Prova Finale (Progetto Reti Logiche)

Prof Gianluca Palermo – Anno Accademico 2020/2021

Francesca Ricter – CODICE PERSONA 10653781 – MATRICOLA 912389

Indice

1 INTRODUZIONE

1.1 Specifica del progetto

La specifica della prova finale (Progetto Reti Logiche) 2020 è ispirata al metodo di equalizzazione dell'istogramma di una immagine. Tale metodo è pensato per ridistribuire il valore delle intensità dei pixel di una immagine, nel caso in cui questi valori risultino essere vicini tra loro, aumentandone il contrasto.

L'algoritmo da implementare ne è una versione semplificata che tiene conto esclusivamente di immagini rappresentate su scala di grigi con 256 livelli. La rielaborazione dei pixel si basa su quattro operazioni fondamentali:

- Calcolo del "*delta value*": intervallo numerico tra il pixel con valore massimo e pixel con valore minimo, presenti nella immagine ;
- Identificazione dello shift level (valore compreso tra 0 e 8) , che permette di semplificare la trasformazione dei pixel nei nuovi valori, semplicemente shiftando i bit del valore del pixel di tante posizioni quanto lo shift value ;
- Applicazione dello shift a ciascun pixel ;
- Settaggio del valore '255' come massimo valore di intensità raggiungibile.

Il modulo da implementare dovrà leggere l'immagine da una memoria in cui è memorizzata, sequenzialmente e riga per riga, l'immagine da elaborare. Ogni byte corrisponde ad un pixel dell'immagine. La dimensione della immagine è definita da 2 byte, memorizzati a partire dall'indirizzo 0. Il byte all'indirizzo 0 si riferisce alla dimensione di colonna; il byte nell'indirizzo 1 si riferisce alla dimensione di riga. La dimensione massima dell'immagine è 128x128 pixel. L'immagine è memorizzata a partire dall'indirizzo 2 e in byte contigui, mentre l'immagine equalizzata deve essere scritta in memoria immediatamente dopo l'immagine originale.

1.2 Interfaccia del componente

Il componente presenta 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:

- 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: Macchina a stati finiti

2.1 Algoritmo

L'algoritmo sviluppato per la realizzazione del componente hardware è composto dalle seguenti fasi:

1. calcolo delle dimensioni della immagine;
2. prima lettura di tutti i valori dei pixel, individuando il massimo e il minimo ;
3. calcolo del **delta_value** e dello **shift level**;
4. seconda lettura dei valori: partendo dal primo pixel, viene calcolato il nuovo valore tramite le operazioni indicate, il nuovo dato viene scritto in memoria subito dopo l'ultimo pixel, si passa poi al pixel successivo iterando il procedimento per ciascun pixel fino al termine della immagine;
5. termine: il segnale done viene posto ad '1'.

Il modulo può eventualmente ripercorrere nuovamente l'algoritmo subito dopo per elaborare altre immagini.

2.2 Descrizione e gestione dei segnali

Per implementare tale algoritmo, il componente è stato modellato mediante macchina a stati finiti, che transita da uno stato a quello successivo sul fronte di salita del segnale *i_clk*. Per la prima codifica il modulo deve riceve in ingresso il segnale *i_rst*, il quale lo porterà nel primo stato, **IDLE** . Da qui, attende che il segnale *i_start* in ingresso venga portato a "1" per poter iniziare l'elaborazione vera e propria. Quest'ultimo segnale rimane alto fin quando il segnale *o_done* non sarà portato alto: ciò avviene una volta terminata la computazione, dopo aver scritto il risultato in memoria. Il segnale o_done rimane alto fin qaundo il segnale di i_start non è riportato a "0" . Il componente è in grado di codificare più immagini. Una seconda elaborazione o successive non attendono il reset del modulo.

2.3 Scelte progettuali - Stati della macchina

La FSM in questione è composta da 11 stati: 9 stati principali e 2 stati di supporto (**WAIT_CLOCK** e **FETCH_NEXT**). Di seguito è riportata una breve descrizione della funzionalità di ciascun stato:

- **IDLE**
  Stato di partenza che precede l'elaborazione della immagine, a questo si giunge grazie al segnale *i_rst*, oppure dopo aver terminato una elaborazione precedente, dallo stato **DONE_STATE.** Qui vengono azzerati tutti i valori dei segnali necessari per la elaborazione, preparando la macchina alla computazione. Il segnale *i_start, portato ad "1",* permette di transitare allo stato successivo.

- **CALC_DIM**
  Stato in cui si identificano le dimensioni della immagine: il numero di righe, il numero delle colonne e il numero complessivo dei pixel presenti nella immagine. A questo si accede tre volte durante il corso di una singola computazione, usando come supporto lo stato **WAIT_CLOCK**, riportato di seguito.

- **WAIT_CLOCK**
  Stato usato per attendere un ciclo di clock, ogni volta ritenuto necessario, ad esempio, prima di interagire con i dati in ingresso dalla memoria.

- **FETCH_NEXT**
  Stato di supporto impiegato per incrementare l'indirizzo corrente e farne richiesta del contenuto alla memoria.

- **FIND_MAX_MIN**
  Stato adoperato per leggere tutti i valori pixel della immagine, individuando il valore massimo e minimo.

- **FIND_DELTA_VALUE**
  Stato per determinare delta value.

- **FIND_SHIFT_VALUE**
  Stato nel quale viene individuato lo shift value mediante una Lookup Table con valori soglia.

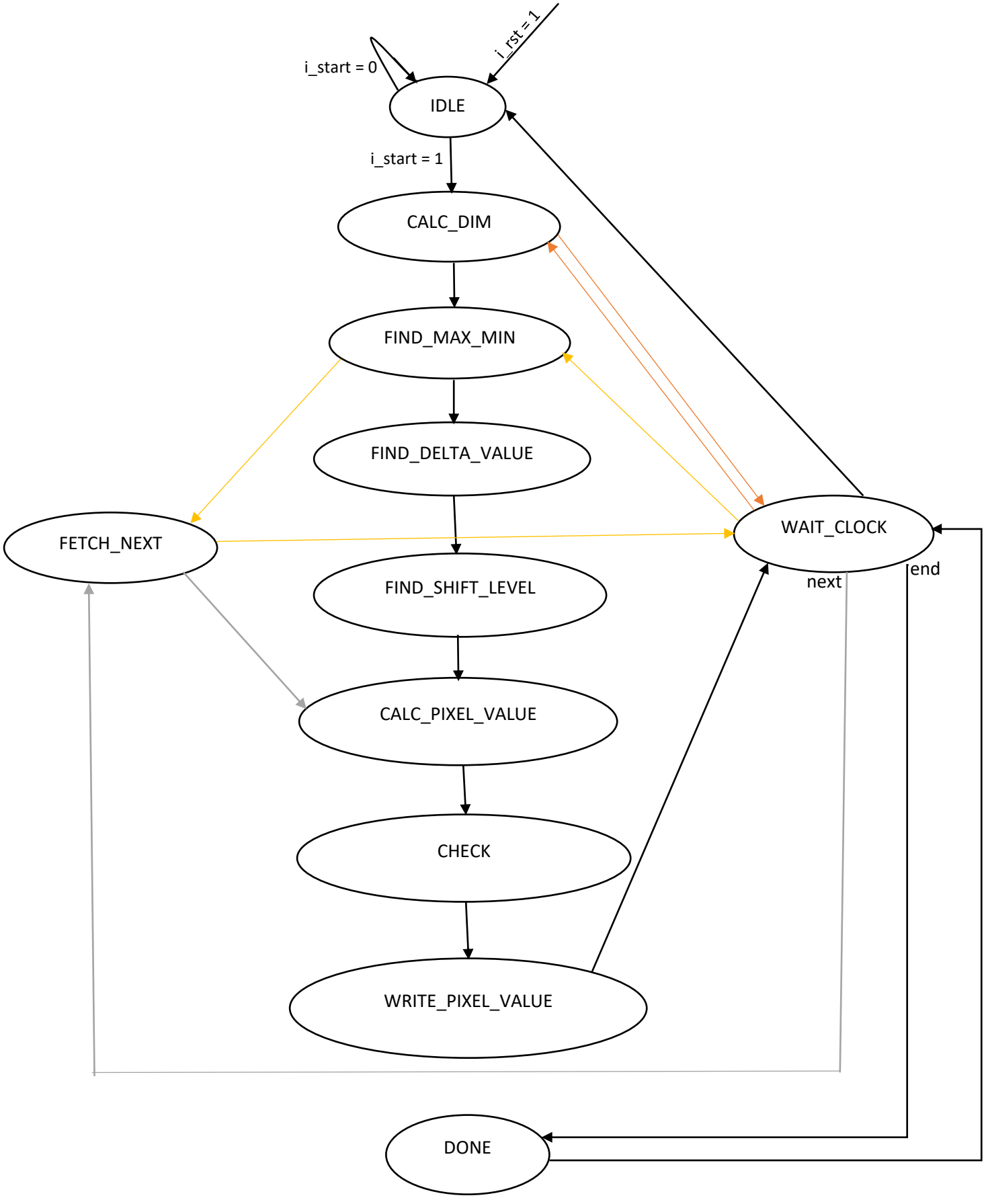
- **CALC_PIXEL_VALUE**
  Stato in cui si ottiene il nuovo valore, risultato delle operazioni sopra citate.

- **CHECK**
  Stato per controllare che nessun valore di pixel superi la soglia massima ‘255’, nel caso in cui questo avvenga il valore del pixel viene aggiornato a tale valore.

- **WRITE_PIXEL_VALUE**
  Stato impiegato per scrivere i valori appena calcolati in memoria, immediatamente dopo l’immagine originale.

- **DONE_STATE**
  Stato terminale, che segna la fine della computazione, nel quale viene portato alto il segnale o_done, il componente è riportato nello stato IDLE.

2.3 Schema funzionale



DESCRIZIONE CICLI SCHEMA

Lo schema è costituito da quattro cicli (uno principale e tre sottocicli):

- CICLO PERCORSO TRAMITE FRECCE NERE : ciclo principale del modulo che rappresenta l'intero algoritmo, esso racchiude i seguenti 3 sotto-cicli :
  - CICLO PERCORSO TRAMITE FRECCE ARANCIONI: usato per determinare in ordine: colonne, righe e pixel dell'immagine;
  - CICLO PERCORSO TRAMITE FRECCE GIALLE : usato per la prima lettura dei pixel in successione;
  - CICLO PERCORSO TRAMITE FRECCE GRIGE : percorso per ciascun pixel successivo al primo per il calcolo e la scrittura del nuovo valore, la computazione dell'ultimo pixel porta nello stato finale.

3 Risultati sperimentali

3.1 Testing

Per accertare il corretto funzionamento del componente, questo è stato testato grazie all'utilizzo di vari testbench. I test ricoperti, sono i seguenti:

3.1.1 TEST GENERALI:

1. **10k casi**
   Testano il componente con 10mila diverse immagini successive di piccole dimensioni (in caso di reset e in assenza di reset tra le immagini);

2. **2k casi**
   Testano il componente con 2mila diverse immagini successive di grandi dimensioni (in caso di reset e in assenza di reset tra le immagini);

3. **Test semplici**

3.1.2 TEST SPECIFICI (CORNER CASES) :

1. **Immagine zero pixel**
   Il componente viene testato nel caso in cui non vi sia alcuna immagine da elaborare, poiché la dimensione indicata in memoria è pari a zero sia per il numero di righe che per numero di colonne. Il componente dunque salta allo stato finale senza procedere con l'algoritmo di elaborazione.

2. **Immagine con solo una delle due dimensione posta a zero**
   Similmente al caso precedente, individua una immagine con zero pixel: la differenza consiste nell'avere uno dei due valori righe/colonne diverso da zero.

3. **Immagine con un solo pixel**
   Testa il componente con una immagine di dimensione minima (1x1).

4. **Immagine di dimensione massima**
   Testa il componente con una immagine di dimensione massima (128x128).

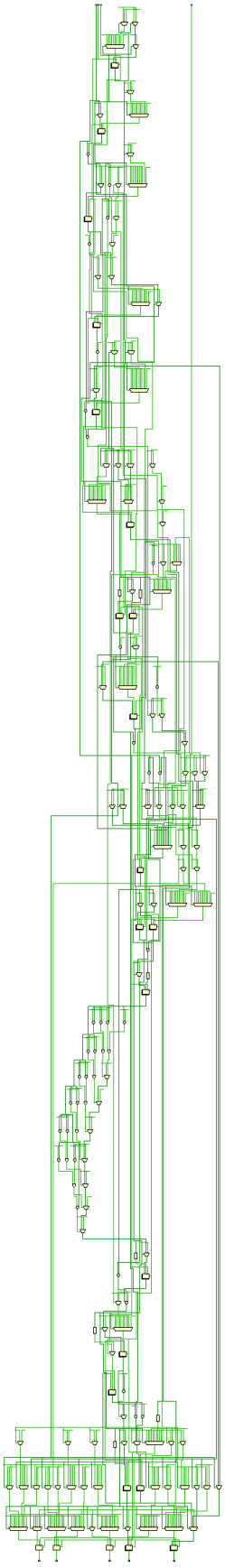
5. **Immagine con tutti valori uguali (posti al valore minimo)**
   Testa nel caso di una immagine con i pixel tutti posti al valore 0.

6. **Immagine con tuti valori uguali (posti tutti al valore minimo)**
   Testa nel caso di una immagine con i pixel tutti posti al valore 255.

7. **Immagine invariata**
   Il testbench in questione contiene una immagine con valori di intensità già ben distribuiti sull'intervallo indicato, in particolare l'immagine ha come valore massimo '255' e come valore minimo '1', l'immagine viene riscritta invariata dopo quella originale.

8. **Reset asincrono**
   Il modulo riceve un segnale di reset durante la computazione, dunque viene resettato e inizia l'elaborazione da principio.

3.3 Schema di sintesi



4 Ottimizzazioni

L'ottimizzazione principale apporta è stata quella di ridurre il numero di stati della macchina:

-Lo stato **CALC_DIM** si occupa di trovare tutte le dimensioni della immagine, invece di usare tre diversi stati, come pensato originariamente;

- Lo stato **WAIT_CLOCK** raggruppa quelli che erano inizialmente 4 diversi stati usati come transizione (o attesa), per passare da uno stadio dell'algoritmo ad un altro, o per settare/resettare determinati valori nel mezzo;

- Lo stato **FETCH_NEXT** racchiude due stati, viene usato per aggiornare l'indirizzo corrente e richiedere alla memoria il contenuto all'indirizzo successivo.

Altri stati secondari sono stati fusi a quelli principali, permettendo di passare dalla Macchina a Stati Finiti ideata inizialmente con 20 stati alla macchina finale con 11 stati.

Sarebbe stata possibile una ulteriore riduzione, ma per avere una visione sequenziale dell'algoritmo implementato si è preferito mantenere questo set di stati.

6 Conclusione

Per lo sviluppo del progetto, sono partita disegnando l'automa corrispondente alla mia FSM, in modo da avere una buona idea dell'algoritmo e degli stati da implementare, questo ha facilitato il compito di scrittura del codice. Prima di procedere con la fase di testing, ho risolto eventuali warnings presenti, questo mi ha permesso di avere pochi problemi in questa fase.

Il modulo rispetta le caratteristiche richieste e supera i test generali e i test specifici a livello *Behavioral* e *Post-Synthesis*, si ritiene dunque di aver realizzato un componente hardware in linea con la specifica del progetto.