POLITECNICO DI MILANO

Progetto di Reti Logiche

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Ingegneria Informatica

Sezione Prof. Salice

A.A. 2020/2021

Studenti: Sofia Martellozzo, Ilaria Muratori

Matricole: 910488, 911815

Codice Persona: 10623060, 10677812

Indice

Introduzione

La seguente documentazione vuole descrivere in maniera dettagliata il processo volto alla risoluzione del problema proposto in forma di Prova Finale per l'Anno Accademico 2020/2021 dell'insegnamento di Reti Logiche.
La specifica della Prova finale (Progetto di Reti Logiche) 2020 è ispirata al metodo di equalizzazione dell'istogramma di una immagine.
Il metodo di equalizzazione dell'istogramma di una immagine è un metodo studiato per aumentare il contrasto di un' immagine quando l'intervallo dei valori di intensità sono molto vicini effettuandone una distribuzione su tutto l'intervallo di intensità, al fine di incrementare il contrasto.

fig.1 - Esempi di immagine pre e post equalizzazione (sorgente Wikipedia)

Ogni immagine è rappresentata da una successione di pixel, con valori compresi tra 0 e 255 (inclusi) equivalenti ad una scala di grigi: dal bianco al nero. L'algoritmo svolge le seguenti operazioni:

In primo luogo calcola il *delta* e lo *shift_level*

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

Successivamente applica a ciascun pixel un'operazione di shift a sinistra (moltiplicazione per una potenza di due, essendo numeri in base2) di *shift_level* posizioni e lo confronta con il valore massimo assumable dai pixel (255); il minore dei due sarà scritto in output coma valore del pixel equalizzato

TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL
NEW_PIXEL_VALUE = MIN(255 , TEMP_PIXEL)

L' immagine fornita per l'equalizzazione è memorizzata come una sequenza di byte :

- all'indirizzo 0 il byte indica il numero di colonne da cui è composta l'immagine (leggendo il byte più significativo)
- all'indirizzo 1 il byte indica il numero di righe da cui è composta l'immagine (leggendo il byte meno significativo)
- la dimensione massima dell'immagine è di 128X128 pixel

- dall'indirizzo 2 sono memorizzati i byte dell'immagine in sequenza
- ciascun pixel dell'immagine è composto da 8 bit

L'output effettivo, ovvero l'immagine equalizzata viene scritta in memoria subito dopo quella originale (ovvero dall'indirizzo pari alla dimensione immagine + 2) .

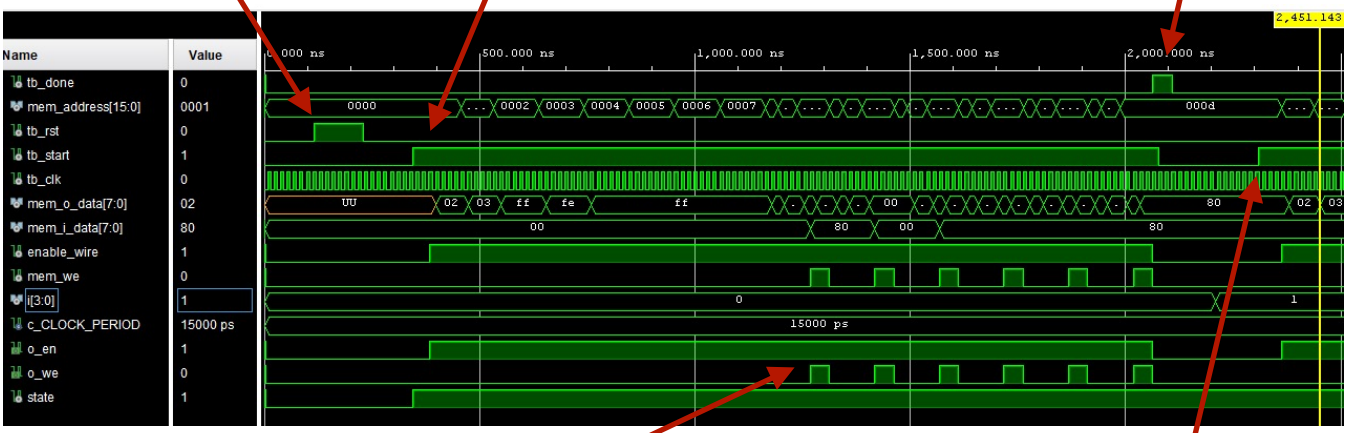
Deve esser possibile elaborare più immagini in sequenza senza che interferiscano tra loro, ovvero non iniziando un'altra elaborazione finché quella attuale non ha notificato la terminazione del processo alzando il segnale di *done.*

Una volta portato il segnale di *start* in ingesso alto, ha inizio l'elaborazione mantenendo tale segnale a valore alto fino a fine processo, quando il segnale di *done* viene portato a valore 1. Sempre quest' ultimo deve mantenere il suo attuale valore finché il segnale di *start* non cambia valore, a zero. Prima di cominciare una nuova elaborazione oltre al alzare il segnale di *start* quello di *done* deve tornare basso; non è necessario un segnale di *reset* a uno per una nuova equalizzazione (soltanto per la prima).

1) Inizio elaborazione: *reset* portato alto

4) fine elaborazione: *done* portato a 1

2) Inizio lettura immagine: *start* alto e *reset* basso



3) segnale di *we* per scrivere in memoria(output)

5) inizio nuova elaborazione, dopo aver riportato *done* a 0 *start* diventa 1 (senza che *reset* diventi di nuovo alto, essendo la seconda immagine)

Esempio generico:

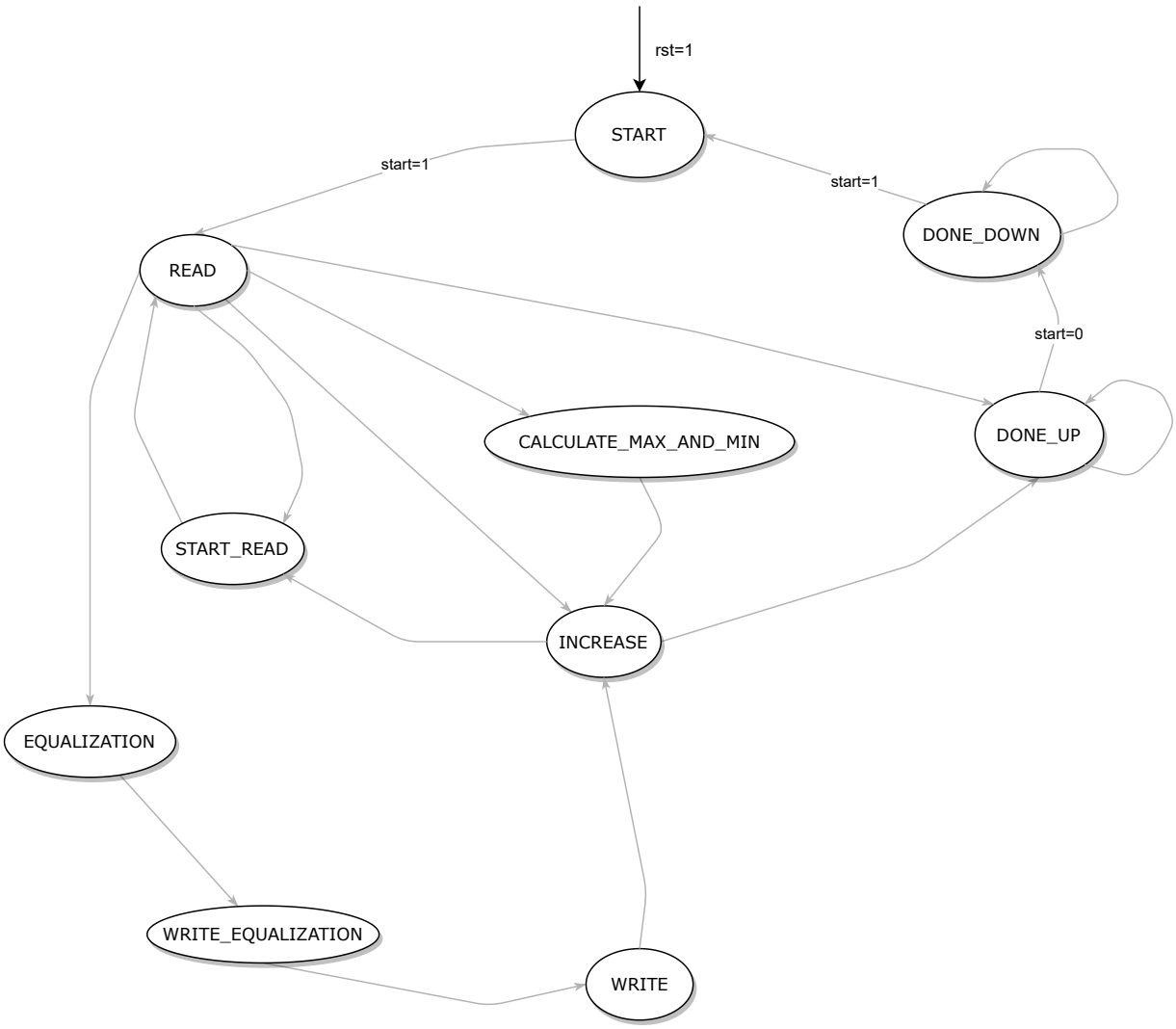
IMMAGINE 4 x 3

INDIRIZZO MEMORIA	VALORE	COMMENTO
0	4	\\ Byte più significativo numero colonne
1	3	\\ Byte meno significativo numero righe
2	0	\\ primo Byte immagine
3	23	
4	46	
5	69	
6	92	
7	115	
8	139	
9	162	
10	185	
11	208	
12	231	
13	255	\\ ultimo Byte immagine
14	0	\\ primo Byte immagine equalizzata (risultato)
15	23	
16	46	
17	68	
18	92	
19	115	
20	139	
21	162	
22	185	
23	208	
24	231	
25	255	

Architettura

2.1 Descrizione macchina a stati

Abbiamo optato per un' implementazione a singolo processo, che svolge un'elaborazione in modalità sincrona, sviluppando una MSF con 10 stati, a cui abbiamo aggiunto un 11esimo (DELAY) per permettere il caricamento dei valori dei segnali e di lettura e scrittura in memoria.
Qui di seguito una rappresentazione grafica della macchina:



Stato	Descrizione
START	Stato di inizio processo e equalizzazione di un immagine a seguito di un segnale di *reset* alto (necessariamente per la prima immagine) e un segnale alto di *start*. Vengono inizializzate tutte le variabili e i segnali al valore di default 0.
START_READ	Stato per inizio lettura da memoria impostando il segnale di *enable* a 1 e l'*o_address* pari al segnale *current_address* per indicare l'indirizzo di memoria da cui leggere

Stato	Descrizione
READ	Stato in cui in base all' indirizzo di memoria a cui mi trovo, il numero letto sarà: • memorizzato nel segnale che indica il numero di colonne dell'immagine (se prima lettura ovvero all'indirizzo 0) • memorizzato nel segnale che indica il numero di righe dell'immagine (se seconda lettura ovvero all'indirizzo 1) • il pixel dell'immagine letto, usato per confrontarlo con tutti gli altri allo scopo di trovare il massimo e il minimo (operazione svolta nello stato apposito) • se ho finito la lettura dei pixel dell'immagine originale (ovvero mi trovo all'indirizzo = riga x colonna +2) carico nell'*o_address* l'indirizzo del relativo pixel da equalizzare, il quale sarà poi memorizzato dopo l'elaborazione nell'attuale indirizzo. Inoltre calcolo il *delta_value* avendo già a disposizione il minimo e massimo valore tra i pixel dell'immagine letta tutta precedentemente.
CALCULATE_MAX_AND_MIN	Stato in cui effettuo il confronto effettivo con l'attuale pixel letto e il massimo e minimo attuali, eventualmente aggiornandoli.
INCREASE	Stato in cui incremento (di 1) sia il contatore *step* che mi tiene traccia dell'evoluzione del processo di equalizzazione corrente, sia del segnale per l'indirizzo corrente di memoria a cui si trova l'elaborazione; operazione eseguita a seguito di una fine lettura del pixel dell'immagine originale o dell'equalizzazione e scrittura di un pixel stesso.
EQUALIZATION	Stato in cui inizia effettivamente l'equalizzazione con il calcolo dello *shift_level* da applicare all'algoritmo, con un controllo di soglia del delta value calcolato nello stato di READ. Infine viene caricato nell'apposito segnale il valore del pixel da equalizzare.
WRITE_EQUALIZATION	Stato in cui, preso il valore del pixel da elaborare dalla memoria, procedo con l'algoritmo di equalizzazione: applico lo shift a sinistra di tante posizioni quante indicate dallo *shift_level* e tale risultato lo confronto con 255. Il risultato dell'operazione viene salvato nella variabile *new_pixel_value* e viene riportato l'*o_address* all'indirizzo in cui scriveremo il pixel equalizzato.
WRITE	Stato dove alzo il valore del segnale di *write enable* per poter scrivere in memoria (nel primo indirizzo libero in memoria pari all'indirizzo del pixel equalizzato + (riga x colonna)) il *new_pixel_value*
DONE_UP	Stato che mi segnala la fine elaborazione portando il segnale di *done* alto, tenendolo inoltre tale finché non diventa basso il segnale di *start*

Stato	Descrizione
DONE_DOWN	Stato successivo a DONE_UP a seguito di un abbassamento del segnale di *start* in cui viene portato il segnale *done* basso e rimane in attesa di una nuova elaborazione segnalata con un alzamento del segnale di *start*
DELAY	Stato realizzato con il fine di consentire il caricamento dei valori dei segnali e la lettura/ scrittura in memoria, interponendolo tra i vari stati che ne necessitano

2.2 Descrizione dei segnali e delle variabili

Abbiamo utilizzato una serie di segnali e due variabili (da supporto per le operazioni di equalizzazione, essendo il caricamento di un nuovo valore immediato per loro rispetto ai segnali che necessitano di un segnale di clock) :

- **state**: segnale di tipo state_type per identificare gli stati della macchina
- **current_state**: altro segnale di tipo state_type utilizzato nello stato di DELAY per indirizzare l'operazione successiva in base a questa variabile(ovvero l'ultima operazione svolta)
- **current_address**: segnale per memorizzare l'indirizzo di memoria a cui l'elaborazione si trova attualmente; alla prima lettura sarà il valore da caricare in o_address, alla seconda lettura (per l'elaborazione) si leggerà da memoria all'indirizzo [o_address - (row X column)] e il risultato si scriverà in memoria all'indirizzo current_address stesso
- **column**: segnale per memorizzare il numero di colonne di cui è composta l'immagine
- **row**: segnale per memorizzare il numero di righe di cui è composta l'immagine
- **max_pixel_value** e **min_pixel_value**: segnali in cui vengono memorizzati rispettivamente il valore massimo tra i pixel dell'immagine e il minimo. Alla lettura del primo pixel vengono entrambi inizializzati al valore dello stesso, per poi essere confrontati con i pixel successivi e eventualmente aggiornati (se minori o maggiori).
- **delta_value**: segnale dato dalla differenza tra il max_pixel_value e il min_pixel_value utilizzato nell'algoritmo di equalizzazione fornito da specifica.
- **current_pixel_value**: segnale dove viene caricato il valore del pixel letto da memoria
- **step**: segnale utilizzato per tener traccia dell'evoluzione del processo; in base al valore di questo la macchina valuta se deve leggere da memoria, scrivere o se il processo è terminato.
- **shift_level**: ulteriore segnale necessario per l'algoritmo di equalizzazione; può assumere valori da 0 a 8, inoltre viene calcolato con un controllo di soglia del delta_value:
- **dim_img**: segnale in cui, una volta letti i primi due valori in memoria (numero righe e colonne), viene memorizzata la dimensione intera dell'immagine: data dal prodotto di row X column.

- **temp_pixel**: variabile in cui viene posto il valore dello shift a sinistra del [current_pixel_value - min_pixel_value] di tanti posti quanti indicati dallo shift_level.
- **new_pixel_value**: variabile in cui viene messo il valore più piccolo tra il temp_pixel e 255 per poi essere scritto in memoria, ovvero è il valore dell'equalizzazione del pixel.

2.3 Criticità riscontrate

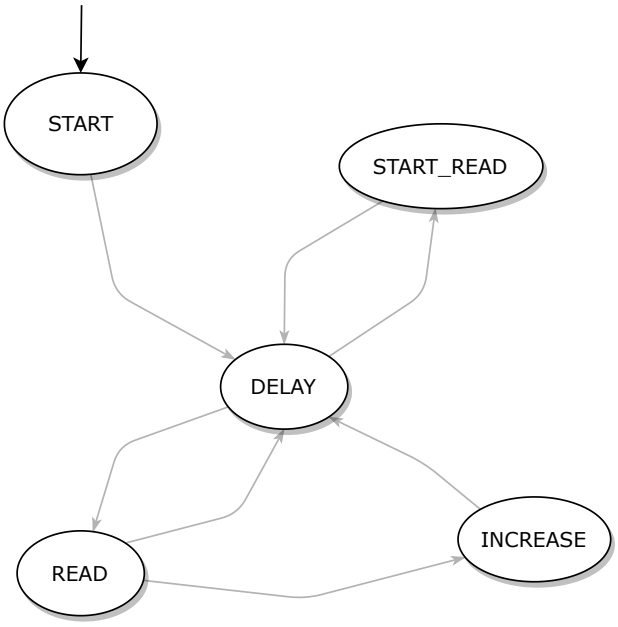
Durante la moderazione della macchina a stati sono emerse alcune criticità:

A. Gestione del segnale di reset (*i_rst*):

i_rst, come da specifica, è il segnale che permette di inizializzare la macchina per ricevere il primo segnale di *i_start* e iniziare quindi l' elaborazione della prima immagine; di conseguenza, ogniqualvolta il segnale di *i_rst* viene portato a valore alto abbiamo fatto si che la macchina resetti tutti i segnali e le variabili al valore di default e ritorni allo stato iniziale di START.

B. Gestione tempi memoria e segnali:

Per risolvere tale problema abbiamo aggiunto alla macchina un undicesimo stato chiamato DELAY volto a creare un ritardo (delay) artificiale, interponendolo tra gli stati che devono leggere o scrivere in memoria o che necessitano del nuovo valore di un segnale appena modificato. Grazie all'ausilio del segnale *current_state* nel DELAY la macchina sa indirizzarsi verso l'effettivo stato successivo.




```
when DELAY =>

    if (current_state = START) then
        state <= START_READ;
    elsif(current_state = START_READ) then
        state <= READ;
    elsif(current_state = READ) then
        dim_img <= TO_INTEGER(column)*TO_INTEGER(row);
                state <= START_READ;
```

Sintesi

3.1 Report di utilizzo

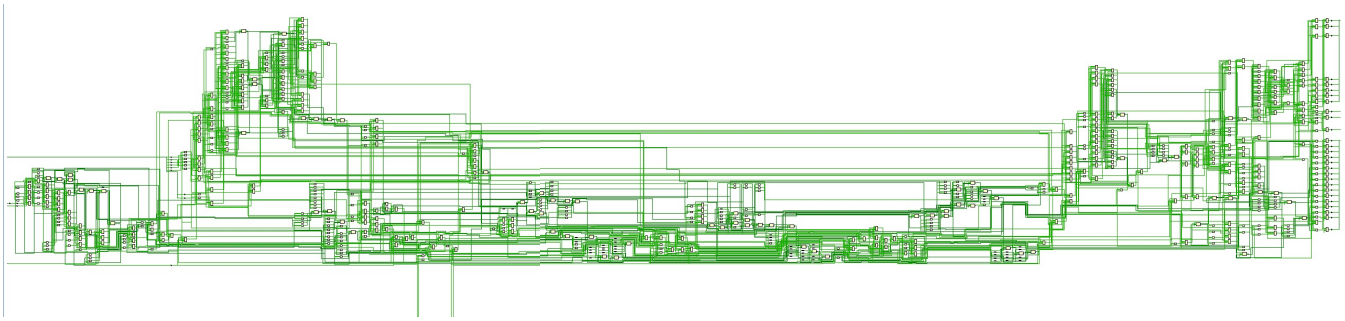
Attraverso il report di utilizzo è stato possibile valutare il numero di componenti e l'assenza di latch:

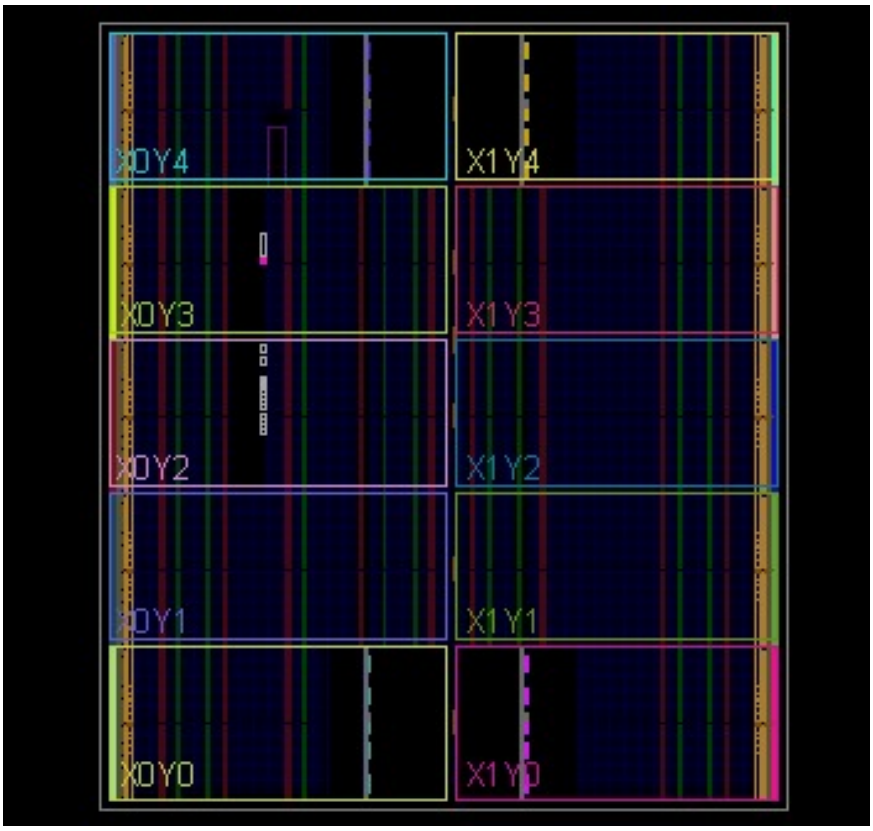
Site Type	Used	Fixed	Available	Util%
Slice LUTs*	360	0	134600	0.27
LUT as Logic	360	0	134600	0.27
LUT as Memory	0	0	46200	0.00
Slice Registers	159	0	269200	0.06
Register as Flip Flop	159	0	269200	0.06
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Nella lista di primitives è possibile valutare il principale utilizzo di flip-flop di tipo D (FDCE e in minor numero di FDRE), 2_Bit Look-Up Table (LUT2)

Ref Name	Used	Functional Category
LUT2	233	LUT
FDCE	127	Flop & Latch
LUT6	81	LUT
CARRY4	74	CarryLogic
LUT1	51	LUT
LUT4	46	LUT
FDRE	32	Flop & Latch
LUT3	30	LUT
OBUF	27	IO
LUT5	27	LUT
IBUF	11	IO
BUFG	1	Clock

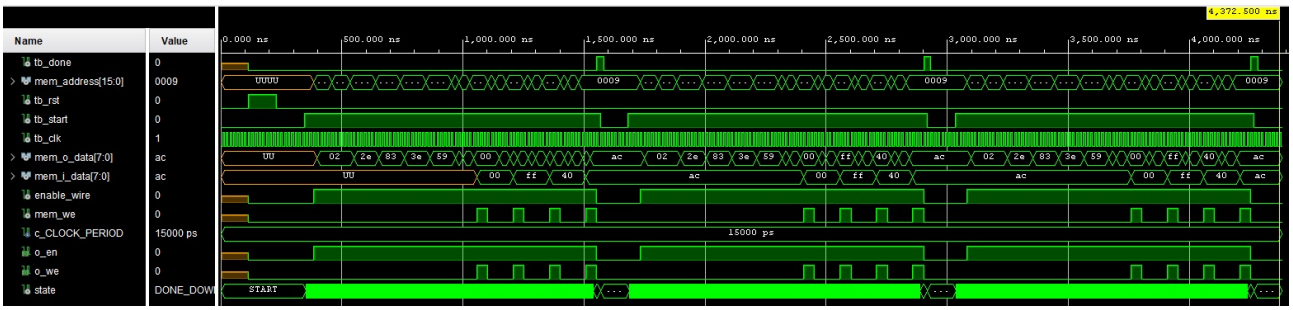
3.2 Schema sintetizzato



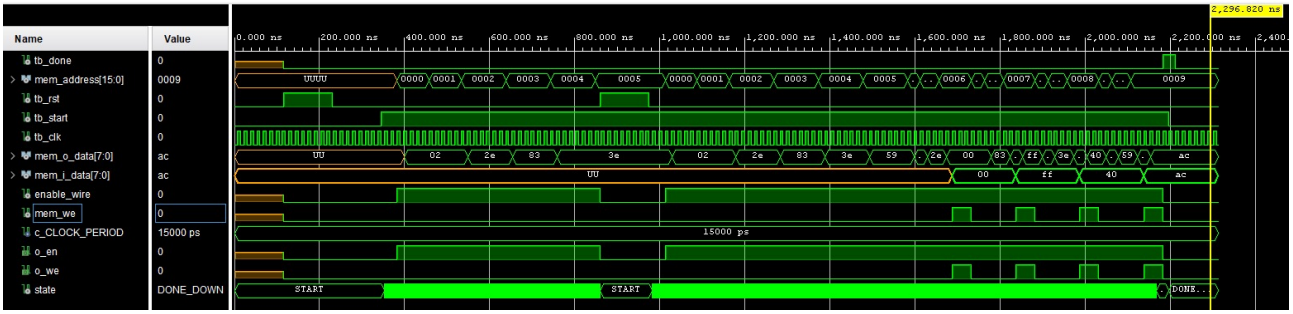


Testing

4.1 Casi normali

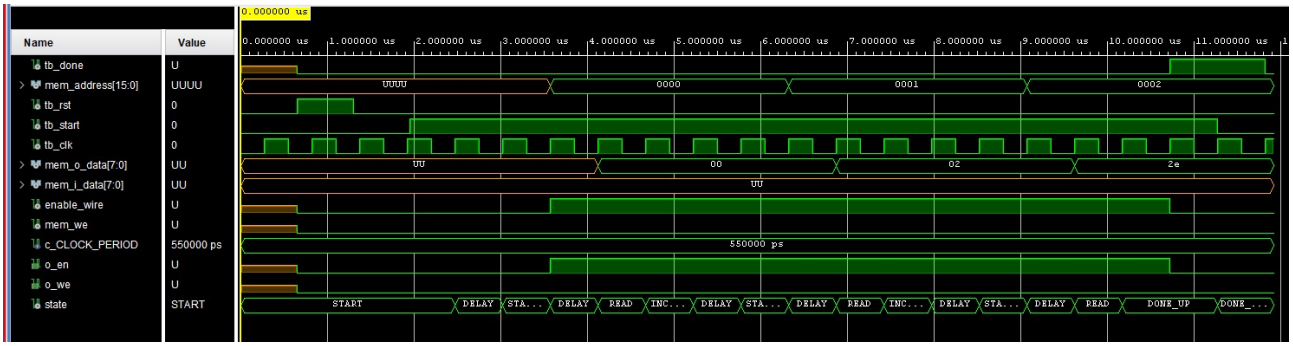


Normale test composto da una stessa immagine elaborata tre volte in successione
Input=[2, 2, 46, 131, 62, 89]
Output=[2, 2, 46, 131, 62, 89, 0, 255, 64, 172]
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 4382500 ps
Time Post Synthesis: 4427600 ps

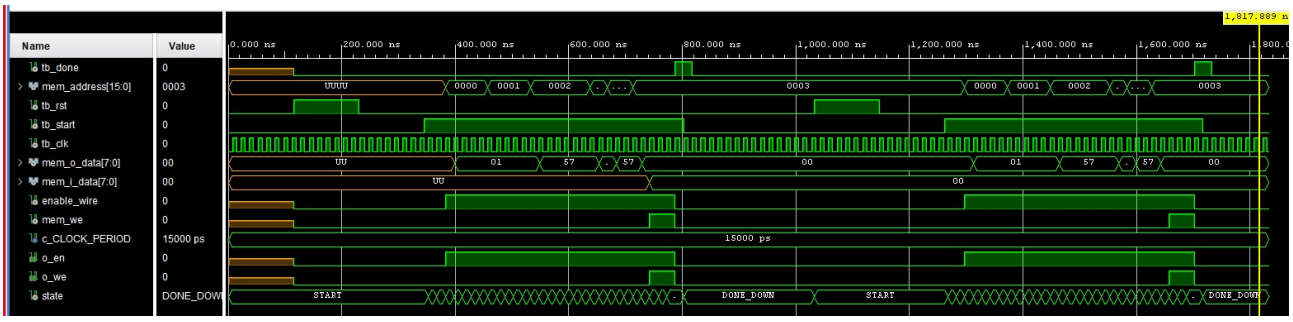


Test con segnale di reset asincrono, ovvero durante l'elaborazione della prima immagine il segnale di *reset* assume valore alto reinizializzando tutti i segnali e le variabili a valore di default (basso) per poi ricominciare una nuova elaborazione a seguito di un segnale di *start* alto.
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 2312500 ps
Time Post Synthesis: 2327600 ps

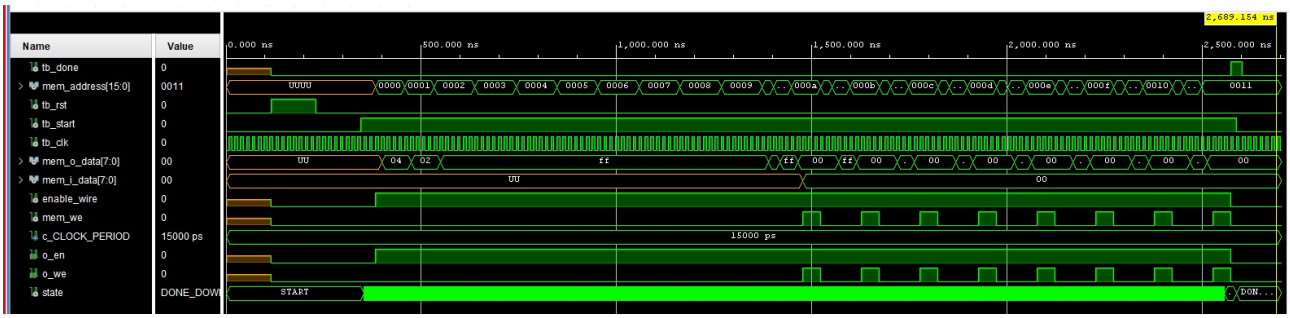
4.2 Casi limite



A. Test composto da un' immagine di dimensione pari a 0, ovvero 0 pixel da equalizzare
Input= [0, 2, 46]
Dopo aver letto i primi due dati in memoria, corrispondenti al numero di colonne e righe dell'immagine, risultando la dimensione di quest' ultima pari a 0 il processo va direttamente in stato di DONE_UP senza apportare modifiche alla memoria
Output= [0, 2, 46]
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 11925 ns
Time Post Synthesis: 12475100 ps



B. Test composto da un' immagine da un pixel (dimensione immagine pari a 1)
Input= [1, 1, 87]
Output= [1, 1, 87, 0]
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 1832500 ps
Time Post Synthesis: 1862600 ps



C. Caso limite di immagine composta da soli pixel di valore 255 (il massimo)
Input= [4, 2, 255, 255, … 255]
Output= [4, 2, 255, … 255, 0, 0, 0, 0, 0, 0, 0, 0]
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 2702500 ps
Time Post Synthesis: 2717600 ps

D. Test con immagine di dimensione massima (128x128)
Risultato: Simulation Ended! TEST PASSATO
Svolgimento: Behavioral simulation e Post Synthesis functional simulation
Time Behavioral: 4178582500 ps
Time Post Synthesis: 4178597600 ps

Conclusioni

Il componente realizzato ha dimostrato di superare correttamente, con tutti i casi di test, la simulazione *Behavioral* e la simulazione *Post-Synthesis Functional*, in linea con quanto richiesto dalle specifiche.
Si ritiene quindi di aver sviluppato un componente hardware in grado di risolvere il problema sottoposto.

Questo progetto ci ha consentito di mettere in pratica gli insegnamenti appresi durante il corso di Reti Logiche ed inoltre di apprendere un nuovo linguaggio di programmazione (vhdl) utilizzando un software mai visto in precedenza (Vivado).