Progetto di Reti Logiche

POLITECNICO
MILANO 1863

Giulio Occhipinti

Professor Fabio Salice

A.A. 2020-2021

Contents

1 Introduzione

1.1 Specifica

La specifica del progetto consiste nella realizzazione di un circuito sequenziale che applichi una semplificazione di un algoritmo di equalizzazione dell'istogramma di un'immagine fornita in input. Ogni pixel dell'immagine dovrà essere trasformato secondo la formula `NEW_PIXEL = min(255, TEMP_PIXEL)`

Dove

`TEMP_PIXEL = left_shift((OLD_PIXEL - MIN_PIXEL) ≪ SHIFT_LEVEL`

`SHIFT_LEVEL =` $[\log_2$`(DELTA + 1)]`

`DELTA = MAX_PIXEL - MIN_PIXEL`

`MAX_PIXEL` e `MIN_PIXEL` sono rispettivamente il valore massimo e minimo dei pixel dell'immagine. Le immagini sono in scala di grigi a 256 livelli, quindi i

valori dei pixel sono a 8 bit.

1.2 Interfaccia del componente

```
entity project_reti_logiche is
    port (
        i_clk:     in std_logic;
        i_rst:     in std_logic;
        i_start:   in std_logic;
        i_data:    in std_logic_vector(7 downto 0);
        o_address: out std_logic_vector(15 downto 0);
        o_done:    out std_logic;
        o_en:      out std_logic;
        o_we:      out std_logic;
        o_data:    out std_logic_vector(7 downto 0)
    );
end project_reti_logiche;
```

i_clk Segnale di clock

i_rst Segnale di reset

i_start Segnale di start, determina l'inizio dell'esecuzione della FSM

i_data Segnale a 8 bit corrispondente al valore contenuto nella cella di memoria all'indirizzo `o_address`

o_address L'indirizzo della memoria al quale si vuole accedere per leggere o scrivere dati

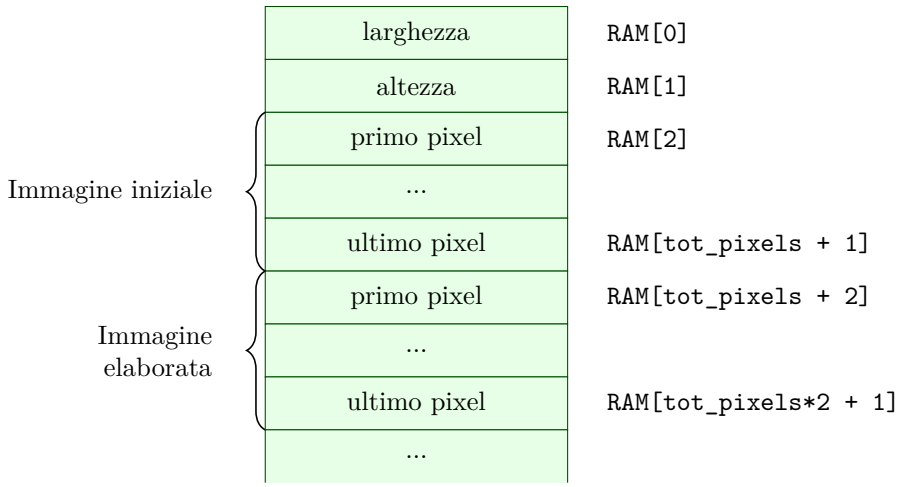
o_done Segnale portato a 1 quando l'esecuzione è terminata

o_en Segnale "enable", deve essere 1 per poter accedere alla memoria (R/W)

o_we "write enable", deve essere 1 per poter scrivere nella memoria

o_data Il valore che si vuole scrivere nella memoria all'indirizzo in `o_address`

1.3 Struttura della memoria



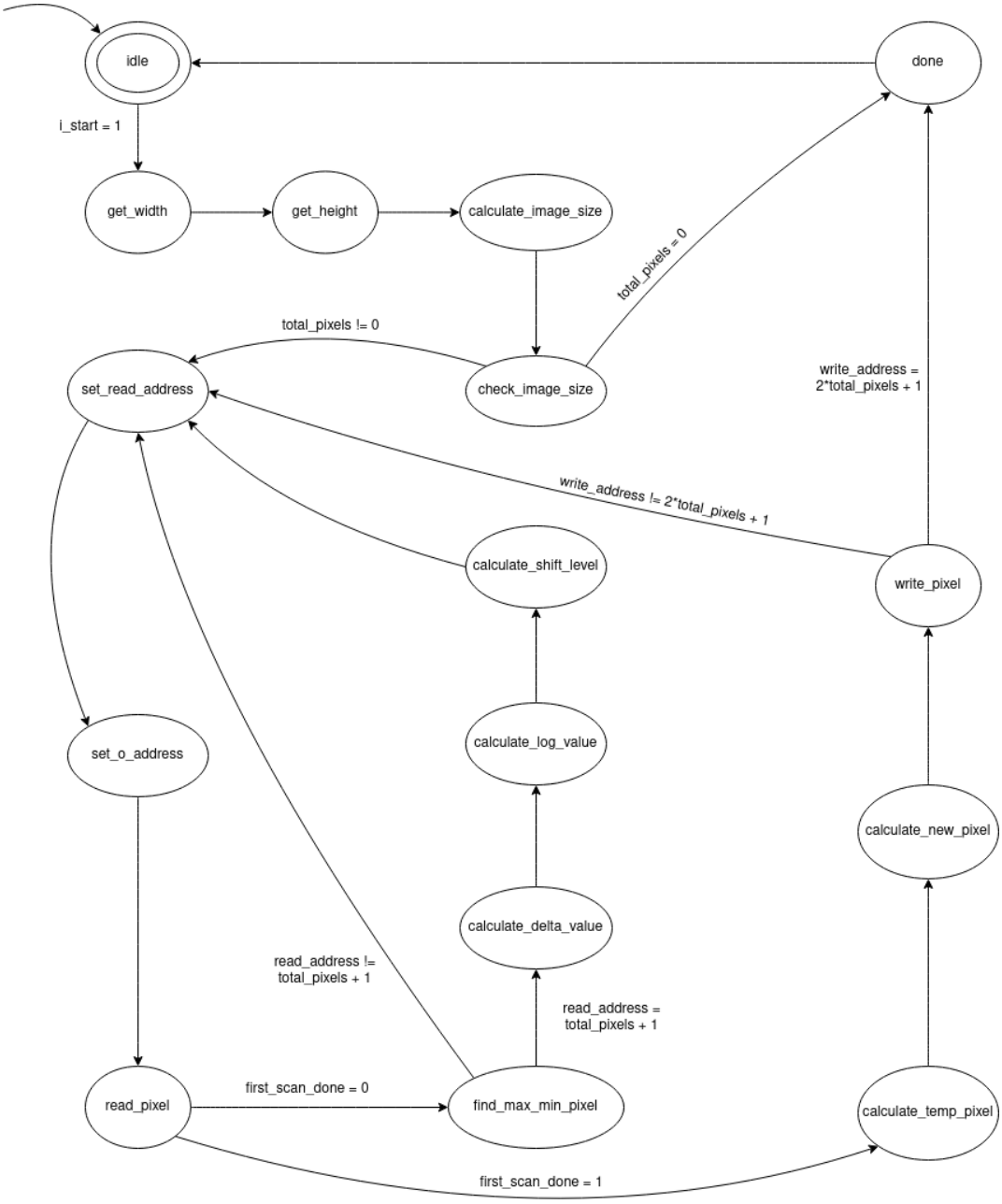
Inizialmente la memoria contiene i valori di larghezza, altezza, e i pixel iniziali. Al termine dell'esecuzione ci saranno anche i pixel elaborati a partire dall'indirizzo `tot_pixels + 2`, a patto che i valori di larghezza e altezza sono entrambi diversi da 0.

1.4 Algoritmo implementato

1. Salva i valori di larghezza e altezza
2. Calcola il numero totale dei pixel dell'immagine
3. Itera tutti i pixel dell'immagine. Per ogni pixel:
   - 3.1 Confronta il suo valore con quelli correnti di massimo e minimo, e se necessario li aggiorna
4. Calcola la costante `SHIFT_LEVEL` utilizzando i valori trovati allo step 3
5. Itera di nuovo l'immagine in input. Per ogni pixel:
   - 5.1 Calcola il valore del nuovo pixel utilizzando le apposite formule
   - 5.2 Scrive il nuovo pixel all'indirizzo `old_pixel_address + tot_pixels`
6. Fine

2 Architettura

2.1 Diagramma degli stati



2.1.1 Descrizione degli stati

idle Lo stato iniziale e finale, viene raggiunto alla fine dell'elaborazione dell'immagine

get_width Prende dall'indirizzo 0 della memoria il valore della larghezza dell'immagine

get_height Analogamente a `get_width`, prende il valore dell'altezza dall'indirizzo 1 della memoria

calculate_image_size Calcola il numero totale di pixel dell'immagine con la formula $base \times altezza$

check_image_size Controlla che il numero di pixel calcolato allo stato precedente sia diverso da zero. Se è zero il prossimo stato è `done` e l'esecuzione termina

set_read_address Incrementa il valore di `read_address` di 1, per poter leggere il pixel successivo

set_o_address Assegna a `o_address` il valore di `read_address`

read_pixel Legge dalla memoria il pixel contenuto in `o_address`. Se il flag `first_scan_done` è 1, ovvero se sono stati trovati i valori `MAX_PIXEL` e `MIN_PIXEL`, passa allo stato `calculate_temp_pixel`.
Altrimenti va in `find_max_min_pixel`

find_max_min_pixel Confronta il valore del pixel corrente con i valori attuali max_pixel e min_pixel, e se necessario li aggiorna. Quando viene letto l'ultimo pixel passa allo stato in cui calcola il `DELTA`, altrimenti torna a `set_read_address`

calculate_delta_value Calcola il `DELTA` facendo `MAX_PIXEL - MIN_PIXEL`

calculate_log_value Calcola $\log_2$(`DELTA + 1`) tramite dei controlli a soglia. Per esempio se `DELTA` è compreso fra 127 e 254, ovvero `DELTA` + 1 è compreso tra 128 e 255, allora il logaritmo è 7

calculate_shift_level Utilizzando il `LOG_VALUE` trovato in precedenza, calcola lo `SHIFT_LEVEL` facendo `8 - LOG_VALUE`. Poi resetta il `read_address` all'indirizzo del primo pixel e dopo aver settato il flag `first_scan_done` a 1 ritorna allo stato `set_read_address`

calculate_temp_pixel Calcola il temp_pixel con la formula `TEMP_PIXEL = (CURRENT_PIXEL - MIN_PIXEL) ≪ SHIFT_LEVEL`

calculate_new_pixel Calcola il pixel elaborato con la formula `NEW_PIXEL = min(TEMP_PIXEL, 255)`, poi prepara l'indirizzo in cui scrivere il pixel con `WRITE_ADDRESS = READ_ADDRESS + TOTAL_PIXELS`

write_pixel Scrive il nuovo pixel al write_address calcolato allo stato precedente. Se tutti i pixel elaborati sono stati scritti in memoria va allo stato `done`, altrimenti torna a `set_read_address`

done Setta `o_done` a 1 e torna allo stato di `idle`

2.2 Registri utilizzati

next_state Il valore dello stato successivo. È di tipo `state_type`, ovvero un'enumerazione con i valori di tutti gli stati

read_address, write_address Contengono rispettivamente l'indirizzo dal quale leggere dati dalla memoria e al quale scrivere i nuovi pixel calcolati

width, height Contengono rispettivamente la larghezza e l'altezza dell'immagine

total_pixels Il numero totale di pixel dell'immagine, usato per capire quando l'ultimo pixel è stato letto/scritto

current_pixel Il valore del pixel attualmente in esame

max_pixel, min_pixel I valori rispettivamente di massimo e minimo dei pixel dell'immagine fornita in input

log_value, delta_value, shift_level Contengono i valori delle costanti omonime utilizzate per calcolare i valori dei pixel elaborati

temp_pixel Il valore temporaneo utilizzato per calcolare il nuovo pixel

new_pixel Il valore del nuovo pixel che verrà scritto nella memoria all'indirizzo in `write_address`

first_scan_done Flag che viene posto a 1 dopo che sono stati trovati i valori di massimo e minimo

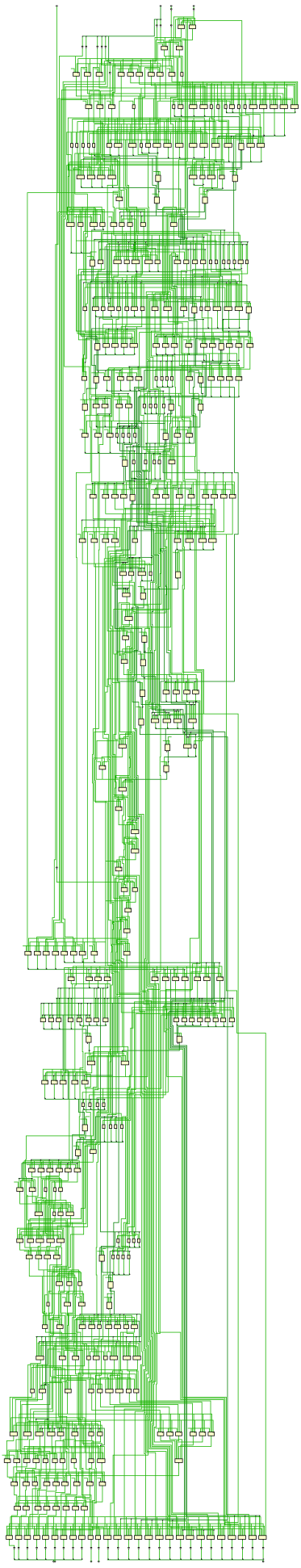
2.3 Scelte progettuali

Ho implementato la macchina a stati tramite un modulo monoprocesso in cui un *process* chiamato `lambda_delta` assegna valori ai segnali di output, ai registri interni e determina lo stato successivo al quale passare al prossimo ciclo di clock. Un altro process chiamato `state_reg`, sensibile ai soli segnali di clock e reset, è responsabile all'assegnamento dei registri per far sì che le informazioni vengano preservate durante l'esecuzione. Per esempio, la larghezza dell'immagine è contenuta in due registri, `width` e `next_width`. Nel process `lambda_delta` il dato viene assegnato al segnale `next_width`. In seguito, sul fronte di salita del clock (ovvero nel process `state_reg`), viene assegnato `next_width` al registro `width`. In questo modo si sono neutralizzati tutti i warning relativi agli *inferring latch* e le testbench passano anche in post-synthesis.

Un'altra scelta che ritengo degna di nota è stata quella di porre il valore di default di `read_address` a 1 invece che a 0. Ho fatto così perché nello stato `set_read_address` il registro `read_address` viene incrementato di 1. Poiché i pixel da leggere cominciano all'indirizzo 2, se il valore iniziale fosse 0 sarebbe stato necessario introdurre uno stato aggiuntivo in cui viene incrementato di 2 solo all'inizio dell'esecuzione. Essendo invece 1, quando viene raggiunto lo stato `set_read_address` viene subito assegnato il valore corretto. I valori `width` e `height` vengono presi settando `o_address` "manualmente" senza passare da `set_read_address`, perché la fase iniziale dell'esecuzione è sempre la stessa e *non* fare così porterebbe a un inutile aumento della complessità del circuito.

Per determinare se i valori di massimo e minimo dei pixel dell'immagine sono stati trovati, e quindi è possibile procedere al calcolo dei nuovi pixel, ho utilizzato un registro chiamato `first_scan_done`. Durante la prima iterazione dell'immagine, nello stato `find_max_min_value` avviene un controllo dell'indirizzo di lettura, e se risulta che si sta analizzando l'ultimo pixel, significa che `MAX_PIXEL` e `MIN_PIXEL` sono stati trovati. Nello stato `calculate_shift_value`, dopo aver calcolato lo `SHIFT_VALUE` viene posto il flag `first_scan_done` a 1 e `read_address` viene riportato a 1 per poter cominciare la seconda iterazione, nella quale per ogni pixel in ingresso verrà calcolato il nuovo valore utilizzando le costanti calcolate in precedenza.

2.4 Schema di sintesi



3 Risultati sperimentali

3.1 Report di sintesi

```
---------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------
Hierarchical RTL Component report
Module project_reti_logiche
Detailed RTL Component Info :
+---Adders :
       2 Input     16 Bit       Adders := 3
       3 Input     16 Bit       Adders := 2
       3 Input      8 Bit       Adders := 1
       2 Input      5 Bit       Adders := 1
+---Registers :
                   16 Bit    Registers := 3
                    8 Bit    Registers := 6
                    5 Bit    Registers := 1
                    4 Bit    Registers := 2
                    1 Bit    Registers := 1
+---Muxes :
      16 Input     16 Bit        Muxes := 2
      16 Input      8 Bit        Muxes := 3
       2 Input      8 Bit        Muxes := 1
       2 Input      4 Bit        Muxes := 6
       5 Input      4 Bit        Muxes := 1
      16 Input      1 Bit        Muxes := 17
       8 Input      1 Bit        Muxes := 1
       2 Input      1 Bit        Muxes := 2
---------------------------------------------------
Finished RTL Hierarchical Component Statistics
```

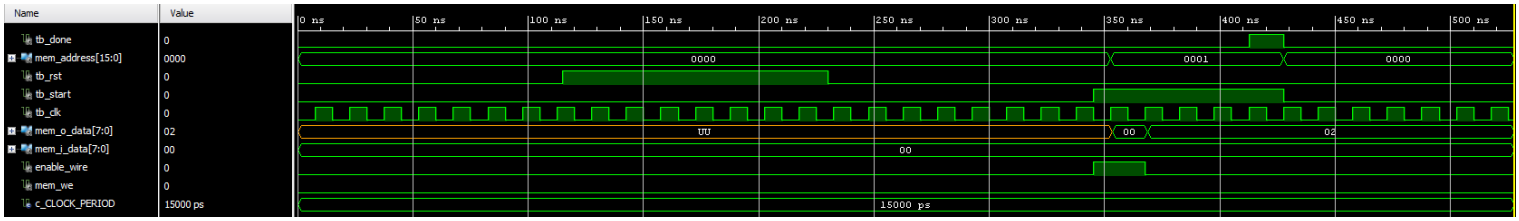
Site Type	Used	Fixed	Available	Util%
Slice LUTs*	213	0	10400	2.05
LUT as Logic	213	0	10400	2.05
LUT as Memory	0	0	9600	0.00
Slice Registers	134	0	20800	0.64
Register as Flip Flop	134	0	20800	0.64
Register as Latch	0	0	20800	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

In seguito al processo di sintesi, Vivado non ha mostrato alcun errore o warning.

4 Simulazioni

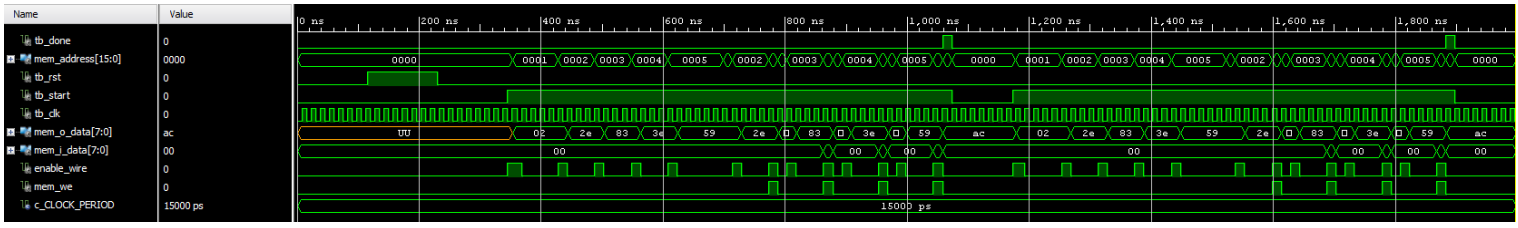
Per verificare il corretto funzionamento del modulo sono state eseguite diverse testbench che sottopongono il circuito a diversi casi limite. Il modulo passa tutte le testbench fornite dal professor Salice, sia in modalità Behavioral che in Post-Synthesis. In seguito riporterò le testbench che si sono rivelate utili a correggere il funzionamento del circuito, in quanto mi hanno permesso di trovare certi *bug* nella mia implementazione del processo di equalizzazione dell'immagine.

4.1 Testbench Zero Pixel



Questa testbench simula l'esecuzione della macchina nel caso in cui l'altezza ha dimensione zero, ma nella memoria è comunque presente il valore di un pixel. Inizialmente il test falliva perché la macchina procedeva lo stesso alla lettura del pixel e al calcolo del nuovo valore. Dopo aver introdotto lo stato `check_image_size` il test passa.

4.2 Testbench a doppia esecuzione



Questa testbench ripete l'esecuzione sulla stessa immagine di input rialzando il segnale `i_start` dopo che rileva `i_done` a 1. Inizialmente il test falliva perché durante la prima esecuzione i pixel iniziali venivano sovrascritti da 0, e quindi alla seconda esecuzione `current_pixel` era sempre 0. Questo accadeva perché lo stato che precedente a `write_pixel`, `calculate_new_pixel`, metteva `o_we` a 1, ed essendo il valore di default di `o_data` 0 veniva scritto 0 nel read address corrente, cioè quello del `current_pixel` da elaborare.

4.3 Altre testbench

Ci sono altre testbench tra quelle fornite che non passavano, ma non reputo necessario riportarle in dettaglio perché i problemi di base sono gli stessi delle testbench descritte in precedenza. Fra le altre testbench che sono state eseguite, fortunatamente la stragrande maggioranza è passata fin da subito senza particolari problemi. Per completezza, elencherò i restanti casi limite testati più importanti:

- Tutti i pixel a 255
- Tutti i pixel a 0
- Maggior numero di pixel possibile (immagine 128x128)
- Nessun pixel (immagine 0x0)
- Reset asincrono
- Elaborazione di molteplici immagini

Settembre 2021