Prova Finale di Reti Logiche

Docente Fabio Salice

Gabriele Lazzarelli
10623766

Riccardo Izzo
10599996

Maggio 2021

POLITECNICO
MILANO 1863

Indice

1 Introduzione

1.1 Scopo

Il progetto consiste nell'implementazione in VHDL del metodo di equalizzazione dell'istogramma di un'immagine. Questo metodo incrementa il contrasto dell'immagine ridistribuendo i valori dei pixel su tutta la scala cromatica.
Il componente hardware sviluppato implementa una versione semplificata dell'algoritmo che prende in considerazione solo immagini in scala di grigi a 256 livelli. La FPGA scelta per l'implementazione è xc7a200tfbg484-1.

1.2 Specifiche generali

La memoria utilizzata ha un indirizzamento al byte, il primo byte della memoria si trova all'indirizzo zero. La dimensione dell'immagine sorgente è definita nei primi 2 byte che rappresentano rispettivamente il numero di colonne e il numero di righe. La dimensione massima è pari a 128×128 pixel. A partire dal terzo byte sono memorizzati i valori dei pixel dell'immagine sorgente. L'immagine equalizzata è scritta a partire dal primo indirizzo libero in memoria.

1.3 Interfaccia componente

Descrizione VHDL dell'interfaccia:

```
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 segnali di interfaccia si dividono in segnali di input, generati dal testbench, e segnali di output, generati dal componente.
In particolare:

- `i_clk` è il segnale di clock, periodico con duty cycle del 50%
- `i_rst` è il segnale di reset
- `i_start` è il segnale di avvio della computazione
- `i_data` è il segnale vettore che porta il valore dell'ultimo byte letto da memoria

- `o_address` è il segnale vettore che porta l'indirizzo di memoria a cui si vuole leggere o scrivere
- `o_done` è il segnale di termine della computazione
- `o_en` è il segnale di enable, quando è alto la memoria è accessibile in lettura o scrittura
- `o_we` è il segnale di write, è alto quando si vuole scrivere in memoria, è basso quando si vuole leggere dalla memoria
- `o_data` è il segnale vettore che porta il valore del byte che si vuole scrivere in memoria

2 Architettura

2.1 Descrizione algoritmo

L'algoritmo si divide in tre fasi:

1. Vengono lette le dimensioni dell'immagine e viene calcolata la dimensione totale
2. Si ha una lettura completa dell'immagine per trovare il minimo e il massimo valore di saturazione e si calcola il valore di shift
3. Si rilegge l'immagine e per ciascun pixel si scrive il nuovo valore a partire dal primo byte libero in memoria

Per chiarezza, si riportano le formule utilizzate:

```
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)
```

Lo shift rappresenta in valore numerico l'alterazione massima possibile dell'istogramma dell'immagine, ovvero maggiore è il valore di shift maggiore sarà il contrasto globale dopo il processo.

Casi particolari: se l'istogramma dell'immagine copre già tutto l'intervallo di saturazione, allora il contrasto è da considerarsi massimo e perciò lo shift ha come valore zero; in caso contrario il valore dello shift aumenta al diminuire della variazione tra il minimo e il massimo di saturazione fino a un valore di 8. Nel caso di un'immagine monocromatica, per la quale si ha un valore di shift pari a 8, non ha senso parlare di equalizzazione dell'istogramma e di fatto l'immagine risultante avrà tutti i valori dei pixel pari a 0.

2.2 Macchina a stati finiti

2.2.1 Descrizione stati

RESET Stato in cui vengono riportati i segnali ai valori di default, escluso `o_done`, preparando la FSM per una nuova esecuzione

IDLE Stato in cui viene abbassato il segnale di `o_done` e si attende il segnale alto di `i_start`

WAIT_CLK Stato in cui si attende la corretta propagazione dei segnali da parte della memoria, segue sempre lo stato di lettura

READ_BYTE Stato in cui viene letto il valore di `i_data`, il suo comportamento varia in base al valore corrente di *status*:

- *status* = 0: `i_data` contiene il numero di colonne, il valore viene passato al segnale `n_col`
- *status* = 1: `i_data` contiene il numero di righe, il valore viene passato al segnale `n_row`
- *status* = 2: `i_data` contiene l'ultimo byte letto dall'immagine, il valore viene confrontato con il massimo e minimo valore di saturazione, quest'ultimi vengono aggiornati dopo ogni lettura
- *status* = 3: `i_data` contiene l'ultimo byte letto dall'immagine, questo viene usato per elaborare il nuovo valore del pixel che viene passato al segnale `o_data`

CALC_SIZE Stato in cui viene calcolata la dimensione dell'immagine usando i valori di `n_col` e `n_row`, il risultato dell'operazione $n\_col \times n\_row + 1$ viene passato al segnale `max_address`, che rappresenta l'indirizzo in memoria dell'ultimo pixel dell'immagine letta

CALC_SHIFT Stato in cui viene calcolato il valore di $shift$ mediante l'uso di una LUT, il valore viene passato al segnale `shift_level`

WRITE_BYTE Stato in cui avviene la scrittura in memoria del valore contenuto in `o_data` all'indirizzo presente in `o_address`

DONE Stato in cui si alza il valore di `o_done`, si segnala la fine della computazione e si passa allo stato di RESET

2.2.2 Diagramma FSM

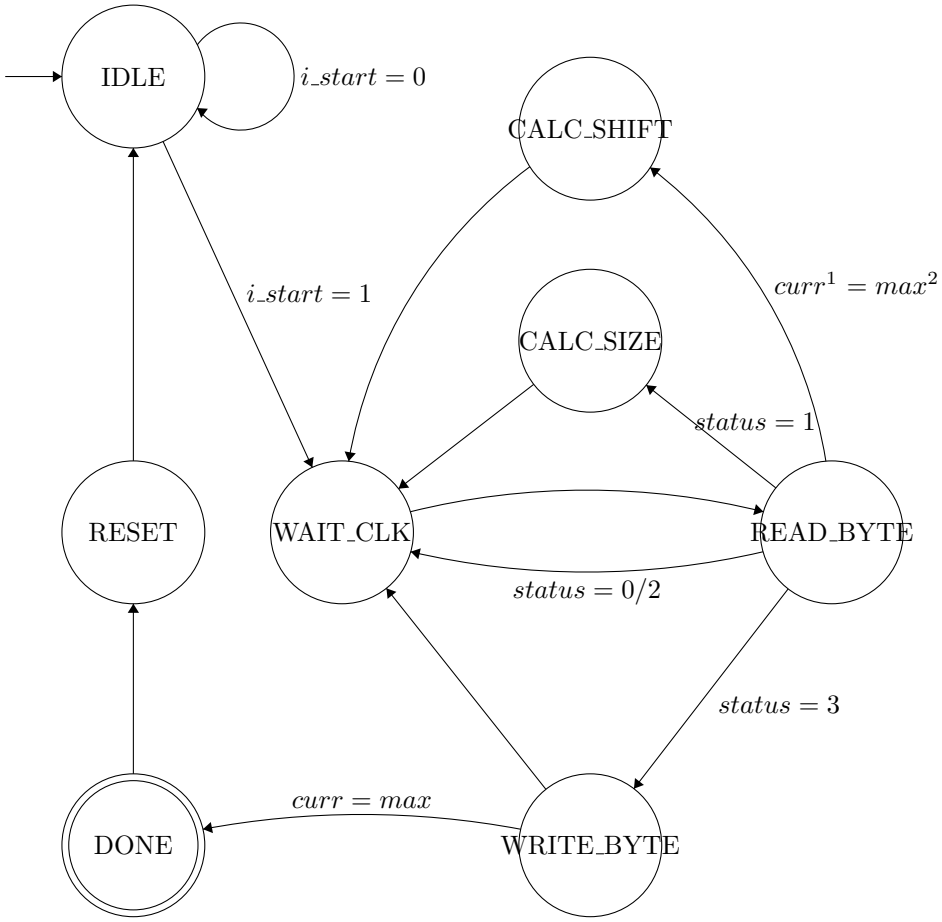

Figura 1: Diagramma FSM

[1] *curr* è il segnale `curr_address` che rappresenta l'indirizzo corrente del byte letto

[2] *max* è il segnale `max_address` che rappresenta l'indirizzo in memoria dell'ultimo pixel letto dall'immagine

3 Risultati sperimentali

Report di sintesi:

- LUT: 178 (0.13% del totale)
- FF (registri): 103 (0.04% del totale)

Type	Used	Available	Util%
Slice LUT	178	134600	0.13
Slice Registers	103	269200	0.04
Register as FF	103	269200	0.04
Register as Latch	0	269200	0.00

L'implementazione scelta permette di evitare l'uso di latch.

4 Simulazioni e test benches

Sono stati eseguiti numerosi test al fine di verificare il corretto funzionamento del programma in situazioni limite, forzando ogni possibile comportamento della macchina. Si riportano di seguito i test eseguiti:

- Immagine vuota (0 pixel): si verifica che, nel caso di un'immagine vuota, non venga scritto nulla in memoria.
- Immagine con dimensione minima (1 pixel)
- Immagine con dimensione massima (16384 pixel)
- Immagine con contrasto massimo: valore di $shift$ uguale a 0.
- Immagini consecutive: in questo caso viene verificata l'elaborazione di più immagini consecutive, in particolare ci si aspetta che vengano ripristinati tutti i segnali ai valori di default e che l'equalizzazione dell'immagine successiva inizi solo dopo il nuovo segnale di start.
- Segnale di reset asincrono: durante l'equalizzazione di un'immagine la macchina deve essere in grado di tornare allo stato di RESET a fronte di un segnale asincrono `i_rst = 1`.
- Immagine monocromatica: nel caso di un'immagine monocromatica l'intervallo d'intensità è minimo e lo `shift_level` è massimo, la nuova immagine risulta essere composta completamente da pixel di valore 0.
- Stress test con 10000 immagini: verifica la corretta elaborazione di 10000 immagini consecutive generate casualmente.

Tutti i test proposti sono stati superati con successo. Per ogni test sono state effettuate le seguenti simulazioni: Behavioral pre-synthesis, Functional e Timing post-synthesis.

5 Conclusione

L'architettura progettata rispetta le specifiche assegnate: sono state verificate sia le funzionalità che le prestazioni mediante numerosi test.
L'architettura risulta essere implementabile sulla FPGA utilizzata e di conseguenza, per completezza, sono state effettuate anche le simulazioni Functional e Timing post-implementation, entrambe superate con successo nei test case proposti.
Le simulazioni Timing sono state eseguite aggiungendo un vincolo al clock nella sezione "constraints" di Vivado, nel rispetto delle specifiche.
Durante le simulazioni sono stati provati diversi periodi di clock. L'implementazione, oltre che a funzionare come da specifica a 100 ns, permette di raggiungere un periodo di clock minimo pari a 1 ns in Behavioral pre-synthesis e 2 ns in Functional post-synthesis.