Prova Finale (Progetto Reti Logiche)

Alessandra de Stefano - Codice Persona: 10606454 - Matricola: 906918
Valeria Detomas - Codice Persona: 10615309 - Matricola: 912207

Prof. William Fornaciari - Anno 2020/2021

Indice

1 Introduzione

1.1 Scopo del Progetto

Lo scopo del progetto è l'implementazione di un componente hardware, in VHDL, in grado di ricalibrare il contrasto delle immagini. Quando i valori di intensità di un'immagine sono molto vicini essa risulta meno chiara ma, ridistribuendo i valori di intensità è possibile aumentare il contrasto.

1.2 Specifiche di Progetto

Per portare un'immagine a basso contrasto a una ad alto contrasto possiamo implementare una versione semplificata dell'algoritmo di equalizzazione. Sono state prese in considerazione solamente le immagini in scala di grigi a 256 livelli, cioè formate da pixel che assumono valori da 0 (nero) a 255 (bianco).
L'algoritmo deve trasformare ogni pixel dell'immagine nel seguente modo:

DELTA_VALUE = MAX_PIXEL_VALUE – MIN_PIXEL_VALUE
SHIFT_LEVEL = (8 – FLOOR(LOG_2(DELTA_VALUE +1)))
TEMP_PIXEL = (CURRENT_PIXEL_VALUE - MIN_PIXEL_VALUE) <<SHIFT_LEVEL
NEW_PIXEL_VALUE = MIN(255 , TEMP_PIXEL)

MAX_PIXEL_VALUE e MIN_PIXEL_VALUE sono rispettivamente il valore massimo e il valore minimo assunti dai pixel che compongono l'immagine. CURRENT_PIXEL_VALUE è il valore del pixel da trasformare mentre NEW_PIXEL_VALUE è il valore del nuovo pixel.

1.3 Dati

Le dimensioni dell'immagine sono di 8 bit, memorizzate con indirizzamento al byte. Il byte all'indirizzo 0 si riferisce al numero di colonne, quello all'indirizzo 1 al numero di righe. Il byte successivo, quello all'indirizzo 2, si riferisce al primo byte dell'immagine. Quest'ultima verrà letta da una memoria in cui è memorizzata sequenzialmente, riga per riga.
L'immagine equalizzata verrà scritta in memoria a partire dall'indirizzo che segue l'ultimo pixel dell'immagine originale.

Indirizzo 0:	Numero Colonne
Indirizzo 1:	Numero Righe
Indirizzo 2:	Primo Pixel immagine originale
....	....
Indirizzo n:	Ultimo Pixel immagine originale
Indirizzo n+1:	Primo Pixel immagine equalizzata
Indirizzo n+2:	Secondo Pixel immagine equalizzata
...	...

Table 1: Rappresentazione Indirizzi della Memoria

1.4 Interfaccia del componente

L'interfaccia del componente da descrivere è definita come:

```
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 dilettura;
- `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
- `o_en` è il segnale di `enable` da dover mandare alla memoria per poter comunicare (sia 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 Scelte Progettuali

2.1 Descrizione generale

L'algoritmo utilizzato dal nostro componente si basa su una macchina a stati finiti, la quale descrive in ogni stato le azioni da compiere. Inizialmente si attende nello stato `INITIAL` che il segnale `i_start` venga portato a 1. In questo stato vengono inoltre inizializzati i segnali necessari, cosicchè sia possibile una seconda elaborazione senza che il segnale `i_rst` torni alto. A questo punto il componente inizia l'elaborazione spostandosi allo stato stato successivo iniziando la computazione. Al termine della computazione si attende che il segnale `i_start` venga portato basso per poi portare a 0 il segnale `o_done`. Se a questo punto il segnale `i_start` torna alto significa che il modulo deve ripartire con la codifica di un'altra immagine.

2.2 Scelte implementative

Per implementare il componente richiesto nella specifica è stato ritenuto opportuna una soluzione a due processi.

Il primo processo ha lo scopo di descrivere la logica sequenziale. In corrispondenza del fronte di salita del clk viene memorizzato nello stato presente il valore dello stato prossimo

Il secondo processo ha invece lo scopo di determinare lo stato prossimo della macchna a stati finiti a partire dallo stato presente e dai segnali d'ingresso

2.3 Stati della Macchina

2.3.1 INITIAL state

Stato iniziale in cui viene fatto un set up di tutti i segnali necessari al funzionamento e si attende che i_start venga portato a 1. Inoltre si torna in questo stato se `i_rst` si alza.

2.3.2 WAIT_DATA state

Stato necessario per attendere la risposta della memoria dopo la richiesta di un dato.

2.3.3 SAVE_DIM state

Stato in cui viene letto il dato che contiene il numero di colonne all'indirizzo 0 e, il numero di righe all'indirizzo 1. In questo modo è possibile calcolare la dimensione dell'immagine allo stato successivo.

2.3.4 COUNT_PIXEL state

Stato in cui si contano quanti pixel compongono l'immagine senza adoperare l'operatore moltiplicazione. Ogni riga contiene tanti pixel quanti il numero delle colonne che compongono l'immagine. Tenendo a mente ciò, è possibile sommare il numero delle colonne tante volte quante le righe.

2.3.5 MAXMINDELTA state

Stato in cui vengono letti uno ad uno i pixel dell'immagine. Se il pixel letto è maggiore del segnale max (inizializzato a 0), allora il pixel appena letto è il "nuovo max". Altrimenti, viene effettuato lo stesso controllo con il segnale min(inizializzato a 255). Iterando per tutti i pixel si determinano massimo e minimo valore di intensità dell'immagine. Ora è possibile calcolare il delta (max - min + 1).

2.3.6 CTR_SOGLIA state

Stato che permette di calcolare di quanto ogni pixel viene shiftato tramite un controllo a soglia del valore delta.

2.3.7 NEW_VALUE state

Stato in cui viene calcolato il nuovo valore di ogni pixel. `o_we_next` è a '1' perché allo stato successivo verrà scritto in memoria il nuovo dato calcolato.

2.3.8 WRITE state

Stato in cui si stabilisce se il valore del pixel calcolato allo stato precedente è maggiore o minore di 255. A questo punto il componente scrive in memoria, a partire dall'inidirizzo successivo all'ultimo pixel dell'immagine, il valore del pixel equalizzato

2.3.9 WAIT_RAM state

Stato che permette alla RAM di elaborare la richiesta.

2.3.10 WAIT_DATADUE state

Stato necessario per attendere la risposta della memoria dopo la richiesta di un dato.

2.3.11 DONE state

Stato in cui il componente attende che la memoria porti a 0 `i_start` per poter abbassare `o_done` e tornare allo stato `INITIAL`.

La memoria richiede un ciclo di clock per rendere disponibile il dato. Per questo motivo, abbiamo deciso di aggiungere uno stato di attesa del dato (`WAIT_DATA`, `WAIT_DATADUE`) seguito dallo stato che esegue effettivamente la lettura di tale dato. Di seguito viene fornita la macchina a stati finiti presa in cosiderazione. Si noti che non sono stati rappresentati gli archi che da ogni stato vanno direttamente allo stato `INITIAL` quando si alza il segnale `i_rst`.

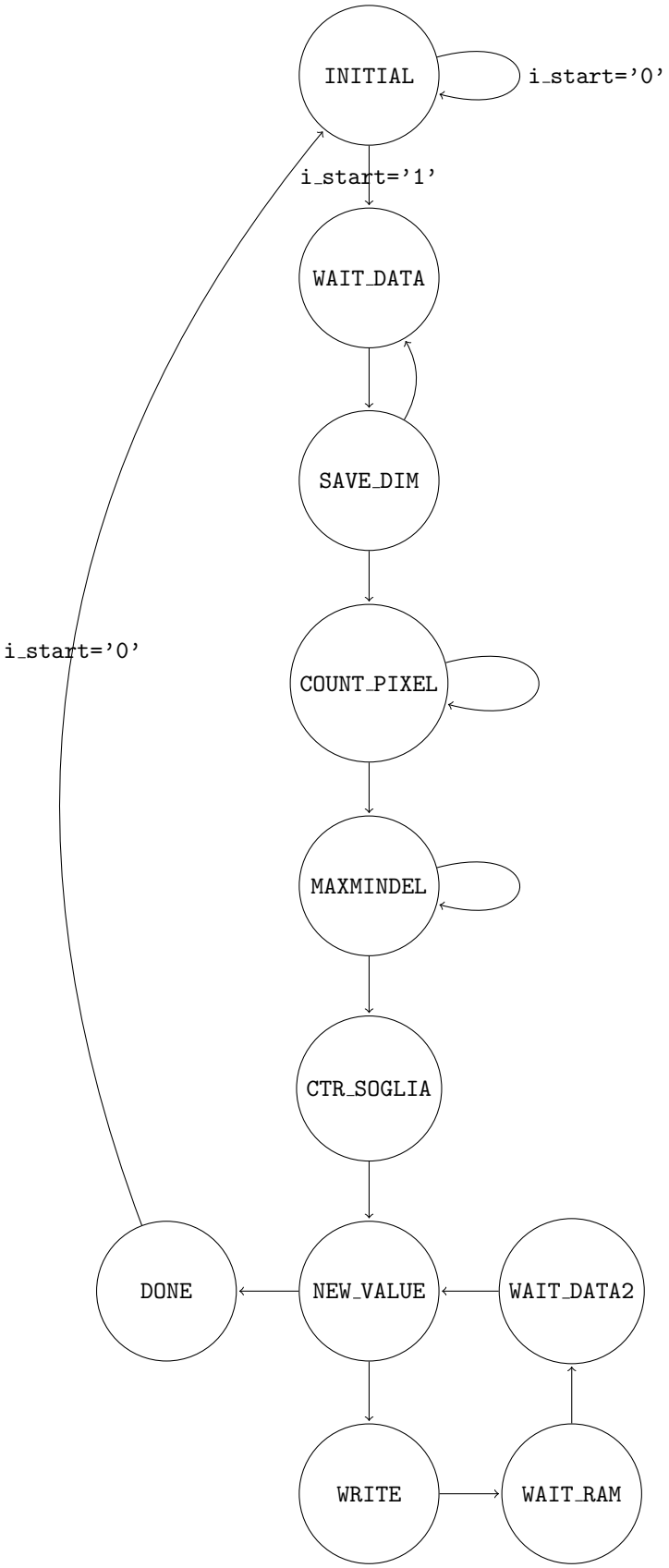

Figure 1: Macchina a Stati Finiti

3 Testing

Per verificarne il corretto funzionamento, il componente è stato testato sia con il testbench d'esempio sia con altri testbench da noi definiti. Questi ci hanno permesso di verificare la correttezza del nostro codice anche nei casi limite della computazione.
Il componente sintetizzato supera correttamente tutti i test a livello di:

- Behavioral
- Post-Synthesis Functional
- Post-Synthesis Timing

I test realizzati verificano i seguenti casi:

3.1 Immagini di dimensioni massime

Verifica la correttezza della lettura delle immagini di dimensioni massime consentite dalla specifica, cioè un numero di righe e di colonne pari a 128. Il componente in questo caso finisce la computazione salvando all'indirizzo 37769 l'ultimo pixel dell'immagine equalizzata.

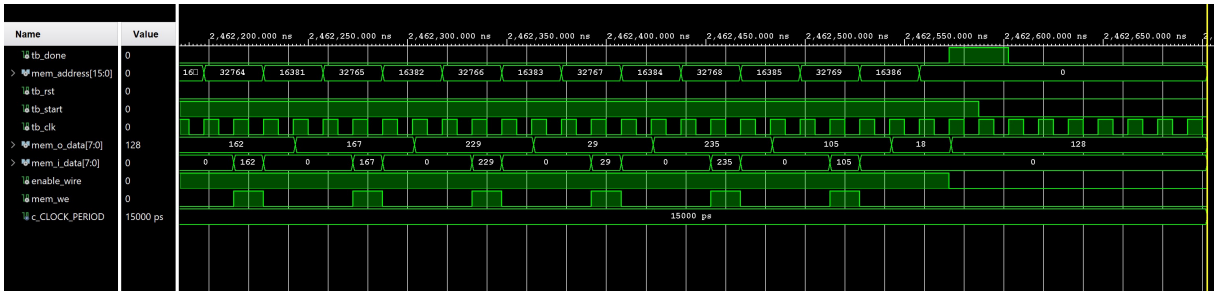

Figure 2: Simulazione Testbench Immagine 128x128 .

3.2 Immagini di dimensioni nulle

Verifica la correttezza della lettura di immagini che hanno dimensioni di righe o colonne uguale a zero. In questi casi non vengono letti altri dati e la computazione termina alzando il segnale di `o_done` senza scrivere nessun dato in memoria.

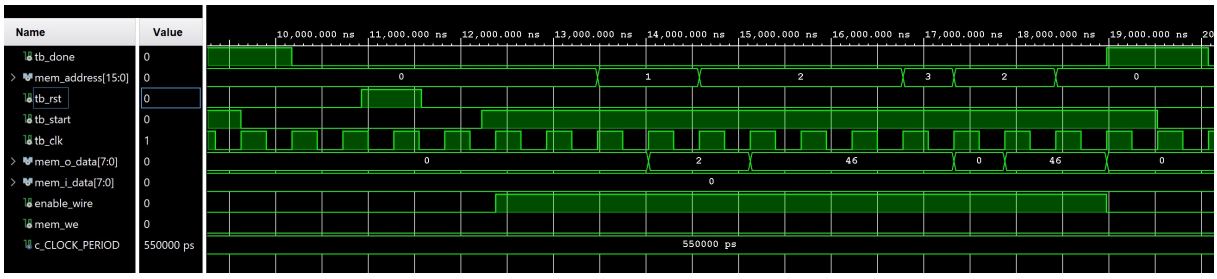

Figure 3: Simulazione Testbench Immagine 0x0.

3.3 Segnale di reset durante l'esecuzione

Verifica che nel caso di ricezione di un segnale di reset la computazione non venga compromessa e che questa riinizi, facendo ritornare la macchina allo stato iniziale INITIAL.

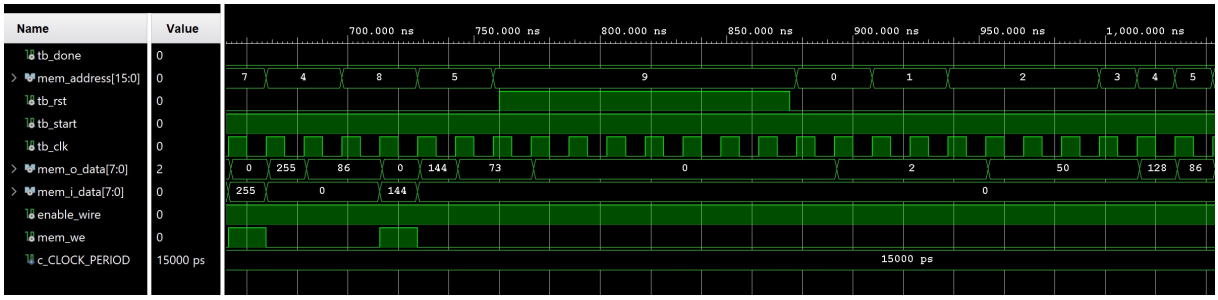

Figure 4: Simulazione Testbench Reset Asincrono .

3.4 Computazioni immagini consecutive

Richiede l'equalizzazione di più immagini consecutive verificando che il componente riesca a computare correttamente le immagini senza aspettare che il segnale di reset torni alto. In questo caso viene inoltre verificato che il componente inizializzi correttamente i segnali tra due computazioni consecutive.

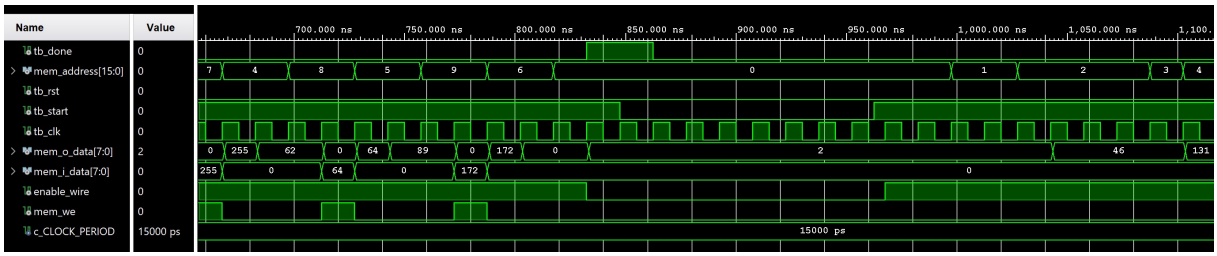

Figure 5: Simulazione Testbench Tre Immagini Consecutive.

3.5 Ulteriori test

Dopo aver verificato il corretto funzionamento del componente nei casi limite, sono stati svolti ulteriori test randomici per testare ulteriormente il componente sintetizzato. Ad esempio, sono stati svolti test in cui l'immagine era formata da tutti pixel con valore di intensità 255 e successivamente, tutti pixel di valore 0. Per accertarsi inoltre che l'individuazione del max e min avvenisse correttamente, abbiamo testato il codice con una immagine i cui valori max e min erano gli ultimi pixel dell'immagine.

4 Conclusione

Il numero di stati della FSM realizzata è stato ridotto grazie a delle ottimizzazioni di lettura di righe e colonne e del calcolo di max min e delta.
Considerando che il componente supera tutti i test a cui è stato sottoposto, si ritiene di averlo sintetizzato correttamente, avendo quindi svolto il lavoro richiesto.