Prova Finale del
Progetto di Reti Logiche

Prof. **Fabio Salice**

Anno Accademico 2020-21

POLITECNICO
MILANO 1863

Lorenzo Iovine

Nicola Landini

Indice

1 INTRODUZIONE

1.1 Scopo del progetto

La specifica del progetto è ispirata al metodo di equalizzazione dell'istogramma di un'immagine. Tale equalizzazione ha come scopo l'aumento del contrasto, dove per contrasto si intende la differenza in intensità tra il livello di grigio maggiore e quello minore.

Immagine originale

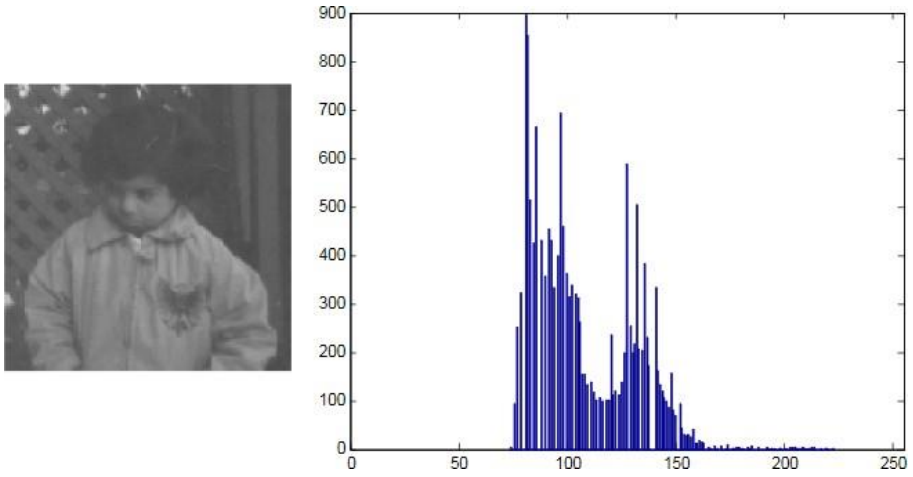
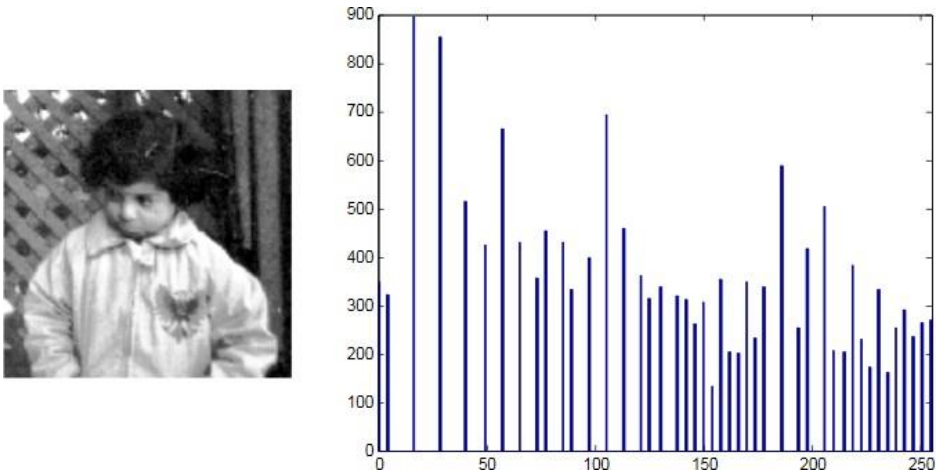


Immagine post-equalizzazione



Nell'esempio mostrato in figura è possibile notare come, a seguito di una equalizzazione, sono cambiati l'immagine ed il relativo istogramma.
Infatti, è possibile osservare come i valori dell'istogramma si sono appiattiti su tutto il range della scala di grigi, che varia tra 0 e 255. Ciò è associato ad un notevole aumento di contrasto nell'immagine.

1.2 Interfaccia del componente

```
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;
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal TestBench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- i_start è il segnale di START generato dal Test Bench;
- i_data è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- o_address è il segnale (vettore) 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 dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_we è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

1.3 Dati e Memoria

All'interno della memoria le immagini sono memorizzate sequenzialmente e riga per riga. La dimensione dell'immagine è definita da 2 byte, il numero di colonne è salvato all'indirizzo 0 e il numero di righe all'indirizzo 1.

La dimensione massima dell'immagine può essere 128x128 pixel (ad ogni pixel corrisponde 1 byte).

Il risultato dell'immagine equalizzata verrà memorizzato anch'esso in modo sequenziale e riga per riga a partire dall'indirizzo 2+(N_COLONNE*N_RIGHE).

2 ARCHITETTURA

2.1 Segnali Interni

Per poter implementare al meglio il componente abbiamo definito i seguenti segnali:

```
--State Machine signals
signal col_load : STD_LOGIC;
signal row_load : STD_LOGIC;
signal dim_load : STD_LOGIC;
signal min_load : STD_LOGIC;
signal max_load : STD_LOGIC;
signal iter_load : STD_LOGIC;
signal delta_load : STD_LOGIC;
signal shift_load : STD_LOGIC;
signal add_addr : STD_LOGIC;
signal iter_end : STD_LOGIC;
signal r1_load : STD_LOGIC;
signal current_pixel_load : STD_LOGIC;
signal r2_load : STD_LOGIC;
signal ftp_load : STD_LOGIC;
signal npv_load : STD_LOGIC;
signal addr_sel : STD_LOGIC;
signal reset : STD_LOGIC;
signal zero : STD_LOGIC;

--Datapath signals
signal reg1, row, col : std_logic_vector (7 downto 0);
signal max : std_logic_vector (7 downto 0) := (others=>'0');
signal min : std_logic_vector (7 downto 0) := (others=>'1');
signal temp_addr : std_logic_vector (15 downto 0) := (others=>'0');
signal dim : std_logic_vector (15 downto 0);
signal iter : std_logic_vector (15 downto 0);
signal shift : std_logic_vector (7 downto 0);
signal delta : std_logic_vector (7 downto 0);
signal current_pixel : std_logic_vector(7 downto 0);
signal reg2 : std_logic_vector(7 downto 0);
signal funct_temp_pixel : std_logic_vector(15 downto 0);
signal new_pixel_value : std_logic_vector(7 downto 0);
```

In particolare:

- I segnali *col_load, row_load, dim_load* e *r1_load* indicano quando devono essere caricati rispettivamente i registri *col, row, dim e reg1.* Questi salvano il numero di colonne, il numero di righe, la dimensione (col*row) e il pixel corrente dell'immagine.
- I segnali *max_load, min_load* e *delta_load* indicano quando devono essere caricati rispettivamente i registri *max, min* e *delta.* Questi salvano il pixel con valore massimo, quello con valore minimo e la differenza tra i due.
- Il segnale *add_addr* gestisce il caricamento del registro *temp_addr*, il quale è un iteratore per lo scorrimento degli indirizzi. Per quanto riguarda *addr_sel,*

questo è un segnale di controllo di un multiplexer, per gestire la scelta tra indirizzi in lettura ed indirizzi in scrittura.

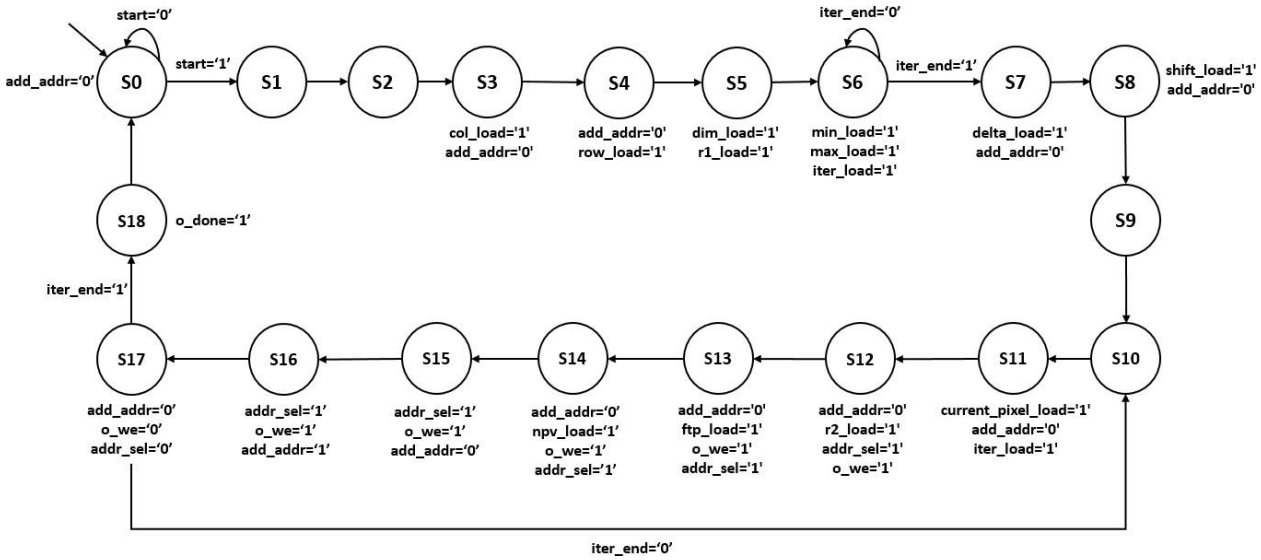
- Il segnale *iter_end* rappresenta l'uscita di un comparatore a 0 della differenza tra i registri *iter* e *dim*. Il segnale *iter_load* indica quando deve essere caricato il registro *iter*.
- I segnali *shift_load, current_pixel_load, r2_load, ftp_load, npv_load* indicano quando devono essere caricati rispettivamente i registri *shift, current_pixel, reg2, funct_temp_pixel, new_pixel_value.* Questi gestiscono la seconda lettura dei pixel dell'immagine e la corrispondente equalizzazione.
- Il segnale *reset* serve a resettare tutti i registri quando termina l'equalizzazione di un'immagine e se ne vuole iniziare subito un'altra.
- Il segnale *zero* serve a gestire il caso particolare in cui le righe e/o le colonne dell'immagine siano uguali a 0, per cui si torna direttamente allo stato iniziale, non essendo necessaria alcuna equalizzazione.

I segnali definiti hanno i seguenti default:

```
add_addr<='1';
o_en<='1';
o_we<='0';
col_load<='0';
row_load<='0';
iter_load<='0';
dim_load<='0';
max_load<='0';
min_load<='0';
r1_load<='0';
o_done<='0';
shift_load<='0';
delta_load<='0';
current_pixel_load<='0';
r2_load<='0';
ftp_load<='0';
npv_load<='0';
addr_sel<='0';
reset<='0';
```

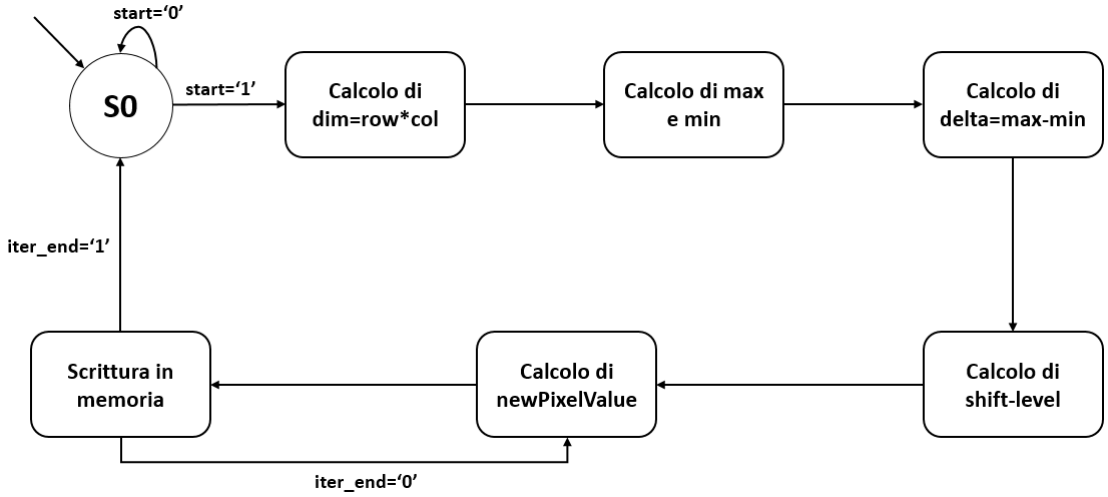
2.2 Macchina a Stati

La rappresentazione completa della macchina a stati è la seguente:



In questa immagine sono omesse le frecce che riportano allo stato S0 da qualsiasi stato nel caso in cui i_rst sia posto ad 1 in modo asincrono

Una versione semplificata è la seguente:



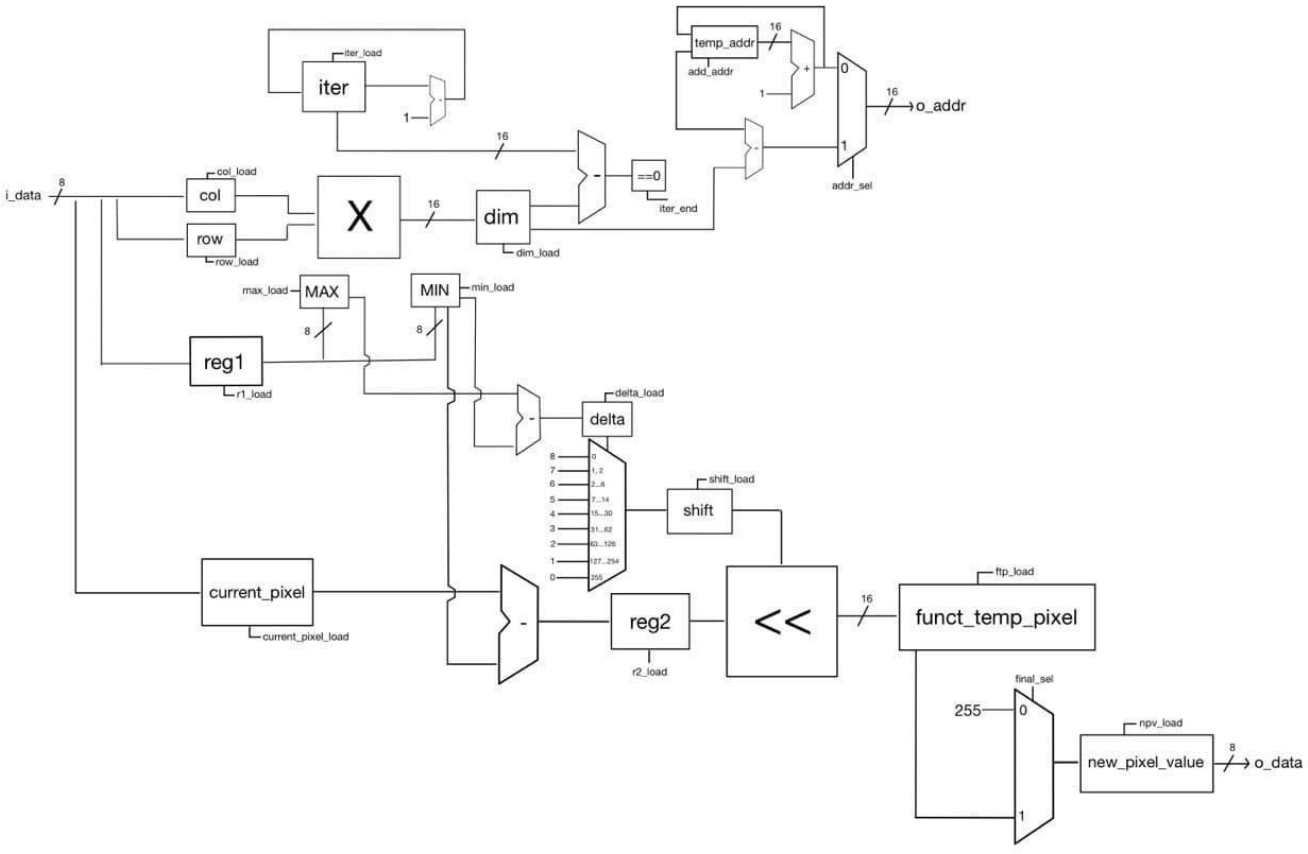
Descrizione della macchina semplificata:

- **Stato S0**: rappresenta l'inizio del programma, il quale si attiverà solo quando lo start sarà posto ad 1.
- **Calcolo di dim=row*col**: questo stato è composto dagli stati S1, S2, S3, S4, S5. Qui vengono salvate il numero di righe e di colonne dell'immagine e successivamente il loro prodotto, così da ottenere la dimensione totale.

- **Calcolo di max e min**: questo stato è composto dallo stato S6, nel quale si rimane fino alla fine della prima lettura di tutti i pixel dell'immagine. Qui vengono salvati il pixel con valore massimo e quello con valore minimo.
- **Calcolo di delta=max-min**: questo stato è composto dallo stato S7, nel quale viene salvato il delta, dato dalla differenza tra il pixel con valore massimo e quello con valore minimo.
- **Calcolo di shift-level**: questo stato è composto dallo stato S8, nel quale viene salvato il valore dello shift da effettuare per equalizzare i pixel, tramite dei controlli a soglia.
- **Calcolo di newPixelValue**: questo stato è composto dagli stati S10, S11, S12, S13, S14. Qui avviene il calcolo dei pixel modificati da scrivere in memoria.
- **Scrittura in memoria**: questo stato è composto dagli stati S15, S16, S17. Qui avviene la scrittura in memoria del new_pixel_value, per poi ritornare allo stato S10 fino a quando il segnale iter_end sarà uguale a 0, altrimenti terminerà la scrittura.

2.3 Datapath

La rappresentazione del Datapath è la seguente:



Nel multiplexer che ha come segnale di controllo il registro *delta* abbiamo calcolato lo shift con dei controlli a soglia in questo modo:

DELTA	**SHIFT**
0	8
1..2	7
3..6	6
7..14	5
15..30	4
31..62	3
63..126	2
127..254	1
255	0

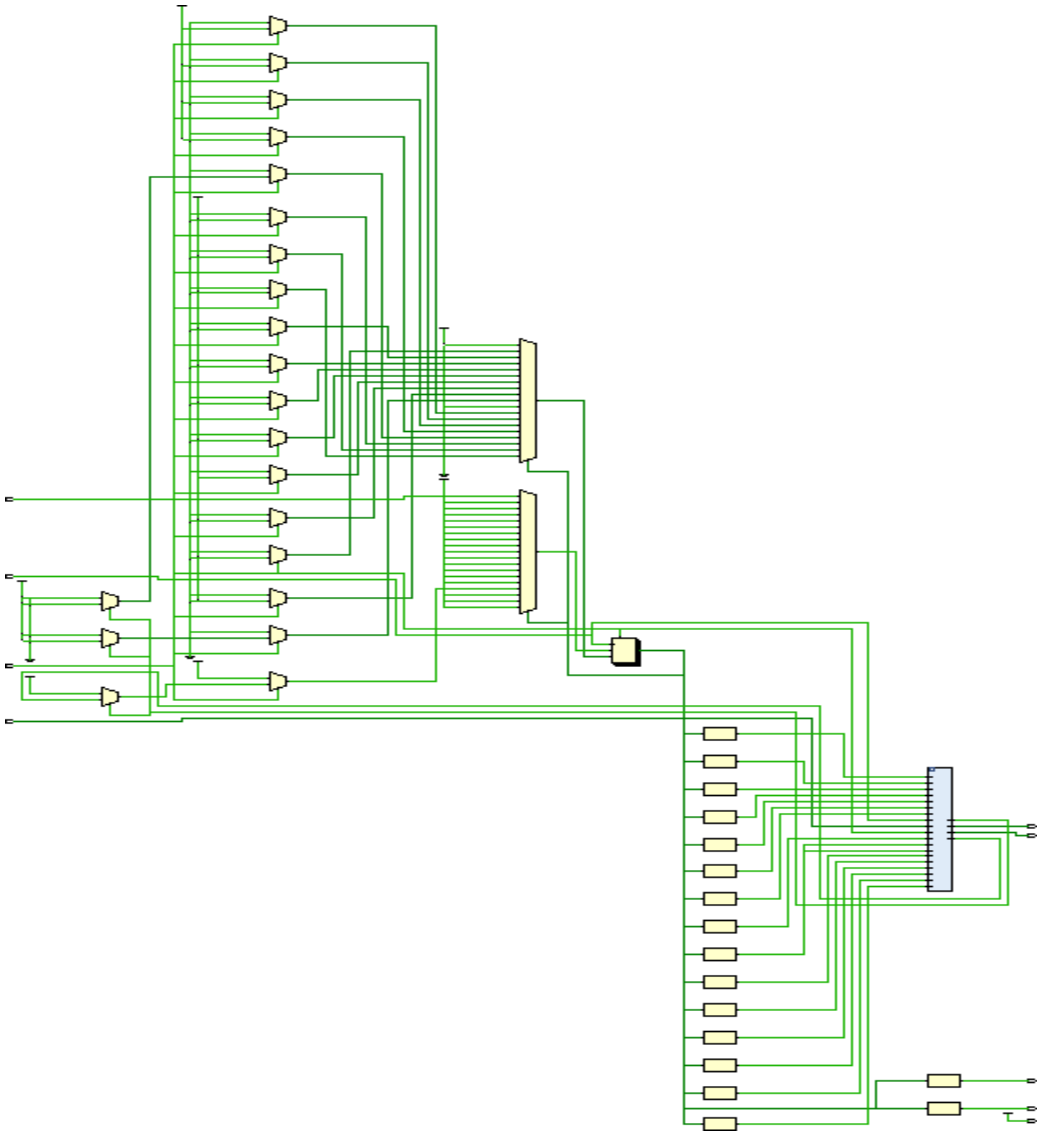
3 RISULTATI SPERIMENTALI

3.1 Utilization-Synth Design

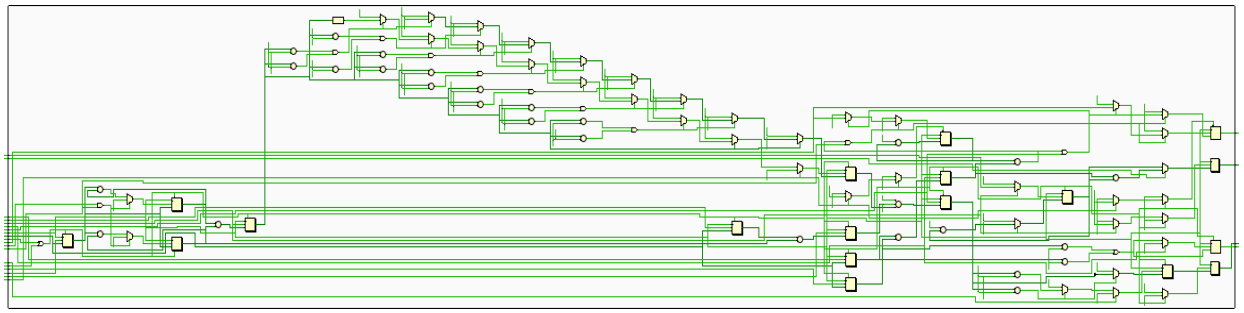
Il componente risulta correttamente sintetizzabile ed implementabile con l'utilizzo di 201 LUT, 185 Flip Flop e 0 Latch, come si può notare dall'immagine che segue.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	201	0	134600	0.15
LUT as Logic	201	0	134600	0.15
LUT as Memory	0	0	46200	0.00
Slice Registers	185	0	269200	0.07
Register as Flip Flop	185	0	269200	0.07
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.2 Schema del componente sintetizzato



La parte in celeste rappresenta il Datapath, il quale esteso è rappresentato in questo modo:



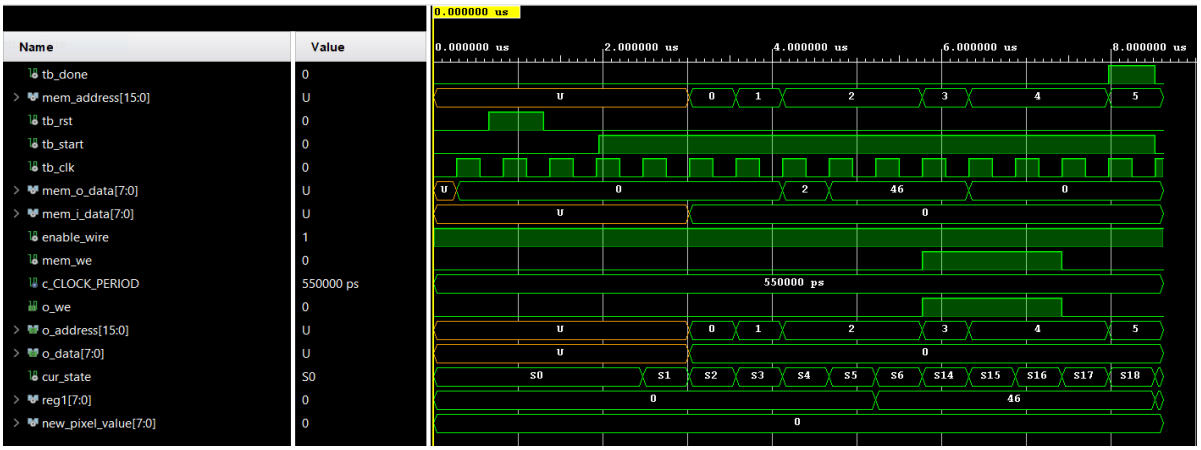
4 SIMULAZIONI

Il nostro progetto è stato sottoposto a numerose simulazioni per testarne l'efficacia, sia in Behavioral che in Post-Synthesis Functional, e sono state tutte superate con successo.

Successivamente verranno riportati i risultati in Behavorial Simulation dei testi più significativi; nei risultati sono stati aggiunti i segnali *o_we, o_address, o_data, cur_state, reg1* e *new_pixel_value* in quanto sono i più rilevanti per capire lo sviluppo del programma.

4.1 Caso limite: Zero Pixel

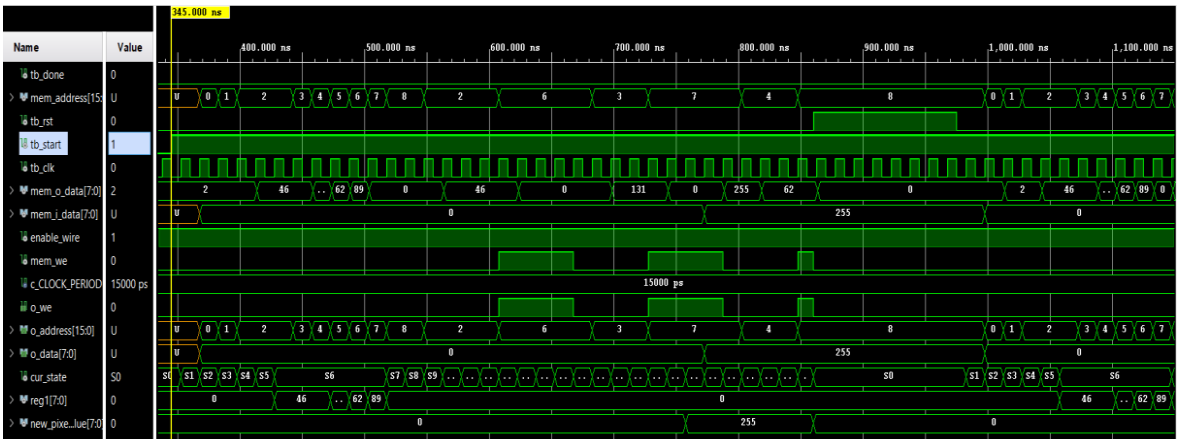
In questo test il numero di pixel da equalizzare è uguale a 0 e il risultato della simulazione è il seguente:



Behavioral Simulation: caso Zero Pixel

4.2 Reset Asincrono

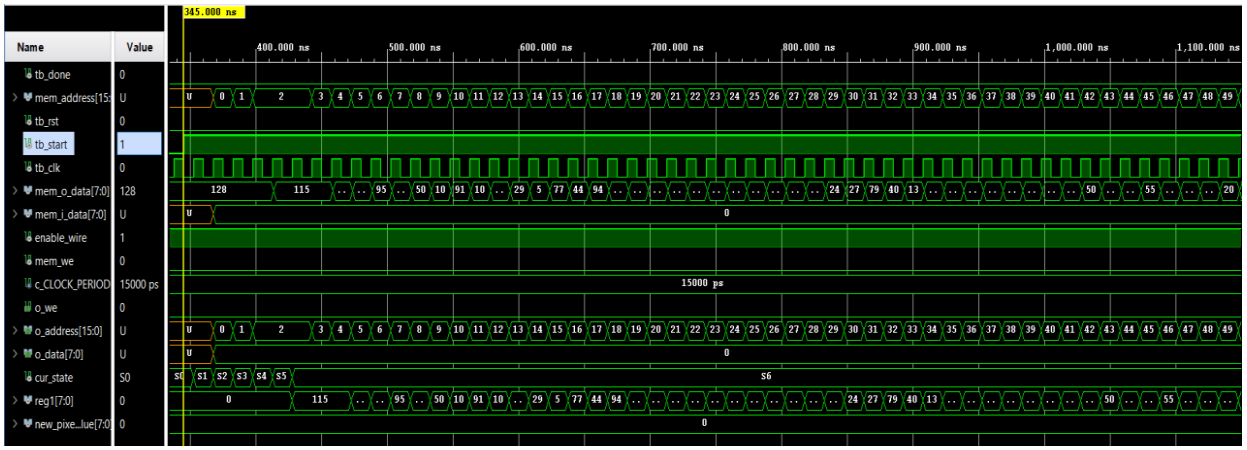
In questo test viene posto i_rst ad 1 prima del termine dell'equalizzazione dell'immagine, per cui il programma ripartirà dallo stato S0; il risultato è il seguente:



Behavioral Simulation: caso Reset Asincrono

4.3 Dimensione massima: 128x128 pixel

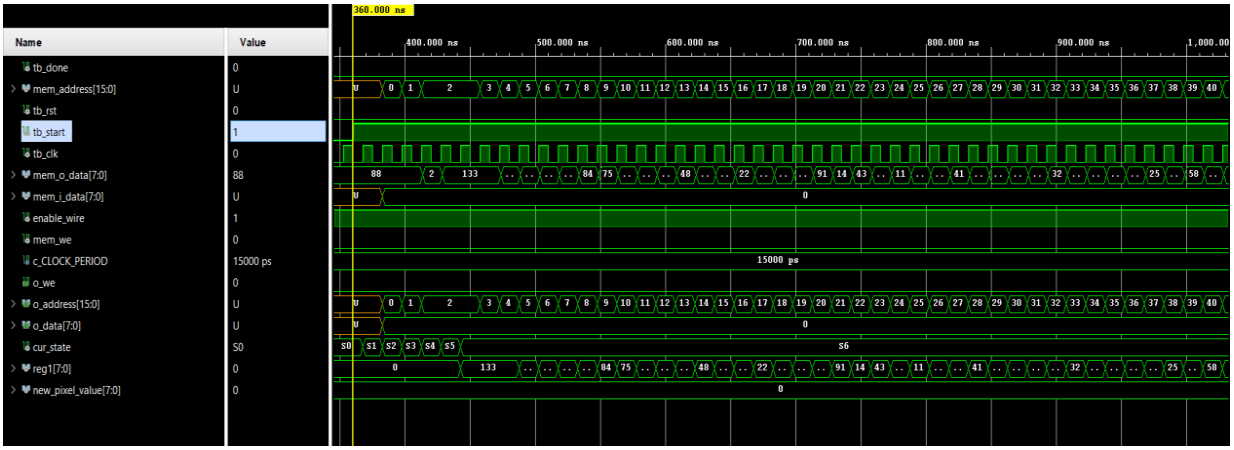
In questo test avviene l'equalizzazione di un'immagine di dimensione massima, ovvero di 128x128 pixel; il risultato è il seguente:



Behavioral Simulation: caso 128x128 pixel

4.4 Stress Test: equalizzazione di 50 immagini consecutive

In questo test da noi creato, equalizziamo 50 immagini consecutive, tra cui anche immagini con 0 o 1 pixel; il risultato è il seguente:



Behavioral Simulation: caso 50 immagini consecutive

5 CONCLUSIONI

Tale progetto ci ha permesso di migliorare le nostre capacità di programmazione e la nostra conoscenza della materia e di affinare le nostre abilità di lavoro in team.

Portare avanti lo sviluppo iniziale di un datapath, per progettare al meglio quello che sarebbe poi diventato il nostro programma, si è rivelato molto interessante. L'analisi delle forme d'onda è stata, poi, un'azione stimolante per comprendere come sviluppare al meglio il progetto.

Durante il lavoro è emerso un problema con un latch, che abbiamo prontamente risolto in coppia.

Per riuscire in ciò, abbiamo dapprima controllato la schematica ed analizzato le Primitives all'interno del report utilization, in modo da comprendere quale fosse il ref name del latch. Successivamente, cercando nella netlist a quale registro fosse stato associato questo latch, abbiamo capito qual era il bit che provocava problemi, per cui abbiamo risolto inserendo un branch inizialmente non considerato.

Lavorare alla risoluzione dei problemi ci ha permesso di comprendere bene il funzionamento di Vivado e di migliorare la capacità di risolvere delle criticità in maniera efficiente.

Con questo progetto abbiamo avuto la possibilità di mettere in pratica le lezioni apprese durante questi mesi, e l'aver raggiunto una conclusione "fisica" a tutto ciò che abbiamo studiato durante il corso è stata una fonte di continui stimoli ed apprendimento.