Prova Finale - Progetto di Reti Logiche

Martina Franzè - Giuseppe Marchesani
10656643 - 10660217

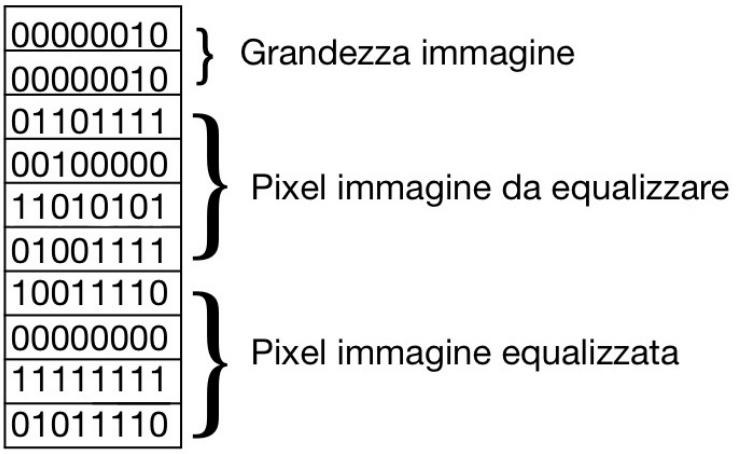
Anno 2020-2021

Indice

1 Introduzione

Il progetto consiste nel descrivere l'architettura di un equalizzatore di immagini semplificato: le immagini sottoposte ad equalizzazione saranno in scala di grigio a 256 livelli. Al componente è richiesto di:

- Leggere la dimensione dell'immagine la quale è definita mediante 2 byte memorizzati nei primi due indirizzi di memoria. Il primo byte (indirizzo 0) indica il numero delle colonne, il secondo byte indica il numero di righe (indirizzo 1).
- Leggere, poi, l'immagine memorizzata il cui primo pixel risiede in memoria all'indirizzo 2.
- Scrivere in memoria, immediatamente dopo l'immagine originale, l'immagine equalizzata.



Al componente è richiesto, anche, di poter codificare più immagini.
Il componente comincerà ad elaborare quando verrà portato il segnale d'ingresso START a 1, il quale rimarrà alto fino a che il segnale DONE, che notifica la fine dell'elaborazione, non verrà portato alto.
Il segnale DONE rimarrà alto fin quando il segnale d'ingresso START non verrà riportato a 0.
Per codificare una seconda immagine, dovrà essere dato un nuovo segnale START, ma purchè questo avvenga è necessario che DONE sia stato prima riportato a 0.

Di seguito l'algoritmo implementativo alla base dell'equalizzazione.

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)

2 Architettura

2.1 Automa a stati finiti

Per affrontare la risoluzione di questo problema è stato pensato un Automa a Stati Finiti con i seguenti stati:

- **IDLE**: E' lo stato iniziale in cui si rimane finchè il segnale d'ingresso START non viene portato a 1. Qui tutti i segnali necessari alla computazione vengono inizializzati.
- **CALC_NUMPIXEL**: In questo stato viene letta da memoria la dimensione di colonna e la dimensione di riga in modo da calcolare la grandezza dell'immagine e quindi il numero totale di pixel.
- **CALC_DELTA**: Scorre tutti i pixel in memoria in modo da calcolare il massimo e il minimo valore dei pixel (rispettivamente maxPixel e minPixel). Una volta trovati questi valori sarà possibile calcolare il deltaValue come differenza fra i due.
- **CALC_SHIFT**: Attraverso un controllo a soglia calcola lo shiftLevel necessario per il calcolo del pixel equalizzato.
- **CALC_NEWPIXEL**: Calcola il valore del nuovo pixel e lo scrive in memoria.
- **READ**: Legge in memoria il valore del pixel dell'immagine da equalizzare e incrementa il currentAddress (indirizzo corrente di memoria).
- **DONE**: Stato finale in cui si aspetta che il segnale START venga riportato a 0, se non già riportato prima, in modo da poter ritornare nello stato IDLE e leggere un'eventuale nuova immagine da equalizzare.

2.2 Processi

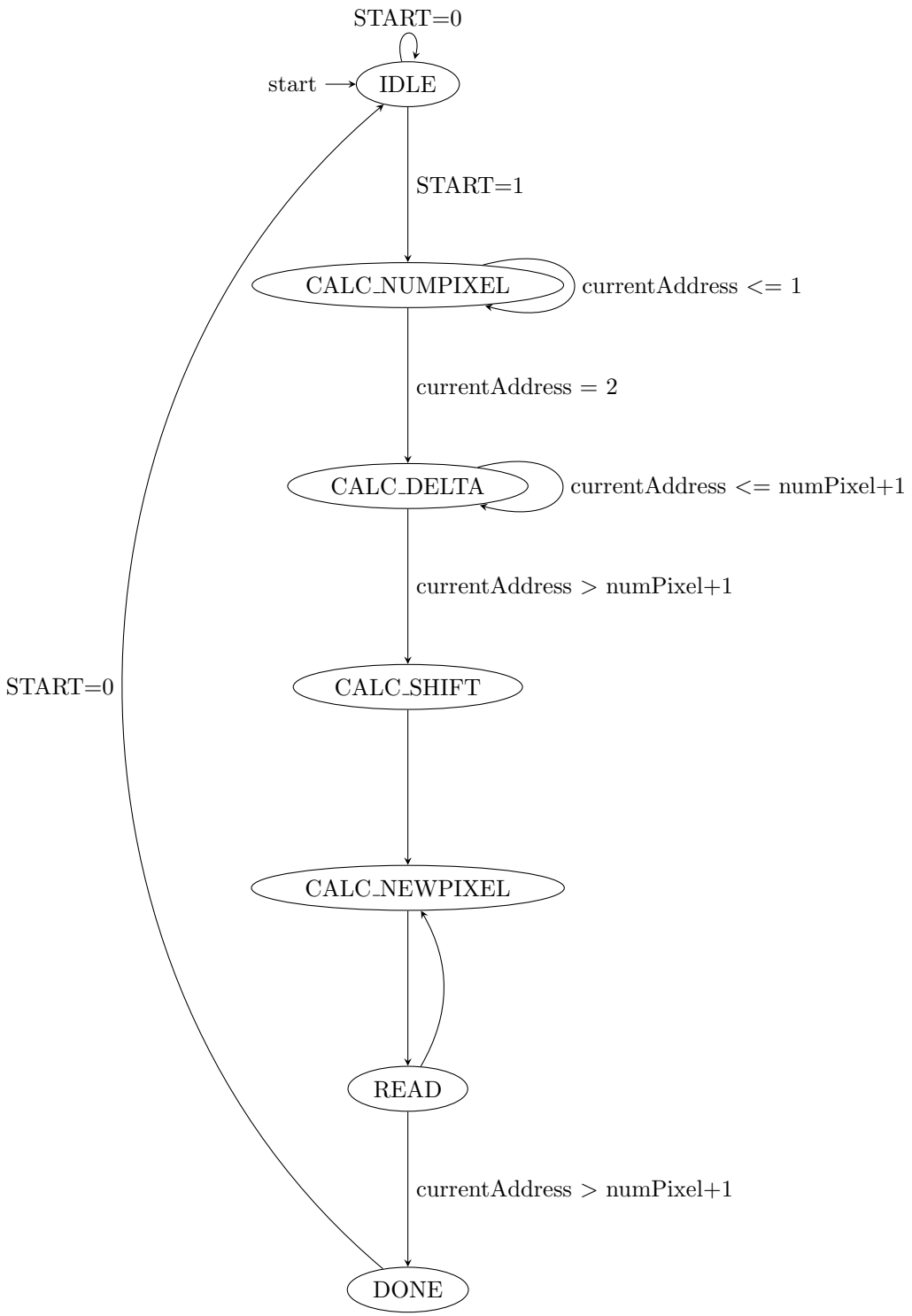
Il funzionamento avviene mediante due processi:

- Il primo ha come lista di sensibilità ***i_clk*** e ***i_rst*** ed assegna le uscite e lo stato interno sul fronte di salita del clock.
- Il secondo, invece, calcola le uscite e lo stato validi per il prossimo fronte di salita del clock.

Per assegnare le uscite dei segnali sono stati utilizzati dei registri 'next' che conterranno il valore che i registri 'veri e propri' assumeranno al prossimo ciclo di clock. Si è fatta particolare attenzione nella progettazione in modo da evitare l'utilizzo di latch.

I segnali utilizzati durante l'elaborazione, oltre alla loro versione 'next' sopra menzionata, sono:

- ***currentState*** per memorizzare lo stato corrente
- ***currentAddress*** per memorizzare l'indirizzo di memoria puntato
- ***maxPixel*** per memorizzare il massimo valore fra i pixel
- ***minPixel*** per memorizzare il minimo valore fra i pixel
- ***righe***, ***colonne*** per memorizzare la grandezza dell'immagine
- ***deltaValue*** utile al calcolo del nuovo pixel equalizzato
- ***numPixel*** grandezza dell'immagine o prodotto fra righe e colonne
- ***shiftLevel*** utile al calcolo del nuovo pixel


START=0
start
IDLE
START=1
CALC_NUMPIXEL
currentAddress <= 1
currentAddress = 2
CALC_DELTA
currentAddress <= numPixel+1
currentAddress > numPixel+1
START=0
CALC_SHIFT
CALC_NEWPIXEL
READ
currentAddress > numPixel+1
DONE

3 Risultati sperimentali

La sintesi del componente descritto ha prodotto i risultati riportati qui di seguito.

3.1 Report di utilizzo

Dal report di utilizzo si può dedurre il numero dei componenti utilizzati. In particolare si è cercato di evitare latch, mentre il modulo utilizza 297 LUT (0,22 % di quelli presenti sulla FPGA assegnata) e sono stati necessari 79 registri (lo 0,03% di quelli presenti sulla scheda) utilizzati come flip-flop per memorizzare i dati necessari alla codifica.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	297	0	134600	0.22
LUT as Logic	297	0	134600	0.22
LUT as Memory	0	0	46200	0.00
Slice Registers	79	0	269200	0.03
Register as Flip Flop	79	0	269200	0.03
Register as Latch	0	0	269200	0.00

3.2 Report di timing

Dal report di timing si può notare come sia rispettato il vincolo sul tempo di clock di al più 100ns. In particolare, il WNS - Worst Negative Slack - fornisce la differenza fra il clock_period della specifica (100ns) e il tempo di arrivo più lungo

```
------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
    -------      -------  ---------------------  -------------------
    92.736        0.000                      0                  171


    WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints
    -------      -------  ---------------------  -------------------
     0.207        0.000                      0                  171


    WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    --------     --------  ----------------------  --------------------
    49.500        0.000                       0                    80


All user specified timing constraints are met.
```

4 Risultati dei test

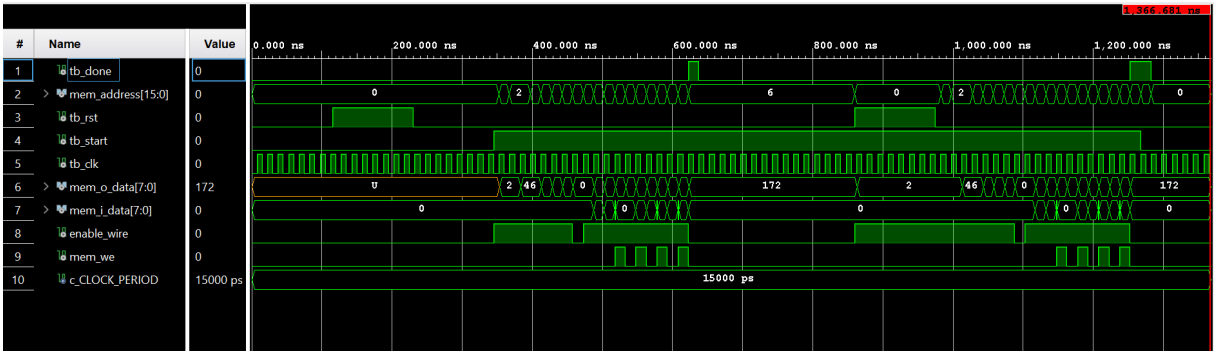
Per controllare il corretto funzionamento del componente e per assicurarsi della corretta implementazione delle specifiche sono stati effettuati dei test. Tutti i test benches riportati qui di seguito sono stati superati con successo.

4.1 Reset asincrono

Il test verifica il corretto funzionamento in caso di reset asincrono, cioè controlla che la macchina ritorni nello stato IDLE resettando tutti i segnali anche se la codifica dell'immagine non è terminata.

Time: 1367600 ps

Di seguito lo screenshot della waveform che permette di notare come il componente si comporta a seguito del segnale di reset asincrono.



4.2 Immagine con tutti i pixel a 1

Questo test bench utilizza un'immagine composta da soli pixel con valore 1.
La scelta di questo test è dovuta al fatto che viene simulato un caso limite dato che il delta-value assumerà valore zero.
Abbiamo quindi testato che il componente trovasse in modo corretto il massimo e il minimo pixel e calcolasse il giusto delta-value al fine di poter concludere la computazione correttamente.

Time: 917600 ps

4.3 Immagine 128x128

Questo test prova il corretto funzionamento del componente stressando particolarmente la memoria.
Viene depositata in memoria un'immagine di dimensioni 128x128 pixel e ci si aspetta che l'algoritmo funzioni correttamente nonostante il numero elevato di calcoli e iterazioni.
La scelta di questo test bench ci ha permesso di notare che nonostante la grandezza dell'immagine, oltre al corretto funzionamento dell'algoritmo, i tempi di esecuzione non siano stati troppo elevati.

Time: 737837600 ps

4.4 Immagine con zero pixel

Il test verifica che il componente funzioni anche nel caso (limite) in cui non venga passata alcuna immagine.
Questo ci ha permesso di capire la corretta implementazione dell'algoritmo e come questo sia in grado di coprire tutti i possibili casi, compresi quelli particolari.
Time: 6425100 ps

4.5 Tre immagini consecutive con reset

Questo test ci ha permesso di testare il soddisfacimento della richiesta di processare più immagini. Finita di leggere un'immagine, viene alzato il reset e si vuole leggere un'altra immagine, il componente resetta tutti i valori nei registri in modo tale da poter procedere con una nuova computazione e quindi equalizzare un'altra immagine. Il test, verifica, quindi, oltre alla corretta implementazione delle specifiche richieste, che l'equalizzazione di una prima immagine non impatti l'equalizzazione di una seconda.

Time: 316787600 ps

5 Conclusioni

Per far fronte alle richieste della specifica si è partiti pensando alla progettazione di un automa a stati finiti per descrivere l'algoritmo implementativo ad alto livello. Successivamente è stato descritto il componente in VHDL e sono stati effettuati dei test in modo da controllare il soddisfacimento delle specifiche richieste e il suo corretto funzionamento. Il componente è stato, poi, sintetizzato superando correttamente i test, prima nelle simulazioni Behavioral e poi anche in quelle Post-Synthesis rispettando tutti i requisiti della specifica. Le scelte implementative e l'algoritmo adottato si sono rivelati adatti. Di seguito lo schema del circuito sintetizzato.