Progetto Reti Logiche A.A 20/21

Docente: William Fornaciari
Progetto Di:
- Matteo Borghetti: **10584179**
- Riccardo Bargauan: **10621473**

1 Introduzione

1.1 Requisiti progettuali

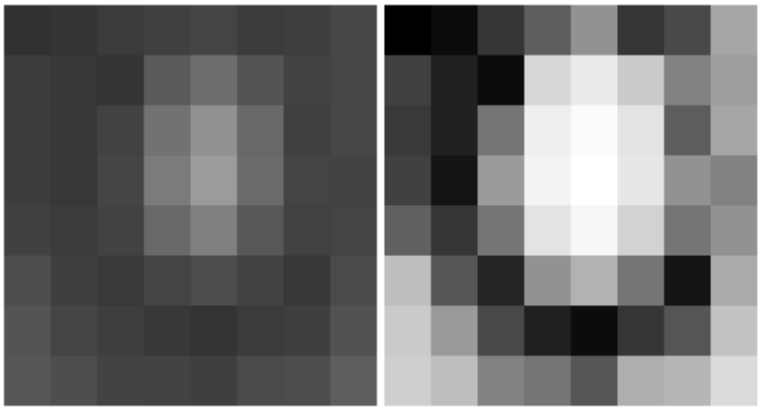
Il progetto richiede l'implementazione in vhdl dell'algoritmo semplificato di equalizzazione di un' immagine in scala di grigi a 256 (8 bit) livelli.
L'equalizzazione è un metodo di elaborazione digitale delle immagini con cui si può calibrare il contrasto di un'immagine usando l'istogramma della stessa.
L'algoritmo da implementare deve eseguire i seguenti passaggi:

1. Data un'immagine, computare la dimensione dell'immagine e salvarla in un registro di 16 bit.
2. Scorrere tutti i bit che compongono l'immagine e calcolare il massimo e il minimo valore della scala di grigi.
3. Una volta calcolato il massimo e il minimo si calcola il **shift level** (logaritmo della differenza tra il massimo valore e minimo valore dell' immagine).
4. Per ogni pixel calcolare il nuovo valore equalizzato.
5. Salvare sulla memoria in nuovo pixel.

Inoltre, l'implementazione deve essere in grado di gestire un segnale di Reset.
Per l'implementazione si è scelto di supporre il Reset asincrono rispetto al segnale di clock. ?
L'implementazione deve essere poi sintetizzata con target FPGA xc7a200tfbg484-1.

1.2 Esempio

Un esempio del processo di equalizzazione è facilmente rappresentabile dalle due immagini sotto riportate.



fonte: Wikipedia

Si riporta in seguito un esempio numerico rappresentando un' immagine 2x2 come una matrice di pixel:
Data un'immagine 2x2 composta da 4 pixels (pixel = 8 bit)

12	24
2	124

l'algoritmo calcola il massimo e il minimo che, in questo caso, sono rispettivamente **2** e **124.**
In seguito viene calcolato lo **shift level** che rappresenta quante volte ogni pixel dovrà subire la procedura di shift a sinistra.
La formula del calcolo dello **shift-level** è la seguente:
$SHIFT\_LEVEL = (8 - FLOOR(LOG2(DELTA\_VALUE + 1)))$
In questo esempio lo **shift level** vale **6**, ogni pixel dell'immagine viene aggiornato tramite la seguente formula:
$(CURRENT\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE) << SHIFT\_LEVEL$
Nel nostro esempio la tabella diventa la seguente

768	1536
128	7936

Poiché i valori sopra il 255 non sono ammessi, l'ultimo passaggio dell'algoritmo associa al nuovo valore del pixel il minimo tra 255 e il valore calcolato precedentemente generando la seguente (e equalizzata) tabella:

255	255
128	255

2 Architettura

2.1 Scelte Progettuali

L’intero componente è stato implementato attraverso la definizione di un datapath , di una macchina a stati finiti (FSM) e da un modulo per il calcolo degli indirizzi di lettura e scrittura nella memoria RAM, in modo da poter gestire l’algoritmo richiesto suddividendo lo svolgimento in una parte di controllo e in una strettamente elaborativa (contenente l’intera logica combinatoria).
È stato quindi trovato un equilibrio fra le due parti, in maniera tale che non vi fosse un numero eccessivo di stati o che il datapath diventasse troppo complesso.

2.2 Struttura Del *Datapath*

Per risparmiare sul numero di stati della FSM è stata usata le seguente proprietà;

- Qualsiasi segnale è disponibile in lettura dopo un ciclo di clock da quando è stato scritto.

Il datapath contiene tutte le informazioni necessarie per poter attuare le operazioni logiche dell’algoritmo.
Per semplicità si è deciso di suddividere il datapath in 4 “blocchi” logici.
Questa distinzione è stata fatta per modularizzare l'algoritmo e poter testare ciascun blocco indipendentemente dall’implementazione complessiva dell’intero algoritmo.
I blocchi logici individuati sono i seguenti:

1. **Calcolo dimensione immagine.**
2. **Calcolo del massimo e del minimo dei pixel.**
3. **Calcolo del logaritmo e dello shift value.**
4. **Equalizzazione dei pixel.**

Per implementare il Datapath sono stati utilizzati tre processi onde evitare condizioni di “*Multiple driven signal”* .
L'esecuzione dei vari blocchi logici viene implementata dalla FSM poichè il datapath è trasparente ai segnali di controllo.
Per fare ciò si è utilizzato un segnale (*machine_counter*) a 2 bit.
La codifica del *machine_counter* corrisponde ai seguenti blocchi logici:

- **00 :** calcolo della dimensione dell’immagine
- **01 :** calcolo del massimo e del minimo
- **10 :** calcolo del logaritmo e shift value
- **11 :** equalizzazione dei pixel

A livello di codice questa logica è stata implementata tramite l'utilizzo del paradigma **case-when.**
DI seguito vengono riportate le descrizioni dettagliate dei vari blocchi, la logica implementata e i segnali utilizzati per manipolarli.

Calcolo dimensione immagine

Quando la FSM commuta il segnale *summ_dimention_load* a **1**, il valore proveniente dalla memoria tramite *i_data* (il bus che mette in comunicazione la memoria e il datapath) viene salvato nel registro *sdm_reg*.
Se la prima dimensione dell'immagine è **'00000000',** il datapath commuta il segnale *end_calc_temp* che indica alla FSM di riportarsi nello stato di inizio e resettare tutti i segnali e portarsi nello stato di INIT.

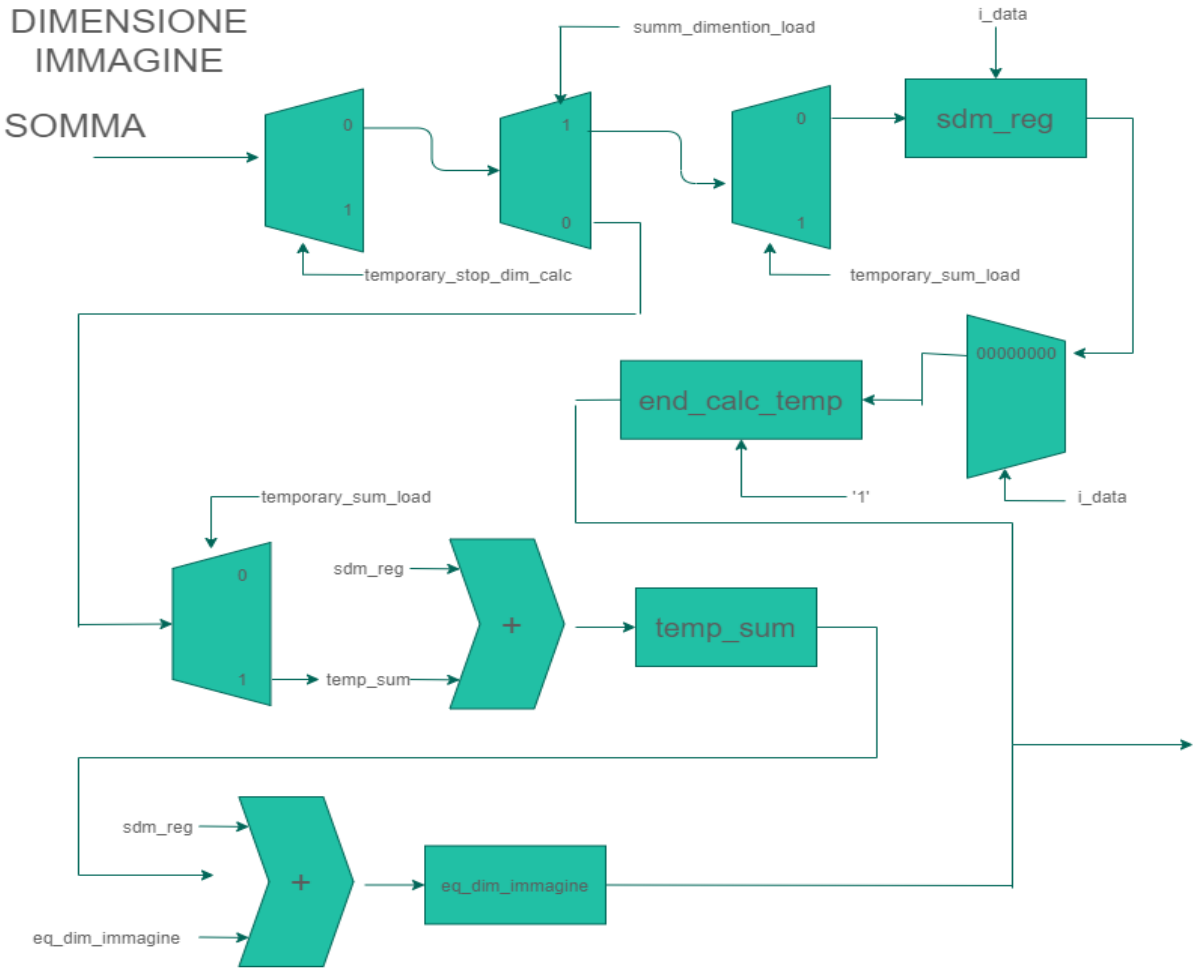
Nel processo che gestisce la sottrazione a fronte della commutazione del segnale *dec_load* a **'1'** viene salvato il valore proveniente da *i_data* nel registro temporaneo *dec_reg*.
Come per la somma , se la dimensione letta equivale a **'00000000'** il segnale *end_calc_temp_2* commuta a **'1'** segnalando alla FSM di resettare tutti i segnali e portarsi nello stato di INIT.

A seguito della fase di load, inizia la fase del computo della somma delle due dimensioni (come descritta precedentemente).
Tramite il segnale *temporary_sum_load* si calcola la dimensione dell'immagine sommando al registro *temp_sum* il suo valore precedente calcolato e il valore contenuto nel registro *sdm_reg*.
Parallelamente a fronte del segnale *decrementor_mux_selector* con valore **'1'** viene effettuata la sottrazione salvando ogni volta il risultato nel registro *dec_reg.*
Questo processo viene ripetuto finché il registro *dec_reg* non ha valore di **"00000001"**, a questo punto viene segnalato alla FSM la fine della computazione della dimensione dell'immagine tramite il segnale *temporary_stop_dim_calc*.
Per risparmiare uno stato della FSM, durante il calcolo della dimensione si è deciso di far terminare la computazione della sottrazione quando *dec_reg* vale **"00000001"**.
Così facendo, nel periodo che intercorre tra la scrittura del segnale di *temporary_stop_dim_calc* e la commutazione del segnale *machine_counter da* **"00" a "01",** il processo che esegue la somma viene effettuato il numero corretto di volte.
Così facendo si è risparmiato un ulteriore stato di "Buffer" nella FSM .

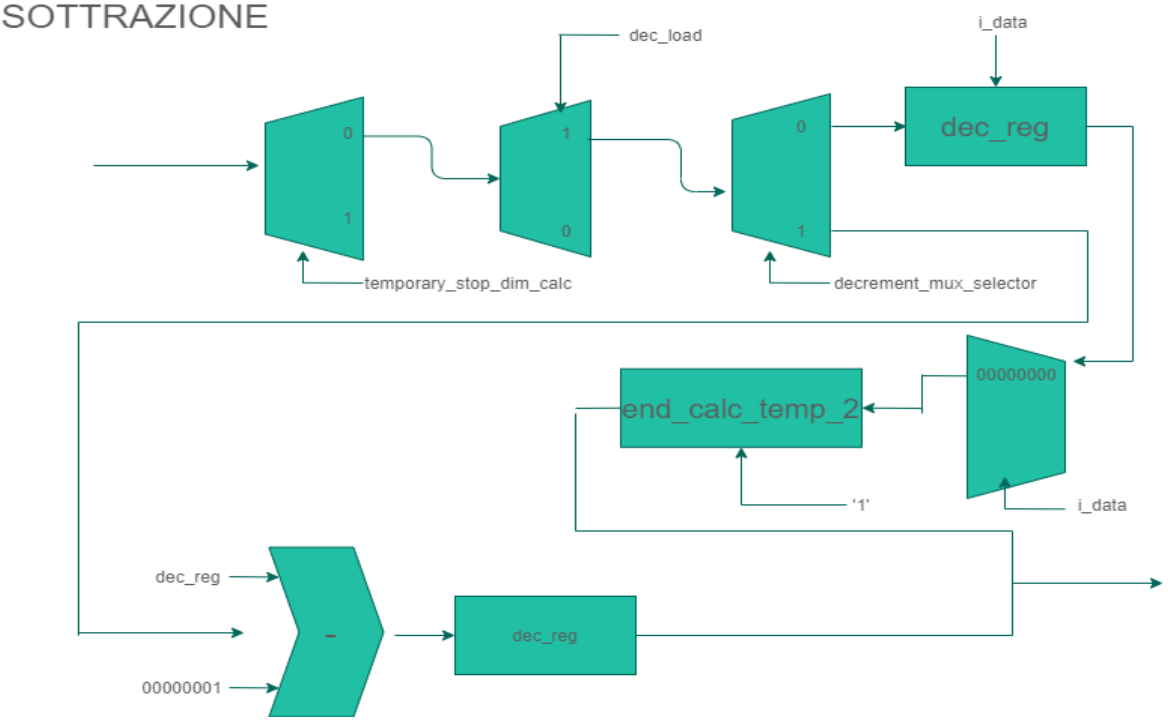
Vedi schema "Calcolo dimensione immagine" a pag. 5

CALCOLO DIMENSIONE IMMAGINE

SOMMA



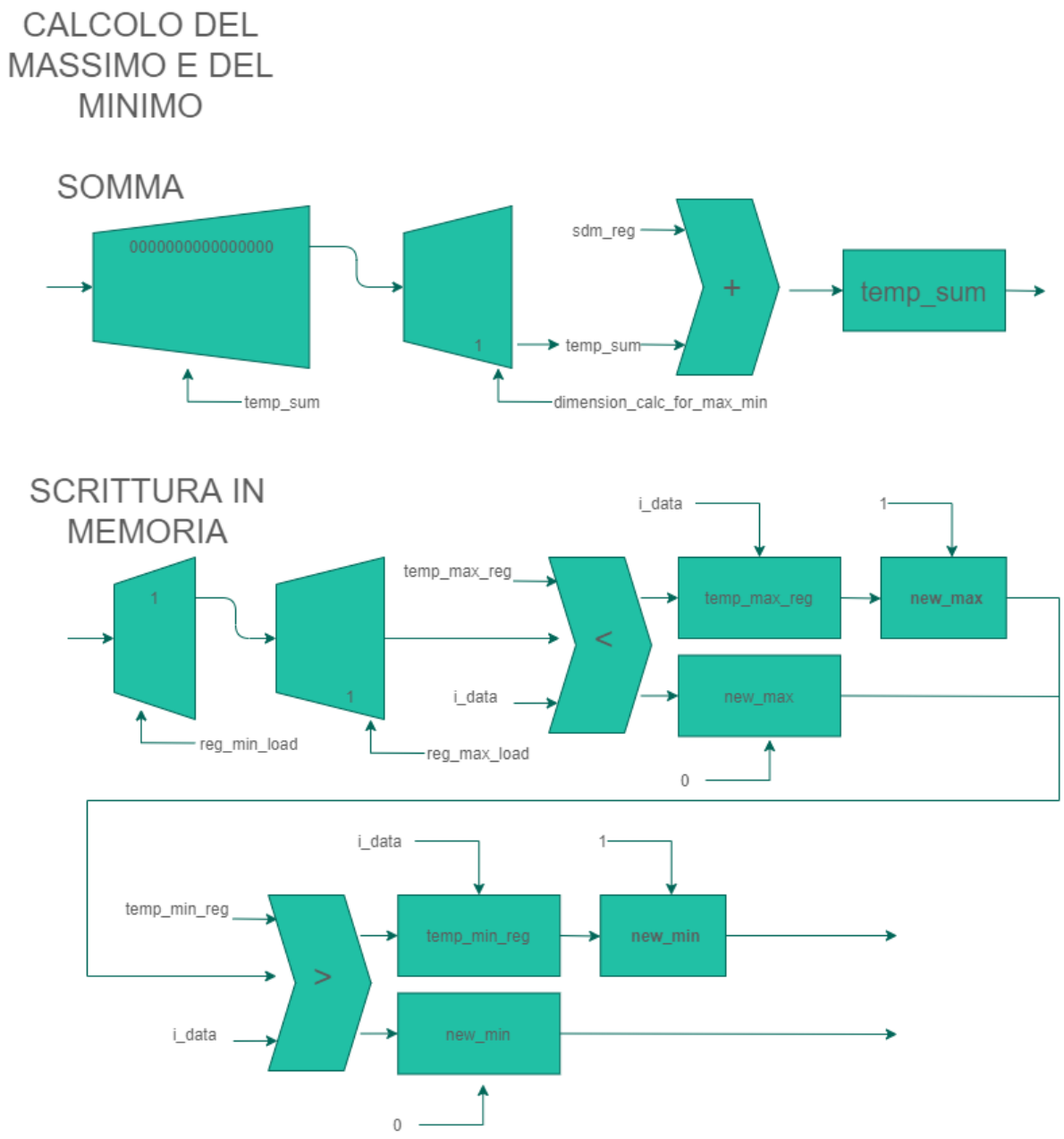
SOTTRAZIONE



Calcolo del Massimo e del Minimo

Durante il processo di calcolo del massimo e del minimo vengono eseguiti parallelamente il decremento del registro *temp_sum (che contiene la dimensione dell'immagine)* e il calcolo del massimo e del minimo.
Nello stato DEC_IMAGE_DIMENSION viene posto a **'1'** il segnale *dimension_calc_for_max_min* , tale segnale decrementa di **'1'** il valore di *temp_sum* , in seguito quando i segnali *reg_min_load* e *reg_max_load* commutano a **'1'** viene effettuato il confronto con il dato presente su *i_data*.
Per far si che il segnale *i_data* contenga il valore corretto si è deciso di affidare il calcolo degli indirizzi di lettura al modulo *address_adder*.
Se il valore contenuto su *i_data* è maggiore del valore presente su *temp_max_reg* (che ha come valore di default **"00000000"**) allora il valore di *i_data* diventa il nuovo massimo.
Se viene trovato un nuovo massimo, il segnale *new_max* commuta da **'0'** a **'1'** indicando alla FSM la necessità di passare per lo stato di REGISTER_NEW_MAX_MIN per poter salvare correttamente nel registro *temp_max_reg* il corretto valore di *i_data*.
Per il calcolo del minimo viene utilizzato lo stesso procedimento del calcolo del massimo ma al posto di far commutare *new_max* viene fatto commutare *new_min* e la logica per la scrittura dei dati rimane la stessa precedentemente descritta.
La fase di computazione del massimo e del minimo termina quando il valore del registro *temp_sum* vale **"00000000"**, a fronte di questo evento il segnale *stop_max_min* commuta da **'0'** a **'1'** segnalando alla FSM di terminare il calcolo degli indirizzi di lettura e modifica del valore del registro *machine_counter* da **"01"** a **"10"**.

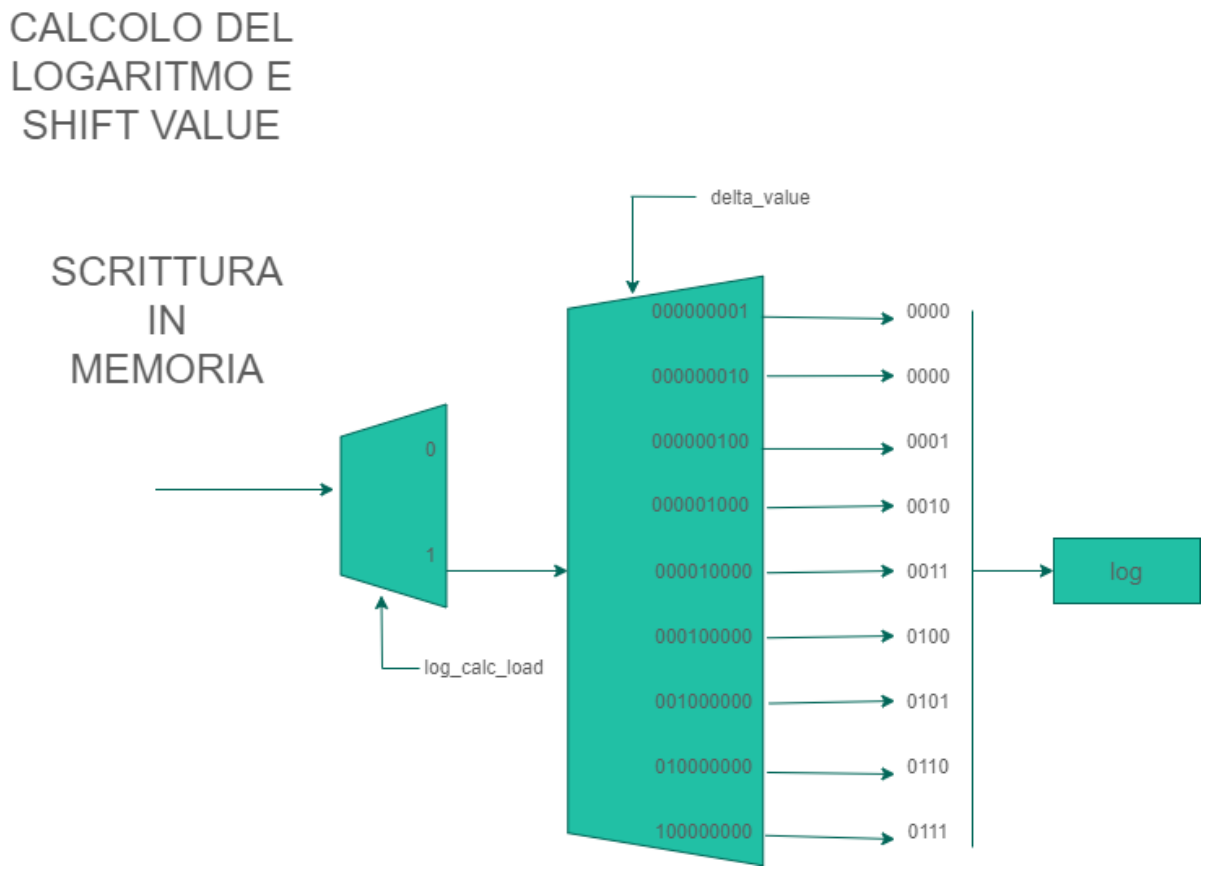
Vedi schema "Calcolo del Massimo e del Minimo" a pag. 7



Calcolo del logaritmo e valore Shift Value

Quando il segnale *delta_value_load* commuta *da* **0** a **1,** nel registro *delta_value* viene salvato l'esito della sottrazione tra il massimo e il minimo valore dei pixel + **"00000001"** (come da specifica).
In seguito quando il segnale *log_calc_load* commuta a **1,** tramite una serie di **if-elsif-else,** viene confrontato il valore presente nel registro *delta_value* con i valori tabulati delle potenze di 2 da 0 a 8 (poiché il valore di delta value può essere al massimo di 256(quindi 8 bit).
A termine del controllo il registro *log* contiene il valore del logaritmo a base 2 del valore di *delta_value.*

Vedi schema "Calcolo del logaritmo e valore Shift Value" a pag. 8

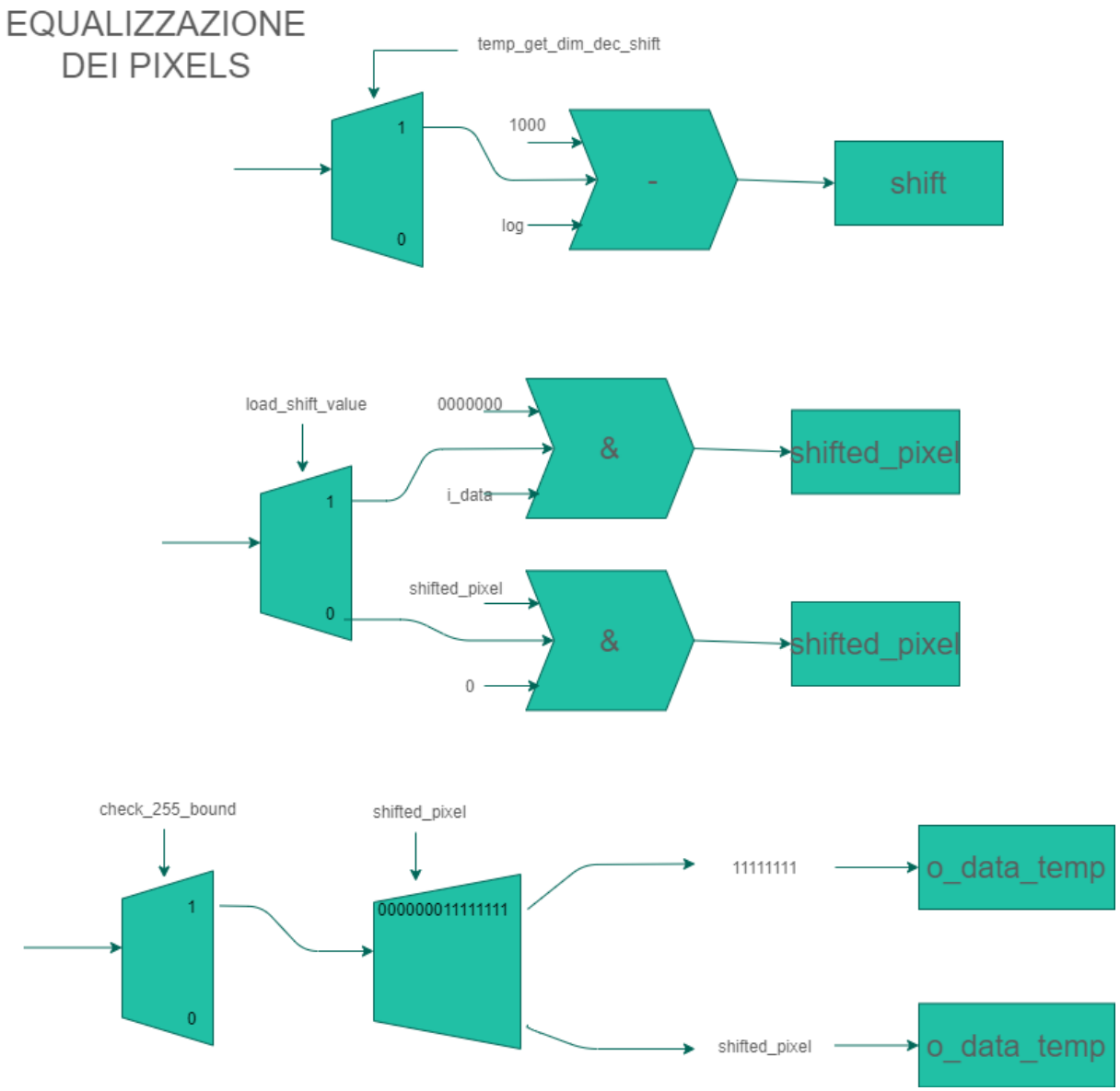


Equalizzazione pixel

L'ultimo passo dell'algoritmo consiste nell equalizzazione dei pixel ovvero nell' effettuare l'operazione di "shift a sinistra" tante volte quanto il valore di *log.*
Per prima cosa il segnale *load_shift_value* viene posto a **1** e il valore proveniente da *i_data* viene salvato nel registro *shifted_pixel (*che ha dimensione 15 bit per poter essere shiftato al più 7 volte).
In seguito quando il segnale *evaluate_equalized_pixel* commuta a **1** il registro *shifted_pixel* viene sovrascritto con la differenza tra il valore precedentemente salvato sullo stesso e il valore contenuto in *temp_min_reg* (come richiesto da specifica).
Quando il segnale *ready_shift* vale **1** e *load_shift_value* vale **0** viene effettuata una volta l'operazione di shift a sinistra concatenando gli ultimi 14 bit di *shifted_pixel* e **0**, salvando il valore ottenuto nel registro *shifted_pixel.*
Parallelamente per tenere traccia di quante volte effettuare l'operazione di shift su un dato pixel viene decrementato il registro *shift* di **"0001"** a seguito della commutazione del segnale *start_dec_for_shift* a **1**. I due segnali *start_dec_for_shift* e *evaluate_equalized_pixel* vengono fatti commutare nello stesso stato della FSM per poter garantire la sincronizzazione tra l'operazione di shift e il decremento del registro *shift (*che contiene il numero rimanente di operazioni da eseguire*)*.
A seguito della corretta equalizzazione del pixel, il datapath segnala il termine dell'operazione con il segnale *dec_shift* che porta la FSM nello stato CHECK_255 , tramite il segnale *check_255_bound* viene scritto su un registro temporaneo *o_data_temp* oppuire

“11111111”, se il valore di *shifted pixel* è superiore a 255 o altrimenti gli 8 bit meno significativi del registro *shifted_pixel*.
Per scrivere il pixel equalizzato sul bus che si interfaccia tra la memoria e il datapath viene impostato il segnale *write_to_memory_pixel* a **1.**
L’operazione di equalizzazione viene ripetuta tante volte quanti sono i pixel che compongono l’immagine, per tenere traccia di questo si è utilizzato un registro temporaneo *eq_dim_immagine,* che contiene la dimensione dell’immagine da equalizzare e con il segnale *do_dec_on_eq_img* viene decrementato il valore del registro di **“0000000000000001”**.

Vedi schema “Equalizzazione pixel” sotto riportato.



2.3 Struttura del Address Adder

Questo componente è stato scritto per calcolare gli indirizzi di lettura e scrittura della memoria RAM.
La scelta di implementare queste funzioni in un componente separato dipende dal fatto che la logica per il calcolo degli indirizzi avrebbe reso troppo complessa la struttura del datapath.

L'interfaccia del componente è la seguente:

- **i_clk:** Segnale di clock per sincronizzare il componente con il datapath
- **compute_read_address:** Richiesta di un indirizzo di memoria
- **compute_write_address:**Richiesta di un indirizzo di memoria
- **read_address_out:** Bus di output che si interfaccia con gli altri componenti contenente il valori di **read_address**.
- **write_address_out:**Bus di output che si interfaccia con gli altri componenti contenente il valori di **write_address**.
- **reset_addres_to_known_value:**Quando il segnale di reset vale **1** vengono resettati i valori dei registri temporanei del componente
- **full_address:**Carica nel registro *read_address_reg* il valore corrente dell'indirizzo di memoria.
- **half_address:**Carica nel registro *write_address_reg* il valore corrente dell'indirizzo di memoria.
- **reset_full_address:** aggiorna i valori temporanei dei registri
- **reset_half_address:** aggiorna i valori temporanei dei registri

Si è scelto di sfruttare il parallelismo tra processi, modellando la sequenza di segnali e la logica come segue:
richiesta indirizzo -> scrittura del valore sul bus di output -> computo del indirizzo di memoria successivo.

Invece di implementare il classico paradigma :
richiesta indirizzo -> computazione indirizzo -> scrittura sul bus di output.

Così facendo si è ridotto il numero di stati della FSM poiché la richiesta di un indirizzo necessita soltanto uno stato mentre il calcolo del successivo è distribuito negli altri stati.
In base alla specifica abbiamo utilizzato due vettori logici di 15 bit per poter gestire il calcolo degli indirizzi di memoria: *read_address_out* e *write_address_out*.
-*read_address_out* contiene gli indirizzi di memoria che vanno da 2 a 2 volte la dimensione dell'immagine e viene usato in fase di calcolo del massimo e del minimo e anche per tenere traccia dell'indirizzo su cui scrivere in fase di equalizzazione.
-*write_address_out* è utilizzato per la rilettura dell'immagine e va quindi da 2 alla dimensione dell'immagine e serve in fase di equalizzazione per gestire gli indirizzi dei pixel da equalizzare.

Per implementare la logica descritta sopra per calcolare *read_address_out* , si sono utilizzati 3 registri:

- **temp_read:** Contiene il valore dell'indirizzo corrente e viene aggiornato all'indirizzo successivo con il segnale di *reset_full_address*.
- **read_address_reg:** Contiene il valore corrente dell'indirizzo di memoria e viene usato per la computazione dell'indirizzo successivo.
- **read_address_reg_next:** Contiene l'indirizzo successivo rispetto a quello presente in *read_address_reg*.

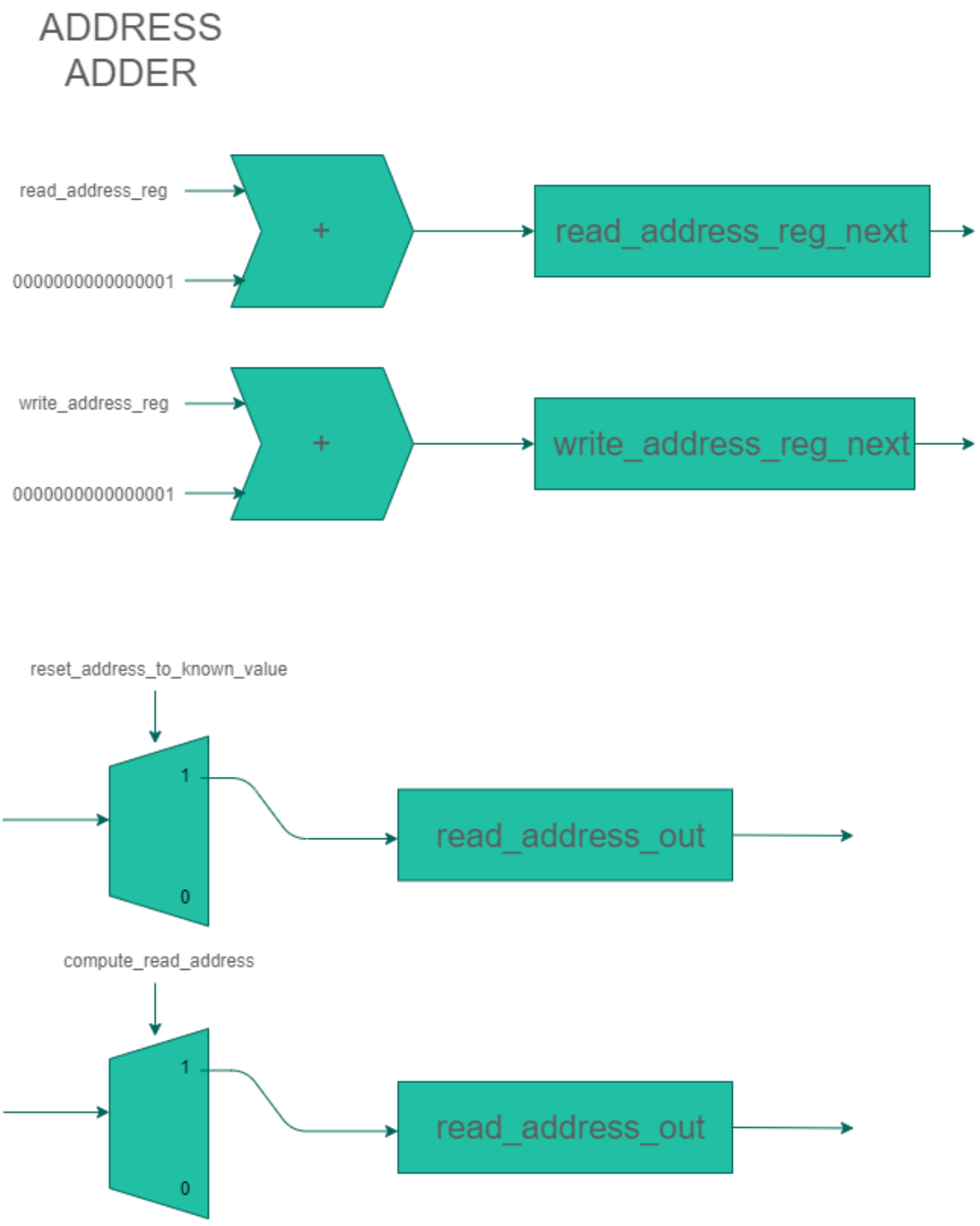
Quando viene richiesto un indirizzo di memoria la FSM commuta a **1** i segnali di *compute_read_address* e *reset_full_address*.
Con il segnale *compute_read_address* viene scritto sul bus *read_address_out* il valore contenuto in *temp_read*, con il segnale *reset_full_address* viene sovrascritto il valore di *temp_read* con il valore contenuto in *read_address_reg_next.*
Dopo un ciclo di clock per asserire i valori sui vari registri, la FSM commuta il segnale *full_address* a **1** e il registro *read_address_reg* viene sovrascritto con il valore contenuto in *temp_read.*

Per calcolare *write_address_out* si sono utilizzati i seguenti registri:

- **temp_write:** Contiene il valore dell'indirizzo corrente e viene aggiornato all'indirizzo successivo con il segnale di *reset_full_address*.
- **write_address_reg:** Contiene il valore corrente dell'indirizzo di memoria e viene usato per la computazione dell'indirizzo successivo.
- **write_address_reg_next:** Contiene l'indirizzo successivo rispetto a quello presente in *write_address_reg*.

Poiché la logica combinatoria necessaria per computare gli indirizzi di rilettura dell'immagine è la stessa sopra descritta, pertanto si è deciso di omettere la spiegazione del funzionamento di quest'ultima.

Vedi schema "Address adder" sotto riportato.



2.4 Macchina a stati finiti

Questo processo gestisce il progredire dell algoritmo quando a fronte di un segnale di clock vengono aggiornati gli stati.
Sono state omesse le frecce che da ogni stato ritornano allo stato iniziale a seguito della commutazione del segnale di reset e anche le frecce che collegano gli stati di HALT con lo stato di INIT_MACHINE.

Nel diagramma della FSM lo stato di HALT è stato ripetuto più volte poichè utilizzarne uno solo avrebbe reso molto disordinato il diagramma.

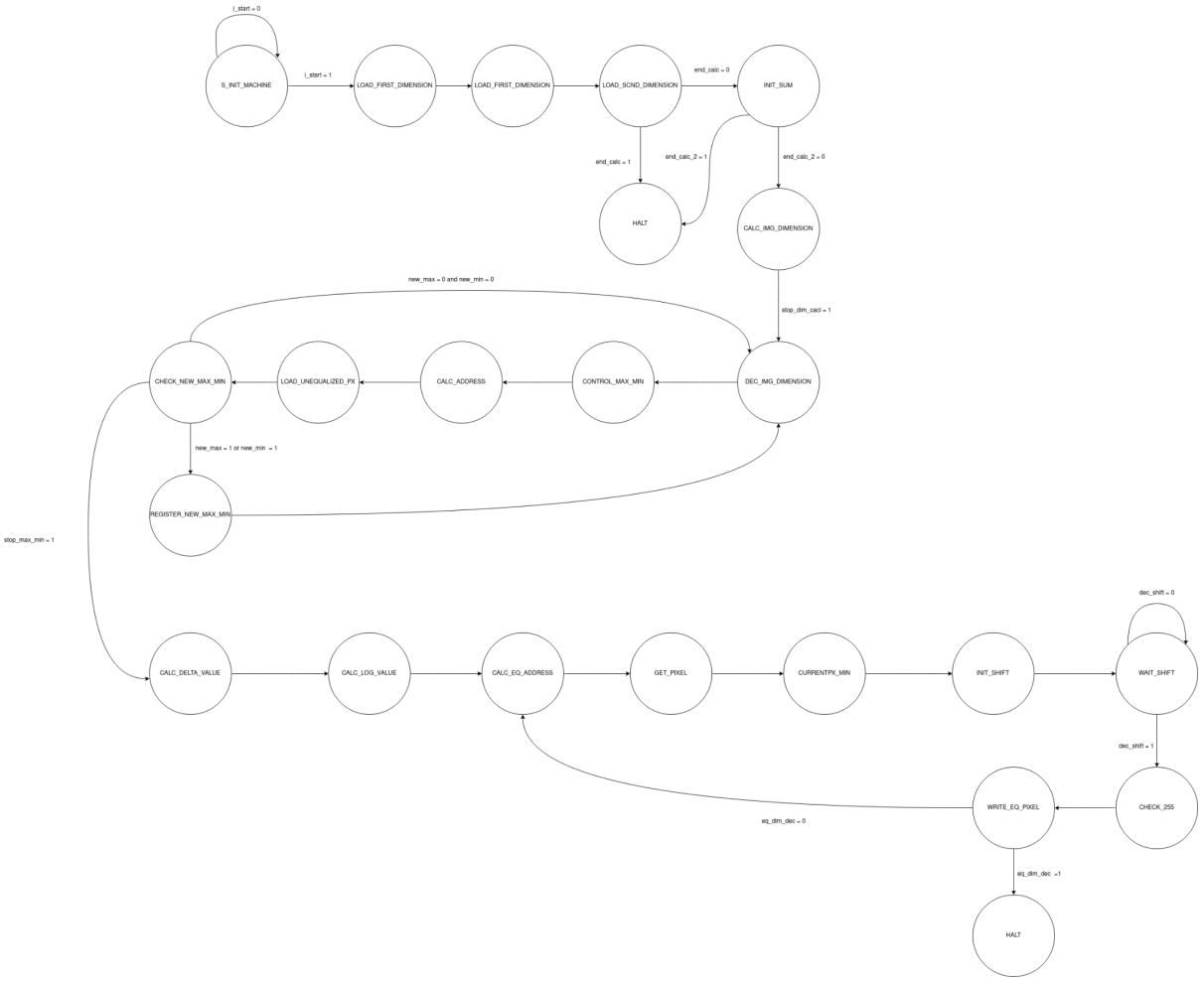
Segue la descrizione degli stati formali della FSM:

- **INIT_MACHINE:** Lo stato in cui la macchina si trova appena viene eseguito l'algoritmo.
  Continua a rimanere in questo finché non viene sollevato il segnale di di start.
- **LOAD_ADDRESS:** Viene caricato nel registro *o_address* il primo indirizzo della memoria RAM per poter leggere la prima dimensione dell'immagine**.**
- **LOAD_FIRST_DIMENSION:** Viene salvata la prima dimensione dell'immagine nel registro e in *o_address* viene caricato il secondo indirizzo della memoria RAM che contiene la seconda dimensione dell'immagine.
- **LOAD_SECOND_DIMENSION:** Viene caricata nel registro la seconda dimensione. Dopo che la RAM ha asserito i valori della prima dimensione se quest'ultima vale "**00000000"** lo stato successivo sarà quello di **HALT.**
- **INIT_SUM:** Stato in cui inizia la computazione della somma delle due dimensioni precedentemente caricate.
  Dopo che la RAM ha asserito i valori della seconda dimensione, se quest'ultima vale **"00000000"** lo stato successivo sarà quello di **HALT.**
- **CALC_IMAGE_DIMENSION:** Stato in cui la macchina rimane finchè non finisce la computazione della dimensione dell'immagine.
- **DEC_IMG_DIMENSION:** In questo stato la macchina decrementa di 1 la dimensione calcolata negli stati precedenti preparando la lettura dei pixel salvati nella memoria RAM.
- **CALC_ADDRESS:** In questo stato viene richiesto al componente *address_adder* l'indirizzo del primo Byte di memoria che contiene il primo pixel dell'immagine.
- **LOAD_UNEQUALIZED_PIXEL_FOR_MAX_MIN:** In questo stato viene letto e memorizzato il byte contenuto all'indirizzo calcolato in **CALC_ADDRESS.**
- **CHECK_NEW_MAX_MIN:** In questo stato viene controllato se il pixel letto sia un nuovo massimo o minimo.
- **REGISTER_NEW_MAX_MIN:** Se il pixel letto è un massimo o minimo allora viene salvato in memoria come nuovo massimo o minimo.
- **CALC_DELTA_VALUE:** In questo stato il segnale *machine_counter* viene aggiornato a **"10"** e viene calcolato il *delta_value.*
- **CALC_LOG_VALUE:** In questo stato, a seguito del calcolo del *delta_value,* viene computato il logaritmo di quest'ultimo.
- **CALC_EQUALIZATION_ADDRESS:** In questo stato il segnale *machine_counter* viene aggiornato a **"11"** e viene calcolato tramite il componente *address_adder* sia l'indirizzo di scrittura del pixel equalizzato che l'indirizzo del pixel da equalizzare.
- **GET_PIXEL:** In questo stato viene letto il pixel da equalizzare e salvato in memoria.
- **CURRENTPX_MIN:** Il pixel letto dalla memoria viene espanso a 14 bit per poter poi essere shiftato.
- **INIT_SHIFT:** Stato di wait per aspettare che la RAM asserisca il valore del pixel da equalizzare.
- **WAIT_SHIFT:** La macchina rimane in questo stato finché la procedura di shift non termina (ossia finchè dec_shift non commuta da **0** a **1**).
- **CHECK_255:** In questo stato viene effettuato il confronto tra il valore shiftato del pixel e il fondo scala della scala di grigi (255).

- **WRITE_EQUALIZED_PIXEL:** Viene salvato Il valore equalizzato in memoria.

In seguito vengono riportati i segnali utilizzati nel diagramma della FSM e i segnali utilizzati nel componente.
Si è scelto di attribuire degli acronimi agli stati della FSM per risparmiare spazio nel diagramma.

segnali del componente	segnali nel diagramma
S_INIT_MACHINE	S_INIT_MACHINE
LOAD_FIRST_DIMENSION	LOAD_FIRST_DIMENSION
LOAD_SECOND_DIMENSION	LOAD_SCN_DIMENSION
S_INIT_SUM	INIT_SUM
CALC_IMAGE_DIMENSION	CALC_IMG_DIMESION
S_DEC_IMG_DIMESION	DEC_IMG_DIMESION
S_CONTROL_MAX_MIN	CONTROL_MAX_MIN
S_CALC_ADDRESS	CALC_ADDRESS
LOAD_UNEQUALIZED_PIXEL_FOR_MAX_MIN	LOAD_UNEQUALIZED_PX
CHECK_NEW_MAX_MIN	CHECK_NEW_MAX_MIN
REGISTER_NEW_MAX_MIN,	REGISTER_NEW_MAX_MIN,
CALC_DELTA_VALUE	CALC_DELTA_VALUE
CALC_LOG_VALUE	CALC_LOG_VALUE
S_CALC_EQUALIZATION_ADDRESS	CALC_EQ_ADDRESS
S_GET_PIXEL	GET_PIXEL
S_INIT_SHIFT	INIT_SHIFT
S_WAIT_SHIFT	WAIT_SHIFT
S_CHECK_255	CHECK_255
S_WRITE_EQUALIZED_PIXEL	WRITE_EQ_PIXEL
HALT	HALT
S_CURRETPX_MIN	CURRETPX_MIN



3 Risultati Sperimentali

3.1 Sintesi (Report di sintesi)

Dal punto di vista dell'area la sintesi riporta il seguente utilizzo dei componenti

- **LUT:** 230 (0.17% del totale).
- **Flip Flop:** 193 (0.07% del totale)

Si è fatta molta attenzione durante la scrittura del codice per evitare l'utilizzo di inferred latch.

3.2 Simulazioni

Si è riservata una particolare attenzione alle simulazioni fornite dalla specifica di progetto e anche a variazioni per testare determinati valori di shift e per controllare che il componente si comportasse come desiderato.
Sono stati creati testbench per testare ogni possibile valore di shift, per verificare che il componente si comportasse come richiesto e shift era 0 oppure massimo.
Sono stati fatti anche test per verificare casi limite dell'implementazione come il caso in cui una delle due dimensioni sia **"00000000"**, in entrambi i casi il componente reagisce correttamente resettando i segnali interni e ponendosi in condizione di ricevere un nuovo input.

4 Conclusioni

L'architettura funziona correttamente sia in simulazioni post-synthesis functional sia in simulazioni behavioural.
Si è cercato di sfruttare il parallelismo tra processi per rendere l'architettura più efficiente a costo di una più lunga fase di progettazione e si è cercato di dividere il più possibile la logica combinatoria dalla FSM.
Nel percorso di sviluppo e scrittura del codice dell'algoritmo ci siamo imbattuti in problemi che avremmo voluto risolvere utilizzando la logica dei linguaggi di programmazione ad alto livello, ci siamo però impegnati a cercare soluzioni che fossero aderenti ai paradigmi dei linguaggi di descrizione hardware.
La scelta di implementare due moduli (uno che contenesse la logica dell'algoritmo e uno che contenesse il calcolo degli indirizzi) ha reso la scrittura e lo sviluppo molto più semplice e ordinato.