1 Requisiti di progetto

1.1 descrizione del problema

Si vuole realizzare un modulo in grado di realizzare una versione semplificate dell’equalizzazione dell’istogramma di un’immagine, ossia di ricalibrare il contrasto di quest’ultima, effettuandone una ridistribuzione dei valori di intensità pixel per pixel.

1.3 interfaccia del componente

Il componente da realizzare ha un’interfaccia così definita:

\*code snippet

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.

1.2 descrizione della memoria e dell’interazione con il componente

Il modulo implementato dialogherà in lettura e scrittura con una memoria RAM, indirizzata al byte.

In particolare, l’algoritmo di equalizzazione sarà applicato ad immagini in scala di grigi a 256 livelli, con una dimensione massima di 128x128 pixel. La loro grandezza effettiva, però, sarà specificata dal contenuto delle celle ad indirizzo 0 ed 1 della memoria, contenenti rispettivamente il numero di colonne (N\_COL) e di righe (N\_RIG) dell’immagine, entrambi di dimensione 8 bit.

Nei byte successivi, dall’indirizzo 2 all’indirizzo N\_COL\*N\_RIG+2, sarà memorizzata sequenzialmente e in modo contigui, l’immagine pixel per pixel su cui operare la trasformazione richiesta.

Infine, dopo opportuni passaggi, a partire dall’indirizzo N\_COL\*N\_RIG+3 all’indirizzo 2\*N\_COL\*N\_RIG+2 verrà salvata l’immagine ottenuta dal processo di equalizzazione richiesto.

1.3 esempio di funzionamento

2 Design

Si è scelto di descrivere tramite linguaggio vhdl un modulo single-process tramite architettura behavioural (comportamentale/algoritmica). Questo è stato reso possibile tramite la progettazione di una macchina a stati finiti, di seguito descritta.

2.1 Macchina a stati e descrizione algoritmica

L’FSM definita è composta da 12 stati suddivisi in 3 sottogruppi.

2.1.1 stati ausiliari

Gruppo di stati di inizio, fine e attesa nel processo.

1. WAIT\_MEM: stato di attesa che permette ai valori di essere correttamente letti dalla memoria e resi disponibili sul segnale i\_data
2. WAIT\_RESET: stato iniziale di attesa del segnale i\_rst, che può essere anche asincrono
3. WAIT\_START: stato di attesa del segnale di i\_start. In qualsiasi momento dell’elaborazione, se il segnale i\_rst è rilevato alto, anche in modo asincrono, il componente viene riportato in questo stato, in attesa di una nuova elaborazione
4. DONE: stato in cui o\_done viene posto ad ‘1’, simboleggiando la fine dell’elaborazione. Si aspetta quindi un valore di start basso per tornare in WAIT\_START

2.1.1 calcolo dimensioni

Gruppo di stati che permette il calcolo della dimensione effettiva dell’immagine.

1. COL\_READ\_REQ: rilevato i\_start alto, viene raggiunto questo stato in cui viene abilitata la memoria per la lettura del byte 0, passando quindi all’attesa in WAIT\_MEM
2. COL\_READ\_ROW\_READ\_REQ: stato in cui N\_COL, letto su i\_data, viene salvato. Viene quindi abilitata la memoria e richiesta la lettura del byte 1, tornando in WAIT\_MEM
3. ROW\_READ: stato in cui anche N\_ROW, letto su i\_data, viene utilizzato per calcolare l’effettiva dimensione dell’immagine da manipolare, passando poi a SFM\_READ\_REQ

2.1.2 ricerca di valori di massimo e minimo dell’immagine

Gruppo di stati che svolge effettua un primo passaggio di tutti i pixel dell’immagine, stabilendo i valori minimo e massimo tra essi. Questo è necessario per l’effettiva elaborazione che sarà effettuata in seguito.

La lettura sequenziale è effettuata tramite un contatore che tiene traccia del pixel corrente.

1. SFM\_READ\_REQ – scan for max read request: stato in cui viene abilitata la memoria per la lettura di un pixel dell’immagine, passando poi in WAIT\_MEM
2. SFM\_READ – scan for max read: stato in cui il valore del pixel corrente è letto da i\_data ed utilizzato per aggiornare opportunamente i valori di massimo e minimo fin a questo punto rilevati nei precedenti pixel. Se dopo questa iterazione è stata scandagliata l’intera immagine, allora si può azzerare count e passare a PREP\_ELAB\_VALUEST. Altrimenti, count è incrementato e si torna a SFM\_READ\_REQ

2.1.3 elaborazione dell’immagine

Gruppo di stati che effettua l’effettiva elaborazione e la scrittura della nuova immagine in memoria tramite un secondo ciclo su di essa.

1. PREP\_ELAB\_VALUEST: stato in cui vengono calcolati, tramite i dati ottenuti negli stati precedenti, il delta value e lo shift level, necessari per l’elaborazione. Si passa quindi per la prima volta a SFE\_REQ, iniziando il secondo ed ultimo ciclo sui pixel dell’immagine originaria, al fine di elaborare il risultato finale
2. SFM\_READ\_REQ – scan for elaboration request: stato in cui viene abilitata memoria per la lettura del pixel corrente, specificato ancora una volta dal contatore count. Si passa quindi a WAIT\_MEM
3. SFE\_READ – scan for elaboration read: stato di effettiva elaborazione del pixel corrente dell’immagine da modificare. Infatti, finche il valore di count non raggiunge la dimensione dell’immagine, il valore del pixel corrente viene utilizzato per calcolarne il valore aggiornato, che verrà predisposto per la scrittura all’indirizzo corretto, previa abilitazione in scrittura della memoria, per poi tornare ad SFM\_READ\_REQ con count incrementato. Se invece, count ha raggiunto la dimensione dell’immagine, si passa ad DONE

2.2 approfondimento sull’elaborazione

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)

\*spiego

2.3 scelte progettuali

Rising edge, forse spostare la roba del behavioural