Prova Finale (Progetto di Reti Logiche)

Prof. Gianluca Palermo

Giorgio Seguini (cod. pers. 10611319)

Anno accademico 2020/2021

Indice

1 Introduzione

1.1 Specifiche

Lo scopo di questa prova finale è la progettazione e la descrizione, tramite VHDL, di un componente hardware che sia in grado di riprodurre una versione semplificata del noto algoritmo di equalizzazione dell'istogramma dell'immagine. Tale algoritmo 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.

L'algoritmo di equalizzazione sarà applicato solo ad immagini in scala di grigi a 256 livelli e deve trasformare ogni suo pixel nel modo 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)
```

Dove MAX_PIXEL_VALUE e MIN_PIXEL_VALUE, sono il massimo e minimo valore dei pixel dell'immagine, CURRENT_PIXEL_VALUE è il valore del pixel da trasformare, e NEW_PIXEL_VALUE è il valore del nuovo pixel.

1.2 Struttura Memoria

Tutti i valori necessari allla computazione sono presenti in una memoria con indirizzamento al Byte, come descritto di seguito. Le dimensioni dell'immagine sono definite da due valori, ciascuno di 8 bit, il numero di righe e il numero di colonne.
Il byte in posizione 0 si riferisce al numero di colonne (N_COL), il byte in posizione 1 si riferisce al numero di righe (N_RIG).
I pixel del'immagine, ciascuno di un 8 bit, sono memorizzati in indirizzi contigui partendo dalla posizione 2.
I pixel della immagine equalizzata, ciascuno di un 8 bit, andranno memorizzati in indirizzi contigui partendo dal primo indirizzo inutilizzato, ossia dalla posizione 2+(N-COL*N-RIG).

0	numero di colonne
1	numero di righe
2	primo pixel immagine originale
...	...
n_rig * n_col +1	ultimo pixel immagine originale
n_rig * n_col +2	primo pixel immagine modificata
...	...
2 * n_rig * n_col +1	ultimo pixel immagine modificata

1.3 Interfccia componente

Il componente da descrivere deve avere 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

2.1 Scelta dell'architettura

In prima istanza si è provveduto ad un analisi superficiale dell'algoritmo da implementare, deducendo alcune importanti considerazione che si sono rivelate poi fondamentali per la scelta dell'adozione di una Machina a Stati Finiti per risolvere il problema proposto.
In particolare si può osservare che :

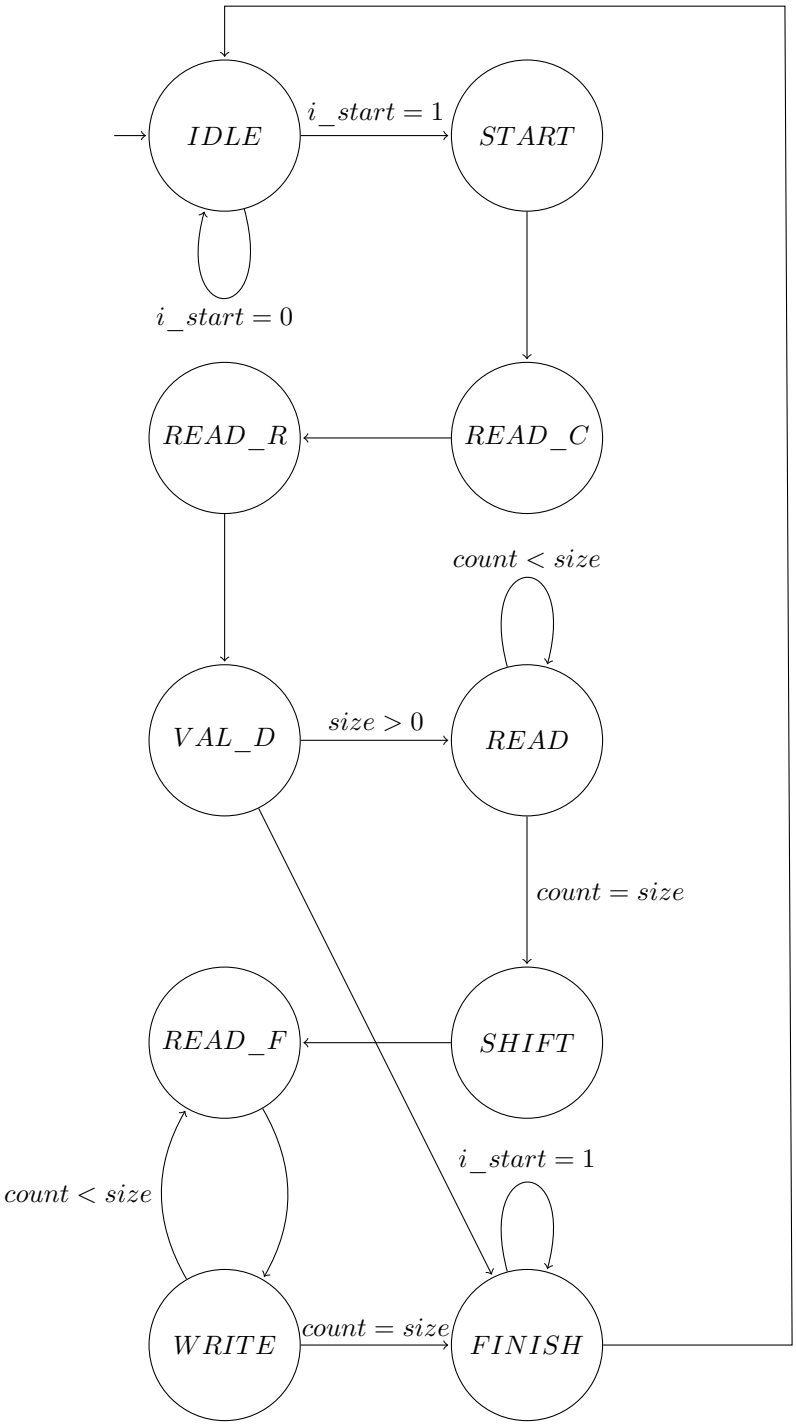
- noto il valore dello SHIFT_LEVEL si può facilmente ricavare il valore finale di ogni pixel in maniera indipendente
- per calcolare il valore SHIFT_LEVEL è fondamentale conoscere il valore di MAX_PIXEL_VALUE e MIN_PIXEL_VALUE
- il valore massimo e minimo assunti da almeno un pixel in tutta l'immagine si possono ottenere solo dopo una lettura esaustiva di tutti i valori dei pixel che compongono l'immagine

Si è optato per dividere il problema in due fasi, una prima fase in cui avviene il calcolo dei valori sopra menzionati e una seconda in cui si vanno a leggere i valori dei pixel, elaborarli e scrivere il risultato finale. Si noti che la memoria a nostra disposizione è in grado di fornire o di scrivere un solo Byte(pixel) alla volta, pertanto quest'ultima operazione sarà sequenziale.

Dato il potenziale alto numero di pixel con cui si ha a che fare (il valore massimo è di 16384 pixels) si è pensato che la soluzione migliore sia quella di NON salvare i valori dei pixel una volta letti durante la prima fase, dunque di andarli a rileggere una seconda volta per poi poterne calcolare il valore finale.

2.2 FSM

La Macchina a stati finiti elaborata è composta da 10 stati. Oltre all'opportuno registro usato per memorizzare lo stato ne vengono adoperati altri per memorizzare ulteriori valori di fondamentale importanza per la computazione. Inoltre al fine di una migliore comunicazione con la RAM esterna, che legge i valori dei segnali sul fronte di salita, si è optato per adottare dei registri come buffer in uscita con aggiornamento sul fronte di discesa, così da evitare comportamenti indesiderati. Quest'ultimo buffer non è strettamente parte della architettura della macchina a stati finiti, tuttavia per semplicità si è pensato di integrare tutti questi processi in un'unica entità, sebbene poi all'interno di essa vi siano 4 processi.


IDLE
i_start = 1
START
i_start = 0
READ_R
READ_C
count < size
VAL_D
size > 0
READ
count = size
READ_F
SHIFT
i_start = 1
count < size
WRITE
count = size
FINISH

2.2.1 Stati della macchina

IDLE: Stato in cui si trova la macchina dopo aver terminato una computazione, o dopo aver ricevuto un segnale di reset. I valori dei registri vengono mantenuti costanti e non vi è interazione con la memoria

FINISH: Stato in cui giunge la macchina al termine della computazione, il segnale `o_done` viene alzato. Quando il segnale `i_start` viene abbassato dal Test-Bench si torna nello stato di IDLE

START: Stato in cui viene portata la macchina quando viene alzato il segnale `i_start`. Questa operazione banale è fondamentale al fine di sincronizzare al meglio tutte le procedure successive dato che il segnale `i_start` è asincrono rispetto a `i_clk`. Viene mandata una richiesta alla memoria per leggere il valore all'indirizzo 0. Lo stato successivo è sempre READ_COL

READ_COL: Viene letto il numero delle colonne e salvato nella parte inferiore del registro `size`. Viene mandata una richiesta alla memoria per leggere il valore all'indirizzo 1. Lo stato successivo è sempre READ_RIG

READ_RIG: Vi ene letto il numero delle righe, moltiplicato per il numero di colonne e il risultato viene quindi salvato nel registro `size`. Il segnale `o_en` viene abbassato. Lo stato successivo è sempre VALIDATE_DATA

VALIDATE_DATA: Viene verificato che `size` sia maggiore di 0. Se non è così si passa direttamente a FINISH, altrimenti si procede a READ. Viene mandata una richiesta alla memoria per leggere il valore all'indirizzo 3

READ: Viene letto il valore in ingresso dalla memoria, confrontato con `min_val` e `max_val`, quest'ultimi vengono eventualmente aggiornati. Viene inoltre aggiornato `count` così da sapere quando la lettura è terminata e passare allo stato COMPUTE_SHIFT, altrimenti si rimane in questo stato. Viene mandata la richiesta per la lettura del valore successivo

COMPUTE_SHIFT: Vengono calcolate i valori dei registri `max_admissable` e `shift_level` mediante opportuni controlli a soglia. Il registro `count` viene riportato a 0, poiché si dovrà ricominciare a leggere da capo i valori. Viene mandata la richiesta di lettura del valore all'indirizzo 3. Lo stato successivo è sempre READ_FINAL

READ_FINAL: Viene mandata la richiesta di lettura del valore successivo. Lo stato successivo è sempre WRITE

WRITE: Viene letto il valore in input e viene mandata la richiesta di scrittura del valore opportunamente rielaborato. Viene inoltre aggiornato `count` così da sapere quando la scrittura è terminata e passare allo stato FINISH, altrimenti si rimane in questo stato.

2.2.2 Registri della macchina a stati finiti

`state`: indica lo stato della macchina tra quelli sopra elencati, inizialmente IDLE

`size` (16 bit): inizialmente 0, contiene poi la dimensione dell'immagine. Data la natura dell'algoritmo si è ritenuto non fosse necessario tenere separate le due dimensioni e calcore direttamente questo valore con una moltiplicazione tra due valori a 8 bit.

`count` (16 bit): inizialmente0, serve per sapere a che punto della lettura/scrittura si è. Non supera mai il valore di `size`

`max_val` (8 bit): valore del pixel "più grande" nell'immagine, inizialmente 0

`min_val` (8 bit): valore del pixel "più piccolo" nell'immagine, inizialmente 255

`shift_level` (4 bit): contiene il valore dello shift level, viene scritto nello stato COMPUTE_SHIFT e mai aggiornato, sebbene non sia strettamente necessario si è ritenuto di inserire questo registro per comprensibilità del codice. Assume valori tra 0 e 8 (inclusi)

`max_admissible` (9 bit): valore soglia usato per calcolare il risultato di ogni pixel. Rappresenta il massimo valore che la differenza tra il pixel preso in considerazione e il `min_val` può assumere senza che l'operazione di shift porti ad un risultato non completamente immagazzinabile in 8bit (ossia maggiore di 255), in quel caso infatti l'output non sarà determinato dallo shift ma è noto a priori (ossia proprio 255). Assume valori tra 1 e 256 (inclusi).

2.2.3 Processi della macchina a stati finiti

`state_reg`: aggiorna i registri della macchina a stati finiti sul fronte di salita. Gestisce eventuali segnali di reset

`delta`: Calcola lo stato prossimo della macchina e l'eventuale aggiornamento dei registri ausiliari

`lambda`: Calcola il valore delle variabili in uscita, i valori calcolati costituiscono in realtà l'input del buffer in uscita

`output_reg`: Aggiorna i registri del buffer in uscita sul fronte di discesa. Gestisce eventuali segnali di reset

3 Risultati Sperimentali

3.1 Prestazioni teoriche

Data la scelta architetturale di non andare a salvare ogni di pixel e dato il throughtput della memoria che abbiamo a disposizione, un evidente limite delle prestazioni, intese come numero di clock necessario per terminare tutte le operazioni, è dato da 3n (2 clock per le due letture, 1 clock per la scrittura), dove n è il numero di pixel dell'immagine. Si noti che anche nel caso si fosse optato per memorizzare i valori dei pixel all'interno del nostro componente hardware le prestazioni non avrebbero comunque potuto superare 2n (1 clock per la lettura, 1 clock per la scrittura). Alla luce di questa analisi teorica appare ancora più giustificata la decisione di non adottare quest'ultima strada dato che il tradeoff tra costi della memoria (che potenzialmente passerebbe da un occupazione O(1) a O(n)) e incremento di prestazioni non risulta conveniente.

3.2 Report di sintesi

La sintesi con il tool proposto(VIVADO 2020.2) viene portata a termine correttamente, senza riportare alcun warning. In particolare vengono riportati i seguenti utilizzi di risorse:

- Flip-Flop: 91
- Look-Up-Table: 208

3.3 Test eseguiti

Tutti i test sono stati simulati con un ciclo di clock di 10ns, ossia 10 volte inferiore a quello richiesto, le simulazioni sono state sia Behavioral (Pre-sintesi) che Functional (Post-sintesi). Si noti che la macchina proposta ha una piccola caratteristica non desiderabile che tuttavia poiché non va ad influire sulla computazione è stata mantenuta al fine di avere una maggiore semplicità sui controlli. Durante la prima lettura dei dati, (nello stato READ) viene sempre mandata una richiesta per un byte oltre a quelli necessari. Tale valore comunque non viene mai effettivamente utilizzato ed inoltre si è certi dell'esistenza di tale indirizzo poiché si tratta del primo indirizzo di scrittura. Per testare l'esattezza del codice si sono utilizzati principalmente tre test, di seguito presentati.

3.3.1 Test a

Il primo test è stato un test di copertura delle istruzioni (branch coverage), in particolare per quanto riguarda i controlli a soglia effettuati nello stato COMPUTE_SHIFT e la condizione di terminazione anticipata dello stato VALIDATE_DATA Il testbench proposto utilizzava quindi 9 immagini di dimensione 2x2, con dei valori opportuni dei pixel in modo da andare a coprire tutti i possibili intervalli di DELTA_VALUE(0, 1-2, 3-6, 7-14, 15- 30, 31-62, 63-126, 127-254, 255) e conseguentemente tutti i possibili valori di SHIFT_LEVEL (0 – 8). Vi erano poi 2 immagini "errate" ossia con dimensione rispettivamente 0x2 e 2x0 con i byte successivi riempiti con dei valori casuali. Si andava a verificare che non venissero modificati i suddetti valori.

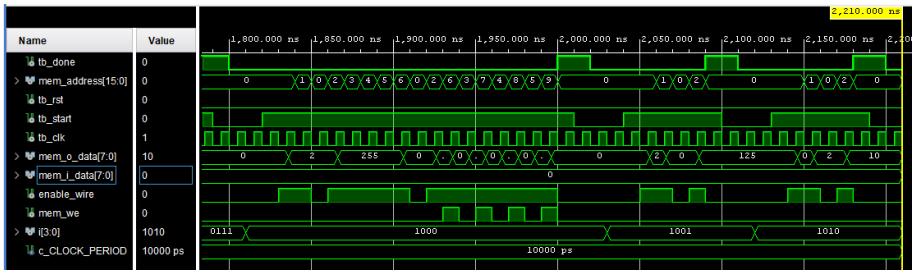


Figure 1: Test A Wave form: ultime tre immagini

3.3.2 Test b

Il secondo test era un test che andava a verificare la corretta computazione per immagine di varie dimensioni. In particolare sono state testati i corner case, ossia un immagine di dimensione 1x1 (quelle di dimensione zero sono state testate nel test precedente) e 128x128(si noti che l'architettura proposta potrebbe supportare immagini con una dimensione fino a 255x255). Anche in questo caso oltre ai valori dei pixel dell'immagine ci si assicurava che anche i byte successivi non venissero modificati. Si riporta la waveform del test dell'immagine 1x1.

3.3.3 Test c

Il terzo test è un test estensivo, composto da 10mila immagini di dimensione variabile, generate in maniera casuale. Al fine di avere valori di DELTA_VALUE differenti si è fissata una dimensione massima dell'immagini a 16x16. Si tratta di un test particolarmente impegnativo che il compontene realizzato è stato in grado di terminare in `23220590 ns`

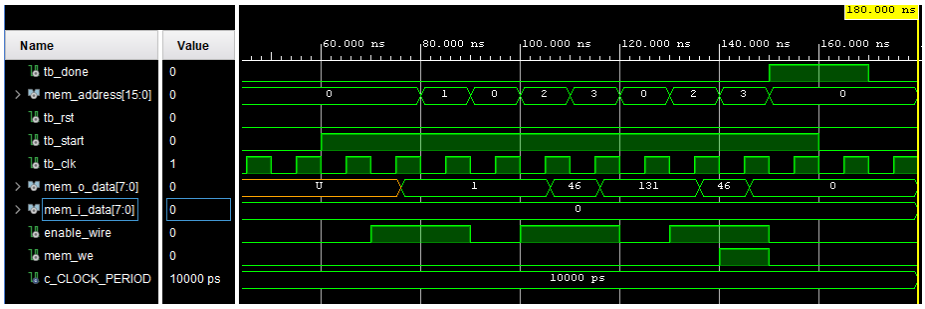


Figure 2: Test B Wave form

4 Conclusioni

È stato progettato e realizzato, tramite descrizione VHDL, un componente hardware in grado di eseguire la computazione dell'algoritmo semplificato di equalizzazione dell'istogramma dell'immagine. Al fine della realizzazione è stata utilizzata un architettura di una Macchina a Stati Finiti (FSM) che meglio si prestava a risolvere il problema proposto. Al fine di minimizzare il numero di cicli di clock per interagire con la memoria del testbench è stato inserito un registro con fronte di aggiornamento sul fronte di discesa che funziona dei buffer dei segnali in uscita. Questa scelta ha permesso di raggiungere prestazioni consistenti e ritenute soddisfacenti, inoltre sono stati superati test esaustivi sia in Behavioral Simulation che in Functional Simulation.