Progetto di Reti Logiche

Anno Accademico 2020-2021

Filippo Ranieri Pantaleone 10664253
Elisa Parlati 10656371

Docente **Gianluca Palermo**

POLITECNICO
MILANO 1863

Sommario

1. Introduzione

Questo progetto si ispira al metodo di equalizzazione dell'istogramma di un'immagine, con l'obiettivo di ricalibrare il contrasto distribuendone i valori di intensità su tutto l'intervallo ammesso. Le immagini fornite in input sono in scala di grigi a 256 livelli, con una dimensione massima di 128x128 pixel. Ogni immagine è letta sequenzialmente pixel-per-pixel da una memoria RAM presente su un testbench mappato sul modulo in esame. L'immagine risultante dall'elaborazione viene salvata sulla medesima memoria RAM, a partire dall'indirizzo di memoria successivo all'immagine originale.

Il calcolo del nuovo valore che un pixel deve assumere è stabilito attraverso il seguente algoritmo:

```
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 il massimo e minimo valore dei pixel dell'immagine; `CURRENT_PIXEL_VALUE` è il valore del pixel da trasformare; `NEW_PIXEL_VALUE` è il valore calcolato del nuovo pixel.

Immagine non equalizzata fornita in input

Immagine dopo l'equalizzazione dell'istogramma

2. Architettura

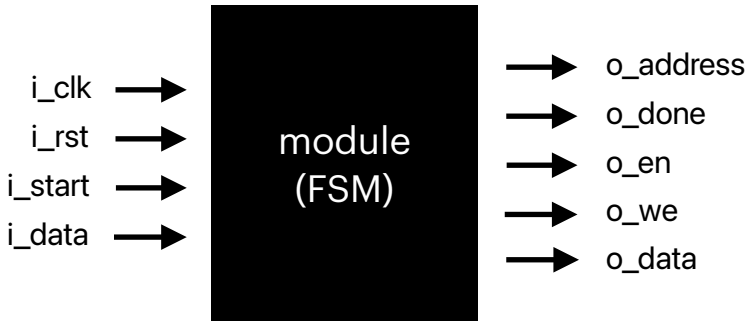


Figura 1 - Rappresentazione semplificata del modulo

Il progetto consiste di un solo modulo figura 1 che svolge tutte le operazioni necessarie all'elaborazione dell'immagine in input.

Questo modulo implementa una Final State Machine figura 2 per la visione complessiva della FSM all'interno di un solo processo sensibile esclusivamente al segnale di clock fornito dal testbench. La FSM è composta dai seguenti stati:

A. **`FIRST`** lo stato iniziale in cui si trova la FSM all'avvio;

B. **`RESET`** inizializza il contenuto dei segnali del modulo (interni e di output).
   La FSM transita su questo stato ogni qualvolta il segnale in input di reset `i_rst` sia alzato, oppure quando `i_start` viene abbassato mentre la FSM si trova in **DONE**;

C. **`WAITING`** mantiene la FSM in attesa finché il segnale di start `i_start` non è alto;

D. **`START`** prepara la FSM alla lettura del primo indirizzo di memoria della RAM;

E. **`READCOL`** salva nel segnale interno `N_COL` il numero di colonne di pixel salvato nella prima cella di memoria e prepara la lettura della cella successiva;

F. **`READRIG`** salva nel segnale interno `N_RIG` il numero di righe di pixel salvato nella seconda cella di memoria e prepara la lettura della cella successiva;

G. **`LOADTOT`** calcola il numero totale di pixel presenti nell'immagine da leggere tramite il calcolo `N_COL * N_RIG` il cui risultato è salvato nel segnale interno `TOT_PIXEL`.
   Prepara la lettura della terza cella di memoria;

H. **`MAXMIN`** salva il valore massimo e minimo tra i pixel dell'immagine.
   La FSM si mantiene su questo stato leggendo un pixel per ciclo di clock.
   La transizione allo stato successivo avviene quando il valore di `TOT_PIXEL` (decrementato ad ogni ciclo di lettura) viene posto a zero;

I. **`LOADDELTA`** calcola la differenza tra valore massimo e minimo salvati nello stato precedente nel segnale `DELTA_VALUE`. Re-imposta i valori di conteggio per puntare al primo pixel dell'immagine;

J. **LOADSHIFT** ripristina il valore di TOT_PIXEL precedente alla decrementazione e calcola il valore del segnale SHIFT_LEVEL associando ogni valore assumibile da quest'ultimo ad un range di valori di DELTA_VALUE. Predispone per la lettura della terza cella di memoria;

K. **CALCULATENEWVALUE** se ci sono pixel da leggere, salva temporaneamente il pixel corrente, lo clona, sottrae il valore MIN_PIXEL_VALUE e applica uno shift di SHIFT_LEVEL bit verso sinistra. Poi predispone per la scrittura nella cella di memoria (il cui indirizzo è calcolato sommando l'indirizzo dell'immagine corrente al numero totale di pixel di quest'ultima).
Infine la FSM transiziona verso lo stato **HASOVERFLOWED**.
Se non ci sono altri pixel da leggere, la FSM transiziona verso lo stato **DONE**.

L. **HASOVERFLOWED** controlla se sul pixel corrente (non shiftato) si potrebbe verificare l'overflow durante lo shift di SHIFT_LEVEL bit verso sinistra. In base al valore assunto da SHIFT_LEVEL (che deve essere maggiore di zero, altrimenti non si verifica overflow) viene eseguito un loop ad esso associato. Nel loop si verifica se uno qualunque tra i SHIFT_LEVEL bit più significativi è pari a uno. In tal caso, il nuovo pixel da scrivere ha un valore necessariamente maggiore di 255 (a causa dell'operazione di shift), quindi viene alzato il segnale interno OverFlow per segnalare questo evento. Altrimenti il segnale rimane basso. Infine si predispone la memoria RAM alla scrittura alzando il segnale di output o_we.

M. **WRITENEWVALUE** scrive il nuovo valore calcolato negli stati precedenti sul segnale di output o_data: se OverFlow è alto, allora viene scritto 255. Altrimenti si riporta il valore del pixel shiftato. In ogni caso OverFlow viene abbassato.

N. **SYNC** blocca la scrittura in memoria abbassando il segnale di output o_we e predispone la lettura della cella di memoria successiva dell'immagine originale.

O. **DONE** alza il segnale in output o_done per segnalare che è terminata la computazione della precedente immagine se il segnale in input i_start è alto. Altrimenti abbassa o_done e, per decisione progettuale, porta la FSM sullo stato di **RESET** per preparare il modulo ad una eventuale computazione successiva.

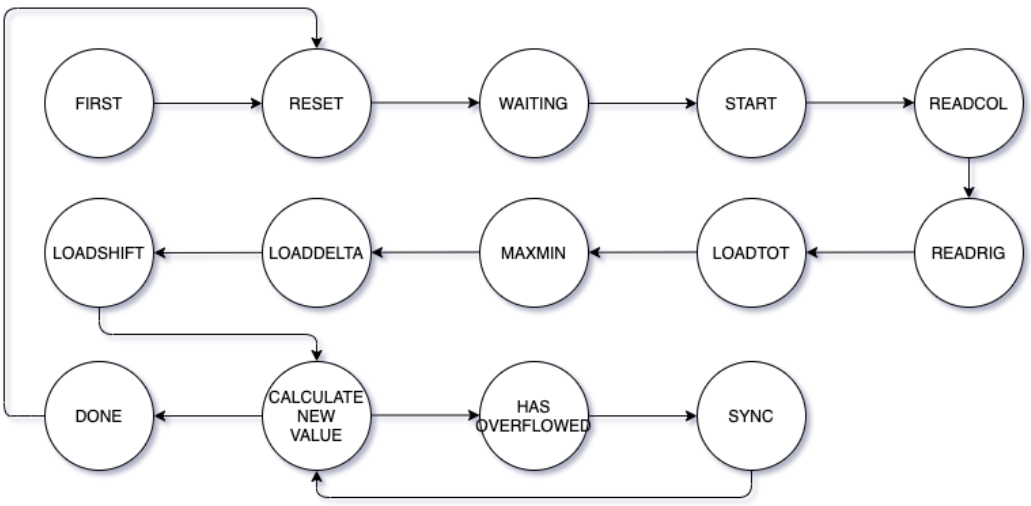


Figura 2 - Stati della FSM che governa il modulo
(si omette per semplicità gli archi da ogni stato verso quello di RESET)

2.1 Synced

Durante la fase di progettazione si è reso evidente il problema della sincronizzazione delle richieste di lettura/scrittura dal modulo verso il testbench. I dati vengono resi disponibili dopo due cicli di clock, comportando un ritardo nella propagazione dei dati richiesti e errori certi di elaborazione se non dovesse essere prevista una corretta sincronizzazione, siccome il modulo stabilisce in autonomia quando passare da uno stato all'altro.

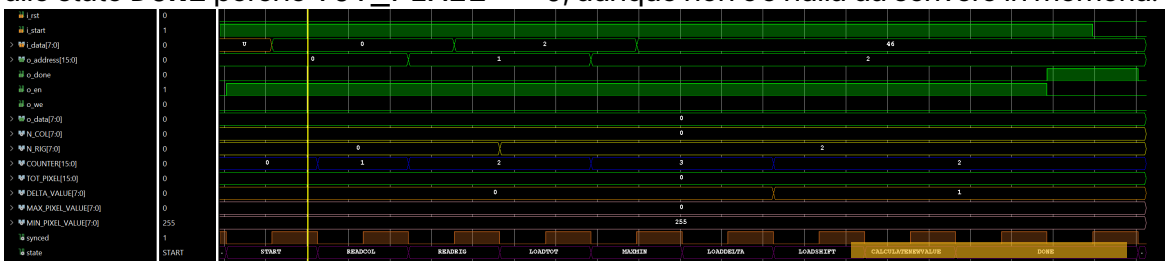
`synced` è un segnale che viene abbassato ogni volta che il modulo esegue delle operazioni proprie di uno specifico stato. Nel ciclo di clock successivo viene ignorata la funzione principale dello stato corrente e viene rialzato `synced`. In questo modo il modulo attende il tempo necessario a vedere i dati aggiornati in seguito alla propria richiesta.

3. Test significativi

Segue una descrizione accompagnata dal eventuale diagramma d'onda di alcuni test utilizzati e ritenuti più significativi ai fini di questa relazione. Alcuni verificano il comportamento di fronte a corner-cases relativi alla dimensione dell'immagine.

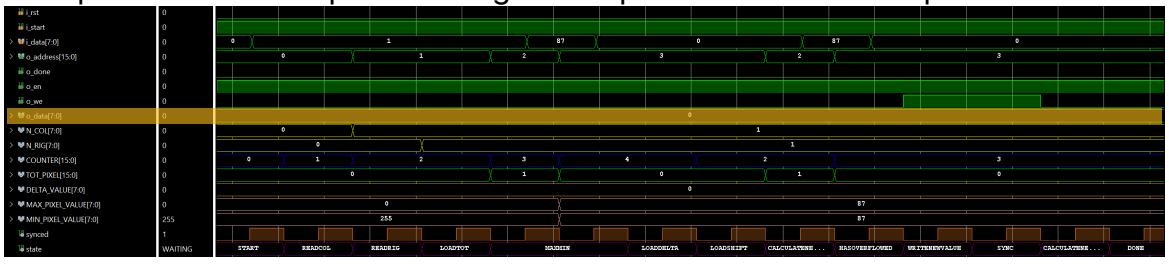
3.1 zero pixels

Questo testbench fornisce in ingresso nelle prime due posizioni della memoria valori di `N_RIG` e `N_COL` tali che `TOT_PIXEL` risulti essere pari a zero. Di conseguenza i valori di minimo e massimo pixel, `DELTA_VALUE` e `SHIFT_LEVEL` vengono ignorati perché, una volta entrata la FSM nello stato **`CALCULATENEWVALUE`**, avviene la transizione allo stato **`DONE`** perché `TOT_PIXEL = 0`, dunque non c'è nulla da scrivere in memoria.



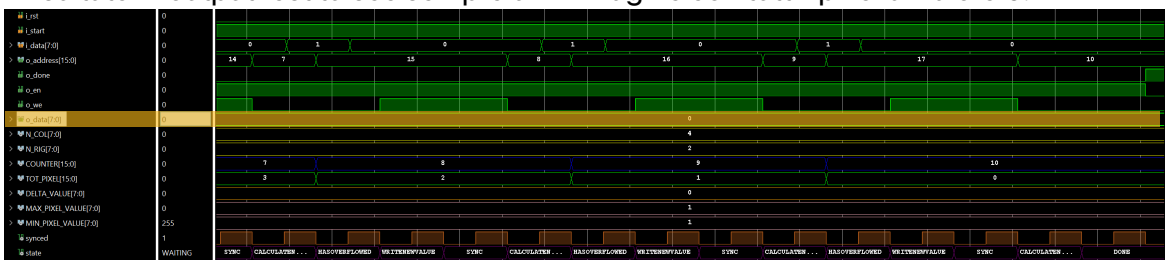
3.2 one pixel

Questo testbench fornisce in ingresso nelle prime due posizioni della memoria valori di `N_RIG` e `N_COL` tali che `TOT_PIXEL` risulti essere pari a uno. Di conseguenza `MAX_PIXEL_VALUE` e `MIN_PIXEL_VALUE` assumono il valore dell'unico pixel da cui è composta l'immagine, quindi `DELTA_VALUE = 0` e `SHIFT_LEVEL = 8`. Il risultato in output restituisce sempre un'immagine a un pixel di dimensione 1 e pixel di valore 0.



3.3 each pixel equals `value`

Questo testbench fornisce in ingresso un'immagine contenente tutti pixel di valore costante `value`. Di conseguenza
`MAX_PIXEL_VALUE = MIN_PIXEL_VALUE = value`,
`DELTA_VALUE = 0` e `SHIFT_LEVEL = 8`.
Il risultato in output restituisce sempre un'immagine con tutti i pixel di valore 0.



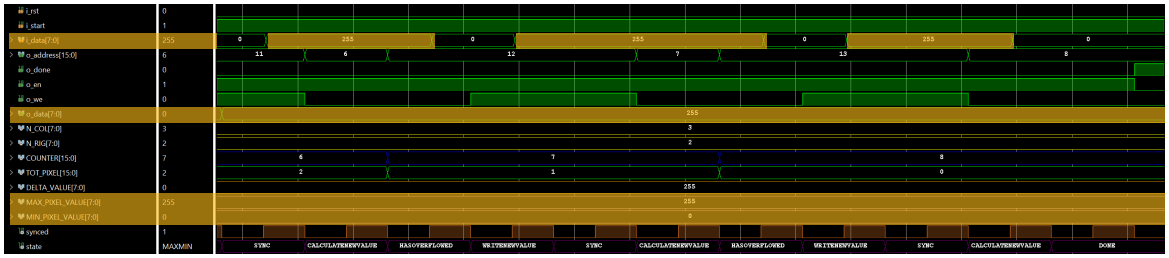
Si evidenzia il fatto che `o_data` rimane sempre zero nonostante si richieda la scrittura in celle di memoria diverse

3.4 maximum size (128x128)

Sono stati generati casualmente dei test contenenti immagini di dimensione 128x128 che vengono correttamente elaborate.

3.5 full range

Questo testbench fornisce in ingresso un'immagine di qualunque dimensione e tale che `MIN_PIXEL_VALUE = 0`, `MAX_PIXEL_VALUE = 255`, `DELTA_VALUE = 255` e `SHIFT_LEVEL = 0`. In questo modo l'immagine in output corrisponde esattamente all'immagine in input.



Sono evidenziati i valori di massimo e minimo, il valore di `o_data` che rimane costante perché gli ultimi pixel letti valgono tutti 255.

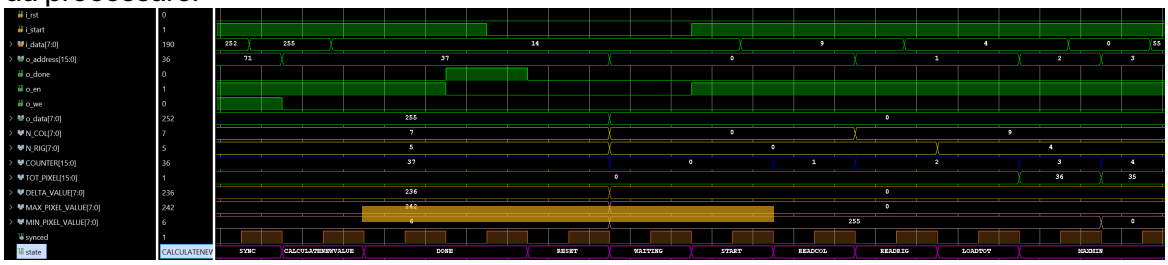
3.6 reset in the middle

Questo testbench testa il comportamento del modulo a fronte del sollevamento del segnale `i_rst`. Il modulo viene correttamente inizializzato e le informazioni relative alla computazione corrente vengono definitivamente eliminate.

Sono evidenziati la transizione verso lo stato di **RESET** e il sollevamento del segnale di reset `i_rst`.

3.7 multiple images

Questo testbench testa il comportamento del modulo quando sottoposto a più computazioni di immagini diverse consecutive. Alla fine di ogni computazione la FSM transita sullo stato di **RESET** (per scelta progettuale) in modo da eliminare ogni riferimento alla computazione precedente ed accettare una nuova immagine da processare.



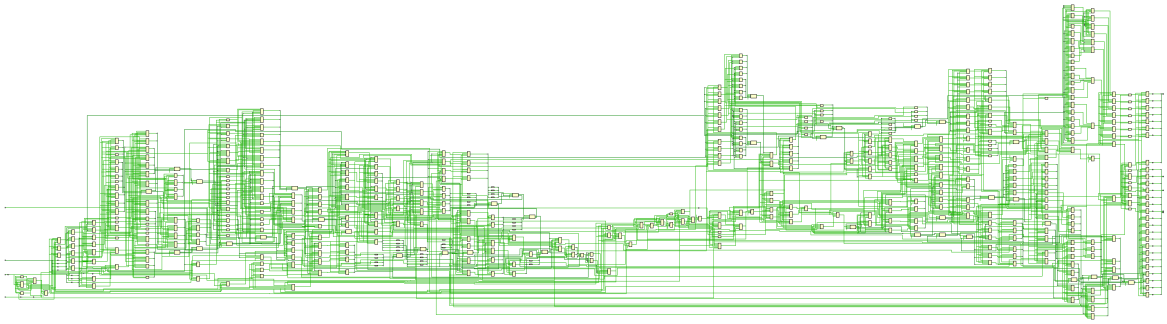
Sono evidenziate le transizioni che avvengono in seguito al termine di una computazione.

4. Report di sintesi

Sfruttando la funzione di sintesi di Vivando 2021.1 su board 7k70tfbv676-1 si ottiene il seguente report di utilizzo:

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	302	0	0	41000	0.74
LUT as Logic	302	0	0	41000	0.74
LUT as Memory	0	0	0	13400	0.00
Slice Registers	173	0	0	82000	0.21
Register as Flip Flop	173	0	0	82000	0.21
Register as Latch	0	0	0	82000	0.00
F7 Muxes	0	0	0	20500	0.00
F8 Muxes	0	0	0	10250	0.00

Report di utilizzo.



Schematic del modulo in post-sintesi.

La post-sintesi è soddisfacente data l'assenza di latches e per il fatto che tutti i testbench utilizzati terminano correttamente l'esecuzione durante la behavioral simulation sia prima della sintesi sia in post-sintesi.

5. Conclusioni

Constatiamo che la FSM è stata una scelta adatta alla specifica perché separa in modo netto le operazioni all'interno di stati ben definiti. Ciò consente di sviluppare ulteriormente il modulo al fine di aggiungere nuove funzionalità definendo nuovi stati e correggendo le transizioni esistenti.

Questo progetto è stato molto interessante e utile perché ci ha permesso di imparare ad utilizzare un software ad uso commerciale che prevede un tipo di programmazione diverso da quello a cui siamo abituati solitamente.