Prova Finale - Reti Logiche (prof. William Fornaciari)

Corigliano Emilio 10627041 - 907936

A.A. 2020/21

1 Introduzione e spiegazione del problema

Il progetto consiste nella progettazione di un circuito per l'equalizzazione di immagini in scala di grigi a 256 valori mediante il linguaggio di definizione hardware VHDL, attraverso il software *Vivado*. L'equalizzazione è un processo di elaborazione digitale che consiste nell'elaborare l'immagine affinchè i suoi colori coprano tutto lo spettro disponibile. Nel nostro caso, dopo l'elaborazione vogliamo ottenere delle immagini che abbiano pixel di valore 0 e 255 rispettivamente nei punti di valore minimo e massimo nell'immagine originale; tutti gli altri pixel verranno ridistribuiti su tutto lo spettro in maniera coerente.

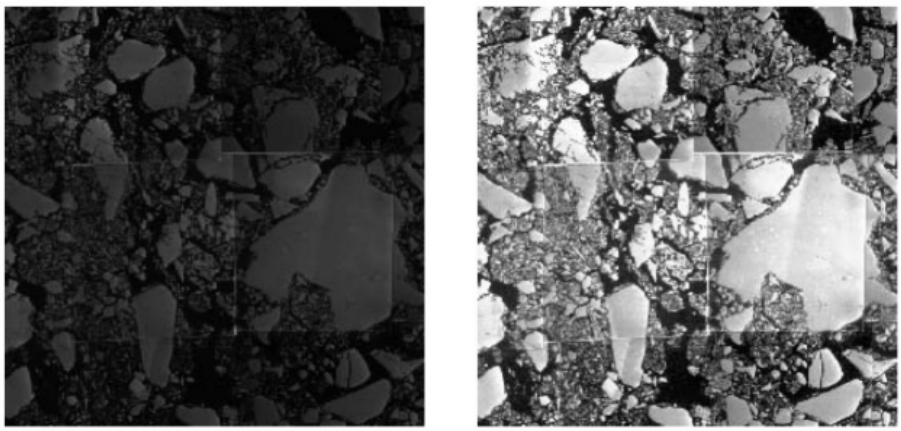

Figure 1: esempio di equalizzazione di un'immagine

2 Considerazioni e scelte implementative

- Non è necessario distinguere le varie righe, si può trattare la memoria semplicemente come un vettore di dimensione $N = R \cdot C$ da analizzare sequenzialmente
- Ogni byte in input all'indirizzo $mem[2+i]$ dovrà essere processato e l'output dovrà essere scritto in memoria all'indirizzo $mem[2 + N + i]$, con $i \in [0, N)$

2.1 Fasi

Ci sono 4 fasi fondamentali che occorre attraversare per elaborare l'immagine, ognuna delle quali ha bisogno di dati elaborati precedentemente. Per questo non sono ulteriormente parallelizzabili.

Si è scelto di affrontare ogni fase singolarmente per semplificare l'esposizione; questo approccio ha semplificato anche l'implementazione, di conseguenza le diverse fasi sono indipendenti tra loro. Seguendo questo approccio, però, si è limitata l'ottimizzazione della progettazione, facendo sì che alcuni componenti (come per esempio i contatori) non fossero in quantità minima ma sono stati replicati, in maniera leggermente diversa, per ogni fase in cui erano necessari. Questo però non è stato ritenuto un problema data la dimensione del progetto e la qualità dell'hardware per il quale è stato progettato; si è preferita la qualità e semplicità espositiva alla ottimizzazione progettuale.

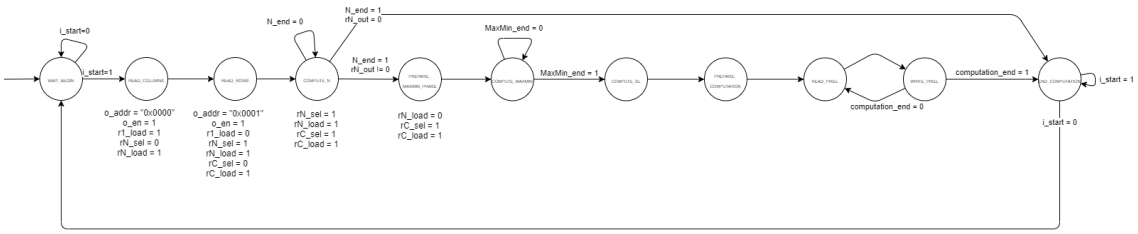

Figure 2: FSM dell'elaborazione

2.1.1 computazione della quantità di pixel

La prima fase è quella in cui si calcola la dimensione dell'immagine N da equalizzare. Questo valore è ottenuto moltiplicando $mem[0]$ e $mem[1]$ tramite un moltiplicatore per somme successive. Alla fine dell'operazione viene portato in alto il segnale N_end che indica la fine dell'operazione. Da questo momento fino alla prossima esecuzione nel registro rN ci sarà il valore $N = R \cdot C$.

2.1.2 Ricerca dei valori massimi e minimi dei pixel

Nella seconda fase si cercano il valore massimo (MAX_PV) e il valore minimo (MIN_PV): si scorre una prima volta tutta l'immagine e si cercano contemporaneamente il valore massimo e minimo dei pixel presenti nell'immagine. Per tenere conto dei pixel letti e per generare l'indirizzo al quale leggere il valore si usa un contatore che parte da 0 e ad ogni ciclo di clock incrementa di 1, fino a raggiungere N; a questo punto verrà portato in alto il segnale di $MaxMin_end$.

2.1.3 Computazione del valore SHIFT_LEVEL

La terza fase prevede il calcolo dello $SHIFT\_LEVEL$; questo viene ottenuto implementando una funzione combinatoria (nell'immagine rappresentata con il blocco $f(x)$

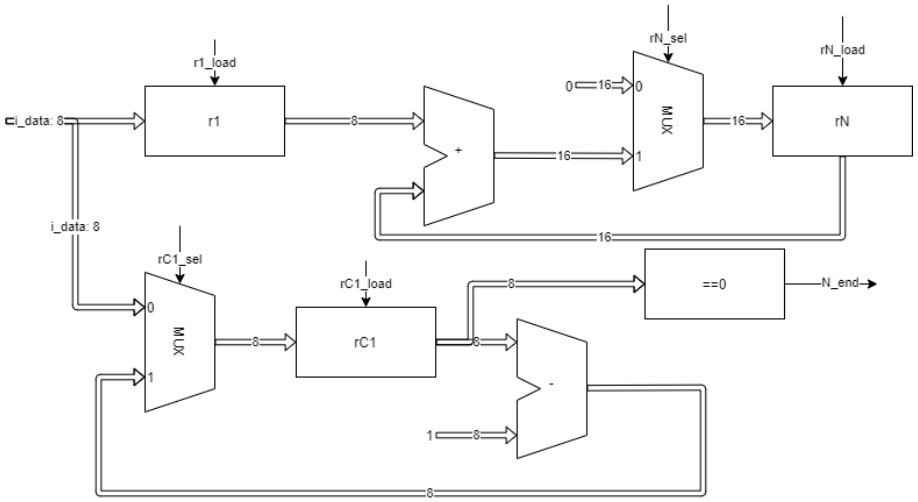

Figure 3: computazione della quantità di pixel

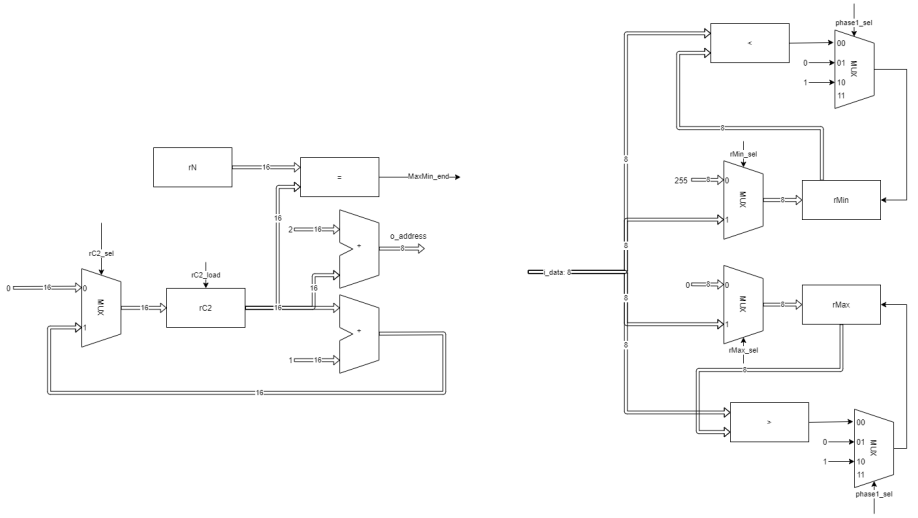

Figure 4: Ricerca dei valori massimi e minimi dei pixel

avente come input 8 bit e come output 4 bit). La funzione combinatoria prende in input $MAX\_PV - MIN\_PV + 1$ (quindi il $DELTA\_VALUE$ incrementato di 1) perchè si era pensato che sarebbe stato più semplice la generalizzazione, per non avere le varie soglie fisse. Infatt

Listing 1: Generazione di soglie

```
for i in range(256):
        print(str(i) + ": " + str(8 - math.floor(math.log2(i+1))))
```

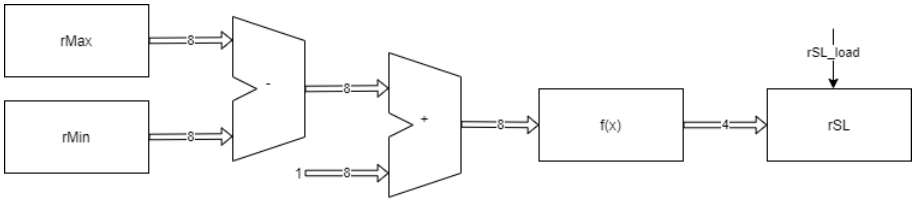

Figure 5: Ricerca dei valori massimi e minimi dei pixel

2.1.4 Equalizzazione

La quarta fase, quella finale, provvede a equalizzare l'immagine tramite l'algoritmo fornitoci. Per ogni pixel che compone l'immagine (scorsi grazie ad un contatore che parte da $N-1$ e decrementa fino a 0) si legge il valore originale. Al ciclo successivo si provvede a scrivere in memoria il valore computato all'indirizzo del pixel originale incrementato di N. Sono necessari due cicli per ogni pixel da equalizzare poichè la memoria prevede un solo segnale per codificare l'indirizzo su cui leggere o scrivere; visto che l'immagine originale non va sovrascritta si deve alternare una fase di lettura con una di scrittura.

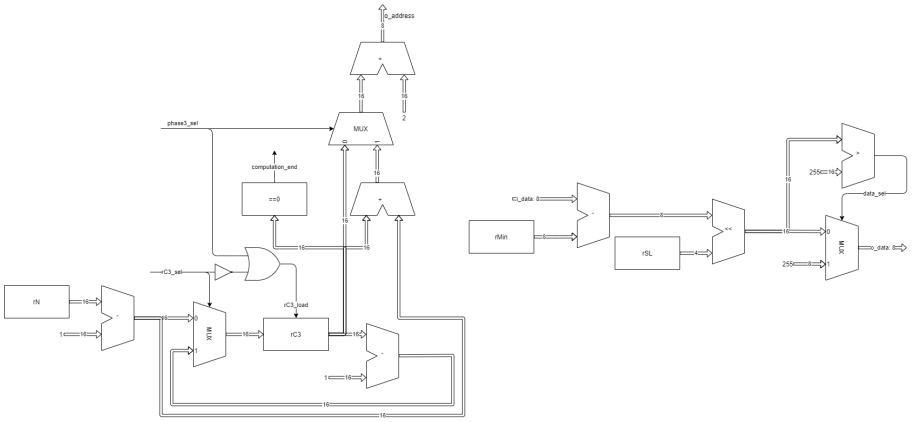

Figure 6: Ricerca dei valori massimi e minimi dei pixel

- una quarta fase che è la vera e propria elaborazione dell'immagine, nella quale si scorre pixel per pixel, elaborandolo e scrivendolo nella regione di output della memoria (input: $mem[2+i]$; output: $mem[2+N+i]$)

2.2 Registri

- $RegN$: memorizza il numero totale di byte che compongono l'immagine
- $RegMaxPL$: memorizza il livello massimo dei pixel presente nell'immagine
- $RegMinPL$: memorizza il livello minimo dei pixel presente nell'immagine

- $RegShiftLevel$: memorizza lo $SHIFT\_LEVEL$, può essere computato a soglie dato che il delta value è l'unica variabile.
- un contatore caricabile con un valore variabile che permetta di eseguire la moltiplicazione iniziale o che possa scorrere per tutto il vettore l'immagine

3 testing

Corner cases testati:

- grossa immagine
- zero righe
- zero colonne
- stesso valore in tutta l'immagine
- più immagini consecutive