Relazione di Progetto - Reti Logiche

Elia Maggioni
Ingegneria Informatica, Scuola 3I, Polimi
C.P. 10610008

Marco Fasanella
Ingegneria Informatica, Scuola 3I, Polimi
C.P. 10617541

Questa relazione descrive il codice VHDL sintetizzato per l'equalizzazione di un'immagine in bianco e nero su piattaforma FPGA. L'implementazione si basa sulla espansione della gamma cromatica utilizzata al fine di sfruttare tutta quella rappresentabile. Lo sviluppo ha incluso la definizione di test bench per testare la correttezza e l'affidabilità del risultato prodotto.

1. Introduzione

Il circuito descritto si occupa di leggere e rielaborare i dati da RAM producendo un'immagine con nitidezza più alta e quindi più leggibile.

Quando il segnale start è fornito al circuito, viene attivata la prima macchina a stati ed in base al bias tra la saturazione dei pixel e l'offset, ovvero il valore minimo di saturazione, viene restituita nei valori di memoria immediatamente successivi un'immagine con pixel che meglio ricoprono i valori rappresentabili.

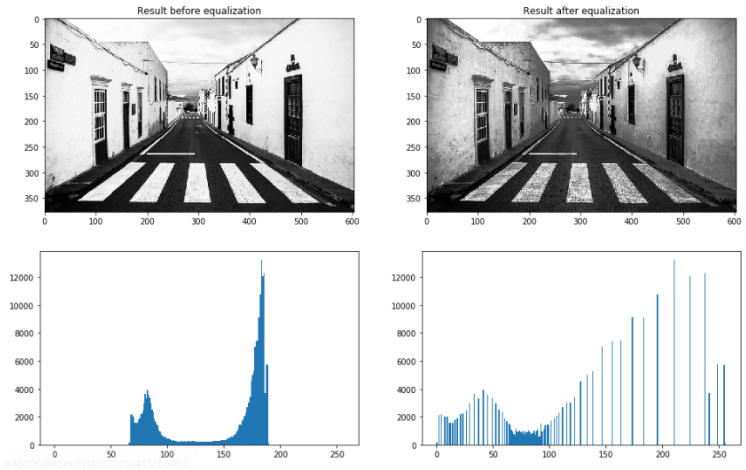


Figure 1. Esempio di Equalizzazione (con algoritmo completo)

Quando l'elaborazione è terminata, viene restituito un segnale di done ed il circuito è pronto per una nuova immagine a partire dall'indirizzo 0. L'algoritmo usato nel progetto è solo una semplificazione di quello che realmente sarebbe possibile con questa tecnica.

Questo per sottolineare il potenziale dell'equalizzazione dell'istogramma e di come riesca a migliorare in modo veramente apprezzabile la qualità di una immagine digitale.

Nel circuito è stato scelto di dividere in 3 la computazione con moduli in cascata.

1.1 Specifica

```
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;
```

Figure 2. Interfaccia del Componente)

I segnali da considerare sono i seguenti:

- 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;
- i_start è il segnale di START generato da TestBench;
- i_data è il segnale (vettore) che arriva dalla memoria;
- 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;
- o_we è il segnale di WRITE ENABLE;
- o_data è il segnale (vettore) di uscita dal componente verso la memoria;

Mentre i valori da calcolare durante l'elaborazione saranno:

- 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)

Ogni byte corrisponde ad un pixel dell'immagine, che sarà quindi formata in modo sequenziale riga per riga. Bisognerà quindi leggere i valori e rielaborarli, per poi scriverli in memoria (a partire dall'indirizzo 2 e in byte contigui).

1.2 Esempio

Di seguito è riportato un esempio per esplicare meglio il funzionamento dell'algoritmo su valori di memoria realistici.

52	55	61	59	79	61	76	61
62	59	55	104	94	85	59	71
63	65	66	113	144	104	63	72
64	70	70	126	154	109	71	69
67	73	68	106	122	88	68	68
68	79	60	70	77	66	58	75
69	85	64	58	55	61	65	83
70	87	69	68	65	73	78	90

Figure 3. Esempio di Pixel da Equalizzare

L'equalizzazione incrementa il contrasto globale delle immagini, specialmente quando i pixel di cui è composta sono rappresentati da valori di intensità molto vicini. Le intensità vengono perciò distribuite sull'istogramma, permettendo alle aree a basso contrasto locale di ottenere un alto contrasto.

0	12	53	32	190	53	174	53
57	32	12	227	219	202	32	154
65	85	93	239	251	227	65	158
73	146	146	247	255	235	154	130
97	166	117	231	243	210	117	117
117	190	36	146	178	93	20	170
130	202	73	20	12	53	85	194
146	206	130	117	85	166	182	215

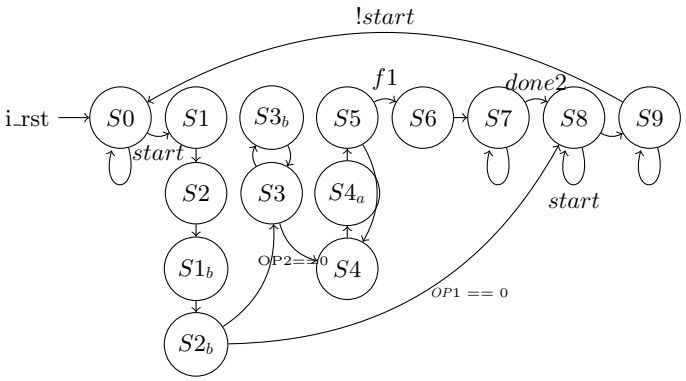
Figure 4. Esempio di Pixel Equalizzati

2. Architettura

La scelta architetturale ricade sull'utilizzo di 3 moduli, descritti ognuno da una propria FSM, attraverso segnali in cascata per l'avvio e la terminazione delle varie parti. Si avranno quindi i segnali *start2, start3, done2,* e *done3* che avranno come scopo la comunicazione fra i vari moduli e i loro processi.

1. **Modulo 1**

   Il primo modulo si occupa dell'inizio della computazione, della lettura e ricerca di massimo, minimo, e delta. Inoltre attende la discesa del segnale start in modo da abbassare done e mettersi in attesa di una nuova immagine. La macchina a stati di riferimento (FSM1) è composta dai seguenti stati:



   - F1S0 (*Ready*)

     Stato iniziale che attende il segnale di start. Si entra in questo stato per azzerare i registri e prepararsi a processare l'immagine.

   - F1S1,F1S2,F1S1b,F1S2b (*Operands*)

     Questi stati si occupano della lettura di *OP1* e *OP2* (numero righe e colonne) attraverso il processo *MULT*. In *F1S2b* un controllo si accerterà che *OP1* non sia nullo, altrimenti si passerà allo stato di terminazione e reset dei valori per una nuova lettura.

   - F1S3,F1S3b (*Multiply*)

     A questo punto viene calcolato M (cioè $OP1 * OP2$) iterando in questi due stati attraverso il segnale *o_m*. Il calcolo avviene per somme successive fino a quando *OP2* sarà zero.

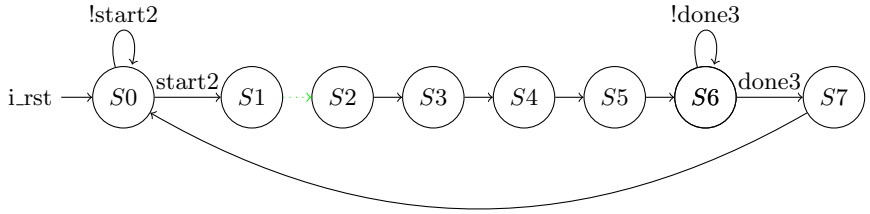
   - F1S4,F1S4a,F1S5 (*Pixels*)

     Attraverso i processi *ADDRHandler* e *MINeMAX*, verrà incrementato *REGAddr* (registro dell'indirizzo di memoria) e controllato Pixel (registro che salva ogni current pixel). Il segnale f1 detta il passaggio fra questi stati attraverso la condizione $REGAddr > M + 2$. Il calcolo del delta invece, avviene ad ogni nuovo assegnamento del Massimo o del Minimo.

- F1S6,F1S7 (*Finalize*)
  Una volta terminato il proprio compito, nello stato *F1S6* verrà alzato il segnale start2 per far partire il secondo modulo e successivamente, lo stato *F1S7* si metterà in attesa del segnale *done2* per terminare l'elaborazione e reset dei valori.
- F1S8,F1S9 (*Reset*)
  A questo punto, una volta ricevuto il segnale *done2*, lo stato *F1S8* alzerà done ed *endof* (segnale generale di imminente terminazione e reset dei valori), e aspetterà che si abbassi start, per poi passare a *F1S9* che farà riscendere i due precedenti segnali. Si torna quindi a *F1S0* in attesa di una nuova immagine.

2. **Modulo 2**

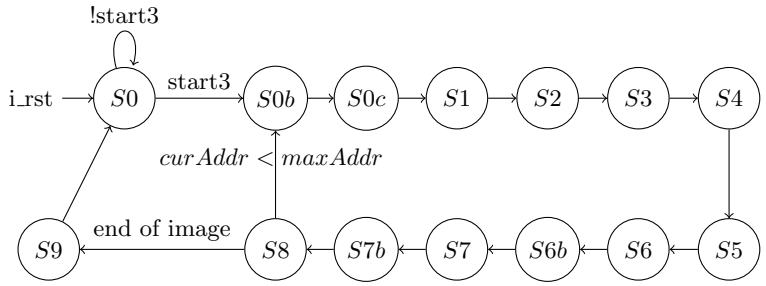
   Il modulo due svolge il calcolo dello shift level noto il data value e lo rende disponibile per l'utilizzo al modulo tre. Una volta completato il calcolo, alza il segnale *start3* per fare partire la FSM3, dopodichè si mette in attesa di *done3* per poi terminare alzando per un ciclo di clock il segnale *done2*. Di seguito la descrizione della FSM2:



- F2S0 *(Ready)*
  Attende l'inizio dell'esecuzione del programma che avviene tramite l'innalzamento del segnale start2.
- F2S1 *(Get_Delta)*
  Carica il valore di delta nel registro *S2R1*.
- F2S2,F2S3,F2S4 *(LUT)*
  Attende la propagazione del segnale nel datapath e carica il valore in uscita dalla LUT in *S2R2*.
- F2S5 *(Start3)*
  Avvia il modulo tre alzando il segnale *start3*.
- F2S6 *(WaitDone3)*
  Attende in questo stato finché *done3* non viene alzato dal modulo tre
- F2S7 *(Finalize)*
  Alza il done2 per segnalare al modulo due l'avvenuta terminazione dell'elaborazione.

3. **Modulo 3**

Il modulo tre è dove avviene la parte più delicata dell'elaborazione, in particolare a questa parte è delegata la lettura da memoria del valore del pixel corrente, il calcolo sequenziale per ogni pixel del nuovo valore e la riscrittura in memoria.



- F3S0 *(Ready)*

  Stato di attesa della macchina, passa a *F3S0b* quando start3 è portato a 1

- F3S0b, F3S0c *(Memory_Load)*

  In questi due stati viene prelevato da memoria il valore corrente del pixel ed assegnato al registro *F3R3* che simboleggia il *CurrPixel* value. La gestione della memoria viene effettuata tramite i flag *o_f3addr_read* e *o_f3addr_write* in un processo separato. Viene quindi assegnato il valore di ritorno *i_data*.

- F3S1 *(Get_Calculated_Values)*

  Avviene ora il prelievo dei segnali provenienti dai moduli uno e due e il caricamento degli stessi nei registri *F3R1*, *F3R2* e *F3R4*, rispettivamente *shift_level, min pixel value* e *max address*.

- F3S2, F3S3 *(Zeroing)*

  In questi stati avviene il calcolo di $CurrPixel - minV$ e viene propagato tramite il segnale *o_f3sub* che poi viene caricato in *F3R5*.

- F3S4 *(Shift)*

  Da *shift_level* e *F3R5* viene calcolato il valore di *o_f3shift*, vettore di 16 bit per evitare il possibile overflow.

- F3S5, F3S6 *(Overflow_Check)*

  Un mutex viene inserito per portare *shift_level* da 16 bit a 8, nel caso in cui sia minore di 255 gli 8 bit più significativi vengono scartati, altrimenti viene assegnato il valore massimo rappresentabile di 255.

- F3S6b, F3S7 *(Memory_Write)*

  Tramite il flag *o_f3addr_write* viene assegnato in *o_data* il valore finale del registro *F3R6* in uscita al mutex.

- F3S7b, F3S8 *(Current_Pixel_Increment)*

  Il valore di *F3R7* viene incrementato di uno, questo registro serve alla macchina per sapere il valore corrente di memoria. Viene usato nel

processo di gestione della RAM sia per la lettura che per la scrittura dei valori e viene inoltre usato per la condizione d'uscita del ciclo di stati *F1S0b* $\rightarrow$*F3S8* insieme a *max address*.

- F3S9 *(Finalize)*
  Stato finale della macchina, viene portato a 1 il segnale *done3*

3. Risultati sperimentali

Di seguito i risultati della sintesi, ottenuti attraverso la Simulazione in ambiente Vivado. Sotto viene evidenziato il modo in cui lavorano i vari segnali a cascata per passare tra i vari moduli.

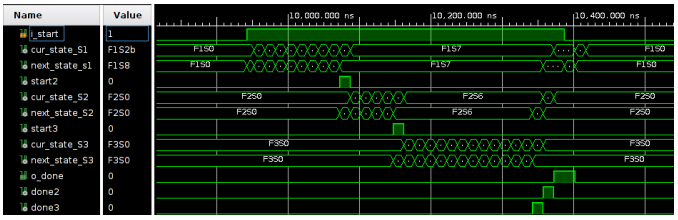


Figure 5. Waveform della simulazione

A seguire sono evidenziati i report di timing della sintesi e di utilizzo delle risorse della FPGA:

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	3,984 ns	Worst Hold Slack (WHS):	0,137 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	515	Total Number of Endpoints:	515

All user specified timing constraints are met.

Figure 6. Tempi della simulazione

Site Type	Used	Fixed	Available	Util %
Slice LUTs	180	0	134600	0.13
Slice Registers	207	0	269200	0.08
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Table 1. Utilization Design Information - Slice Logic

4. Simulazioni

Al fine di testare la correttezza del codice, abbiamo creato le seguenti simulazioni che sono state inserite in un solo test bench per testare anche l'elaborazione di più immagini in sequenza, sia con che senza un reset al termine di ogni immagine.

- TB Delta=255:

  input=[2,3,255,0,1,1,0,255] output=[255,0,1,1,0,255].

  Il delta sarà massimo nel momento in cui $MAXPixel = 255$ e $MINPixel = 0$. Il valore del logaritmo è quindi massimo e lo $shiftlevel = 0$ (caso minimo) che porterà quindi ad avere in output gli stessi valori di input.

- TB tutti 0 e tutti 255 $Delta = 0$:

  input=[2,3,0,0,0,0,0,0] output=[0,0,0,0,0,0]

  input=[2,3,255,255,255,255,255,255] output=[0,0,0,0,0,0]

  Il delta sarà nullo nel momento in cui tutti i valori dell'immagine saranno uguali. Sarà infatti MAXPixel=MINPixel, generando così uno $shiftlevel = 8$ (caso massimo) che porterà quindi ad avere in output solo valori nulli. Questo è anche intuitivamente immaginabile con una monocromia dell'immagine che non può quindi essere equalizzata in alcun modo.

- TB con Shift massimo:

  input=[2,3,128,128,128,128,128,128] output=[0,0,0,0,0,0]

  Lo Shift sinistro di 8 posizioni è un caso ottenibile solo in circostanze banali ovvero quando $CurrentPixel = MinPixel$ e viene dunque shiftato un vettore di valori nulli che quindi rimane invariato. In output avremo quindi il valore $CurrentPixel - MinPixel = 0$.

- TB con Massimo e Minimo agli estremi di lettura :

  input=[2,3,255,132,254,2,2,1] output=[255,255,255,2,2,0]

  Potrebbe capitare la lettura non completa dell'input. Caso molto comune nei due estremi RAM(3) e RAM($COL * RIG + 2$), a volte non compresi nella ricerca del minimo e del massimo.

- TB con lunghezza 0:

  input=[2,0,110,115,124,110,111,120] output invariato

  input=[0,3,110,115,124,110,111,120] output invariato

  Se l'immagine ha lunghezza o larghezza nulla, allora non verrà elaborata e il contenuto della RAM rimarrà invariato.

- TB con caso limite per LUT:

  input=[2,3,200,199,170,177,180,171] output=[255,2552,0,112,160,16]
  Delta=30 Shift=4
  input=[2,3,201,199,170,177,180,171] output=[248,232,0,56,80,8]
  Delta=31 Shift=3
  Bisogna considerare che $log(delta + 1)$ risulta caso limite nel momento in cui il valore di delta assume valore $2^n - 1$ (con $0 <= n <= 8$). La LUT dell'implementazione, tiene già conto dell'argomento del logaritmo, cambiando quindi il valore ogni volta che viene raggiunto dal delta una potenza di $2^n - 1$.

- In fase di testing abbiamo raggiunto tramite test successivi una copertura delle istruzioni $> 99\%$ ed una copertura delle decisioni $> 99\%$, escludendo da entrambe la LUT che è stata invece testata nei casi precedenti e successivi a tutti i cambiamenti del valore in uscita. E' stato testato inoltre l'invariante di rappresentazione pubblico della non modifica di celle di memoria non interessate dall'elaborazione ovvero maggiori di $2 * (COL * RIG) + 2$

5. Conclusioni

Il progetto è stato sviluppato in un team di due persone, fin da subito il nostro approccio è stato quello di applicare i metodi insegnataci nel corso di Ingegneria del Software per la compartimentazione del lavoro di scrittura del codice e la definizione di interfacce comuni, per questo il progetto è stato diviso in 3 componenti *stadio1DataPath, stadio2DataPath* e *stadio3DataPath* le quali rendevano disponibili dei segnali ad uso interno che poi venivano usati negli stadi successivi, senza necessariamente sapere come essi venissero generati. Oltre a questo abbiamo sviluppato un sistema di start dei tre moduli in propagazione a cascata e dei done in senso contrario. Nello scrivere la relazione abbiamo usato il software LaTeX.

Ringraziamo infine i professori per averci guidato in questa stimolante esperienza.