PROVA FINALE
PROGETTO RETI LOGICHE

Scaglione Prof.: Salice Fabio

Composizione gruppo:

Davide Franchi - 10614321
Francisco Nieto - 10532531

1. Introduzione

Ci è stato chiesto di progettare un modulo che implementi la funzione di equalizzazione dell'istogramma di un'immagine di dimensioni massime 128x128 pixels in scala di grigi.

L'algoritmo in considerazione (semplificato rispetto a quello standard) è il seguente:

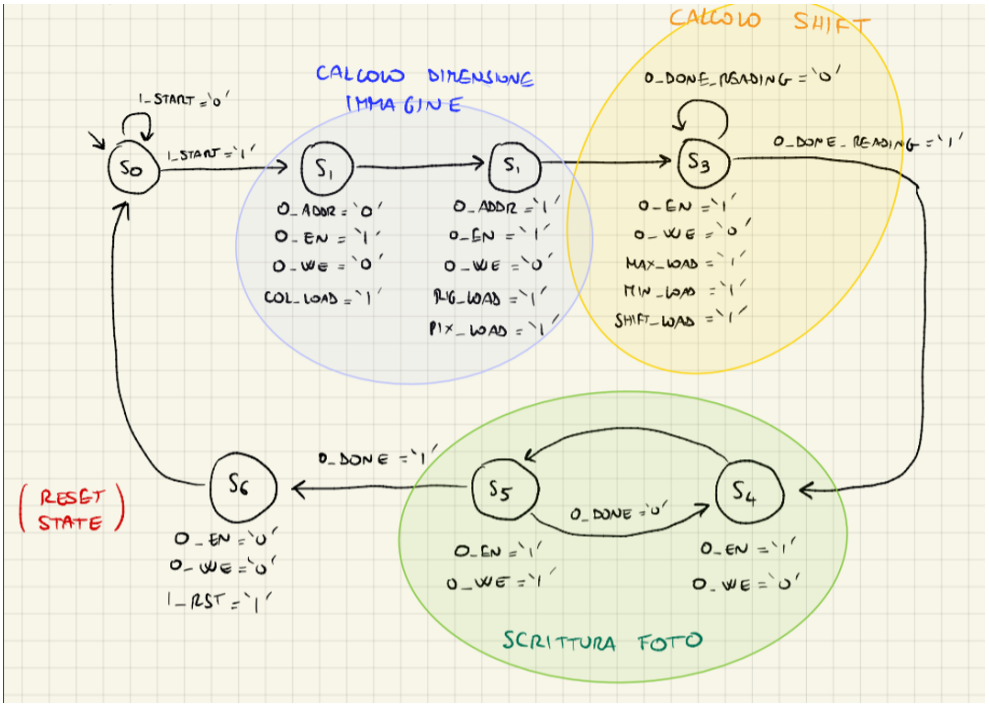
DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE
SHIFT_LEVEL = (8 – FLOOR(LOG2(DELTA_VALUE +1)))
TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) << SHIFT_LEVEL
NEW_PIXEL_VALUE = MIN(255 , TEMP_PIXEL)

dove MAX_PIXEL_VALUE e MIN_PIXEL_VALUE sono rispettivamente il valore più alto e più basso tra i pixel dell'immagine presa in analisi.
L'immagine dovrà essere letta sequenzialmente dalla memoria in cui è memorizzata, riga per riga, modificata in base all'algoritmo sopra descritto, e scritta in memoria contiguamente all'ultimo pixel della prima immagine.

La memoria è indirizzata al byte, e le dimensioni dell'immagine sono espresse rispettivamente dal numero di colonne (byte contenuto nel primo indirizzo) e numero di righe (byte contenuto nel secondo indirizzo). Il primo pixel dell'immagine, quindi, sarà contenuto in memoria al terzo indirizzo.

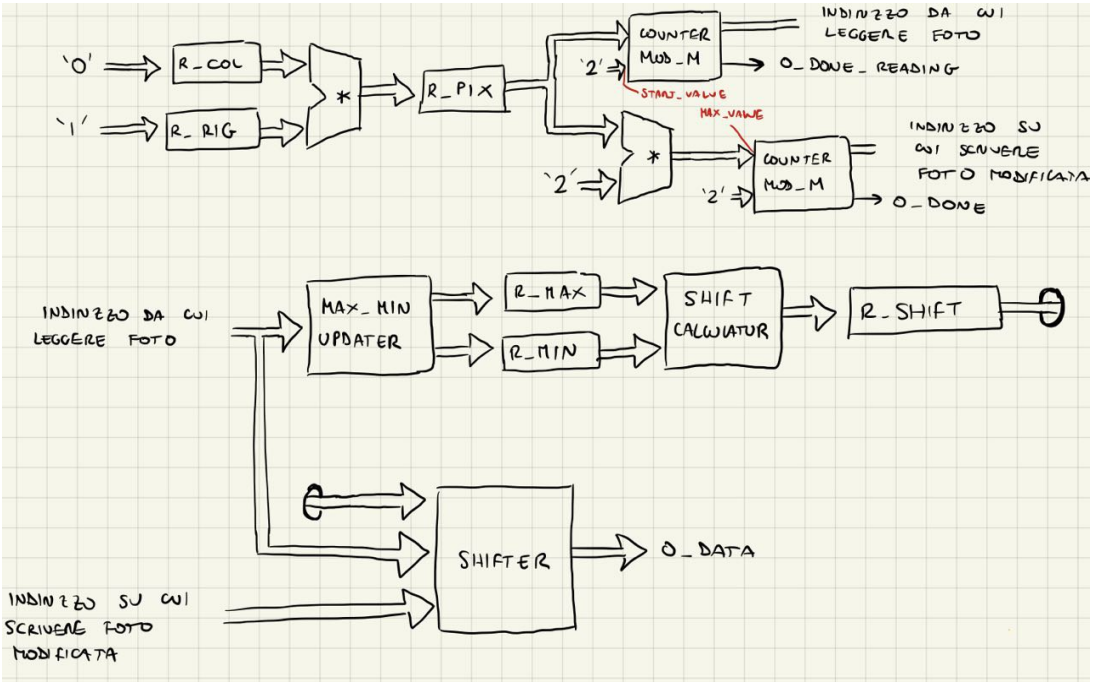
Il modulo quindi dovrà inizialmente calcolare la dimensione dell'immagine, successivamente dovrà trovare il valore MAX_PIXEL_VALUE e MIN_PIXEL_VALUE ed infine scrivere in memoria pixel per pixel l'immagine modificata.

Un funzionamento intuitivo del modulo è presentato nel disegno riportato qui sotto.



2. Scelte progettuali e descrizione del componente

Data la complessità delle operazioni da svolgere abbiamo deciso di non optare per un modulo unico, bensì nell'impiego di più moduli.



I. Contatori

Un elemento fondamentale per il funzionamento del progetto è la gestione degli indirizzi da cui leggere e scrivere.
Per fare ciò, abbiamo deciso di utilizzare due contatori, uno per tenere traccia dell'indirizzo di lettura e uno per quello di scrittura.
I contatori partono quando ricevono un segnale di avvio dallo stesso indirizzo, ovvero quello del primo pixel.
Il contatore utilizzato per tenere traccia dell'indirizzo di lettura, dopo essere giunto all'ultimo indirizzo dell'immagine, riparte da capo. Il secondo contatore, invece, prosegue per tenere traccia dell'indirizzo dove viene scritta l'immagine modificata.
Entrambi i contatori sono pilotati dal clock, ed alla ricezione del segnale di reset vengono riportati all'indirizzo del primo pixel.
Quando il contatore raggiunge l'indirizzo massimo, impostato da un segnale in input, viene alzato un segnale di output per notificare agli altri moduli di essere giunto a destinazione, e successivamente riparte da capo.

II. Calcolatore per il valore di shift:

Una volta avviata la scansione dell'immagine data in input con questo modulo ricaviamo i valori MAX_PIXEL_VALUE e MIN_PIXEL_VALUE dai quali di conseguenza ricaviamo il Delta. Ottenuto il valore di Delta avviamo un controllo a soglia per il valore che andrà in uscita dal modulo cioè quello del SHIFT_LEVEL.
Per implementare questo modulo abbiamo creato tre processi che lavorano in parallelo, i primi due ricevono i_data in input, fanno un continuo confronto per calcolare massimo e minimo, mentre il terzo processo si occupa di calcolare il valore di shift, ricevendo in input massimo e minimo e di conseguenza facendo un controllo a soglia del delta. Tutti e tre i moduli vengono gestiti tramite un segnale che fa da interruttore e permette quindi una gestione manuale della loro attivazione e disattivazione.

III. Flip flop e registri

Durante l'esecuzione si è dimostrato necessario salvare in memoria alcuni valori, ad esempio la dimensione dell'immagine oppure il segnale di enable di un contatore.
Per fare ciò abbiamo deciso di implementare due moduli: un semplice flip flop ed un registro a N bit, con N arbitrario.
Entrambi i moduli sono pilotati dal clock, tuttavia ricevono in input anche un segnale di load, che quando posto in alto permette al registro (o al flip flop) di sovrascrivere il valore immagazzinato con il valore in ingresso.
L'output del flip flop è un segnale std_logic, mentre l'output del registro a N bit è uno std_logic_vector (N-1 downto 0).

IV. Modulo principale

Il modulo principale è la black box che contiene tutti gli altri moduli descritti, e che riceve come input e fornisce in output i segnali descritti dalla specifica del progetto (i_clk, i _rst, i_data, o_address …).
Oltre all'istanziazione dei diversi componenti, abbiamo inserito nell'architettura anche tre processi per modellare il funzionamento di una macchina a stati.
Abbiamo quindi usato un'architettura di tipo mixed, in quanto prevede logiche strutturali, di data-flow e comportamentali (behavioral).
In seguito, una descrizione del comportamento di ciascuno stato:

RST:
Resetta tutti i segnali di enable e quelli utilizzati all'interno del process.

IDLE:
Stato in cui la macchina si trova in attesa di ricevere il segnale di start.

WAIT_START:
In questo stato viene salvato in registro il numero di colonne.

S1:

Stato di stallo in cui viene dato il tempo alla macchina di leggere dall'input.

S2:

In questo stato viene salvato in registro il numero di righe e viene avviato il contatore dell'indirizzo di lettura.

CALC_PIX:

In questo stato viene calcolato il numero di pixels, cioè la dimensione dell'immagine.

S3:

La macchina resta in questo stato finché non raggiunge l'ultimo indirizzo dell'immagine, nel mentre i moduli si occupano di calcolare lo shift value.

STALE_STATE:

Stato di stallo in cui viene dato il tempo alla macchina di recuperare l'ultimo indirizzo e salvare in memoria il valore corretto di shift.

S4:

Primo stato del "loop di scrittura". In questo stato viene preso il valore del pixel da shiftare tramite l'indirizzo dato dal counter di lettura.

PAUSE_COUNTERS:

Secondo stato del "loop di scrittura". In questo stato viene dato il tempo alla macchina di calcolare il pixel shiftato da scrivere in memoria.

S5:

Terzo ed ultimo stato del "loop di scrittura". In questo stato viene scritto in memoria il pixel shiftato, all'indirizzo fornito dal counter di scrittura. Quando il counter di scrittura alza il segnale "done", allora si esce dal ciclo.

S6:

In questo stato viene alzato il segnale o_done, e viene tenuto alto finché il segnale i_start non si abbassa. Successivamente la macchina viene portata nello stato di reset per resettare tutti i moduli, infine la macchina ritorna in IDLE in attesa di un nuovo start.

3. Risultati Sperimentali

La sintesi è stata conclusa con con zero errori e con zero warning, inoltre sono stati risolti tutti i casi di possibili inferred latches grazie all'instanziamento dei flip-flop e dei registri.
Analizzando il report di sintesi si evincono dati importanti quali il valore dello *slack* e quello del *Data Path Delay*, con i seguenti valori:
-Slack: 1.693ns
-Data Path Delay: 8.372ns
Ciò implica che, con il constraint di clock impostato a 10.000ns, la macchina riesce tranquillamente ad eseguire le istruzioni e ad avere del tempo di "pausa", quindi il periodo del clock potrebbe potenzialmente essere più stretto.
Sempre dal report di sintesi seguendo ciò che dice il report utilization evinciamo le risorse consumate dal modulo implementato tra cui il dato importante è (dato dallo Slice Logic) : *"0 Slice Registered as Latch"*.
Infine, il messaggio di avvenuta sintesi:

"Finished Writing Synthesis Report : Time (s): cpu = 00:00:40 ; elapsed = 00:00:42 . Memory (MB): peak = 1064.195 ; gain = 410.629"

4. Simulazioni behavioral

Per testare il progetto abbiamo usato diversi test bench, alcuni generati dal generatore, altri selezionati apposta per stressare determinati casi limite. Qui di seguente vengono riportati i più significativi.

I. Test 128x128

Questo test, se vogliamo il "più standard", simula l'input da memoria di un'immagine di dimensioni massime consentite (128x128).
Il motivo per cui abbiamo deciso di partire con questo test è per capire se la macchina adempie alla richiesta fondamentale di equalizzare una foto.

II. Test min first max last

In questo test abbiamo posto il valore massimo del pixel e il valore minimo rispettivamente come primo pixel e come ultimo pixel.
Questo per verificare che il calcolo del valore massimo e minimo sia corretto anche se il valore viene aggiornato poco prima del passaggio allo stato prossimo, e non solo quando si trova "comodamente" all'interno dell'immagine.

III. Test reset in mezzo all'esecuzione

Come da specifica, durante l'esecuzione la macchina deve ignorare segnali di reset finché non ha finito di elaborare l'output, e quindi finché o_done resta basso.
Il test non ha altre particolari peculiarità se non quella che il segnale i_rst è alzato più volte durante l'esecuzione.

IV. Test zero pixel

Questo test ha lo scopo di stressare un altro caso particolare: ovvero un'immagine senza alcun pixel.
Il caso particolare viene gestito normalmente dalla macchina senza causare problemi.

V. Test più immagini con dimensioni diverse

Questo test verifica un'altra specifica del progetto, ovvero quella di poter ricevere diverse immagini una dopo l'altra (a patto di terminare l'esecuzione di volta in volta) senza la necessità di ricevere segnali di reset tra un'elaborazione e l'altra.
Nella selezione delle immagini ci siamo preoccupati di verificare che le immagini abbiano tutte dimensioni diverse, così da verificare che effettivamente ogni volta vengano calcolate le dimensioni corrette della foto in questione.
Anche in questo caso, il test ha dato esito positivo.

VI. Test più iterazioni sulla stessa immagine

Per verificare che durante l'esecuzione del processo soltanto la porzione di ram designata venisse modificata con l'immagine nuova, e che non avvenissero sovrascrizioni illegali, abbiamo testato anche l'evenienza dell'esecuzione molteplice sulla stessa foto.
La porzione di ram contenente la foto originale non è stata mai sporcata, quindi l'esito del test è stato un successo.

5. Simulazioni post sintesi

Per testare il funzionamento del modulo in post sintesi abbiamo usato gli stessi testbench usati nella simulazione Behavioral.
Tutti i test usati hanno avuto esito positivo.

5. Conclusioni

Il modulo progettato, sulla base delle simulazioni svolte, è in grado di adempiere alla specifica.
Per progettare la soluzione abbiamo prima di tutto steso il datapath del modulo per renderci conto della complessità implementativa, rendendoci conto che era necessario usare più componenti.
Il passo successivo è stato quello di ragionare sulla macchina a stati per capire in che modo gestire le risorse ed i registri definiti in fase di progettazione del datapath.
Un altro motivo per il quale abbiamo deciso di usare un approccio mixed con l'impiego di più moduli è per ottimizzare dal punto di vista temporale, sfruttando il funzionamento in parallelo intrinseco della fpga.