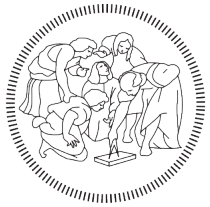
Progetto di Reti Logiche

Stefano Carraro 10583364
Francesco Colabene 10675235

14 maggio 2021



POLITECNICO
MILANO 1863

Dipartimento di Elettronica, Informazione e Bioingegneria

Indice

1 Introduzione

Il progetto sviluppato ha lo scopo di riprodurre l'equalizzazione dell'istogramma di un'immagine.
Il processo prevede l'aumento del contrasto tramite il ripristino dei punti più chiari e più scuri, nonché la successiva distribuzione uniforme dei valori su questi due tipi di pixel.

L'immagine da modificare è salvata in una memoria indirizzata al Byte partendo dall'indirizzo 2: nei primi due si trovano rispettivamente la grandezza delle colonne e delle righe. Il risultato sarà poi scritto in memoria nel primo Byte utile successivo all'immagine data. Le figure avranno una dimensione massima di 128x128 pixel, ed il loro valore cambierà in base al colore in scala di grigi.

L'algoritmo implementato, da specifica, è il seguente:

$$\text{DeltaValue} = \text{MaxPixelValue} - \text{MinPixelValue} \tag{1}$$

$$\text{ShiftLevel} = (8 - \text{FLOOR}(\log_2(\text{DeltaValue} + 1))) \tag{2}$$

$$\text{TempPixel} = (\text{CurrentPixelValue} - \text{MinPixelValue}) << \text{ShiftLevel} \tag{3}$$

$$\text{NewPixelValue} = \text{MIN}(255, \text{TempPixel}) \tag{4}$$

Tramite la differenza tra massimo e minimo valore dei pixel si calcola un parametro chiamato ShiftLevel. Esso sarà utilizzato per ricavare i valori dei pixel ricalibrati.

2 Architettura

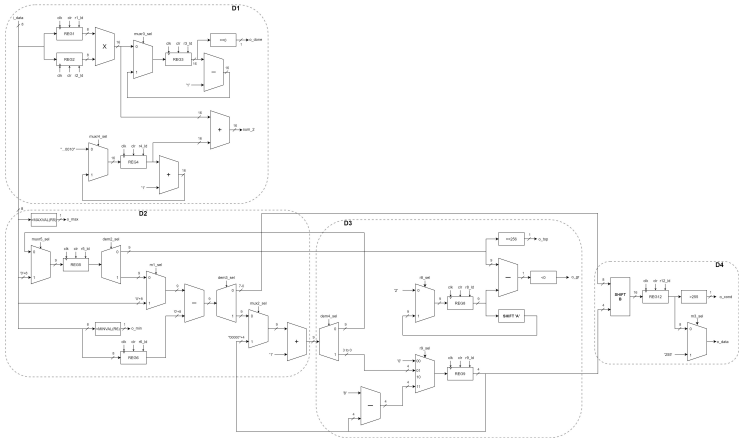
Il funzionamento del circuito equalizzatore consiste in:

1. leggere la dimensione dell'immagine;
2. leggere ogni pixel per trovare massimo e minimo;
3. calcolare il livello di shift, basandosi sui dati raccolti;
4. rileggere nuovamente tutta l'immagine scrivendo man mano nella memoria i pixel modificati.

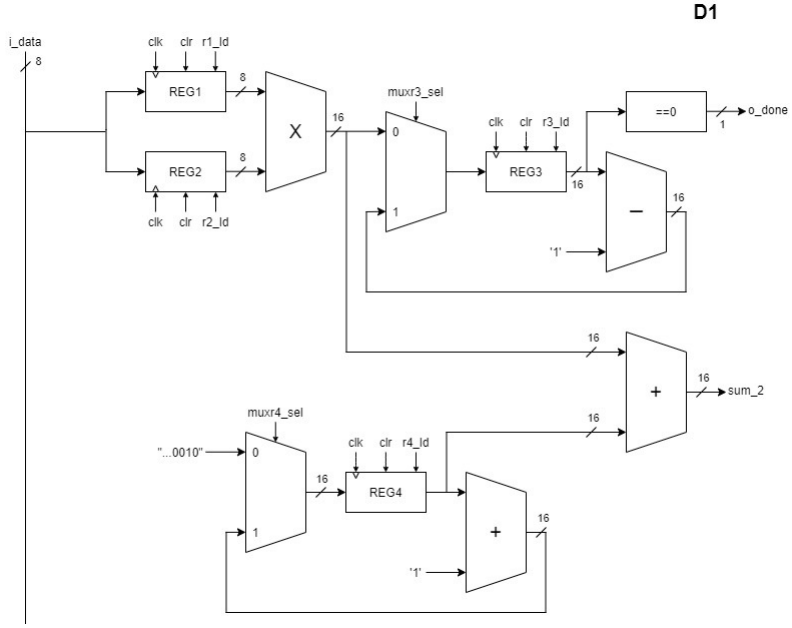
Di seguito figurano il datapath e la macchina a stati, successivamente studiati nella loro divisione in blocchi d'interesse.

2.1 Datapath

Per via delle grosse dimensioni del datapath, alla fine del documento è presente un'immagine più grande e visibile.



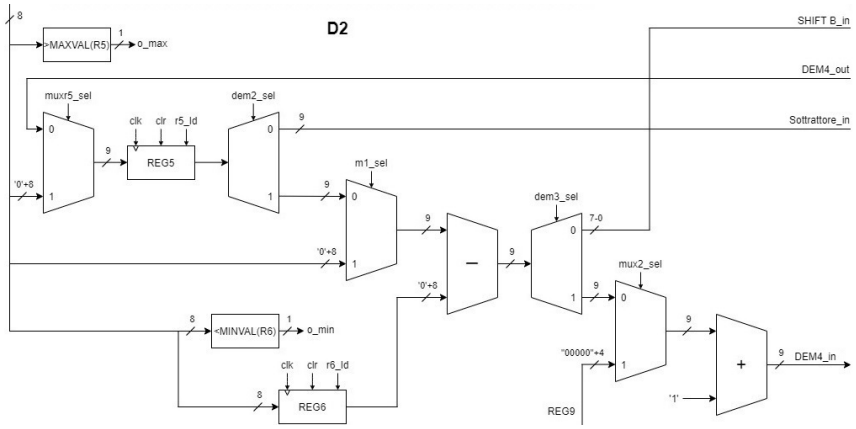
2.1.1 D1



Questa sezione del datapath è divisa in due parti: una contiene la circuiteria atta a salvare il numero di righe e colonne (tramite REG1 e REG2), caricando

successivamente in REG3 il numero di pixel dell'immagine. L'altra, invece, salva l'indirizzo di memoria della RAM utile alla macchina in quel momento nel REG4. Per semplicità di progettazione è stato deciso di utilizzare il registro 4 come sorgente di tutti gli indirizzi di memoria della RAM (o_en attivo), se non diversamente specificato. Dopo una prima inizializzazione di questo registro, il suo valore verrà incrementato di 1 ogni ciclo di lettura tramite il sommatore. Quest'ultimo, sommato al contenuto di REG4, costituisce l'indirizzo di memoria (sum_2 nell'immagine) in cui salvare il Byte processato. Altra funzione di REG3, tramite il sottrattore, è quella di tener conto del numero di cicli rimanenti.

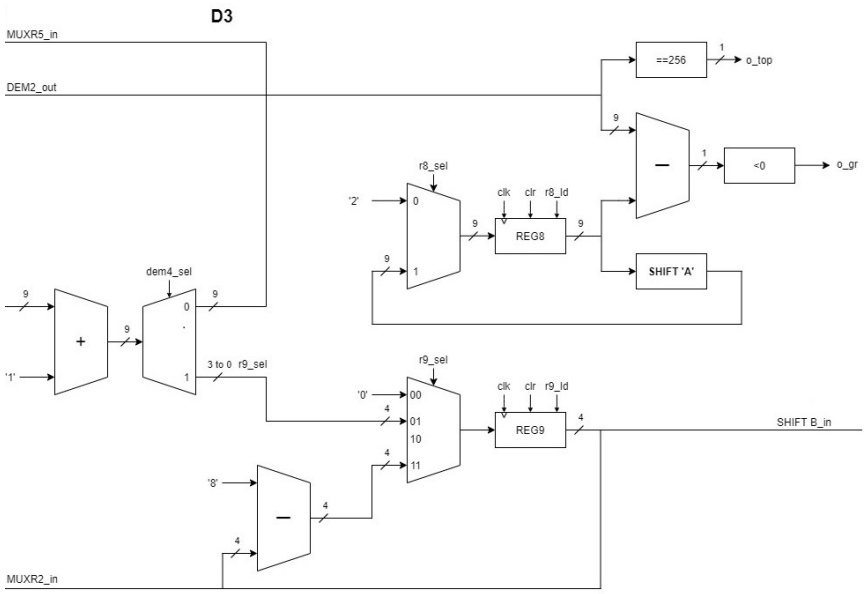
2.1.2 D2



Questo macro blocco si occupa della ricerca del massimo e minimo valore dei pixel, i quali verranno salvati rispettivamente in REG5 e REG6. Nella parte destra dell'immagine è presente una rete adibita a calcolare il parametro DeltaVal+1 e, successivamente, il pixel temporaneo (3). È di rilevanza l'architettura scelta, dato che si fa un buon utilizzo del sommatore, del sottrattore e di REG5 (usato per immagazzinare DeltaVal+1), riutilizzandoli per calcolare valori diversi in cicli differenti. Come si vedrà nella prossima sezione, il sommatore sarà utile anche per il calcolo del logaritmo.

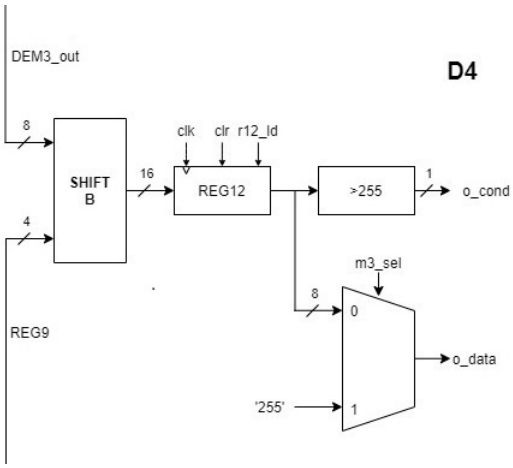
2.1.3 D3

Questa è la parte di circuito in cui avviene il calcolo del logaritmo (2). In particolare, nel registro 8 si trova il valore della potenza del 2 partendo da 2^1, che verrà sottratta ad ogni ciclo al DeltaVal+1. Fino a che il risultato è un numero positivo, il blocco "shift A" raddoppia il contenuto del registro 8. Quando la sottrazione genera un numero negativo (o_gr=1), nel registro 9 troveremo il risultato. Qui si trova l'esponente della potenza del 2 che stiamo cercando, partendo da 0, che verrà incrementata ogni ciclo. Perché la computazione parte



da 2^1 invece che da 2^0? Partire da quest'ultima non porta alcun vantaggio: dalla (1) si evince che il valore minimo di DeltaVal+1 sarà 1, quindi $1 - 2^0 = 0$ che non è negativo. Avendo delle coppie $(2^{k+1}, k)$ che simboleggiano i valori presenti nei registri 8 e 9, l'approssimazione per difetto è già inclusa nel risultato. Per esempio, assumendo che DeltaVal+1 sia 18, la sequenza di cicli risulta essere $(2^1, 0) \rightarrow (2^2, 1) \rightarrow (2^3, 2) \rightarrow (2^4, 3) \rightarrow (2^5, 4)$, quindi il risultato sarà 4 ($\log_2(18) = 4.16$). Questo sistema permette di calcolare in modo corretto anche valori limite come tutte le potenze del due (infatti se DeltaVal+1 $= 2^m$, al ciclo $(2^m, m-1)$ la sottrazione è nulla, non negativa, quindi è necessario un ulteriore ciclo per ottenere $(2^{m+1}, m)$). Alla fine della computazione, non resta che sottrarre a 8 il risultato per ottenere il valore dello ShiftValue (2). Il caso in cui DeltaVal+1 è uguale a 256 invece è trattato separatamente: c'è un confronto il cui risultato (o_top=1) determina se la macchina si trova in questo caso particolare. Se così fosse, verrebbe selezionato l'ingresso "00" del mux che precede il registro 9, e caricato tale valore. Corrisponde infatti ad $8 - 8 = 0$.

2.1.4 D4



Nella parte finale del datapath si trova lo "shift B": quest'ultimo ha il compito di calcolare il valore finale del TempPixel (3). Il valore del pixel corrente sottratto del minimo viene fatto slittare a sinistra dello ShiftValue calcolato precedentemente (2.1.3). Il risultato verrà salvato nel REG12. L'uscita del registro verrà poi confrontata con il numero 255 e nel caso in cui sia minore (o_cond=0), il NewPixelValue salvato sarà quello del TempPixel (4). In caso contrario, il valore caricato in memoria sarà 255.

2.2 Macchina a stati

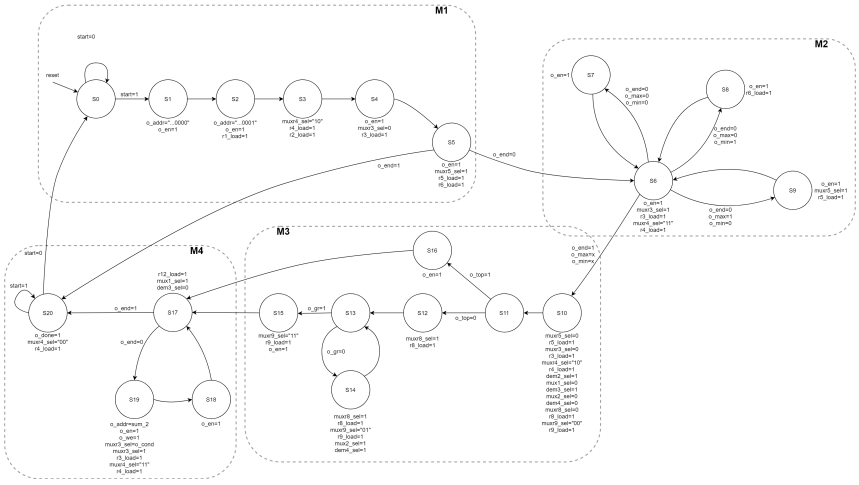
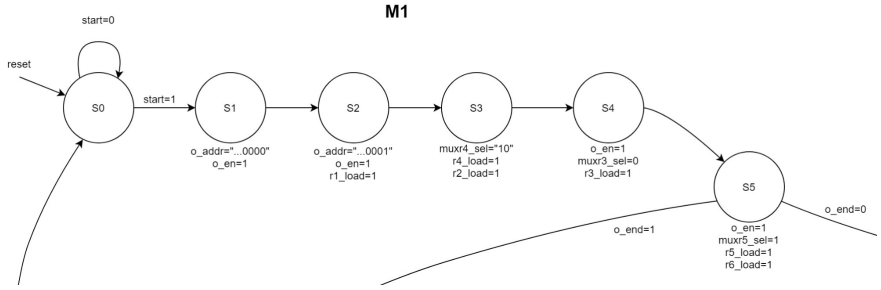


Figura 1: FSM completa

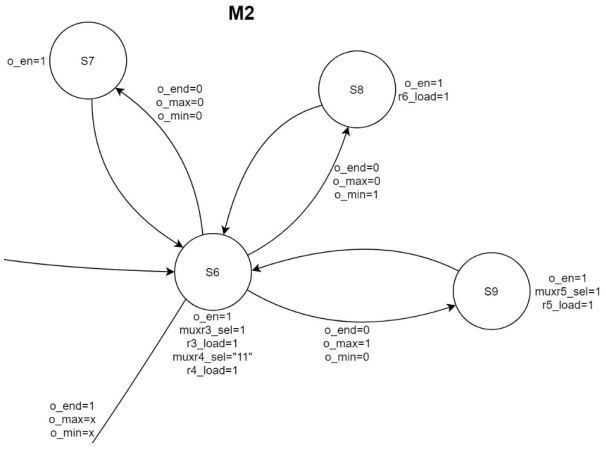
La FSM (Finite State Machine) è composta da 20 stati e comanda tutti i segnali di controllo necessari al datapath per funzionare nella maniera corretta.

2.2.1 M1



Nel primo stato della macchina ($\mathcal{S}_0$), questa rimane in attesa del segnale di start, dopodiché negli stati $\mathcal{S}_1$ e $\mathcal{S}_2$ chiede alla memoria il valore dei primi due indirizzi. Essendo la memoria di tipo sincrono in lettura, il dato viene fornito al periodo di clock successivo alla richiesta. In $\mathcal{S}_2$ salva il valore delle colonne nel registro 1 mentre in $\mathcal{S}_3$ salva quello delle righe nel registro 2. Nello stesso stato inizializza al valore corretto il registro 4 e nel successivo ($\mathcal{S}_4$) carica il registro 3, con il risultato della moltiplicazione fra i registri 1 e 2. Qui viene anche attivata la memoria (si ricorda che il valore di default dell'indirizzo di memoria è il valore del registro 4) per salvare allo stato $\mathcal{S}_5$ il primo pixel dell'immagine. Per scegliere il prossimo stato è presente un controllo sulla dimensione dell'immagine: se risulta essere nulla (o_end = 1) si passerà direttamente a fine computazione allo stato $\mathcal{S}_{15}$.

2.2.2 M2

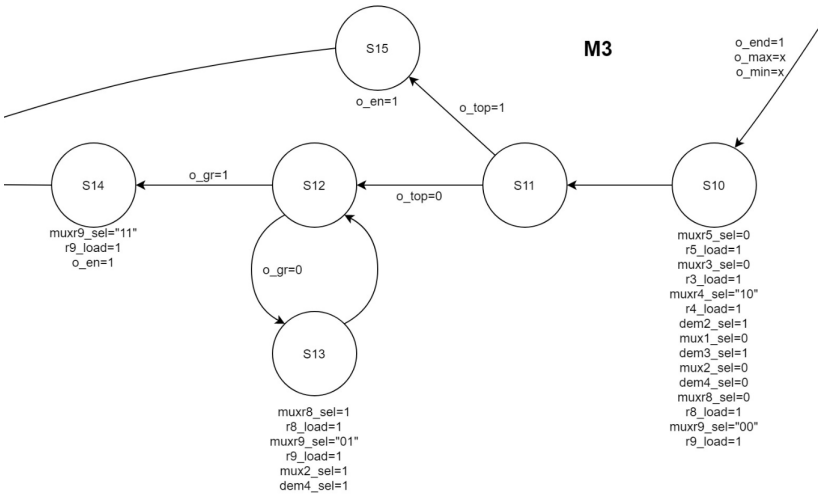


Qui inizia il ciclo per il calcolo del valore massimo e minimo dell'immagine: in $\mathcal{S}_6$ c'è il valore del bit corrente su i_data, la memoria viene attivata per il prossimo stato (nel caso di scrittura di nuovo massimo o minimo) e vengono aggiornati i valori dei registri 3 e 4 (rispettivamente del contatore di cicli rimanenti e dell'indirizzo). Negli altri stati, oltre ad attivare la memoria per il prossimo dato, la macchina fa una delle seguenti tre cose:

- Stato $\mathcal{S}_7$: non trova né un massimo né un minimo, quindi non aggiorna nessun valore;
- Stato $\mathcal{S}_8$: ha trovato un nuovo minimo quindi sovrascrive il registro 6;
- Stato $\mathcal{S}_9$: ha trovato un nuovo massimo quindi sovrascrive il registro 5.

Quando o_end=1 l'immagine è finita, quindi procede allo stato $\mathcal{S}_{10}$.

2.2.3 M3



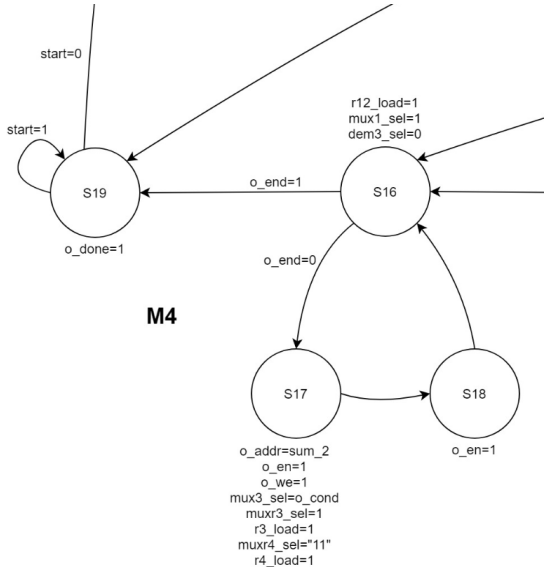
Nello stato $\mathcal{S}_{10}$ avvengono una serie di cambiamenti chiave:

- Tutti i selettori sono impostati per permettere di sovrascrivere il valore del registro 5 con DeltaVal+1, da utilizzare nel calcolo del logaritmo, vedi (1), (2);
- Vengono ripristinati i valori iniziali del registro 3 e 4 per ricominciare da capo il conteggio;
- I registri 8 e 9 vengono inizializzati per il calcolo del logaritmo.

Lo stato $\mathcal{S}_{11}$ è necessario per aggiornare tutti i registri; viene poi controllato il valore all'interno del registro 5. Se quest'ultimo è 256 (o_top=1) significa che

l'immagine fornita in ingresso copre già in partenza tutta la dinamica disponibile, per cui non serve equalizzare. In questo caso, si passa dallo stato $\mathcal{S}_{15}$ evitando i passaggi che calcolano il logaritmo. Altrimenti entra nello stato $\mathcal{S}_{12}$ in cui controlla se o_gr=1, che comunica la fine del calcolo del logaritmo ((2.1.3) per la spiegazione del calcolo). Nello stato $\mathcal{S}_{13}$ viene incrementato il valore sia sul registro 8 che 9, nel primo incrementandolo di 1, nel secondo moltiplicandolo per due. Ritorna poi allo stato $\mathcal{S}_{12}$ dove verifica nuovamente la condizione. Se invece la condizione precedente è vera, allora procederà allo stato $\mathcal{S}_{14}$ e nel registro 9 sarà presente il risultato del logaritmo già approssimato per difetto. Secondo la (2), il valore del registro 9 verrà sovrascritto con 8−DeltaVal.

2.2.4 M4



Nello stato $\mathcal{S}_{16}$ è presente su i_data il valore del primo pixel dell'immagine, che viene caricato nel registro 12 selezionando coerentemente i valori dei selettori. Qui comincia il ciclo di lettura, modifica del bit (3) e scrittura in memoria del valore modificato se minore di 255 (4). Esattamente come succedeva nel ciclo di M2, viene controllato o_end per capire quando tutta l'immagine è stata modificata. Nello stato $\mathcal{S}_{17}$ la macchina modifica l'indirizzo di memoria su cui deve scrivere (si ricorda che sum_2 è la somma del registro 4 e la dimensione dell'immagine), la memoria viene impostata in scrittura, aggiorna i registri 3 e 4 e imposta il selettore del mux3 secondo il valore di o_cond. Successivamente allo stato $\mathcal{S}_{18}$ viene attivata la memoria per la lettura del prossimo valore. Una volta finita l'immagine, allo stato finale $\mathcal{S}_{19}$ viene impostato o_done=1 e, come da specifica, rimane in attesa del segnale di start per una prossima computazione.

2.3 Scelte implementative

Quanto spiegato precedentemente deve essere implementato in linguaggio VHDL. Sono state effettuate alcune scelte particolarmente importanti su cui vogliamo soffermarci.

Si è scelto di suddividere il progetto in due entity principali, il datapath e la macchina a stati. Nel primo sono presenti, oltre al datapath stesso, alcune entity di supporto: un moltiplicatore con ingressi a 8 bit e dei demux a 2 ingressi. La FSM è composta dalla funzione stato prossimo che sceglie quale sarà lo stato seguente e la funzione di stato che specifica come vengono comandati i segnali di controllo. È stato evitato l'utilizzo di "clk'event and clk = '1'" in favore di "rising_edge". La principale differenza risiede nel loro comportamento. Il primo controlla che ci sia una transizione, e che il valore finale sia '1', includendo quindi tutte le possibili transizioni non pulite (come 'Z' => '1', etc). La seconda implementazione invece controlla che il valore iniziale sia '0' e che quello finale sia '1'.

3 Risultati sperimentali

3.1 Sintesi

Il report di sintesi di Vivado effettuato sulla board xc7a200tfbg484-1 presenta l'utilizzo di 97 registri e 259 LUTs. Il percorso critico del sistema è di 12.201 ns causato dal moltiplicatore situato nel blocco M1.

3.2 Simulazioni

I test e le simulazioni sono una parte fondamentale della corretta progettazione di un oggetto. Questi permettono di creare situazioni ad hoc per verificare il funzionamento dello stesso. Sono utili per capire se tutto funziona sia nelle situazioni normali (in questo caso una normale immagine in ingresso), sia in quelle particolari (per esempio, immagini con un valore massimo minimo identici). In questo caso specifico, questo è reso possibile grazie a delle asserzioni in VHDL: permettono di assegnare un livello di gravità ad un problema in base al risultato di alcune condizioni, fino a bloccare l'esecuzione del test in situazioni critiche.

3.3 TestBench

Sono stati creati quattro testbench, ognuno contenente al suo interno otto test, che racchiudono una serie di circostanze normali e non, per garantire la corretta esecuzione dell'algoritmo in ogni possibile caso.

3.3.1 Shift

Il primo testbench permette di testare tutti i possibili livelli di shift: dal caso in cui tutti i valori dell'immagine sono identici fra loro, a quello in cui l'immagine copre già in partenza tutta la dinamica disponibile di 256 valori, e non deve far altro che riscrivere l'immagine originale.

3.3.2 Zeri

Il secondo testbench, oltre a test normali, prova diversi casi limite: immagini di grandezza 0x0, 0xn, nx0, ingressi tutti uguali e nulli.

3.3.3 Reset

Il terzo testbench inserisce durante la computazione alcuni reset non desiderati. In particolare, nel primo test il reset avviene durante la fase M4 della FSM, dopodiché viene effettuato un controllo sulla scrittura della memoria. Un altro test è un reset, sempre durante la computazione, e la successiva equalizzazione della stessa immagine.

3.3.4 Dimensione massima

Il quarto ed ultimo testbench ha un'immagine di grandezza massima che è 128x128.

4 Conclusioni

In conclusione, si sottolinea come sono state ridotte le risorse utilizzate e il numero di stati. Si è deciso di salvare entrambi i valori di riga e colonna nei REG1 e REG2, perché utili nella seconda fase di scrittura dell'immagine modificata: sia per ricominciare il conteggio dei pixel, sia per calcolare a ogni ciclo l'indirizzo di scrittura. La serie di mux e demux nel blocco M3 è nata dall'idea di riutilizzare alcune parti del circuito: il sommatore viene usato per il calcolo del DeltaVal+1 (1) e per l'incremento del REG8 durante il calcolo del logaritmo. Analogamente, il sottrattore è usato per il calcolo del DeltaVal+1, ma anche per il calcolo dei pixel temporanei. Ci sono inoltre due registri riutilizzati per altri scopi grazie all'utilizzo dei mux:

- REG5, dopo aver calcolato il DeltaVal+1 il MaxPixelVal sarà superfluo e quindi verrà scartato (3).
- REG8, il risultato del logaritmo non sarà più utile dopo aver ottenuto lo ShiftLevel (2).

5 Disegno Datapath