Report per il progetto di Reti Logiche

Politecnico di Milano
Autori:
Matteo Regge(10619213)
Nicolas Rossi(10662457)
Docente: Gianluca Palermo

1 Scopo del progetto

Il progetto consiste nell'implementare un componente hardware descritto in VHDL, che preso in ingresso la sequenza di bit che compone un immagine, ne calcoli una versione equalizzata. In particolare implementiamo un algoritmo semplificato nel quale le dimensioni massime dell'immagine sono 128x128, su una scala di grigi composta da 256 livelli. Il componente deve essere in grado di accettare immagini multiple che verranno analizzate in sequenza.

1.1 Specifiche generali del progetto

L'algoritmo di equalizzazione dell'istogramma è formato da varie funzioni :

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)

Lo shift level rappresenta di quanto dobbiamo shiftare ogni pixel durante l'esecuzione dell'algoritmo.
Temp pixel è il valore del pixel una volta shiftato.
Nel caso questo valore superi 255, come new pixel value prendiamo comunque 255 in quanto lavoriamo con una scala di grigi a 256 livelli.
L'immagine viene letta da una memoria dove è memorizzata sequenzialmente.
Ogni byte corrisponde ad un pixel dell'immagine, al di fuori dai primi due byte che corrispondono alla dimensione dell'immagine, rispettivamente al numero di colonne e righe.

1.2 Interfaccia del componente

```
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;
```

- 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

2 Design

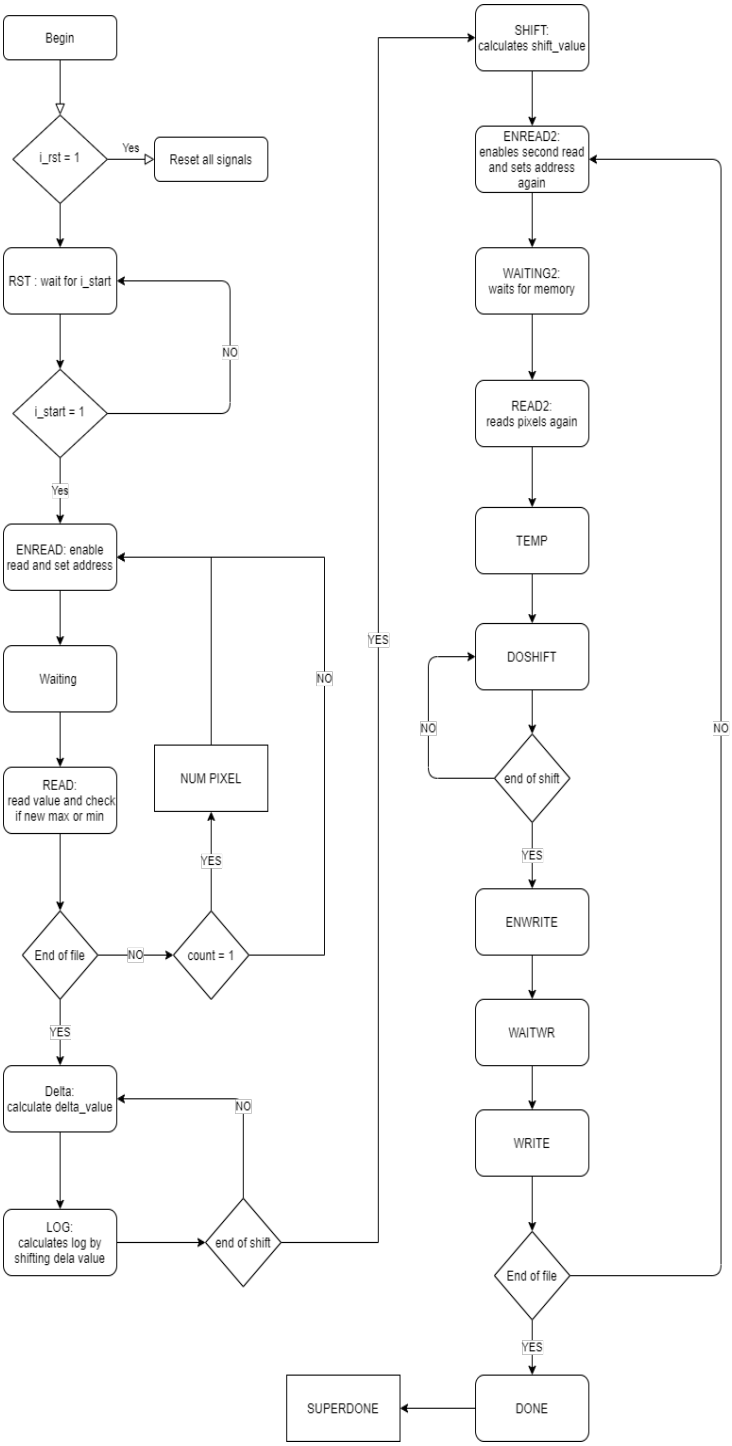
Abbiamo scelto di implementare un unico process che rappresenta la nostra macchina a stati.
Nel momento in cui il segnale di start viene portato a 1, la macchina si sposta sullo stato ENREAD e il componente comincia l'elaborazione.In attesa del segnale di reset la macchina rimane ferma nello stato di RST, ovvero lo stato in cui inizializziamo tutti i segnali.
La macchina termina l'esecuzione nello stato done dove alza il segnale o_done e va poi in superdone dove aspetta l'eventuale reset per la lettura di una nuova immagine.

Lo stato pixelNum è stato creato a seguito di problematiche che sorgevano nel momento di sintetizzare il componente. Abbiamo infatti notato che il modo in cui facevamo il check per vedere se avevamo finito di leggere tutti i pixel, rompeva l'esecuzione del componente post-synthesis.

2.1 Stati della macchina

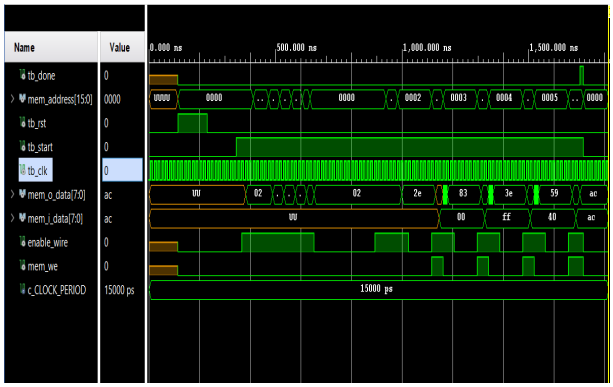
- RST : Stato in cui si attende l'arrivo del segnale i_start e in cui si ritorna nel caso venga alzato il segnale i_rst.
- ENREAD : Stato in cui alziamo il segnale o_en per abilitare la lettura da memoria e in cui aggiorniamo l'indirizzo di memoria dal quale leggere il byte.
- WAITING: Stato in cui aspettiamo la risposta della memoria a seguito di una richiesta di lettura.
- READ : Stato in cui operiamo la prima lettura dei pixel dell'immagine, e in cui calcoliamo il max e il min valore. Durante le prime due letture invece leggiamo e memorizziamo le dimensioni dell'immagine.
- PIXELNUM : Stato in cui calcoliamo le dimensioni dell'immagine dopo averne letto i valori.
- DELTA: Stato nel quale calcoliamo la delta_value.
- LOG: Stato che svolge l'operazione di logaritmo necessaria per calcolare la shift_value. Il valore delta_value viene shiftato di uno, e si ritorna in questo stato fino a che delta_value non è uguale a 1.
- SHIFT: Stato in cui calcoliamo la shift_value utilizzando il valore trovato allo stato precedente.
- READ2: Stato in cui leggiamo nuovamente i pixel dell'immagine.
- WAITING2 : Stato in cui aspettiamo la risposta alla memoria in seguito alla seconda richiesta di lettura.
- DOSHIFT : Stato dove viene svolta l'operazione di shift, su temp_pixel, necessaria per equalizzare l'immagine. Viene svolta utilizzando un counter e ciclando su questo stato fino a che il counter non è uguale alla shift_value.
- TEMP: Stato in cui calcoliamo il valore temp_pixel sottraendo al pixel corrente il valore di min.
- ENWRITE: Stato in cui alziamo i segnali o_n e o_we per abilitare la scrittura su memoria e in cui passiamo l'indirizzo dove andremo a scrivere.
- WRITE : Stato che svolge la scrittura su memoria del pixel equalizzato.
- DONE : Stato nel quale alziamo il segnale o_done;
- SUPERDONE : Stato in cui aspettiamo l'abbassamento del segnale i_start , una volta arrivato resettiamo i segnali della macchina e torniamo nello stato di RST dove attendiamo un nuovo start per una nuova immagine.

3 Diagramma macchina a stati

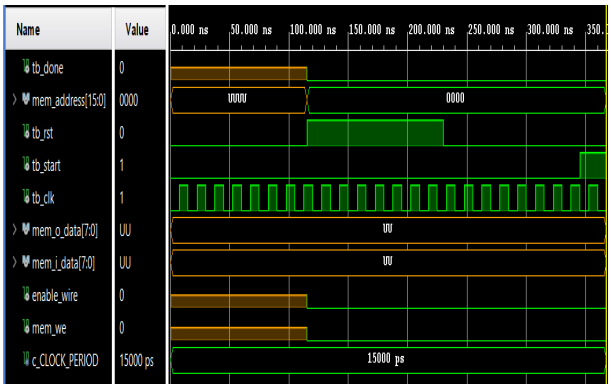


4 Risultati dei test

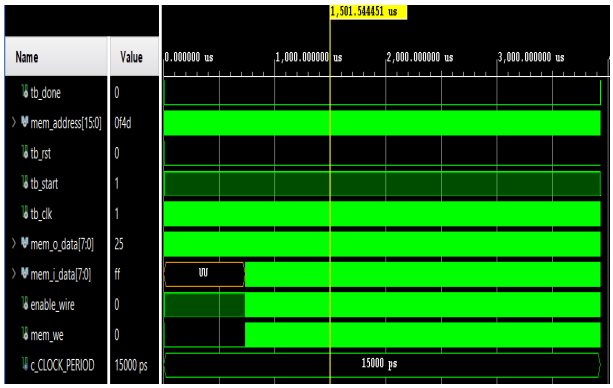
Test di base immagine 2x2 :



Abbiamo inoltre creato dei test aggiuntivi per testare alcuni casi limiti come quello dell'immagine di dimensione massima o l'immagine vuota.
Abbiamo inoltre testato la possibilità di equalizzare più immagini sequenzialmente.
Test immagine vuota :



Test immagine dimensioni massime :



5 Risultati sperimentali

Per verificare la correttezza del nostro algoritmo abbiamo deciso di selezionare alcune immagini, scalarne la risoluzione a 128x128, convertirle in scale di grigi e modificarne il contrasto in modo da rendere difficoltosa la visione del soggetto rappresentato.

Successivamente tramite uno script python abbiamo preso le immagini e abbiamo applicato lo stesso algoritmo sviluppato su Vivado e abbiamo stampato un testbench con l'immagine non corretta come input e l'immagine corretta come valori che ci aspettavamo. L'algoritmo ha terminato correttamente su entrambi i testbench riproducendo i risultati che combaciano con quelli dati dallo script python. Includiamo di seguito i due esempi delle immagini in input e degli output ottenuti.

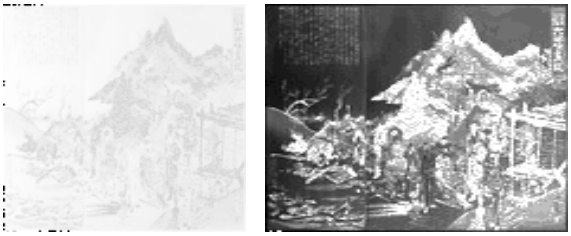
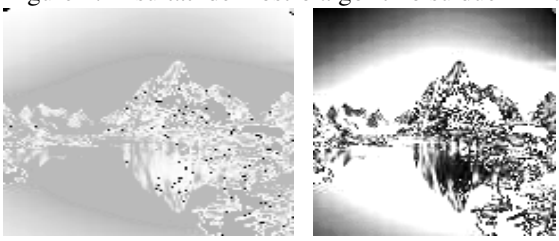


Figure 1: Risultati del nostro algoritmo su due immagini



6 Conclusioni

Il componente funziona correttamente sia in pre-sintesi che in post, in quanto supera i nostri test come ci aspettavamo. Pensiamo di aver raggiunto un buon livello di conoscenza delle funzionalità di vivado e del linguaggio VHD e che questo progetto lo dimostri.