Prova Finale di Reti Logiche 2020/21

Dean Nicola

April 1st, 2021

Matricola:	911817
Codice Persona:	10674826
Docente:	William Fornaciari

Contents

1 Requisiti del Progetto

Il progetto prevede la realizzazione tramite VHDL di un circuito di equalizzazione delle immagini in bianco e nero. Tale circuito poi dovrà essere sintetizzabile per un ipotetica FPGA.
La specifica prevede che il modulo sia conensso ad una RAM di tipo sincrono e che essa contenga un immagine da equalizzare.
Il modulo dovrà leggere tale immagine, elaborarla e scrivere in RAM il risultato.
Dovrà inoltre potrer elaborare eventuali immagini multiple (vedi specifiche tecniche per dettagli).

1.1 Pseudocodice

```
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)
```

1.2 Struttura della RAM

La ram scelta da specifica è di tipo sincrono

1. Dati in lettura vengono ottenuti nel clk successivo rispetto all'indirizzo.
2. Dati in scrittura vengono ceduti assieme all'indirizzo.

Di seguito il contenuto della ram:

Position	**Numer of Bytes**	**Description**
0	1	Column dimension
1	1	Row dimension
2...	Dim=Row*Column	Input Pixels
2+Dim	Dim=Row*Column	Output Pixels

1.3 Specifiche tecniche

- Il modulo dovrà iniziare ad elaborare una volta che il segnale "start" in input verrà alzato.
- Una volta terminata l'elaborazione il segnale Done verrà alzato e si attenderà che il segnale start torni basso prima di eseguire una nuova elaborazione.
- Il reset per la prima elaborazione viene dato dall'esterno per le successive immagini sarà necessario un reset interno al modulo.

- E' consigliato l'utilizzo dell'FPGA **xc7a200tfbg484-1** con un periodo di clock maggiore o uguale a 100MHZ.

2 Implementazione

Per la realizzazione delle specifiche è stata scelta l'opzione di dividere il modulo in 2 Entity :

- `project_reti_logiche` Contiene una macchina a stati che gestisce tutti i flag di stato necessari.
- `Datapath` Contiene tutti i circuiti sequenziali e combinatori controllati dagli stati dell'FSM.

2.1 Descrizione ad alto livello

1. Si attende il segnale di start.
2. Si leggono i primi due byte della Ram contenenti le dimensioni dell'immagine.
3. Si scorre l'immagine una prima volta così da poter calcolare Massimo, Minimo, Delta e Log(Delta+1).
4. Si esegue una seconda passata dei pixel, questa volta per calcolare il valore dei pixel di output sfruttando i risultati ottenuti nella prima passata (qui si provvede pure a salvare i pixel in RAM).
5. Una volta terminata la seconda passata si pone il segnale di done ad alto, si attende che il segnale di start torni a basso e nel frattempo si resettano i valori di tutti i registri così da poter riportare il modulo ad uno stato iniziale pronto ad elaborare altre immagini.
6. Ritorno al punto 1.

2.2 Macchina a Stati Finiti

La **Macchina a Stati Finiti** (FSM) è il circuito che si occupa di guidare il Datapath nei suoi diversi stadi tramite la generazione di segnali di controllo al momento più opportuno che permettono a quest'ultimo di eseguire le varie operazioni nell'ordine corretto.
L'FSM è composta da 2 process: `NEXT_STATE`, sequenziale, che ha il compito di decidere quale sarà il prossimo stato in base agli input forniti; `SET_OUTPUT` è invece il processo combinatorio che calcola le uscite in base allo stato corrente. Segue una descrizione degli stati formali della FSM:

- `START`: Stato in cui registri e flag vengono posti ai valori di default in attesa del segnale di `i_start`. (Questo è anche lo stato di reset)
- `SIZE`: Stato in cui vengono letti i primi 2 byte della RAM (dimensioni dell'immagine) per poi essere salvati nei rispettivi registri di colonna e riga : `COL_REGISTER`,`ROW_REGISTER`
- `STAGE_1`: Stato in cui il flag `Load_Delta` viene settato ad alto e la RAM viene letta un Byte per volta; ognuno di queti byte contribuirà al calcolo dei seguenti registri: `CURRENT_MIM`,`CURRENT_MAX`,`DELTA_REG`.
- `STAGE_2`: Stato in cui vengono incrementi i registri di indirizzo (lettura e scrittura) per la fase di elaborazione dei pixel.
- `WAIT_RAM_i`: Stato di "idle" che permette al modulo di eseguire la lettura di un byte dalla RAM. (Sono presenti 3 stati di wait nei seguenti casi: `SIZE`,`STAGE_1`,`STAGE_2`).
- `ELABORATE`: Stato in cui, dopo la lettura di un pixel in `WAIT_RAM_2`, si eseguono i calcoli necessari per l'equalizzazione (Log,Shift...).
- `WRITE`: si occupa di scrivere i nuovi pixel nella RAM; per farlo assegna `REGISTER_W` a `o_address` e pone ad alto il flag `o_we` per abilitare la RAM in scrittura.
- `FINISH`: Stato in cui si pone `o_done` ad alto e si attende che `i_start` venga riportato a basso per poter terminare l'esecuzione(in questo stato vengono anche resettati tutti i flag e registri) .

Nella figura a seguito è riportato il disegno dell'automa. Sono state usate le seguenti abbreviazioni degli stati per chiarezza:

`i_start`:	`i_s`
`InputRead`:	`siz`
`image_end`:	`im_e`
`Real_End`:	`rl_e`

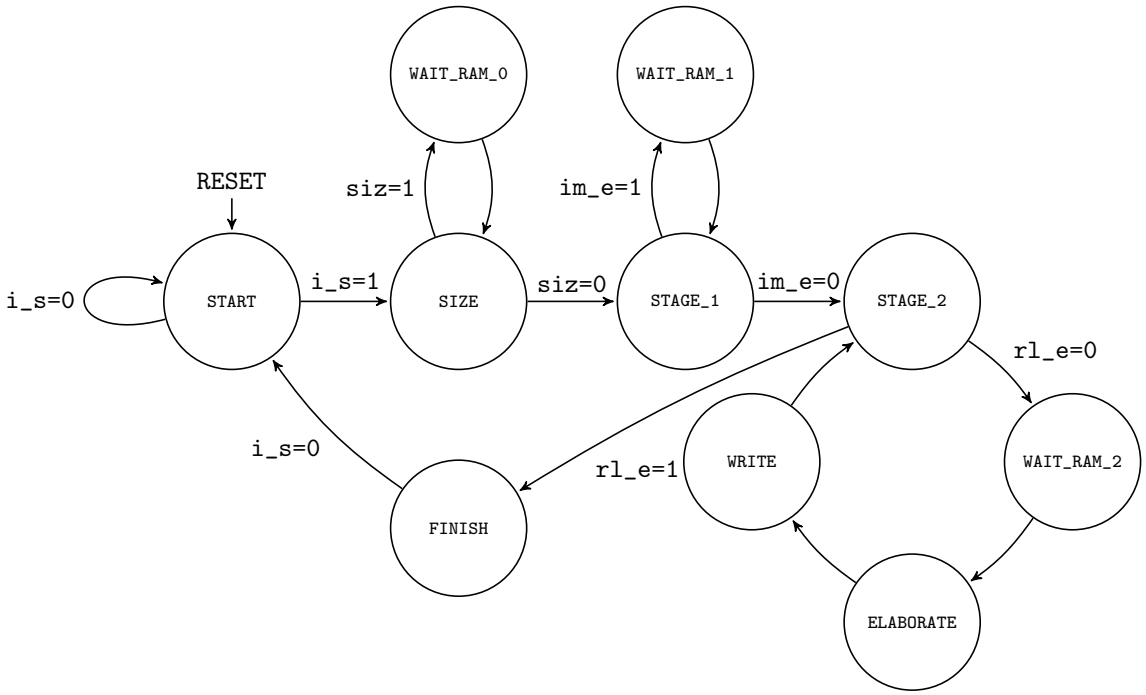

Figure 1: Diagramma degli stati della Macchina a Stati Finiti

2.3 Schema circuitale ad alto livello

Il **Datapath** è quella parte di circuito che svolge effettivamente il lavoro tramite l'implementazione di circuiti sequenziali e combinatori che vengono attivati nei momenti adatti tramite l'ausilio del flag generati dall'FSM. Di seguito una lista dei componenti più rilevanti di tale circuito

- `COUNTER` In breve si occupa di incrementare gli indirizzi e verificare se l'input è terminato. Questo modulo è suddiviso in 3 conatori minori:

  1. **Colonne** Si attiva ogni volta che il clk è alto e il flag increment è attivo e si resetta ogni qualvolta viene raggiunta la dimensione delle colonne salvata nell'apposito registro.
  2. **Righe** Si attiva ogni volta che finiscono le colonne (se finiscono le righe trigghera un flag di fine immagine).
  3. **Indirizzi** Si attiva in contemporanea al contatore di colonne ma incrementa i registri di indirizzo (sia REGAddrW che REGAddrR).

- `MAX/MIN` Un circuito combinatorio controlla quando il pixel corrente soddisfa i requisiti per sostituire l'attuale Max/Min mentre un circuito sequenziale si occupa di effettuare l'effettivo aggiornamento dei registri.

- `LOG` Un circuito sequenziale composto da una serie di controlli a soglia assegna al registro Log il risultato di un operazione di logaritmo su (Delta+1).
- `SHIFT` Un circuito sequenziale esegue, tramite dei controlli a soglia sul registro Log lo shift senza perdita di informazione su (CurrentPixel-Min).
- `OutputAddressSelect` Si occupa di selezionare il corretto registro da porre in ouptut sulla RAM;
  1. write=0 => si pone o_address = REGAddressR
  2. write=1 => si pone o_address = REGAddressW
- `OutputDataSelect` Si occupa di verificare se l'oputput corretto è 255 oppure NewPixel. (prende il minore dei 2)

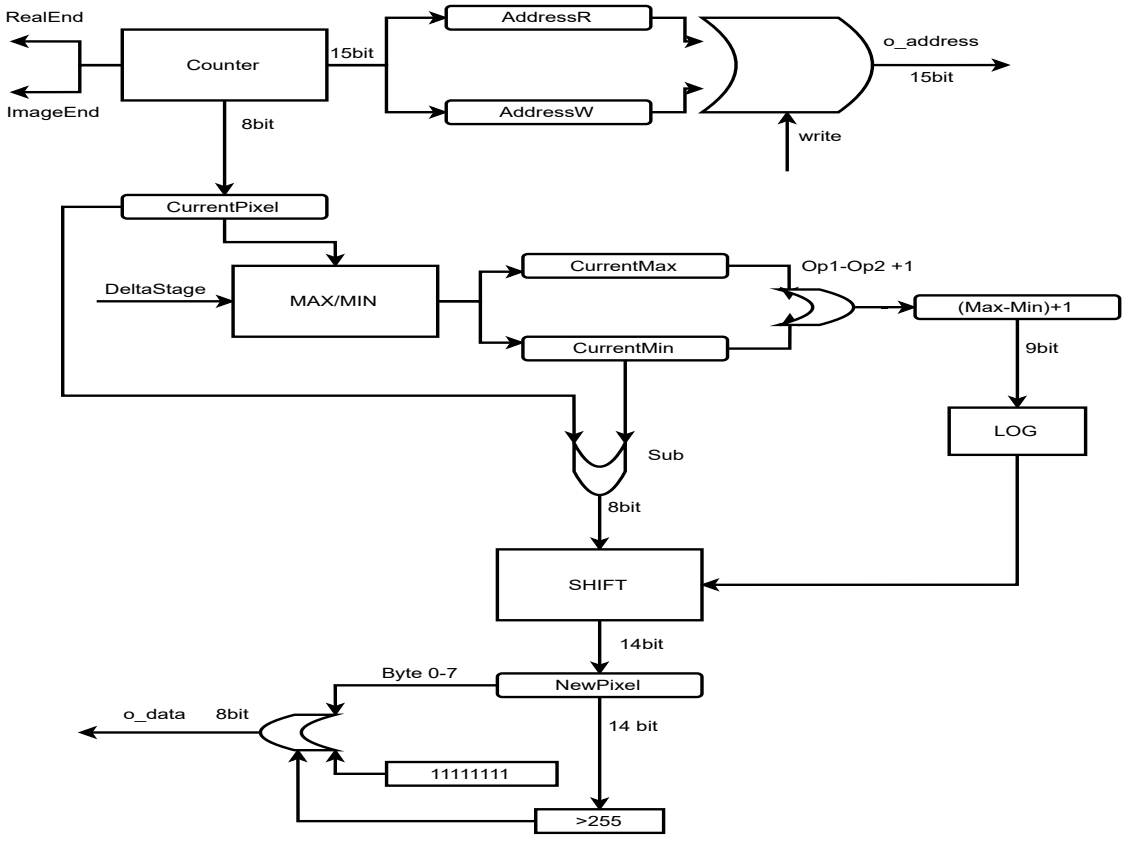

Figure 2: High level circuit scheme

3 Risultati Sperimentali

3.1 Report di Sintesi

Dal punto di vista dell'area occupata dal circuito il report di sintesi riporta il seguente utilizzo dei componenti:

Risorsa	Utilizzo	Disconibilità	Utilizzo%
LUTs(LockUpTables)	160	134600	0.12
Registri(FlipFlop)	112	269200	0.04

Questa tabella mostra le percentuali di hardware utilizzato per la sintesi del modulo.
Altre informazioni fornite dal report sono:

1. **Codifica FSM**: Il sintetizzatore vivado ha ritenuto opportuno sintetizzare gli stati dell'FSM tramite codifica `ONEHOT`.
2. **Registri**: Dal report si evince che solo 8 degli 11 segnali registro utilizzati sono stati sintetizzati come effettivi registri mentre Delta,NewPixel, e ShiftInput sono stati evidentemente eliminati in fase di ottimizzazione.
3. **Adder**: Il report evidenzia che sono stati sintetizzati un totale di 5 sommatori rispettivamente da (16,9,8,8,8) bit.
4. **Mutex-Decoder**: Sono stati sintetizzati un totale di 20 fra mutex e decoder di varie dimensioni.

```
---------------------------------------------
---------------------------------------------
Start RTL Component Statistics
---------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input   16 Bit       Adders := 1
       2 Input    9 Bit       Adders := 1
       2 Input    8 Bit       Adders := 1
       3 Input    8 Bit       Adders := 2
+---Registers :
                  8 Bit    Registers := 8
                  4 Bit    Registers := 1
                  1 Bit    Registers := 2
+---Muxes :
       2 Input   16 Bit        Muxes := 3
      10 Input   11 Bit        Muxes := 1
       2 Input   11 Bit        Muxes := 4
       2 Input    8 Bit        Muxes := 4
       9 Input    4 Bit        Muxes := 1
       2 Input    1 Bit        Muxes := 2
       9 Input    1 Bit        Muxes := 1
      11 Input    1 Bit        Muxes := 4
---------------------------------------------
Finished RTL Component Statistics
---------------------------------------------
```

Figure 3: RTL Component statistics

3.2 Timing Report

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1,067 ns	Worst Hold Slack (WHS):	0,114 ns	Worst Pulse Width Slack (WPWS):	4,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	190	Total Number of Endpoints:	190	Total Number of Endpoints:	88

All user specified timing constraints are met.

Figure 4: Timing Report

Dal Timing Report si evince che il `Worst negative Slack` è pari a **1,067 ns** perfettamente dentro le specifiche di clock.

3.3 Performance

Nel sottocapitolo precedente si è mostrato il risultato del timing report che purtroppo non misura le effettive performance del modulo. Durante il testing si è provveduto a fornire al modulo immagini di ogni dimensione per valutarne i tempi di esecuzione. Un immagine di dimensione massima (128 x 128) richiede un tempo pari a **9832350ps** ovvero una media di **600ps a pixel**.

3.4 Warning

Durante la fase di realizzazione del progetto sono emerse diverse problematiche durante la sintesi riguardanti prevalentemente warning di due tipi:

1. `Infering Latch` L'utilizzo scorretto dei process o dell'operatore di assegnamento potrebbe causare dei latch indesiderati nel circuito; tali latch causano anomalie nel funzionamento del modulo.

2. `Sensitivity List` Registri mancanti nella sensitivity list di alcuni process.

L'attuale versione del circuito non presenta più nessuna delle sopracitate problematiche che sono state eliminate singolarmente in fase di debugging.

4 TestBench

Di seguito una serie di test che mettono alla prova le specifiche o che si sono rivelati utili per far emergere e risolvere alcune problematiche del circuito.

4.1 Test di verifica segnali in input

1. **Test di reset asincrono**: Questo test verifica il corretto funzionamento del reset asincrono richiesto da specifica.Come si vede in figura appena il reset sale alcuni registri critici vengono inizializzati ad un valore di default.`(CurrentMax,CurrentMin,o_address,SubC,SubR,ImageEnd)`.

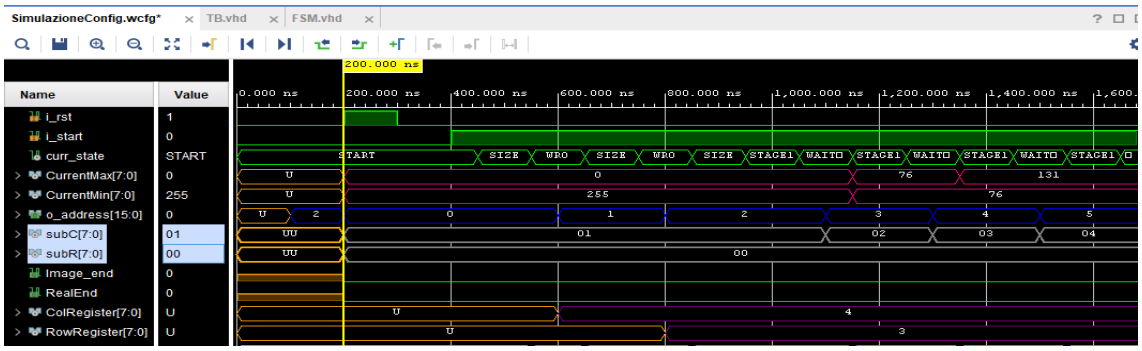

Figure 5: Tb reset asincrono

2. **Test ad immagini multiple**: Questo test è volto a controllare il corretto funzionamento dell'equalizzazione di più immagini consecutivamente senza necessità di un segnale di reset esterno.

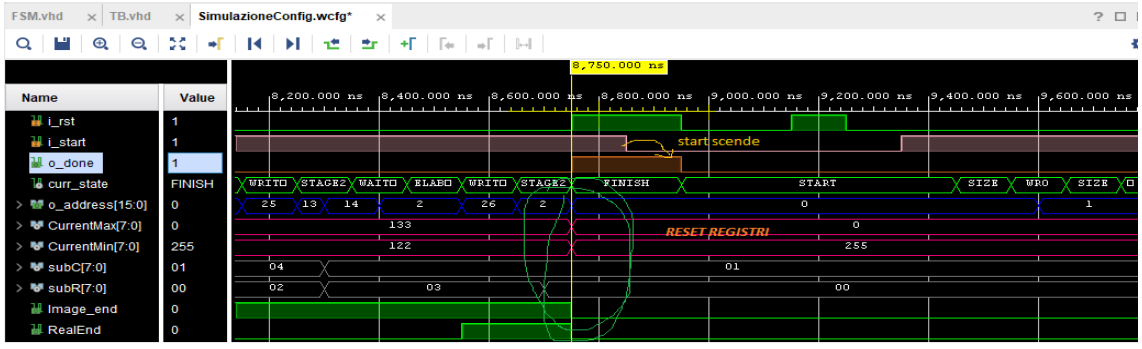

Figure 6: Doppia immagine

4.2 Test che hanno rilevato "BUG" nel circuito

1. **Massimo e Minimo coincidenti (img di 1 pixel)**: Questo test non è molto rilevante ma ha permesso di evidenziare una problematica all'interno di una versione passata del circuito che non prevedeva appunto che massimo e minimo potessero coincidere.

2. **Output selection (Min(NewPixel,255))**: Tramite tale test si è riscontrata la problematica di uno scorretto dimensionamento del registro DELTA contenente l'operazione ((Max-Min) +1)

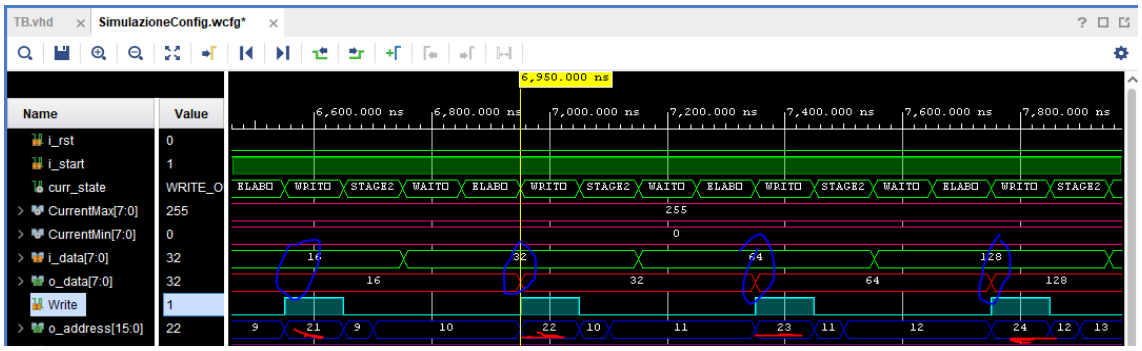

Figure 7: Tb potenze di 2

4.3 Test Generici

1. **Immagine di dimensione massima (128X128)**: Questo test è stato pensato per testare la massima dimensione di input da specifica per vedere come il modulo si sarebbe comportato in termini di prestazionali; il modulo impiega circa: **9832350ps** per terminare l'elaborazione di un immagine di tali dimensioni.

2. **Test con dimensioni e pixel casuali**: Il modulo è stato testato con successo su un centinaio di immagini generate casualmente.

5 Conclusioni

Si ritiene che le specifiche siano state pienamente soddisfatte;
Tale conclusione è dimostrabile in due modi:

1. Report di sintesi discusso nell'apposito capitolo.
2. Testing intensivo sia casuale che mirato a corner case.

Durante la progettazione del circuito si è cercato di essere quanto più "modulari" possibile cercando di separare i diversi task dell'algoritmo in moduli più piccoli dedicati a specifiche operazioni. Questo garantisce al circuito di essere reattivo e sfruttare a pieno il parallelismo delle FPGA.

5.1 Possibili ottimizzazioni e migliorie

1. L'FSM è sicuramente ottimizzabile unendo gli stati di `waiting_ram_i` in un unico stato
2. Si potrebbe ottimizzare l'elaborazione dei pixel leggendo un pixel mentre ne eleboro un altro così da raddoppiare il "througput" del circuito.