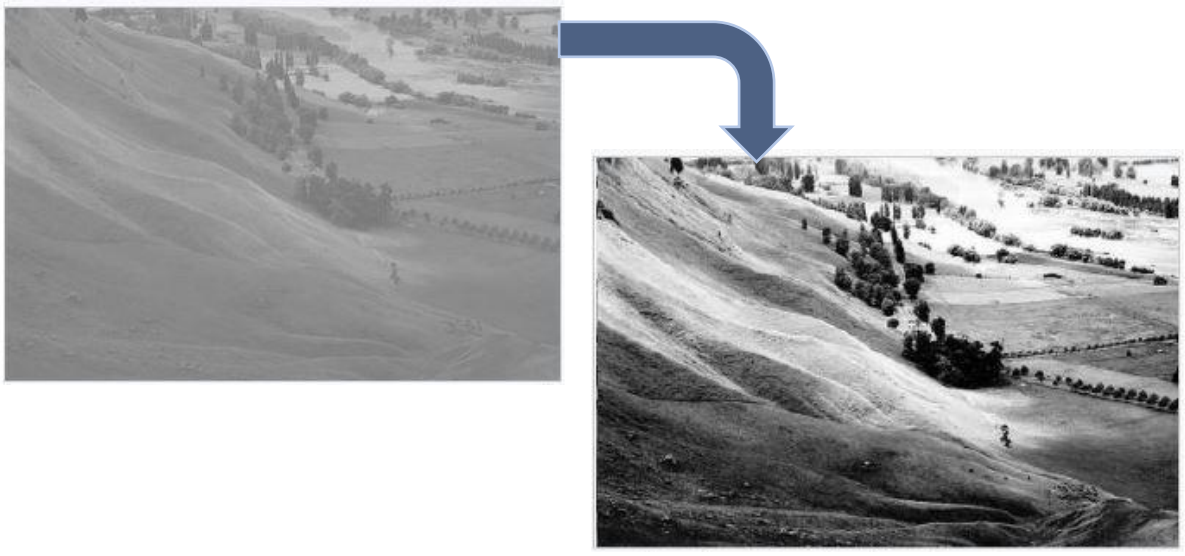
Relazione della Prova finale di Reti logiche

Alfredo Landi 10586178

Politecnico di Milano



1. Introduzione

Obiettivo di questa prova finale è realizzare un circuito, in linguaggio VHDL, che implementi una versione semplificata dell'algoritmo di equalizzazione delle immagini.

All'indirizzo 0 della memoria si trova il numero N-Col di colonne di pixel dell'immagine, mentre all'indirizzo 1 il numero N-Rows di righe. Il prodotto N-Col*N-Rows rappresenta il numero totale di pixel da cui è composta l'immagine. Il primo di questi pixel nella memoria RAM può essere trovato all'indirizzo 2, mentre l'ultimo all'indirizzo N-Col*N-Rows+1. A partire dall'indirizzo N-Col*N-Rows +2 si troverà il corrispondente del primo pixel equalizzato, a N-Col*N-Rows + 3 il secondo e così via fino ad arrivare all'ultimo pixel che è stato equalizzato. L'algoritmo scansiona, una prima volta, tutti i pixel dell'immagine per trovare quello con valore più alto e quello con valore più basso ed in seguito sottrae al massimo il minimo: tale differenza viene chiamata "delta". In seguito, si somma al "delta" un 1 e viene eseguito il logaritmo, in base 2, di questo valore arrotondando sempre per difetto il risultato. Queste operazioni consentono di poter implementare il logaritmo in maniera semplice e veloce tramite i controlli a soglia (Sarebbe infatti abbastanza oneroso per l'hardware fare il logaritmo con una libreria matematica di cui dispone il VHDL). L'aggiungere al "delta", infatti, sempre un 1 evita di avere uno 0 di cui non si può calcolare il logaritmo; arrotondando sempre per difetto il risultato, inoltre, si può semplicemente vedere in che posizione si trova il primo 1, leggendo in codifica binaria, da sinistra verso destra, il numero di cui si vuole fare il logaritmo: quella posizione sarà il risultato del logaritmo. A questo punto si sottrae ad 8 il valore del logaritmo arrotondato e si ottiene un valore che ci servirà per fare uno "shift" a sinistra. Ora si deve nuovamente leggere la RAM dal valore 2 a N-Col*N-Rows + 1 e per ogni pixel della nostra immagine si fa una sottrazione con il minimo, il cui risultato verrà "shiftato" a sinistra con il valore di shift calcolato precedentemente. Si deve, infine, scrivere in memoria il nuovo valore "shiftato" se questo è minore di 255, mentre 255 se questo valore è maggiore o uguale allo stesso 255 (Non si possono, infatti, scrivere valori maggiori sulla RAM dal momento che essa dispone di massimo 8 bit per rappresentare ciascun risultato dell'equalizzazione).

Una volta che tutti i pixel sono stati letti ed equalizzati si deve alzare il segnale d'uscita "o_done" ad 1 ed aspettare che anche il segnale d'inizio "i_start" si abbassi a 0, con conseguente abbassamento anche di "o_done", per poter equalizzare la prossima immagine di dimensione uguale o diversa (Il circuito deve infatti consentire equalizzazioni multiple).

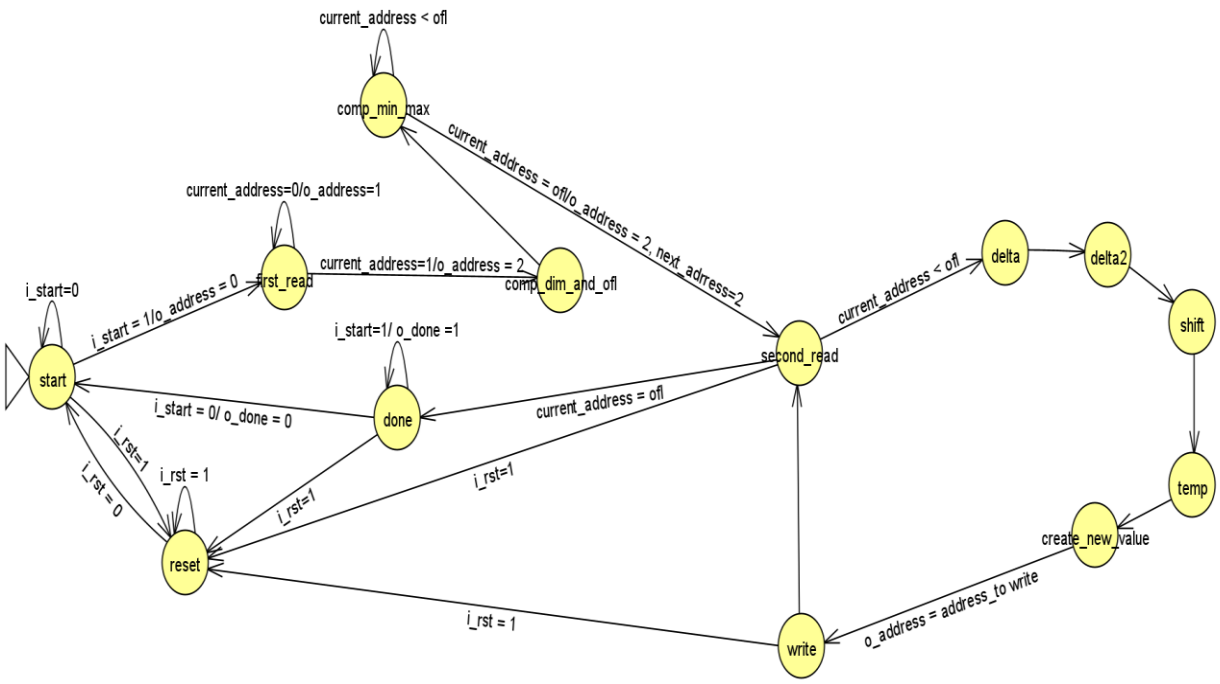
Di seguito si riporta un semplice esempio per una maggiore comprensione complessiva:

Indirizzo RAM	Valore	Note	Delta	Shift value	Min	New value
0	2	N-Col				
1	2	N-Rows				
2	46	Primo pixel	85	2	46	(46-46)*2^2 = 0
3	131		85	2	46	(131-46)*2^2 = 340 > 255
4	62		85	2	46	(62-46)*2^2 = 64
5	89	Ultimo pixel	85	2	46	(89-46)*2^2 = 172
6	0	Nuovo primo pixel				
7	255					
8	64					
9	172	Nuovo ultimo pixel				

2. Architettura

Si è optato per descrivere il componente in VHDL usando un'architettura completamente "Behavioral" e a singolo processo. Tale processo è "sensibile" a variazioni del clock e del segnale di reset. Ad ogni variazione del clock il processo può eseguire una commutazione di stato, se si verificano opportune condizioni, e ogni qualvolta il reset va ad 1 il processo si porterà nello stato di "reset" e, quando "i_rst" si abbasserà, potrà ricominciare dallo stato iniziale.

Per maggiore chiarezza si riporta alla pagina successiva lo schema della macchina a stati ed una descrizione sintetica di ogni stato.



*Nota bene: da tutti gli stati si può andare nello stato reset, non solo da start, done, write e second_read, non ci sono gli archi solo per non appesantire troppo la notazione.

Lista degli stati:

1. Reset: è lo stato nel quale si va se viene alzato il segnale di reset, quando abbassato si ripartirà dallo stato start;
2. Start: E' lo stato che attende il segnale i_start per dare inizio alla computazione vera e propria;
3. First_read: legge gli indirizzi 0 e 1 per determinare il numero di colonne e il numero di righe;
4. Comp_dim_and_ofl: calcola l'ofl, ovvero da quale indirizzo nella RAM si dovrà iniziare a scrivere;
5. Comp_min_max: trova il pixel più grande di tutti e quello più piccolo;
6. Second_read: legge un pixel alla volta per poi avviarlo agli stati che lo equalizzeranno;
7. Delta: fa la differenza tra max e min;
8. Delta2: supporta delta nella memorizzazione del valore della differenza tra max e min;
9. Shift: calcola di quanto dovrà essere shiftato il pixel corrente e fa la differenza tra il pixel corrente e il più piccolo di quelli forniti;
10. Temp: "shifta" a sinistra dello "shift_value" il pixel corrente;
11. Create_new_value: compara il valore ottenuto dallo "shift" con 255 e decide cosa scrivere in memoria, incrementato anche l'indirizzo di scrittura;
12. Write: stato di supporto alla scrittura;
13. Done: aspetta che i_start venga abbassato a 0 per preparare la macchina alla prossima immagine.

3. Risultati sperimentali

Senz'altro potevano essere fatte ulteriori ottimizzazioni, come portare gli stati inerenti al calcolo del delta e dello "shift_value" dopo quello del calcolo del min e max per non ripetere queste operazioni per ogni pixel letto, ma dopo un'attenta analisi del report di sintesi (si veda la prossima immagine per maggiore chiarezza) è stato notato che anche con questo e altri accorgimenti non variava né il numero di LUT sintetizzate né quello di Flip-Flop. Inoltre, anche con 1000 immagini 128x128 (massima dimensione consentita) la post-sintesi impiega meno di 7 secondi.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	240	0	134600	0.18
LUT as Logic	240	0	134600	0.18
LUT as Memory	0	0	46200	0.00
Slice Registers	244	0	269200	0.09
Register as Flip Flop	244	0	269200	0.09
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Si segnala, inoltre, che il circuito riesce a funzionare con un periodo di clock di almeno 100 ns e la FPGA usata è la xc7a200tfbg484-1 della famiglia delle Artix-7, come da specifica.

4. Simulazioni

Oltre ai numerosi test bench forniti su Beep, che già coprivano un gran numero di casi, si è usato un generatore in python per stressare il funzionamento del segnale di reset e/o la robustezza del circuito a numerosi input di grandi dimensioni.

Seguendo, inoltre, i principi del testing ingegneristico sono stati evidenziati i seguenti casi limite che non avevano copertura certa e per i quali sono stati scritti dei test bench "ad hoc" in VHDL. Di seguito viene riportata una lista:

1. Pixel tutti decrescenti a partire da 255 per stressare il calcolo del minimo;
2. Pixel tutti crescenti a partire da 0 per stressare il calcolo del massimo;
3. N-Col = 0 e N-Rows diverso da 0 con 4 pixel a partite dall'indirizzo 2 non validi (dimensione=0x2=0);
4. N-Col=N-Rows=0 con 4 pixel a partire dall'indirizzo 2 non validi (dimensione = 0x0=0).

Per maggiori informazioni su questi test bench si rimanda ad un link dove è possibile scaricarli: https://mega.nz/folder/nRph2I4Z#iUPNkYbA_xWXw52Uz-2uqQ

5. Conclusione

A differenza delle prove finali di ingegneria del software e di algoritmi e strutture dati, questa prova è l'unica che richiede lo sviluppo di un componente circuitale mediante un linguaggio di descrizione del hardware. Sebbene le scelte progettuali abbiano portato alla descrizione del componente in Behavioral, la più vicina, tra le varie possibili, ad una descrizione algoritmica del problema e ad un linguaggio di programmazione come può essere il C o Java, sussistono, tuttavia, forti differenze con tali linguaggi. Il progetto, infatti, nelle varie fasi di realizzazione, porta a focalizzarsi sulle differenze con un classico programma software quali possono essere la diversa gestione dei segnali basati su bit rispetto ai tipi di dati, sia rispetto a quelli forniti dalla programmazione strutturata classica che a quelli forniti dalla OOP, o sull'evoluzione nel tempo dei segnali hardware all'interno di un circuito (aspetto totalmente assente nei linguaggi di programmazione per il software). Bisogna, tuttavia, puntualizzare, che i linguaggi per la descrizione del hardware sono sì differenti da quelli per il software, ma lo sviluppo di questo componente per le equalizzazioni delle immagini ha seguito, nelle sue fasi, procedure ingegneristiche affini a quello dello sviluppo del software, quali:

1. Progettazione con l'ausilio di modelli teorici (FSA vs studio della complessità/UML);
2. Implementazione (VHDL vs C/Java);
3. Debugging e testing (Test benches vs verificatori automatizzati/ JUnit).