1 Requisiti di progetto

1.1 descrizione del problema

Si vuole realizzare un componente in grado di svolgere una versione semplificata del processo di equalizzazione dell’istogramma di un’immagine, ossia di ricalibrare il contrasto di quest’ultima, effettuando una ridistribuzione dei valori di intensità pixel per pixel.

Le immagini che da manipolare saranno in scala di grigi a 256 livelli e avranno una dimensione massima di 128x128 pixel.

1.2 interfaccia del componente

Il componente da realizzare ha un’interfaccia così definita:

\*code snippet e immagine tikz

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.

1.3 descrizione della memoria e dell’interazione con il componente

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

In particolare, l’algoritmo di equalizzazione sarà applicato ad immagini ad immagini pre-salvate in memoria la cui grandezza effettiva (in pixel), sarà specificata dal contenuto delle celle ad indirizzo 0 ed 1 della RAM, 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 salvata pixel per pixel, sequenzialmente e in modo contiguo, l’immagine 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 scritta l’immagine ottenuta dal processo di equalizzazione richiesto.

1.3 esempio di funzionamento

2 Design

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

2.1 Macchina a stati

L’FSM definita è composta da 10 stati, suddivisibili 3 sottogruppi che realizzano altrettante fasi principali del processo.

2.1.1 stati ausiliari

Gruppo di stati che realizzano: inizio, richiesta di lettura, attesa della memoria e fine del processo.

1. WT\_RST – wait reset: stato iniziale di attesa del segnale i\_rst, che può essere anche asincrono
2. WT\_STR – wait start: stato di attesa del segnale di i\_start. In qualsiasi momento dell’elaborazione (a partire quindi da qualsiasi stato successivo a questo), se il segnale i\_rst è rilevato alto, anche in modo asincrono, il componente viene riportato a WT\_STR, in attesa di un nuovo i\_start.

NB: In caso di reset abbiamo si è presupposto che anche il segnale dei start venga riportato basso per il periodo in cui il i\_rst è alto

Quando i\_start è rilevato alto, vengono inizializzati tutti i valori necessari al processo

1. RD\_REQ – read request: stato di abilitazione della memoria in lettura. Viene preparato su o\_address l’indirizzo della RAM che deve essere letto, identificato dalla variabile count, gestita dagli altri stati della FSM
2. WT\_MEM – wait memory: stato di attesa che permette ai valori di essere correttamente letti dalla memoria e resi disponibili sul segnale i\_data al ciclo di clock successivo. Si occupa inoltre della corretta decision dello stato successivo tramite il valore della variabile count, che aggiorna anche ad ogni sua iterazione, e della variabile shift\_value, utilizzata per stabilire se si è già nel gruppo di stati elaborazione dati
3. 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 WT\_STR

2.1.1 calcolo dimensioni

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

1. RD\_COL – read column: stato in cui il valore N\_COL relativo all’immagine, pronto su i\_data, è salvato su una variabile temporanea
2. RD\_ROW – read row: stato in cui N\_ROW, pronto su i\_data, viene utilizzato insieme al valore N\_COL salvato precedentemente per calcolare la dimensione effettiva dell’immagine e successivamente determinare se è adatta (>0) oppure no per il proseguimento dell’esecuzione

2.1.2 ricerca di valori di massimo e minimo dell’immagine

“Gruppo” di stati scandaglia tutti i pixel dell’immagine, grazie ad un ciclo tra gli stati RD\_REQ – WT\_MEM e CMP\_DT, 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 la variabile count, aggiornata in WT\_MEM.

1. CMP\_DT – compare data: stato in cui il pixel dell’immagine all’indirizzo count-1 (WT\_MEM ha già incrementato la variabile) è pronto per l’utilizzo sul segnare i\_data. Il suo valore viene utilizzato per aggiornare i valori di massimo e minimo dell’immagine.   
   Si procede quindi a passare al prossimo pixel o alla effettiva fase di elaborazione dell’immagine. Nel secondo caso, il valore di count viene riportato a 2, ossia all’inizio dell’immagine originale

2.1.3 elaborazione dell’immagine

Gruppo di stati che effettua l’effettiva elaborazione, pixel per pixel, dell’immagine da trasformare, grazie ad un ciclo tra gli stati RD\_REQ – WT\_MEM – EL\_DATA e ai valori calcolati in PREP\_EL.

1. PREP\_EL – prepare elaboration: stato in cui vengono calcolati, tramite i dati ottenuti negli stati precedenti, il delta value e lo shift level, necessari per l’elaborazione dell’immagine. Si passa quindi a un nuovo ciclo sull’immagine sorgente
2. EL\_DATA – elaborate data: stato in cui
   1. Si abilita in scrittura la memoria, ponendo in o\_address il valore di count – 1 + dimesione immagine
   2. il valore del pixel (corrispondente all’indirizzo count-1, perché già incrementato in WT\_MEM) dell’immagine originale, disponibile su i\_data, è utilizzato per calcolare il valore del rispettivo pixel nell’immagine trasformata. Questo viene quindi posto in o\_data per essere scritto in memoria
   3. se la fine dell’immagine è stata raggiunta, si passa a DONE, altrimenti si continua a scandagliare l’immagine da modificare

\*immagine FSM

2.2 approfondimento sull’elaborazione

La manipolazione del contrasto dell’immagine è fondata su 4 istruzioni fondamentali. Le prime due vengono svolte a priori del secondo ciclo, nello stato PREP\_EL, mentre le seconde due sono svolte successivamente alla seconda visita di ogni cella di memoria contenente l’immagine originale.

* DELTA\_VALUE = MAX\_PIXEL\_VALUE – MIN\_PIXEL\_VALUE   
  Il delta\_value rappresenta il valore della differenza tra il pixel più chiaro (valore maggiore) e il più scuro (valore minore) dell’immagine.
* SHIFT\_LEVEL = (8 – FLOOR(LOG2(DELTA\_VALUE +1)))

Lo shift\_value è il valore per cui sarà moltiplicata la differenza tra il pixel corrente e il pixel di valore minore dell’immagine. La funzione FLOOR(X) non è altro che un arrotondamento per difetto del valore X fornitogli come argomento.

* TEMP\_PIXEL = (CURRENT\_PIXEL\_VALUE - MIN\_PIXEL\_ VALUE) << SHIFT\_LEVEL

temp\_pixel contiene il possibile valore da attribuire al pixel corrisponente a CURRENT\_PIXEL\_VALUE nell’immagine finale. Non è però sempre possibile utilizzare direttamente questo valore, dato che potrebbe superare il limite massimo di 255 imposto dalla codifica in scala di grigi a 256 livelli.

* NEW\_PIXEL\_VALUE = MIN( 255 , TEMP\_PIXEL)

NEW\_PIXEL\_VALUE contiene il minimo valore tra 255 e il valore TEMP\_PIXEL precedentemente calcolato. Rappresenta l’effettivo valore che verrà scritto in memoria per il pixel considerato in questa iterazione.

2.3 scelte progettuali

Si è scelto di realizzare un componente sensibile al clock su rising\_edge.

L’algoritmo scelto si focalizza sulla facilità di lettura e comprensione, tralasciando possibili ottimizzazioni. Si è scelto però di cercare di ridurre al minimo il numero di operazioni di moltiplicazione nel codice, ad esempio separando gli stati RD\_COL e RD\_ROW, e di evitare totalmente l’operatore esponenziale, preparando il valore SHIFT\_VALUE tramite delle semplice condizioni IF/ELSE, in quanto portano a un incremento molto rapido delle risorse hardware necessarie al crescere della dimensione degli operandi, imponendo al calcolatore di essere svolte in un ciclo di clock.

2.4 algoritmo base?

3 Testing e Risultati sperimentali

3.1 casi di test

Il componente realizzato è stato testato tramite diversi TestBench. In particolare, ci si è concentrati su diversi casi critici possibili e sul corretto calcolo di tutti i valori utilizzati:

* Il corretto utilizzo di tutti i possibili SHIFT\_VALUE
* La condizione particolare in cui N\_COL = N\_ROW = 0
* Il caso limite di grandezza 128x128, non documentato nel nostro TestBench, in quanto non diverso dal funzionamento di un test 2x2
* Il caso in cui il segnale i\_rst venga portato alto. Il componente si riporta correttamente nello stato WT\_STR, anche in modo asincrono
* Il corretto rapporto tra i segnali i\_rst, i\_start e i\_done

Nella verifica degli ultimi due punti, è risultata utile l'analisi grafica dei segnali d'uscita del modulo, di cui si riporta immagine.

\*immagine

Si è usufruito di TestBench, di dimensioni e caratteristiche variabili dalla singola alle 10000 immagini (TB10K), utilizzandone di redatti manualmente (da colleghi e da noi) ma anche di auto-generati.

4 conclusioni

Per tutti i casi di test e TestBanch utilizzati, sono state svolte con successo le seguenti simulazioni, di cui si riporta come riferimento i risultati del TB 10K:

* simulazione behavioral: \*ris
* simulazione functional post-synthesis: \*ris
* sumilazione timing post-synthesis: \*ris

il report di sintesi ha evidenziato l’utilizzo nell’area di sintesi del modulo dei seguenti componenti:

* LUT:
* FF:
* LATCH:

Si ritiene di aver progettato un’architettura che rispecchi le specifiche di progetto assegnateci e di aver meglio compreso il comportamento e la progettazione di un componente dalle caratteristiche simili a quello da noi realizzato.