Prova Finale (Progetto di Reti Logiche)

Prof William Fornaciari – Anno 2020/2021

Hanson Obinna Anozie (Codice Persona 10581641 – Matricola 889941)

Indice

Introduzione

Scopo del progetto

Lo scopo del progetto è di implementare un componente hardware descritto in VHDL, in grado di equalizzare l'istogramma di un'immagine. In generale si otterrà un'immagine con un contrasto maggiore, soprattutto quando l'intervallo dei valori di intensità sono molto vicini. Quindi, i vari valori saranno meglio distribuiti sull'istogramma, in modo da consentire alle zone con un basso contrasto di ottenerne uno più alto.

Specifiche generali

In ingresso il componente riceverà il numero di colonne dell'immagine (N_COL), il numero di righe (N_RIG) e (N_COL*N_RIG) numeri che rappresentano il valore assunto da ogni pixel. Il componente dovrà poi trasmettere in uscita il valore dei rispettivi pixel equalizzati, in base ai seguenti calcoli:

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 DELTA_VALUE rappresenta la differenza tra il massimo e il minimo valore assunto dai pixel dall'immagine, CURRENT_PIXEL_VALUE è il valore del pixel all'i-esima computazione e NEW_PIXEL_VALUE è il valore che dovrà assumere in uscita.

Interfaccia del componente

Il componente da descrivere possiede 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 test bench;
- i_rst è il segnale di RESET che inizializza la macchina pronta per ricevere il 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 poterci scrivere. Per leggere da memoria esso deve essere 0;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria.

Dati e descrizione della memoria

I dati, ognuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte:

- L'indirizzo 0 è usato per memorizzare il numero di colonne;
- L'indirizzo 1 è usato per memorizzare il numero di righe;
- Gli indirizzi da 2 a (N_COL * N_RIG +1) sono utilizzati per memorizzare i pixel dell'immagine originale;
- Gli indirizzi da (N_COL * N_RIG +2) a (2*N_COL * N_RIG +1) sono utilizzati per memorizzare i pixel dell'immagine equalizzata.

Design

Quando il segnale i_start in ingresso viene settato a 1, il componente sviluppato inizia l'elaborazione spostandosi dallo stato IDLE al primo stato della computazione. Una volta terminata la computazione, dopo aver scritto l'ultimo pixel equalizzato in memoria, il componente alza il segnale o_done. Il test bench risponde portando i_start a 0 e, di conseguenza il componente riporta a 0 o_done. Il componente ritorna nello stato IDLE in attesa che il segnale i_start ritorni alto. Il componente possiede in aggiunta, il segnale i_rst che, insieme agli altri precedentemente elencati, mi ha portato realizzare una macchina a stati finiti con datapath, cioè abbina una normale FSM con dei circuiti sequenziali.
Nel seguito verrà presentata sia la descrizione della FSM sia la descrizione della parte sequenziale del componente che consente la gestione dei vari registri.

Stati della FSM

1.IDLE

Stato iniziale in cui si attente che il segnale i_start venga portato a 1. Ogni volta che il segnale i_rst viene alzato si ritorna in questo stato.

2.ASK_CONST

Stato in cui viene richiesto il numero di colonne dell'immagine e in seguito il numero di righe. Il tutto viene eseguito tornando due volte in questo stato, dato che non è possibile ottenerle entrambi nello stesso tempo.

3. WAIT_RAM

Stato in cui si attente la risposta della memoria dopo la richiesta di un dato.

4.GET_CONST

Stato in cui vengono salvati in due rispettivi registri, il numero di colonne dell'immagine e il numero di righe.

5.MULT
Stato in cui vengono controllati il numero di colonne e di righe dell'immagine. Se uno dei due (o entrambi) è uguale a 0 il componente alza il segnale o_done e cambia il suo stato in DONE. Altrimenti, viene calcolato N_COL*N_RIG mediante somme successive. Tale valore serve alla macchina per sapere quando terminare la computazione.

6.WAIT_MULT
Stato in cui si attende l'aggiornamento dei registri utilizzati per calcolare N_COL*N_RIG.

7.ASK_CURR
Stato in cui viene semplicemente richiesto l'i-esimo pixel per proseguire la computazione.

8.GET_CURR
Stato in cui viene memorizzato in un registro l'i-esimo pixel della computazione.

9.CALC
Stato in cui si confronta l'i-esimo pixel con MAX_PIXEL_VALUE e MIN_PIXEL_VALUE quelli salvati nei registri, eventualmente possono essere sovrascritti. Se tale operazione è già stata eseguita, viene calcolato il valore di NEW_PIXEL_VALUE.

10.WRITE_OUT
Stato in cui viene scritto il nuovo valore dell'i-esimo pixel. Se il componente si trova all'ultimo pixel, esso alza il segnale o_done.

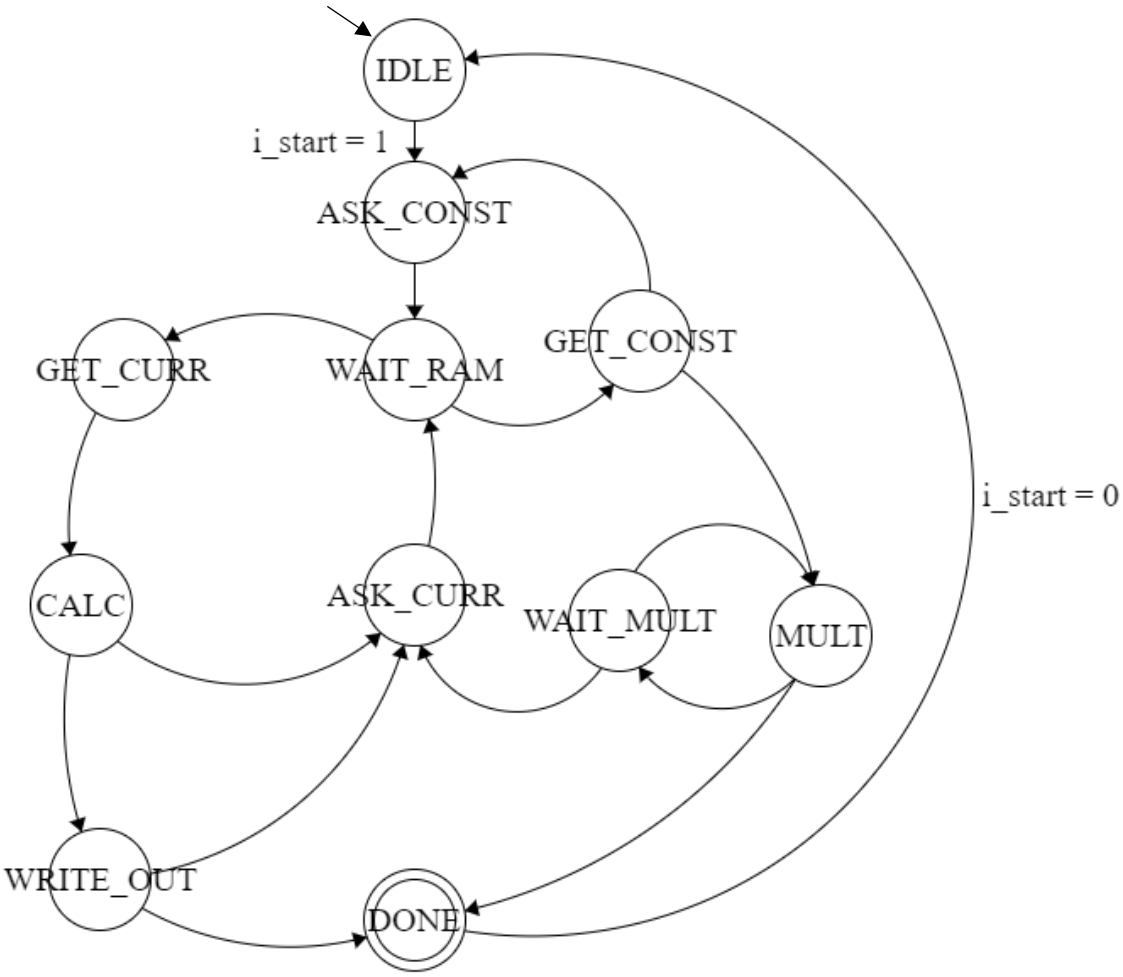
11.DONE
Stato in cui il componente attende che il test bench abbassi il segnale i_start per poter abbassare o_done e tornare nello stato IDLE, ed eseguire eventualmente una nuova computazione.

Scelte Progettuali

Il componente è dotato di due processi:

1. Il primo rappresenta la parte sequenziale e serve a gestire i registri
2. Il secondo rappresenta la FSM che in base ai vari segnali e lo stato corrente determina il prossimo stato in cui si troverà il componente.

L'algoritmo calcola il risultato finale visitando una volta tutti i pixel dell'immagine per poter calcolare MAX_PIXEL_VALUE e MIN_PIXEL_VALUE e una seconda volta per calcolare ogni NEW_PIXEL VALUE. Il tutto con una complessità temporale O(n) e spaziale O(1) in quanto in memoria non sono mai stati salvati dati non necessari.
Ho utilizzato operazioni di SOMMA per incrementare l'indirizzo di memoria, SHIFT_LEFT per eseguire il calcolo di TEMP_PIXEL e diversi CAST per convertire il tipo dei vari segnali per eseguire determinate operazioni.



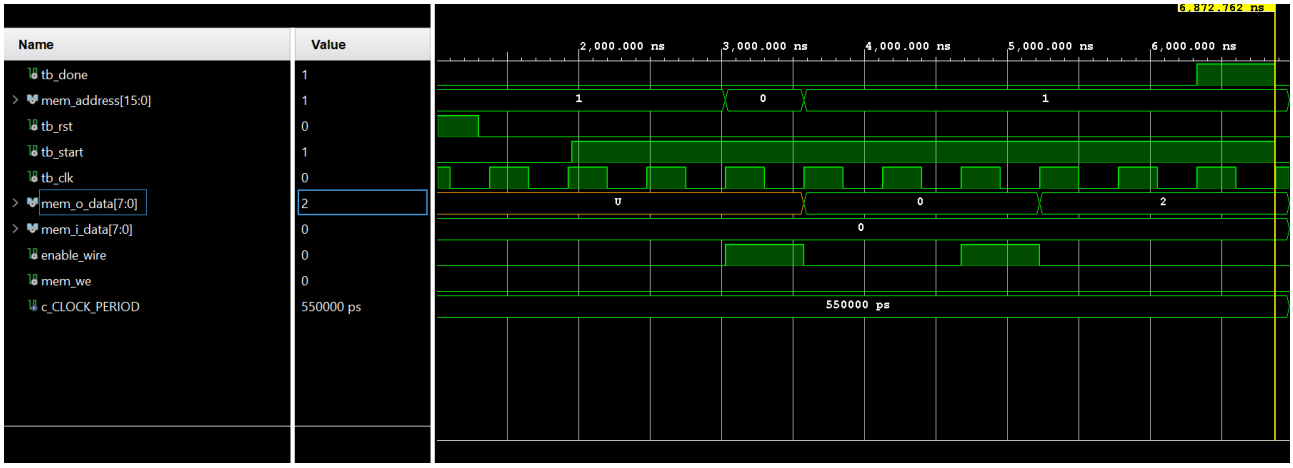
Rappresentazione della Macchina a Stati

Risultati dei test

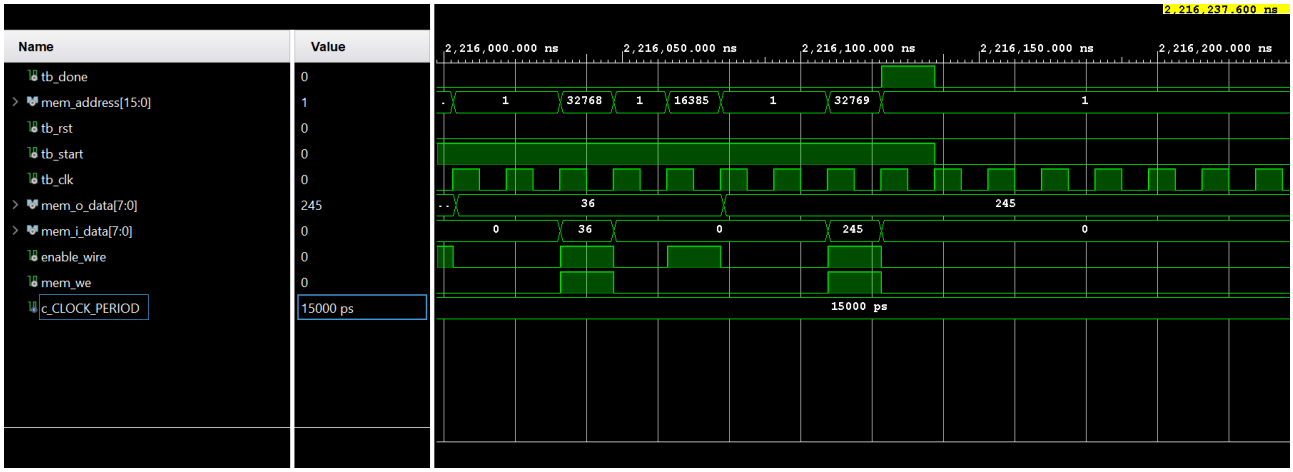
Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con i test bench di esempio, ho definito altri test che verificano il corretto comportamento del componente durante i corner case, in modo di massimizzare la copertura di tutti i possibili cammini che l'automa può effettuare durante la computazione. In seguito, verrà mostrata una breve descrizione dei test più significativi con immagini dell'andamento dei segnali durante la simulazione.

I test bench per la verifica dei corner case sono 2:

1. **Immagine 0p**: l'immagine in ingresso ha dimensione nulla. Il test verifica che il componente non scriva niente in memoria.

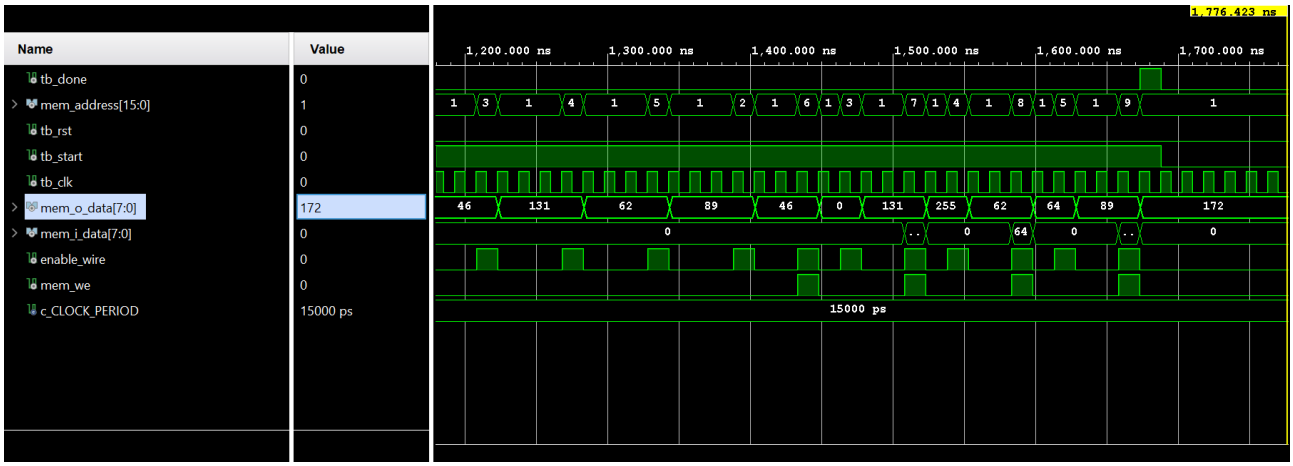


2. **Immagine 128p x 128p:** L'immagine in ingresso ha dimensione massima. Il test verifica che il componente restituisca un'immagine equalizzata correttamente.

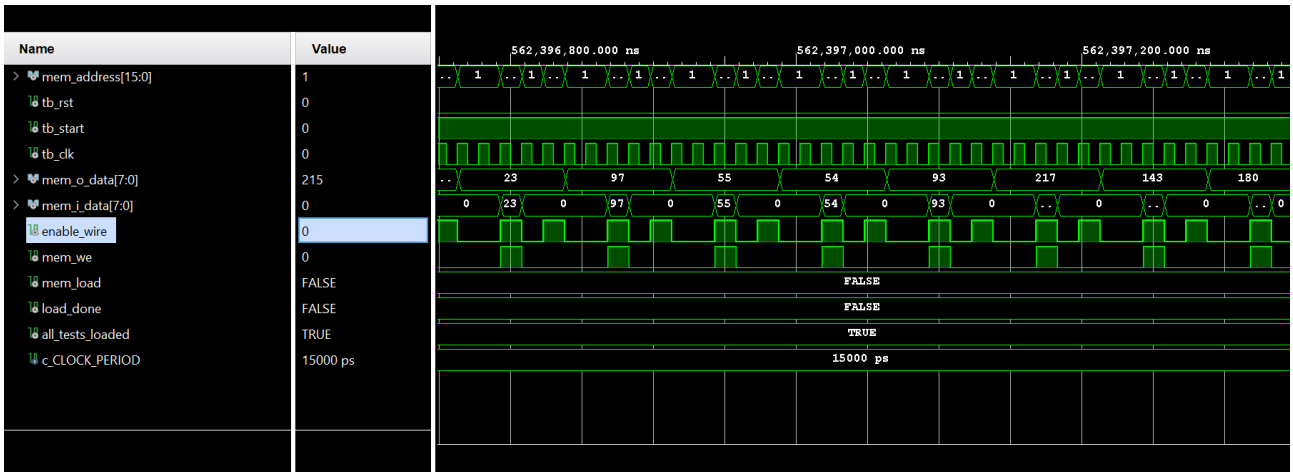


I test bench che verificano il corretto funzionamento dei segnali sono i seguenti:

1. **Reset Asincrono:** il test verifica che un segnale di reset inaspettato non comprometta la computazione e che essa ricominci facendo ritornare l'automa nello stato iniziale IDLE.



2. **1000 immagini con dimensione massima 128p x 128p:** il test verifica la corretta sincronizzazione dei segnali i_start, i_rst, o_done e la capacità del componente di equalizzare immagini una dopo l'altra.



Quest'ultimo test è stato realizzato grazie a uno script in Python che genera un gran numero di casi di test. L'output della computazione viene salvato in un file .txt indicando quanti test sono terminati correttamente.

Conclusioni

Risultato della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle tre simulazioni: Behavioral, Post-Synthesis Functional e Post-Synthesis Timing. Qui di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner case che portano la macchina verso la computazione più corta e quella più lunga.

Immagine 0p

Failure: Simulation Ended! TEST PASSATO
Time: 6975100 ps Iteration: 0 Process: /project_tb/test File: C:/Users/hans2/Downloads/Libri/Progetto RL/test_Salice/TB2/ZeroPixel.vhd
$finish called at time : 6975100 ps : File "C:/Users/hans2/Downloads/Libri/Progetto RL/test_Salice/TB2/ZeroPixel.vhd" Line 107

Immagine 128p x 128p

Failure: Simulation Ended! TEST PASSATO
Time: 2216237600 ps Iteration: 0 Process: /project_tb/test File: C:/Users/hans2/Downloads/Libri/Progetto RL/test_Salice/TB2/Generato128_128_1.vhd
$finish called at time : 2216237600 ps : File "C:/Users/hans2/Downloads/Libri/Progetto RL/test_Salice/TB2/Generato128_128_1.vhd" Line 32875
run: Time (s): cpu = 00:00:04 ; elapsed = 00:00:10 . Memory (MB): peak = 2246.320 ; gain = 0.000

Ottimizzazioni

Le ottimizzazioni che ho attuato sono state principalmente nella riduzione del numero di stati. Inizialmente avevo realizzato un automa che richiedesse il numero di colonne e righe, calcolasse massimo, minimo e NEW_PIXEL_VALUE in stati differenti. Ho quindi deciso di raggruppare la richiesta del numero di colonne e righe in un singolo stato, stessa cosa anche per le operazioni di calcolo di massimo/minimo e calcolo di NEW_PIXEL_VALUE. Inoltre, inizialmente calcolavo N_COL*N_RIG mediante l'operatore * ma dopo la mail di spiegazione inviata dal Prof. Terraneo ho capito che nonostante tale metodo fosse decisamente più intuitivo da implementare, non è altrettanto efficiente dal punto di vista hardware.
Un'altra ottimizzazione a cui ho pensato è stata quella di eseguire lil calcolo solo quando necessario. Infatti, se il numero di righe o di colonne è uguale a 0 la macchina termina immediatamente la computazione.