successivo;
- S14: la macchina legge il valore del pixel richiesto;
- S15: la macchina inizia la trasformazione del pixel salvando il valore sottratto del minimo e shiftato;
- S16: la macchina prepara il valore da scrivere in memoria;
- S17: la macchina scrive il nuovo valore del pixel in memoria;
- S18: la macchina controlla se è stato scritto l'ultimo pixel ed in tal caso termina l'iterazione altrimenti torna allo stato S13;
- S19: la macchina resetta i registri, comunica di aver finito l'elaborazione e attende che il segnale di start torni a 0 prima di tornare allo stato iniziale S0.

Anche qui, per chiarezza espositiva, non ci si è preoccupati di ridurre il numero di stati della macchina. Eventualmente sarebbe sicuramente possibile eseguire già la prima lettura dei pixel calcolando lo shift necessario per l'equalizzazione già durante la prima fase in cui viene calcolata la moltiplicazione righe per colonne. Probabilmente utilizzando alcuni algoritmi o euristiche apposite sarebbe possibile ridurlo ulteriormente.

La macchina a stati è stata descritta in VHDL seguendo un approccio interamente comportamentale come una collezione di 3 process. Il primo assegna semplicemente il valore dello stato corrente ad ogni ciclo di clock, il secondo calcola lo stato successivo e il terzo assegna, in base allo stato corrente, tutti i segnali necessari.



In ogni stato sono indicati solo i segnali che non valgono zero.

Se i_rst = 1 la macchina si porta nello stato S0 indipendentemente dallo stato in cui si trovi attualmente

3 Risultati Sperimentali

3.1 Sintesi

L'operazione di sintesi è stata effettuata, utilizzando il software Vivado, considerando come FPGA target il dispositivo xc7a200tfbg484-1 della Xilinx.

Di seguito sono riportati i componenti della FPGA utilizzati.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	170	0	134600	0.13
LUT as Logic	170	0	134600	0.13
LUT as Memory	0	0	46200	0.00
Slice Registers	136	0	269200	0.05
Register as Flip Flop	136	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 ritardo massimo del data path risulta essere di 4.531 ns con uno slack di 95.318 ns rispetto al periodo di clock richiesto di massimo 100 ns.

3.2 Test

Per verificare il corretto comportamento del componente sono stati eseguiti diversi test simulandolo in pre-sintesi e post-sintesi utilizzando il software Vivado. Con i test si è cercato di coprire la maggior parte dei cammini che il componente potrebbe intraprendere durante la computazione e si è verificato il comportamento nei casi limite. Nella simulazione comportamentale pre-sintesi oltre al controllo del contenuto finale in memoria è stato controllato anche il corretto comportamento dei segnali interni ed esterni generati dal componente durante tutta l'elaborazione.

Test casuali

Prima di tutto sono stati eseguiti numerosi test casuali per mettere in luce eventuali criticità della macchina nel suo comportamento solito senza considerare casi limite. Sono poi stati sottoposti test con immagini a pixel tutti uguali o seguendo pattern specifici.

Test a dimensione massima

Il componente è stato poi sottoposto ad alcuni test alla dimensione massima di 128 × 128 pixel.

Test a dimensioni minime

Sono stati eseguiti poi due test con immagini ad un solo pixel e alcuni test con numero di colonne o numero di righe a 0 per verificare che il percorso che porta la macchina a stati direttamente all'ultimo stato si attivasse correttamente.

Test reset

Il componente è stato sottoposto a due test che verificassero il corretto comportamento del segnale di reset utilizzato sia in modo sincrono che asincrono.

Test con più computazioni

Infine, le tipologie di test precedenti sono state ripetute in altri test presentandole al componente in successione senza resettarlo tra una computazione e l'altra per verificare che il reset dei registri e il ritorno allo stato iniziale alla fine di un'elaborazione avvenissero correttamente.

4 Conclusioni

Il componente ha superato tutti i test che gli sono stati sottoposti sia in *Behavioral Simulation* che in *Post-Synthesis Functional Simulation*. Il componente, come previsto, non utilizza che una piccola percentuale dell'intera FPGA e il ritardo massimo del data path rientra largamente all'interno del margine di clock richiesto.