Prova Finale
Progetto di Reti Logiche

Anno Accademico 2020/2021

POLITECNICO
MILANO 1863

Docente

Prof. Palermo Gianluca

Esaminandi

Matteo Scaccabarozzi
Codice Persona 10708407 - Matricola 939702

Marcos Alexander Tenemaya Moyolema
Codice persona 10629376 – Matricola 911229

SOMMARIO

INTRODUZIONE

Scopo del progetto

Lo scopo del progetto è descrivere in linguaggio VHDL il seguente algoritmo di equalizzazione dell'istogramma di una immagine in scala di grigi:

Delta = Valore pixel MAX – Valore pixel MIN
Numero di SHIFT = 8 - ⌊ $\log_2$(Delta + 1) ⌋
Valore temporaneo pixel = (Valore pixel CORRENTE - Valore pixel MIN) << SHIFT
Pixel equalizzato = MIN(255, Valore temporaneo pixel)

I pixel sono codificati in 8 bit e hanno un corrispettivo valore decimale che parte da 0 (bit spento, ovvero pixel nero) e arriva fino a 255 (bit accesso avente massima intensità luminosa, ovvero bianco).

L'obiettivo è analizzare la distribuzione dell'intensità e ridistribuirla in maniera omogenea, migliorando il contrasto dell'immagine.
L'algoritmo sopra mostrato calcola il DELTA, cioè la differenza tra il valore massimo e il valore minimo dei pixel presenti nell'immagine. Questa variabile verrà utilizzata per calcolare il numero di SHIFT, valore che indica quante volte al valore temporaneo del pixel (indicato in binario) verrà aggiunto uno 0 come LSB. Infine, il valore del pixel equalizzato sarà il valore più piccolo tra 255 e il valore temporaneo appena calcolato.

Schema grafico della memoria

Data una immagine con 4 righe e 3 colonne, possiamo descrivere la memoria attraverso la seguente rappresentazione grafica

Indirizzo di riferimento	Valore in memoria	Descrizione	
0	2	Numero colonne	RAM[0]
1	3	Numero righe	RAM[1]
2	15	Primo pixel originale	RAM[2 + 0]
3	41		RAM[2 + 1]
4	6		…
5	70		…
6	13		…
7	38	Ultimo pixel originale	RAM[2 + (n – 1)]
8	36	Primo pixel equalizzato	RAM[2 + (n + 0)]
9	140		RAM[2 + (n + 1)]
10	0		…
11	255		…
12	28		…
13	128	Ultimo pixel equalizzato	RAM[2 + (n + (n – 1))]

n = numero di pixel totali

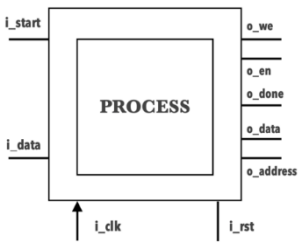
ARCHITETTURA

Interfaccia del componente

Nella specifica fornitasi è presente un'interfaccia da implementare nel nostro progetto che descrive le porte di input e output:

```
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;
```

Le prime quattro porte ricevono un segnale dall'esterno:



- **i_clk** è il segnale di CLOCK in ingresso generato dal test bench;
- **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 esterna in seguito ad una richiesta di lettura;
- **o_address** è il segnale (vettore) di uscita con cui pilotare l'indirizzo di memoria esterna;
- **o_done** è il segnale di uscita che comunica la fine dell'elaborazione;
- **o_en** è il segnale di ENABLE per poter comunicare con la memoria esterna (sia in lettura che in scrittura). Il segnale è sempre alto tranne nello stato di DONE;
- **o_we** è il segnale di WRITE ENABLE di uscita: '1' per poter scrivere e '0' per poter leggere da memoria esterna;
- **o_data** è il segnale (vettore) di uscita dal componente verso la memoria esterna.

Scelte progettuali

È stato scelto di sintetizzare in VHDL una macchina a stati finiti avente 28 stati totali.

L'esecuzione della FSM è eseguita da un singolo processo per semplificare lo sviluppo del codice e per non appesantirlo ulteriormente.

Il non utilizzo degli stati di transizione, descritti nella sezione *Descrizione degli stati*, ha comportato, in fase di testing, sporadici errori di computazione. Si è preferito quindi un approccio meno efficiente ma efficace.

L'idea progettuale iniziale era quella di impiegare l'operatore * moltiplicazione per calcolare il numero totale di pixel. In fase di Post-Synthesis Timing Simulation, in alcuni rari casi (circa 10 su 1000), il prodotto risultava errato. Nello specifico, il vettore rappresentante il prodotto presentava o il quinto o sesto bit, a partire dal LSB, con valore logico errato.

Descrizione funzionamento generale

Il processo è attivato dai segnali `i_clk` o `i_rst`.
Lo stato iniziale è IDLE e rimane tale finché un segnale `i_rst` alto non resetta tutti i segnali. Successivamente, un segnale `i_start` alto avvia l'esecuzione del processo di equalizzazione. Una volta letto il numero di colonne e di righe, un ciclo iterativo calcola il numero totale di pixel sommando il numero di colonne, tante volte quante sono il numero di righe.
Calcolato il numero totale di pixel, occorre leggere il valore di ogni singolo pixel e valutare se è il massimo o minimo valore letto. Queste operazioni sono svolte ciclicamente finché un contatore che viene incrementato ad ogni ciclo, supera il valore del numero di pixel totali.
Si passa quindi al calcolo dei parametri utili al processo di equalizzazione. Fondamentale è stato l'uso della Lookup Table per rappresentare il risultato del logaritmo arrotondato per difetto in quanto si evitano calcoli molto costosi in termini di tempo e di hardware per l'FPGA. Nella sezione *ottimizzazioni* è spiegato il funzionamento della LUT.
Calcolati i parametri, si può passare all'equalizzazione vera e propria.
Come per la lettura dei pixel, viene impiegato un contatore per considerare correttamente il numero delle iterazioni. Ad ogni ciclo la FSM rilegge il valore originale del pixel, lo elabora e scrive in output il valore equalizzato.
Il processo termina con un segnale `o_done` alto e con il ritorno allo stato IDLE.

Segnale di clock

Il progetto è stato testato nella fase iniziale ad una frequenza di clock di 10 MHz, come da specifica.
Conclusa la scrittura del codice e risolti i principali errori, la frequenza è stata innalzata dapprima a 100 MHz e infine a 200 MHz. In fase di testing non è emersa nessuna criticità. La modifica della frequenza di clock è stata eseguita modificando il file di test bench.

Schematico Post-Synthesis

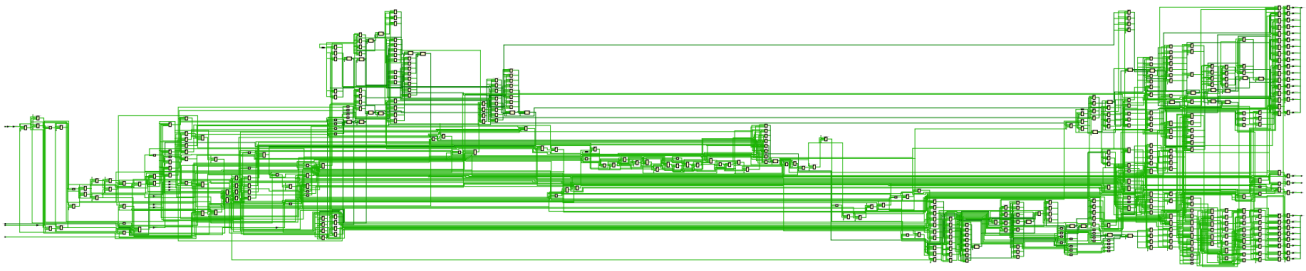
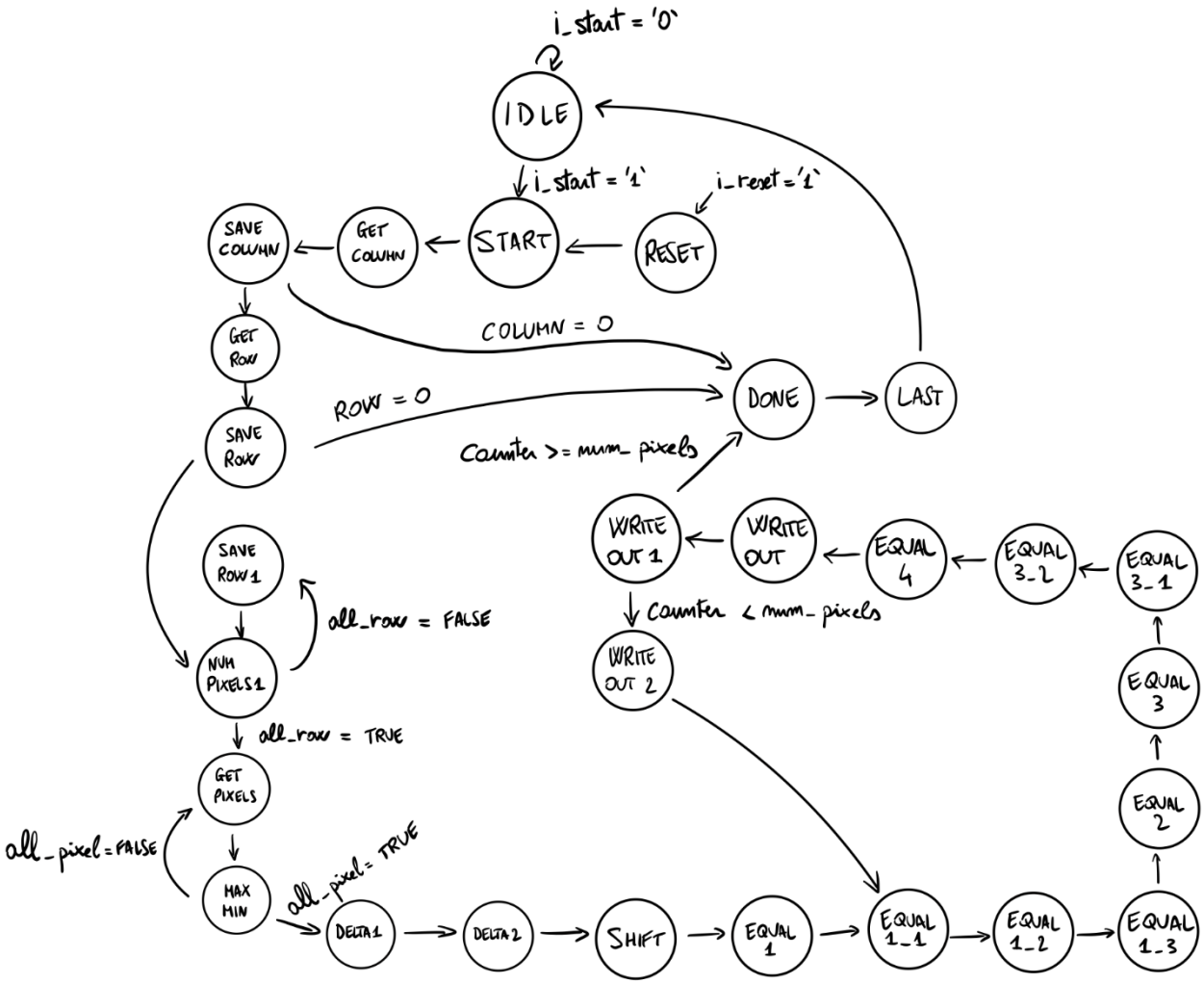


Grafico degli stati

La FSM è descritta da 28 stati, qui sottorappresentati graficamente.



Descrizione stati della FSM

Nella FSM implementata sono presenti i seguenti stati:

- **IDLE:** Stato iniziale. Si attende che il segnale `i_start` diventi alto
- **START**: Segnale `i_enable` alto. Avvio del processo
- **RESET**: Inizializza tutti i segnali
- **GET_COLUMN:** Lettura numero di colonne. Aggiornamento indirizzo per la lettura del numero di righe
- **SAVE_COLUMN**: Stato di transizione[1]
- **GET_ROW**: Lettura numero di righe
- **SAVE_ROW**: Stato di transizione[1]. Aggiornamento indirizzo per la lettura del primo pixel
- **SAVE_ROW1**: Controlla se è terminato il calcolo del numero di pixel totali
- **NUM_PIXELS_1**: Calcola il numero totale dei pixel
- **GET_PIXELS**: Legge il valore del singolo pixel. Aggiorna l'indirizzo di lettura e il contatore
- **MAX_MIN**: Stabilisce valore massimo e minimo dei pixel
- **DELTA1**: Calcola la differenza tra il valore massimo e il valore minimo dei pixel. Resetta indirizzi di lettura e contatore
- **DELTA2**: Stato di transizione[1]
- **SHIFT**: Calcola il numero di shift da effettuare
- **EQUALIZATION1**: Inizia l'equalizzazione. Definisci l'indirizzo di partenza del pixel originale da equalizzare
- **EQUALIZATION1_1**: Legge il pixel
- **EQUALIZATION1_2**: Calcola il segnale temp. Differenza tra pixel originale e pixel di valore minore
- **EQUALIZATION1_3**: Converte in 16-bit il parametro *temp* per poterlo shiftare correttamente
- **EQUALIZATION2**: Esegue da 0 a 8 shift su *temp*
- **EQUALIZATION3**: Calcola pixel equalizzato. Prende il valore minore tra *temp* e 255
- **EQUALIZATION3_1**: Stato di transizione[1]. Segnale `o_we` alto
- **EQUALIZATION3_2**: Restituisce il pixel equalizzato in output
- **EQUALIZATION4**: Incrementa il contatore. Segnale `o_we` basso
- **WRITE_OUT**: Stato di transizione[1]
- **WRITE_OUT1**: Controlla se è stata eseguita l'equalizzazione per ogni pixel. Aggiorna tutti gli indirizzi
- **WRITE_OUT2**: Ricomincia ciclo di equalizzazione del prossimo pixel
- **DONE**: Segnale `o_en` basso. Segnale `o_we` basso. Segnale `o_done` alto
- **LAST**: Segnale `o_done` basso. Torna nello stato IDLE

[1] Per stato di transizione si intende quello stato in cui i segnali hanno tempo di aggiornarsi correttamente.

Ottimizzazioni

La presenza di un numero di righe e/o colonne pari a 0 comporterebbe un'esecuzione critica, seppur tale condizione sia improbabile è stato scelto di gestire tale condizione per rendere più robusto e affidabile il codice.

L'uso della Lookup Table ha facilitato il calcolo del logaritmo in quanto l'FPGA non è in grado gestire agevolmente numeri reali e, dato che il valore da calcolare è un'approssimazione per difetto, la LUT è la struttura ideale per la nostra applicazione.
Il funzionamento è riconducibile ad un array, in cui l'indirizzo di ogni cella rappresenta il valore di input e il suo contenuto rappresenta il valore di output.

Fondamentale è stato anche l'utilizzo di segnali `std_logc_vector` e `unsigned` al posto di segnali `integer`.

RISULTATI SPERIMENTALI

Simulazioni

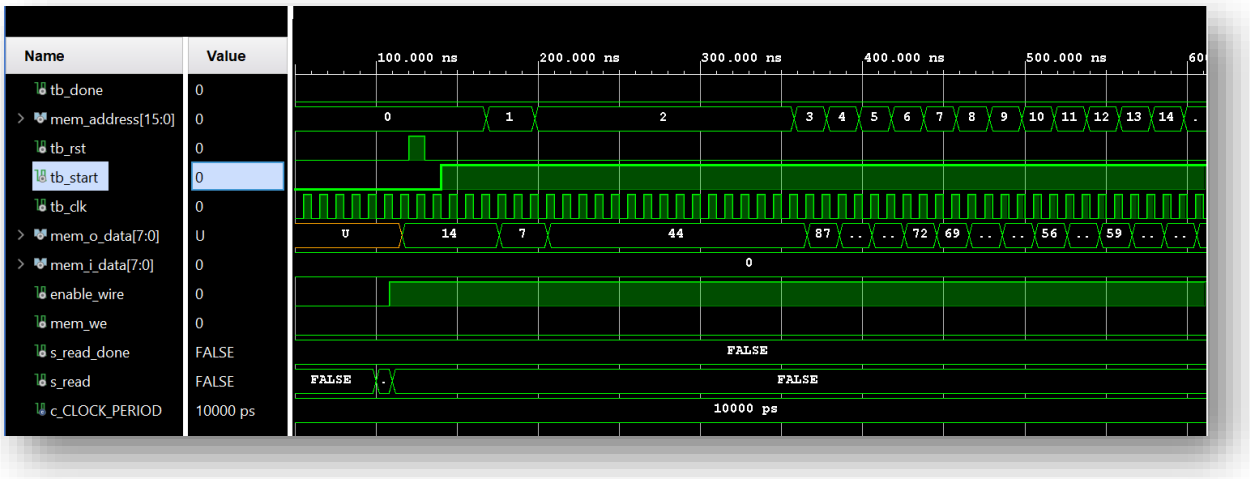
Il progetto è stato testato diverse volte durante la fase di sviluppo. All'inizio è stato impiegato il test bench fornito insieme alla specifica, utile per dettare la struttura generale. Superati con successo tutti i test, sono stati generati dei test casuali con immagini aventi una dimensione più contenuta per rendere più veloce la fase di testing (per esempio, 1.000 immagini con massimo 40x40 pixel).

Infine, è stato impiegato un test bench con 2000 immagini con dimensione fino a 128x128 pixel per la fase di consolidamento del codice.

Sono state riscontrate alcune problematiche soprattutto nella Post-Synthesis Timing Simulation, successivamente risolte.

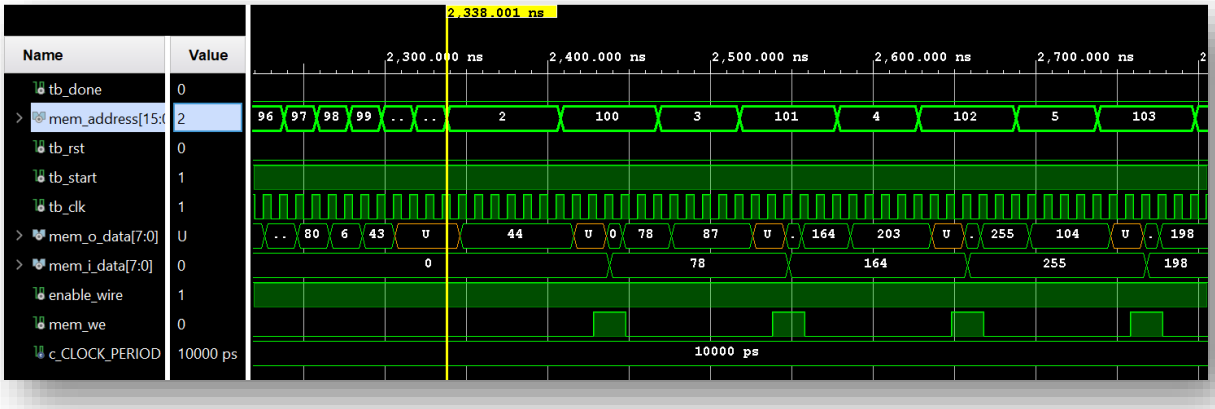
Le immagini più avanti mostrano l'andamento dei segnali principali nei punti salienti dell'esecuzione di una Post-Synthesis Functional Simulation.
La simulazione è stata eseguita ad una frequenza di clock di 200 MHz.

START



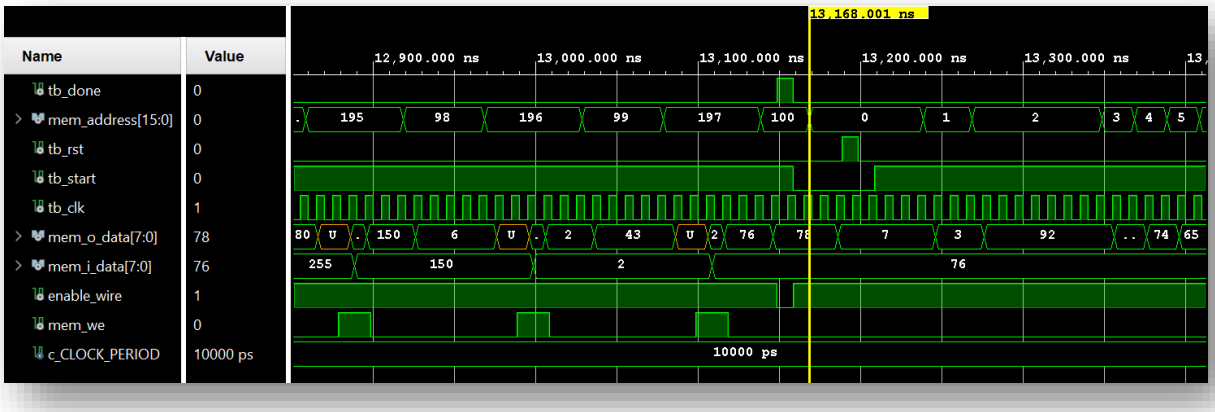
L'immagine letta ha 14 colonne e 7 righe.
Si può notare il segnale di `reset` che avviene un ciclo di clock in anticipo al segnale di `start`. Vengono letti il numero di colonne all'indirizzo 0 e il numero di righe all'indirizzo 1. Da notare la lunghezza atipica del segnale `mem_address` quando vale 2, rispetto agli altri segmenti, conseguenza del calcolo iterativo del numero di pixel totali.

EQUALIZZAZIONE



In questa seconda immagine è evidenziato, dal marker giallo, l'inizio dell'equalizzazione del primo pixel. Viene letto all'indirizzo 2 e salvato all'indirizzo 100. Alla fine dell'elaborazione viene attivato il segnale di `mem_we` per poter scrivere in output il valore del pixel equalizzato.

DONE



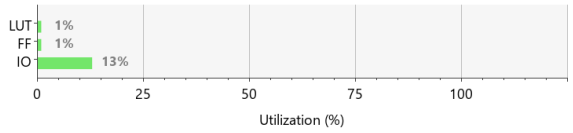
In questa immagine è rappresentata la parte finale dell'esecuzione. L'ultimo pixel in posizione 99 viene elaborato e il segnale `o_done` è settato alto. Il segnale `start` torna basso e l'indirizzo torna a 0 in attesa di un nuovo ciclo di equalizzazione.

Report di sintesi

Primitives

Ref Name	Used	Functional Category
FDRE	229	Flop & Latch
LUT4	94	LUT
LUT6	57	LUT
LUT2	40	LUT
CARRY4	31	CarryLogic
OBUF	27	IO
LUT5	24	LUT
IBUF	11	IO
LUT3	10	LUT
FDSE	8	Flop & Latch
LUT1	3	LUT
BUFG	1	Clock

Resource	Utilization	Available	Utilization %
LUT	178	41000	0.43
FF	237	82000	0.29
IO	38	300	12.67



CONCLUSIONI

Il progetto ha superato positivamente tutti i test eseguiti nelle seguenti simulazioni:

- Behavioral Simulation
- Post-Synthesis Functional Simulation
- Post-Synthesis Timing Simulation
- Post-Implementation Functional Simulation

Si considerano quindi assolte le richieste indicate nella specifica.