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 di cui è richiesta la manipolazione saranno in scala di grigi a 256 livelli e avranno una dimensione massima di 128x128 pixel.

1.2 interfaccia del componente

Il componente realizzato 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 per comunicare quale operazione si vuole svolgere su di essa. Può assumere valori 0 e 1, rispettivamente per lettura e scrittura
* 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 pre-salvate in memoria, la cui grandezza effettiva (in pixel) sarà specificata dal prodotto del contenuto tra le celle ad indirizzo 0 ed 1 della RAM, contenenti rispettivamente il numero di colonne n\_col e di righe n\_row dell’immagine, entrambi di dimensione 8 bit.

Nei byte successivi, dall’indirizzo 2 all’indirizzo n\_col\*n\_row+2, sarà contenuta, 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\_row+3 all’indirizzo 2\* n\_col\*n\_row+2, verrà scritta l’immagine ottenuta dal processo di equalizzazione richiesto, seguendo la stessa formattazione in memoria.

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 da noi prodotta 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 non in corrispondenza di i\_clk, il componente viene riportato a in questo stato, nuovamente in attesa di un nuovo segnale di inizio elaborazione.

NB: In caso di reset si è supposto che il segnale i\_start venga riportato basso per il periodo in cui il i\_rst è alto.

Quando si rileva i\_start = 1, vengono inizializzati tutti i valori necessari al processo, prima di passare al prossimo stato.

Di particolare importanza per l’algoritmo da noi sviluppato è il segnale count (inizializzato a 0), che indica quale sarà l’indirizzo alla quale sarà effettuata l’operazione di read alla prossima richiesta di lettura del componente.

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
2. WT\_MEM – wait memory: stato di attesa che permette ai valori di essere correttamente caricati dalla memoria sul segnale i\_data al ciclo di clock successivo.

È uno stato decisivo per l’FSM in quanto costantemente rivisitato nei cicli di lettura dei pixel dell’immagine, e responsabile del corretto instradamento del processo, grazie a condizioni su count e shift\_value. Si occupa inoltre di aggiornare la variabile count, e quindi della corretta gestione del successivo dato letto in memoria

~~Si occupa inoltre della corretta decisione 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~~

1. 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 e poter ricominciare il processo

2.1.1 calcolo dimensioni

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

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 per essere utilizzato in seguito
2. RD\_ROW – read row: stato in cui n\_row, pronto su i\_data, viene moltiplicato con il valore n\_col salvato precedentemente per calcolare la dimensione effettiva dell’immagine e, successivamente, determinare se essa è adatta - n\_col\*n\_row>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 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.~~

“Gruppo” di stati che permette di stabilire i valori minimo e massimo tra i pixel dell’immagine sorgente, grazie ad un ciclo tra RD\_REQ – WT\_MEM e CMP\_DT. Questi, sono necessari per l’effettiva equalizzazione dell’immagine, svolta dal gruppo di stati successivo.

La lettura sequenziale è mediata dal corretto aggiornamento della variabile count in WT\_MEM.

1. CMP\_DT – compare data: stato in cui è pronto sul segnare i\_data il valore del pixel dell’immagine all’indirizzo count-1 (WT\_MEM ha già incrementato count). Quest’ultimo viene confrontato con i valori minimo e massimo stabiliti fin ora nel processo, che vengono aggiornati se necessario.  
   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’indirizzo del primo pixel 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 ad un nuovo ciclo sull’immagine sorgente
2. EL\_DATA – elaborate data: stato in cui si svolge l’elaborazione del pixel count-1 dell’immagine. In particolare:
   1. Si abilita in scrittura la memoria, ponendo in o\_address il valore dell’indirizzo destinazione per la scrittura: (count-1)+n\_col\*n\_row
   2. Il valore del pixel 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 – count <= n\_col\*n\_row+2 - 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 nello stato PREP\_EL, in preparazione del ciclo di elaborazione, mentre le seconde due sono svolte per ogni cella di memoria contenente l’immagine originale nello stato EL\_DATA.

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

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.

\*immagine esempio

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

temp\_pixel contiene il possibile valore da attribuire al pixel corrispondente a CURRENT\_PIXEL\_VALUE nell’immagine finale. Non è però sempre possibile utilizzare direttamente, in quanto 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 valore minimo 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 prodotto si focalizza sulla facilità di lettura e comprensione. 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 semplici condizioni IF/ELSE, rispettando le regole nella seguente tabella:

\*tabella regole shift\_value

Questo accorgimento è stato apportato in quanto gli operatori di moltiplicazione ed elevamento a potenza portano, imponendo al calcolatore di essere svolti in un ciclo di clock, ad un incremento notevole delle risorse hardware, proporzionale alla grandezza degli operandi, necessarie per il loro completamento.

2.4 algoritmo base? Direi che non serve.

3 Testing e Risultati sperimentali

3.1 casi di test

Il corretto comportamento del componente è stato verificato tramite dei TestBench. In particolare, ci si è concentrati su diversi casi critici possibili durante l’ e sul corretto calcolo di tutti i valori utilizzati. Di seguito una breve lista dei test e condizioni più significativi

* Corretto utilizzo di tutti i possibili SHIFT\_VALUE
* Condizione particolare: N\_COL = N\_ROW = 0
* Caso limite di grandezza 128x128 (non documentato nel nostro TestBench, in quanto non più rilevante del funzionamento di un test 2x2)
* Caso di reset dell’elaborazione, anche in modo asincrono
* Caso di reset dell’elaborazione seguito da cambi di immagine in memoria
* Corretto rapporto tra i segnali i\_rst, i\_start e i\_done durante l’esecuzione

Nella verifica degli ultimi tre 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 caratteristiche e dimensioni variabili dalla singola alle 10000 immagini (TB10K), utilizzandone di redatti manualmente (da colleghi e da noi) ma anche di auto-generati tramite uno script python pubblico.

4 conclusioni

Per tutti i casi di test e TestBanch utilizzati, sono state svolte con successo le simulazioni richieste dalle specifiche di progetto, di cui si riporta come riferimento i risultati legato al TB10K:

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

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

* LUT:
* FF:
* LATCH:

Si è inoltre appurato che il componente progettato supera inoltre le seguenti simulazioni non richieste:

* Simulazione post-implementation functional:
* Simulazione post-implementation timing:

Si ritiene quindi realizzando un’architettura che rispecchi a pieno le specifiche di progetto assegnateci e di aver meglio compreso il comportamento e il processo di progettazione di un componente dalle caratteristiche simili a quello da noi proposto.