Prova Finale Reti Logiche

Riccardo Luigi Aielli

1 Settembre 2021

Matricola: 907237
Codice Persona: 10621879
Docente: William Fornacciari

Indice

1 Introduzione

Il progetto Prova Finale di Reti Logiche proposto nell'annno accademico 2020/2021 consiste nell'implementazione in VHDL di una rete logica in grado di elaborare l'equalizzazione dell'istogramma di un'immagine.

1.1 Specifiche generali

L'equalizzazione consiste nel ricalibrare il contrasto di una immagine quando l'intervallo dei valori di intensità sono molto vicini effettuandone una distribuzione su tutta la gamma disponibile, al fine di incrementare il contrasto.
L'immmagine da elaborare sarà letta dalla memoria RAM, l'immagine elaborata verrà salvata in successione all'immagine di input.

Più elaborazioni in serie sono supportate senza che venga fornito nuovamente il segnale RESET, il quale è necessario solo in precedenza della prima elaborazione.
L'algoritmo sarà implementato in una versione semplificata e verrà applicato solo ad immagini di dimensione massima 128 x 128 pixel in scala di grigi a 256 livelli.

Nello specifico l'algoritmo è il seguente:

```
      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)
```

1.2 Interfaccia del componente

Il componente da descrivere ha la seguente 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;
```

In particolare:

- Il nome del modulo deve essere `project_reti_logiche`.
- `i_clk` è il segnale di CLOCK in ingresso generato dal TestBench;
- `i_rst` è il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START;
- `i_start` è il segnale di START generato dal Test Bench;
- `i_data` è il segnale (vettore) che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria;
- `o_done` è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;
- `o_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- `o_we` è il segnale di WRITE ENABLE da dover mandare alla memoria (=1) per poter scriverci. Per leggere da memoria esso deve essere 0;
- `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

2 Architettura

Il componente realizzato basa la computazione e il flusso di esecuzione su una macchina a stati finiti (FSM), che permette il succedersi delle operazioni necessarie per la lettura dei dati dalla memoria, l'elaborazione e infine della scrittura in memoria.

2.1 Descrizione ad alto livello

Ad alto livello si possono evidenziare i seguenti passi:

1. Inizializzazione dei segnali interni ai valori di reset.

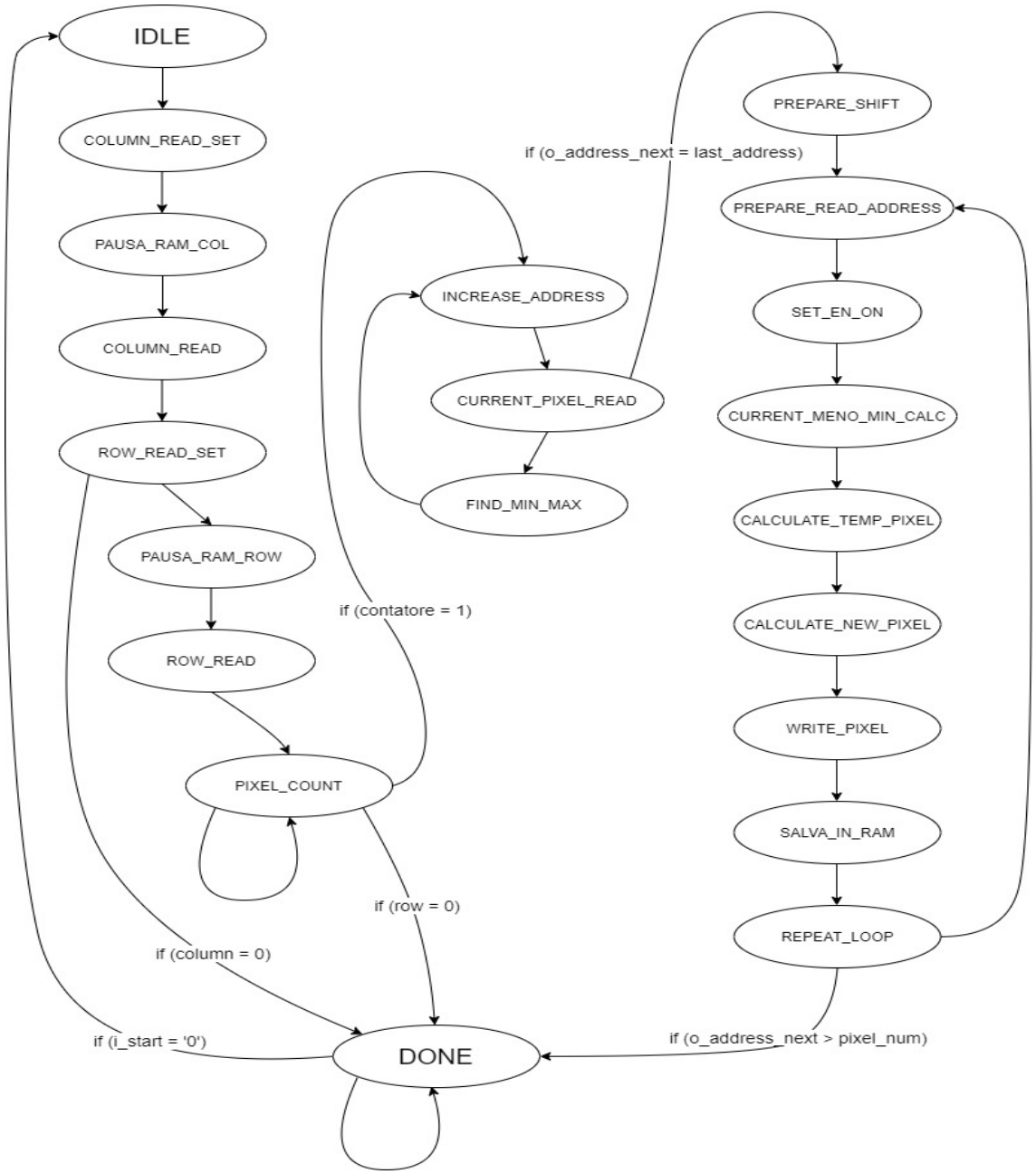
2. Lettura da memoria RAM del numero di colonne e righe dell'immagine da equalizzare.
3. Calcolo del numero di pixel totali che compongono l'immagine.
4. Ricerca del pixel con valore massimo e minimo.
5. Calcolo del `DELTA_VALUE` e dello `SHIFT_LEVEL`
6. Per ogni pixel dell'immagine:
   (a) Lettura del valore del pixel.
   (b) Calcolo del valore del nuovo pixel.
   (c) Scrittura in memoria del valore del nuovo pixel.
7. Segnale DONE alto e attesa di un nuovo segnale START o RESET

2.2 Scelte implementative e problematiche riscontrate

- Durante la realizzazione del progetto si è scelto di favorire la leggibilità e la chiarezza del codice piuttosto che favorire le performance e l'ottimizzazione della macchina.
- Si è scelto per chiarezza e per mantenere una struttura semplice di separare il codice VHDL in tre differenti *sezioni*.
  - Un *prima sezione* che si occupa della gestione dei vari registri necessari per la corretta esecuzione del componente.
  - Un *seconda sezione* che è costituita da tutti i processi nei quali vengono svolti i calcoli per l'elaborazione dei byte letti.
  - Infine un *terza sezione* che si occupa della macchina a stati e dell'esecuzione dei processi delle *sezioni* precedenti.
- Nella stesura del codice VHDL e con l'ausilio dei primi test effettuati si sono riscontrate alcune problematiche che sono state accuratamente risolte:
  - Durante l'esecuzione sono stati notati dei ritardi di propagazione tra la RAM e il componente progettato. Questa problematica ha portato alla scelta di aumentare il numero di stati della macchina distribuendo le operazioni da fare in ogni fase su più stati in modo da permettere la corretta propagazione dei segnali. In alcuni casi particolari sono stati invece introdotti dei veri e propri stati di

pausa. In questi casi è stata fatta l'ipotesi in base ai vari test effettuati che il massimo ritardo di propagazione sia inferiore a un ciclo di clock.

2.3 Macchina a stati finiti



Nel diagramma sono stati omessi gli archi che collegano ciascun stato allo stato di IDLE. Nel caso il segnale `i_reset` fosse posto a 1, qualsiasi sia lo stato corrente, la macchina tornerebbe nello stato IDLE.

Descrizione degli stati della macchina:

1. `IDLE`: Stato iniziale in cui vengono resettati tutti i registri e segnali utilizzati, si attende il segnale `i_start` per iniziare l'elaborazione.
2. `COLUMN_READ_SET`: Seleziona l'indirizzo del primo byte della memoria ram.
3. `PAUSA_RAM_COL`: Stato di *pausa* in cui viene attivato il segnale `o_en`.
4. `COLUMN_READ`: Legge e salva nel registro `column` il byte corrispondente al numero di colonne.
5. `ROW_READ_SET`: Seleziona l'indirizzo del secondo byte della memoria ram, controlla che il numero di colonne sia diverso da zero e setta il registro `contatore` con il valore corrispondente al numero di colonne.
6. `PAUSA_RAM_ROW`: Stato di *pausa* in cui viene attivato il segnale `o_en`.
7. `ROW_READ`: Legge e salva nel registro `row` il byte corrispondente al numero di righe.
8. `PIXEL_COUNT`: Stato *loop* che itera tante volte quante è il numero di colonne attivando il processo che si occupa di calcolare il prodotto righe x colonne, in modo tale da ottenere il numero di pixel totali che compongono l'immagine. Il registro `contatore` viene decrementato a ogni iterazione in modo da interrompere il loop quando il calcolo del prodotto è terminato.
9. `INCREASE_ADDRESS`: Attiva il processo che assegna l'indirizzo contenuto nel registro `o_address_next` al segnale `o_address`.
10. `CURRENT_PIXEL_READ`: Stato che attiva il segnale `o_en` per la lettura di `i_data` nello stato successivo (`FIND_MIN_MAX`) e interrompe il *loop* che legge tutti i pixel dell'immagine originale nel momento in cui è stato letto l'ultimo byte valido passando nello stato `PREPARE_SHIFT`.
11. `FIND_MIN_MAX`: In questo stato si confronta il pixel del segnale `i_data` con i valori dei pixel salvati nei registri `max_pixel`, `min_pixel` e nel caso aggiorna il loro contenuto. Lo stato succesivo è `INCREASE_ADDRESS`.

12. `PREPARE_SHIFT:` Attiva il processo che calcola il valore di *shift* relativo all'immagine corrente.

13. `PREPARE_READ_ADDRESS:` Attiva il processo che assegna l'indirizzo contenuto nel registro `o_address_next` al segnale `o_address`.

14. `SET_EN_ON:` Stato in cui viene attivato il segnale `o_en`.

15. `CURRENT_MENO_MIN_CALC:` Attiva il processo che calcola il valore su cui fare lo *shift* per ottenere il pixel temporaneo.

16. `CALCULATE_TEMP_PIXEL:` Attiva il processo che calcola il valore temporaneo, facendo lo *shift left* del valore calcolato nello stato precedente (`CURRENT_MENO_MIN_CALC`).

17. `CALCULATE_NEW_PIXEL:` Attiva il processo che calcola il valore del nuovo pixel, prendendo il minimo tra 255 e il valore temporaneo.

18. `WRITE_PIXEL:` Scrive sul segnale `o_data` il valore del nuovo pixel.

19. `SALVA_IN_RAM:` Stato di *pausa* per attendere i ritardi di propagazione dovuti alla memoria ram.

20. `REPEAT_LOOP:` Stato che fa ripartire la fase di creazione del valore equalizzato di ciascun pixel, fino a che non è stato processato l'ultimo byte corrispondente all'ultimo pixel dell'immagine originale. (Stato successivo: `PREPARE_READ_ADDRESS`).

21. `DONE:` Alza il segnale `o_done`, attende che il segnale `i_start` si abbassi per poi tornare nello stato di `IDLE` per processare l'immagine successiva.

3 Risultati Sperimentali

3.1 Report di sintesi

Il report di sintesi presenta i seguenti dati:

Dati ottenuti tramite il comando `report_utilization`:
LUT: 163 (0.12%)
FF: 154 (0.06%)
Register as Latch: 0 (0.00%)

Dati ottenuti tramite il comando `report_timing`:

Slack (MET) : 4.086ns (required time - arrival time)
Data Path Delay: 5.532ns (logic 2.735ns (49.440%) route 2.797ns (50.560%))

3.2 Simulazioni

Per verificare il corretto funzionamento del componente progettato sono stati effettuati diversi test specifici, focalizzando l'attenzione sui casi limite. Tutti i test sono stati eseguiti in simulazione behavioural e in simulazione functional post sintesi con un periodo di clock di 10ns. Di seguito una breve descrizione dei principali test:

3.2.1 Test Zero Pixel

Test contenente un'immagine con dimensione nulla ossia con il numero di colonne e/o righe pari a 0.

3.2.2 Test Un solo pixel

Test contenente un'immagine con dimensione 1 ossia con il numero di colonne e righe pari a 1.

3.2.3 Test Delta minimo

Sono stati effettuati due diversi test per verificare il corretto comportamento del componente nel caso in cui il valore delta dei pixel è pari 0:

- Test contenente un'immagine con dimensione arbitraria ma con tonalità di grigio pari a 255 per ogni pixel.
- Test contenente un'immagine con dimensione arbitraria ma con tonalità di grigio pari a 1 per ogni pixel.

3.2.4 Test Immagine di dimensione massima 128x128

Test contenente un'immagine con dimensione massima ossia di 128 x 128 pixel con tonalità di grigio differenti tra loro.

3.2.5 Test Tre immagini consecutive

Test con più elaborazioni nella stessa sessione di immagini con differente numero di pixel. Il segnale `i_rst` (RESET) viene fornito solo in precedenza alla prima elaborazione. Le successive sono eseguite in seguito al segnale `i_start` (START).

3.2.6 Test Reset

Test usato per verificare il corretto funzionamento del circuito di reset. Durante l'esecuzione viene attivato il segnale `i_rst` (RESET) diverse volte e in modo asincrono.

3.2.7 Test Valore pixel 0

Test contenente un'immagine con valore dei pixel pari a 0.

3.2.8 Test Generato 1000 immagini

Test che consiste nell'elaborazione di 1000 immagini generate in modo casuale di dimensione compresa tra 0 e 128 per il numero di righe e di colonne, con valore dei pixel compresi tra 0 e 255.

Tutti i test effettuati hanno dato esito positivo e sono stati utili nella fase di affinamento del codice VHDL così come nella risoluzione di alcuni problemi riscontrati in fase di sviluppo.

4 Conclusione

Durante lo svolgimento di questo progetto è stato realizzato un componente funzionante in pre e post-sintesi su tutti i test utilizzati. Semplice nel funzionamento e descritto in codice VHDL facilmente leggibile. La sintesi mostra un utilizzo di LUT pari a 0.12%, di FF pari a 0.06% e nessun *inferred latch*. Si può concludere che il componente risulti adeguato, in grado di soddisfare la specifica e i requisiti richiesti per il corretto funzionamento.