Prova Finale di Reti Logiche

Giuseppe Serra
Alessandro Zito

A. A. 2020-21

Matricole:	887630 - 890219
Codici Persona:	10566090 - 10617579
Docente:	Gianluca Palermo

Indice

1 Requisiti del Progetto

La specifica della Prova finale (Progetto di Reti Logiche) 2020 è ispirata al metodo di **equalizzazione dell'istogramma di una immagine.** Il metodo di equalizzazione dell'istogramma di una immagine è un metodo pensato per ricalibrare il contrasto di una immagine quando l 'intervallo dei valori di intensità sono molto vicini effettuandone una distribuzione su tutto l'intervallo di intensità, al fine di incrementare il contrasto. Al componente viene richiesto di:

1. Accedere ai primi due indirizzi della RAM per leggere i byte dell'immagine (indirizzo 0 per le colonne e 1 per le righe). La dimensione massima di un'immagine è 128x128 pixel.
2. Leggere ogni pixel dell'immagine
3. Calcolare il nuovo valore del pixel.
4. Scrivere il risultato nella memoria RAM, e rifare lo stesso processo dal punto 4 per tutti i pixel dell'immagine.

Inoltre, l'implementazione deve essere in grado di gestire un segnale di Reset. Per l'implementazione si è scelto di supporre il Reset asincrono rispetto al segnale di clock.
L'implementazione deve essere poi sintetizzata con target FPGA xc7a200tfbg484-1.

1.1 Esempio

1.2 Ipotesi Progettuali

Si sono supposti veri i seguenti fatti:

1. Il nuovo valore del pixel di un massimo di 8 bit.
2. L'immagine non può essere di 0 pixel.
3. Il programma è sintetizzato in modo da poter codificare più immagini mantenendo i vincoli di RESET imposti dalla specifica.

2 Implementazione

L'architettura è stata progettata in maniera modulare, in modo da specializzare i singoli componenti creati e separare le funzionalità di calcolo della codifica dell'indirizzo dalla gestione della macchina a stati finiti.

2.1 Descrizione ad Alto Livello

Da un'ottica di alto livello, l'implementazione esegue i seguenti passi:

1. Legge il primo indirizzo e salva il numero delle colonne delle immagini
2. Legge il secondo indirizzo e salva il numero delle righe delle immagini
3. Calcola il numero dei byte dell'immagine

4. Legge ogni pixel dell'immagine e controlla se è il pixel con valore più grande o più piccolo dell'immagine:
   (a) Se sì, salva il valore in max_pixel_value o min_pixel_value
   (b) Se no, legge il pixel successivo.
5. Calcola la differenza tra il pixel con valore maggiore e pixel con valore minore ($delta\_value$).
6. Calcola il numero di bit da shiftare con la seguente formula:

$$shift\_level = Floor(\log_2(delta\_value + 1) \tag{1}$$

   Il seguente calcolo è stato effettuato tramite un controllo a soglia iterando sulla potenza di 2.
7. Si prende il valore del pixel e lo si shifta di shift_level e codifica il numero a 16 bit aggiungendo "00000000" in coda:
   (a) Se il numero è minore di 255, Carica il risultato nella RAM codificato a 8 bit;
   (b) Altrimenti, carica nella RAM "11111111".
8. Ripete i passaggi dal 7 in poi per tutti i pixel da leggere.
9. Si aggiornano tutti i bit di comando.

Per gestire questo algoritmo si è scelta un'implementazione costituita da una macchina a stati finiti, che rappresenta il top-level component e che gestisce tutto il processo di equalizzazione dell'immagine.

2.2 Macchina a Stati Finiti

La **Macchina a Stati Finiti** (FSM) è stata realizzata con specifica *Behavioural* Segue una descrizione degli stati formali della FSM:

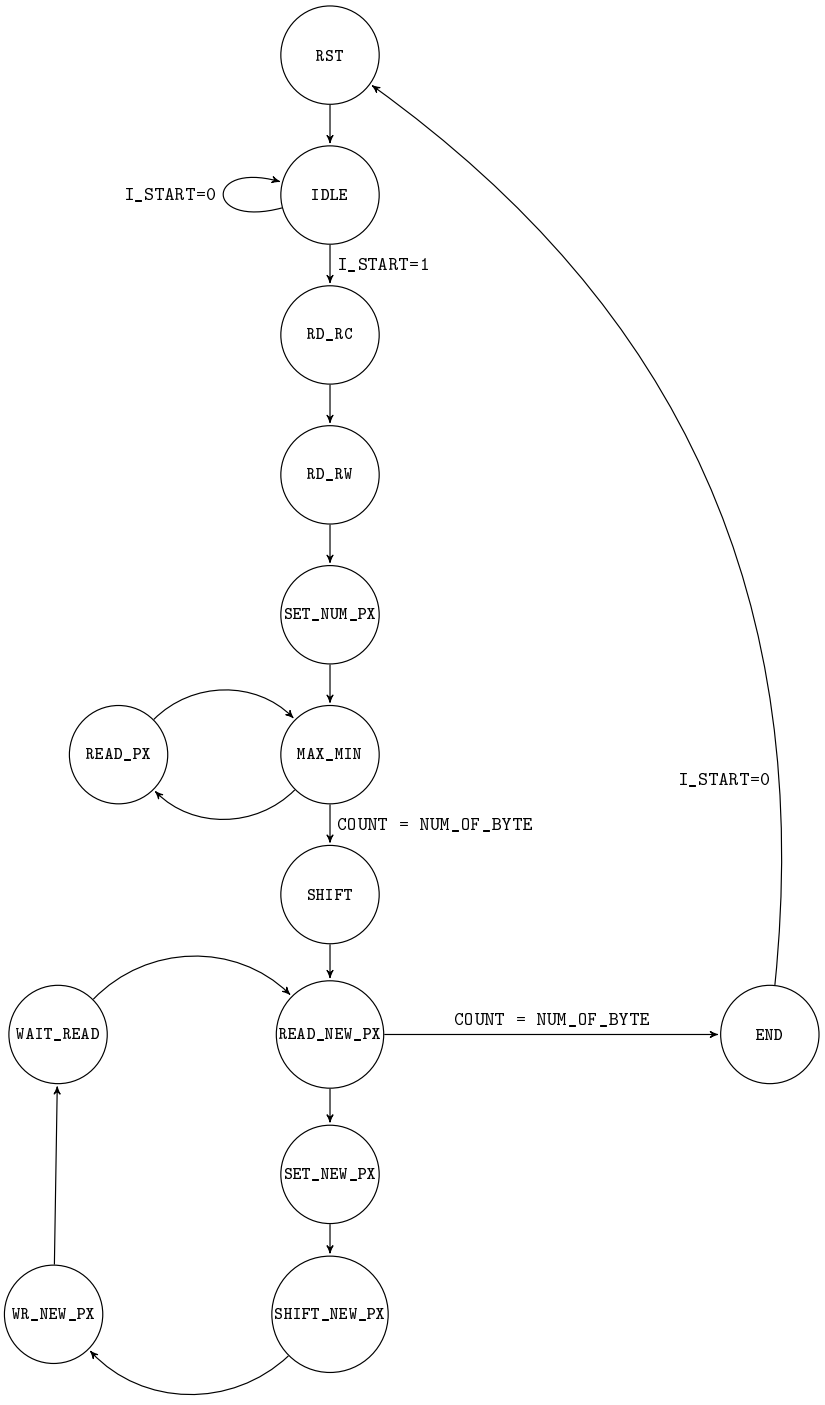
- `RESET`: Stato di idle in cui si posiziona la FSM al reset della computazione. La macchina aspetta che venga asserito il segnale `i_start`.
- `READ_COLUMN`: Stato in cui viene letta la colonna dell'immagine
- `READ_ROW`: Stato in cui viene letta la riga dell'immagine.
- `NUMBER_OF_BYTE`: Stato in cui si calcolano il numero di byte dell'immagine
- `CALC_MAX_AND_MIN`: Stato in cui vengono letti tutti i pixel e vengono trovati il pixel più grande e più piccolo
- `READ_PIXEL`: Stato in cui si aspetta un ciclo di clock per leggere un pixel e ritornare nello stato di calcolo di massimo e minimo
- `CALC_SHIFT_LEVEL`: Stato in cui si calcola lo $shift\_level$

- `READ_PIXEL_VALUE`: Stato in cui, dopo aver calcolato lo shift_level, si ricominciano a leggere i valori dei pixel dell'immagine; se il contatore ausiliare è uguale al numero di byte, si va allo stato finale, altrimenti, si calcola il valore del pixel shiftato.
- `SET_NEW_PIXEL_VALUE`: Stato in cui viene calcolato il valore temporaneo del pixel e codificato in 16 bit.
- `SHIFT_NEW_PIXEL_VALUE`: Stato in cui viene shiftato il nuovo valore temporaneo del pixel ottenendo quindi il valore finale; viene poi settato **o_we** a 1 per scrivere.
- `WRITE_NEW_PIXEL_VALUE`: Stato in cui viene scritto il nuovo valore del pixel sulla RAM sul suo indirizzo **o_address** di uscita che vale esattamente $o\_address = count + 2 + number\_of\_byte$.
- `WAIT_READ`: Stato in cui si aspetta un ciclo di clock per settare l'indirizzo di lettura uguale a count + 3 e settare **o_we** = **0**, per tornare a READ_PIXEL_VALUE
- `FINALIZE`: Stato in cui si aspetta che `i_start` venga portato giù, settando `o_done` a 1 e riportandosi allo stato di `RESET`.

Più precisamente l'insieme degli stati della FSM dovrebbe comprendere almeno anche il counter `count`, che viene usato prima come contatore per leggere tutti gli stati di lettura dei pixel, poi viene utilizzato per il calcolo del logaritmo in base 2 per lo shift_level e, infine, per gli indirizzi di lettura e scrittura sulla RAM.
Nella figura a seguito è riportato il disegno dell'automa. Sono state usate le seguenti abbreviazioni degli stati per chiarezza:

RESET:	RST
IDLING:	IDLE
READ_COLUMN:	RD_RC
READ_ROW:	RD_RW
SET_NUMBER_OF_PIXEL:	SET_NUM_PX
SET_MAX_AND_MIN (With WZ_COUNT=i):	MAX_E_MIN
READ_PIXEL (With COUNT=i):	READ_PX
SET_SHIFT_LEVEL:	SHIFT
READ_PIXEL_VALUE:	READ_NEW_PX
SET_NEW_PIXEL_VALUE:	SET_NEW_PX
SHIFT_NEW_PIXEL_VALUE:	SHIFT_NEW_PX
WRITE_NEW_PIXEL_VALUE:	WRITE_NEW_PX
WAIT_READ:	WAIT_READ
FINALIZE:	END

Figura 1: Diagramma degli stati della Macchina a Stati Finiti


RST
IDLE
I_START=0
I_START=1
RD_RC
RD_RW
SET_NUM_PX
READ_PX
MAX_MIN
I_START=0
COUNT = NUM_OF_BYTE
SHIFT
WAIT_READ
READ_NEW_PX
COUNT = NUM_OF_BYTE
END
SET_NEW_PX
WR_NEW_PX
SHIFT_NEW_PX

2.3 Schema dell'Implementazione

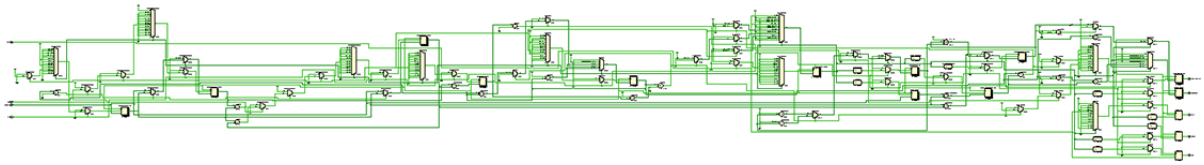


Figura 2: Schema dell'implementazione

3 Test Benches

I test effettuati hanno cercato di effettuare transizioni critiche oppure di verificare possibili configurazioni di memoria estreme. In particolare sono stati creati dei testbench comprendenti le seguenti situazioni:

- Vari test generati casualmente tramite un generatore di immagini in python sia quadrate che rettangolari.
- Caso specifico in cui tutti i pixel sono dello stesso valore
- Caso specifico con $min\_pixel\_value = 0$ e $max\_pixel\_value = 255$
- Vari test casuali con più immagini da equalizzare dentro un unico test.

Questi test hanno evidenziato alcune criticità nel codice iniziale e hanno guidato lo sviluppo della soluzione fino alla sua versione finale.

Per tutti i test riportati e i test successivi è stata effettuata la simulazione behavioural e successivamente la simulazione functional e timing post-synthesis, tutte con successo.

I risultati rilevanti sono riportati nella sezione 4.

4 Risultati Sperimentali

4.1 Report di Sintesi

Dal punto di vista dell'area la sintesi riporta il seguente utilizzo dei componenti:

- LUT: 257 (0.19% del totale)
- FF: 127 (0.05% del totale)

Si è fatta particolare cautela nella scrittura del codice per evitare utilizzo di Latch.

4.2 Risultato dei Test Bench

Lavorando sul testing si è anche provato a variare il periodo di clock, confermando che l'implementazione rispetta le specifiche, funzionando a 100 ns.

Si riportano i risultati di tempo legati agli ultimi 2 test riportati nella sezione 3

Simulazione 1 immagine 128x128 pixel (Functional Post-Synthesis):	1 721 035 100 ps
Simulazione 3 immagini 2x2 Pixel senza RESET (Functional Post-Synthesis):	2 755 100 ps
Simulazione 3 immagini 2x2 pixel con RESET (Functional Post-Synthesis):	3 205 100 ps

Il risultato esprime una variazione del 16% dal caso senza reset al caso con reset, evidenziando come l'architettura, per scelta progettuale, è più efficiente quando non deve fare frequentemente la fase di setup, ossia la fase di reset degli indirizzi di lettura; in qualsiasi dovrà comunque però resettare i segnali ausiliari utilizzati per il calcolo a soglia.

5 Conclusioni

Si ritiene che l'architettura progettata rispetti anzitutto le specifiche, fatto che è stato verificato mediante estensivo testing sia casuale, che con test benches manualmente scritti. Oltre a ciò l'architettura è stata pensata sulla base di una FSM, evitando l'utilizzo di Latch che avrebbero potuto creare l'instaurarsi di cicli infiniti.
Dal punto di vista del design, si è scelto di utilizzare una unica FSM senza componenti esterni.