Prova Finale (Progetto di Reti Logiche)

Politecnico di Milano
Anno Accademico 2020/2021

Prof. Fabio Salice

Elisabetta Fedele (Codice Persona 10631762 - Matricola 911320)
Filippo Lazzati (Codice Persona 10629918 - Matricola 910614)

Indice

1 Introduzione **2**
1.1 Scopo del progetto . . . 2
1.2 Algoritmo di equalizzazione . . . 2
1.2.1 Esempio qualitativo . . . 2
1.2.2 Esempio quantitativo . . . 3
1.3 Interfaccia componente . . . 4
1.4 Dati e descrizione memoria . . . 5
1.5 Protocolli di funzionamento . . . 5
1.5.1 Protocollo di inizio e fine computazione . . . 5
1.5.2 Protocollo di accesso alla memoria . . . 5

2 Architettura **6**
2.1 FSM . . . 6
2.1.1 `IDLE` state . . . 6
2.1.2 `GET` state . . . 6
2.1.3 `COL` state . . . 6
2.1.4 `ROW` state . . . 6
2.1.5 `MIN_MAX` state . . . 6
2.1.6 `ARG` state . . . 6
2.1.7 `LOG` state . . . 6
2.1.8 `UPDATE_R` state . . . 6
2.1.9 `UPDATE_W` state . . . 6
2.1.10 `DONE state` . . . 6
2.2 Uscite di default . . . 8
2.3 Variables, signals e valori di default . . . 8
2.4 Scelte implementative . . . 9
2.5 Scelte di progettazione . . . 9

3 Sintesi **10**
3.1 Area occupata . . . 10
3.2 Codifica degli stati . . . 10
3.3 Report di timing . . . 11
3.4 Warning post synthesis . . . 11

4 Simulazioni **12**
4.1 Obiettivo . . . 12
4.2 Test . . . 12
4.2.1 Casi limite . . . 12
4.2.2 Funzionamento dei segnali . . . 13
4.2.3 Test casuali . . . 13
4.3 Osservazioni . . . 13

5 Efficienza e ottimizzazioni **14**

6 Conclusioni **14**

1 Introduzione

1.1 Scopo del progetto

Sia data un'immagine di 8 bit in scala di grigi di dimensione massima pari a 128 x 128 pixel.
L'obiettivo del progetto è l'implementazione di un componente hardware descritto in VHDL che, ricevuta in ingresso tale immagine e dei segnali di controllo, sia in grado di applicarvi l'algoritmo di equalizzazione fornito nelle specifiche e restituire in uscita i risultati.

1.2 Algoritmo di equalizzazione

Equalizzare l'istogramma di un'immagine significa applicare una funzione di trasformazione ai pixel in essa contenuti con lo scopo di aumentare il contrasto globale. L'algoritmo utilizzato, una versione semplificata del più complesso algoritmo standard di equalizzazione, si articola nei seguenti passi:

1. Ricerca del minimo valore (`MIN_PIXEL_VALUE`) e del massimo valore (`MAX_PIXEL_VALUE`)
2. Calcolo di delta value
   $\texttt{DELTA\_VALUE} = \texttt{MAX\_PIXEL\_VALUE} - \texttt{MIN\_PIXEL\_VALUE}$
3. Calcolo del valore di shift
   $\texttt{SHIFT\_LEVEL} = 8 - \lfloor \log_2(\texttt{DELTA\_VALUE} + 1) \rfloor$
4. Trasformazione di ogni pixel
   $\texttt{TEMP\_PIXEL} = (\texttt{CURRENT\_PIXEL\_VALUE} - \texttt{MIN\_PIXEL\_VALUE}) << \texttt{SHIFT\_LEVEL}$
   $\texttt{NEW\_PIXEL\_VALUE} = min(255, \texttt{TEMP\_PIXEL})$

1.2.1 Esempio qualitativo

Si consideri l'immagine 128 x 128 pixel e il relativo istogramma in Figura 1, realizzato con ImageJ[1]:

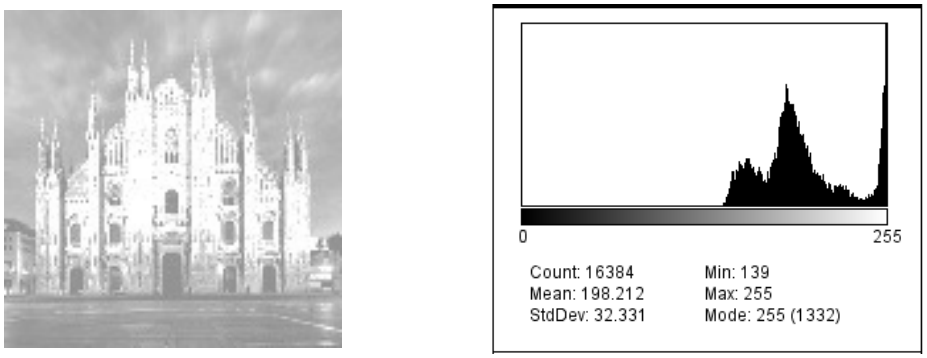


Figura 1: Immagine originale e relativo istogramma

Nell'istogramma si nota come la distribuzione dei grigi nell'immagine non è uniforme, ma è spostata verso i valori alti. Questo lo si può anche comprendere osservando l'immagine, che risulta sbiadita.
Il risultato dell'applicazione dell'algoritmo è mostrato in Figura 2, dove il nuovo istogramma mostra come i valori di intensità dei singoli pixel siano stati ridistribuiti in modo da coprire l'intero range proposto.

[1] ImageJ is a public domain Java image processing program inspired by NIH Image for the Macintosh. It can display, edit, analyze, process, save and print 8-bit, 16-bit and 32-bit images (`https://imagej.nih.gov/ij/docs/intro.html`)

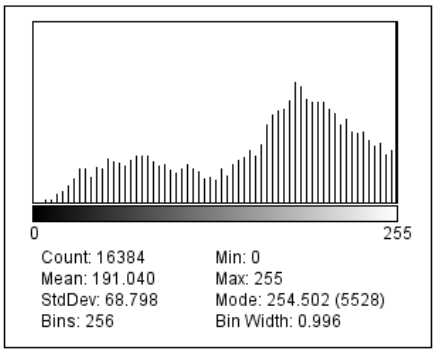


Figura 2: Immagine trasformata e relativo istogramma

1.2.2 Esempio quantitativo

Siano dati in ingresso i seguenti valori corrispondenti ai pixel di un'immagine 3x3:

28	27	27	18	20	19	23	16	30

L'algoritmo procede in questa modo:

1. Calcolo di `MIN_PIXEL_VALUE` = 16
2. Calcolo di `MAX_PIXEL_VALUE` = 30
3. Calcolo di `DELTA_VALUE` = 14
4. Calcolo di `SHIFT_LEVEL` = $8 - \lfloor \log_2(14+1) \rfloor = 8 - 3 = 5$
5. Calcolo per ogni pixel di `CURRENT_PIXEL_VALUE` - `MIN_PIXEL_VALUE`

12	11	11	2	4	3	7	0	14

6. Scorrimento a sinistra di `SHIFT_LEVEL` bit per ogni pixel

255	255	255	64	128	96	224	0	255

1.3 Interfaccia componente

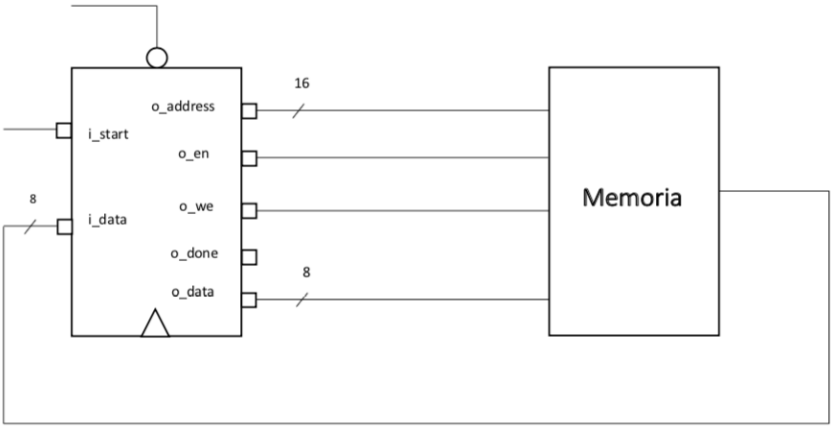
Il componente presenta la seguente interfaccia:

```
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;
```

Nello specifico:

- `i_clk` è il segnale di `CLOCK` in ingresso;
- `i_rst` è è il segnale di `RESET` che inizializza la macchina;
- `i_start` è il segnale di `START`;
- `i_data` è il segnale che arriva dalla memoria in seguito ad una richiesta di lettura;
- `o_address` è il segnale di uscita che trasmette l'indirizzo alla memoria;
- `o_done` è il segnale di fine elaborazione;
- `o_en` è il segnale di `ENABLE` da inviare alla memoria per poter comunicare sia in lettura che in scrittura;
- `o_we` è il segnale di `WRITE ENABLE` per abilitare la scrittura in memoria;
- `o_data` è il segnale di uscita dal componente verso la memoria.

Componente e memoria vengono poi collegati sfruttando i pin `o_address`, `o_en`, `o_we`, in uscita verso la memoria, e `i_data`, in ingresso nel componente, come mostrato nella seguente immagine:



Si noti come le porte `i_start` e `o_done` sono interfacciate ad altri componenti.

1.4 Dati e descrizione memoria

I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte nel seguente modo:

	Numero colonne (C)	Indirizzo 0
	Numero righe (R)	Indirizzo 1
Immagine originale	Pixel 1 - originale	Indirizzo 2
	Pixel 2 - originale	Indirizzo 3
	...	...
	Pixel C*R - originale	Indirizzo 1 + C*R
Immagine trasformata	Pixel 1 - trasformato	Indirizzo 2 + C*R
	Pixel 2 - trasformato	Indirizzo 3 + C*R
	...	...
	Pixel C*R - trasformato	Indirizzo 1 + 2*C*R

Nei primi due byte sono presenti rispettivamente il numero di colonne e il numero di righe dell'immagine da trasformare. I vari pixel, ciascuno di 8 bit, sono memorizzati in memoria a partire dalla posizione 2, riga per riga.
I pixel dell' immagine equalizzata, invece, vengono inseriti in memoria a partire dalla posizione 2+(C*R), anche essi riga per riga.

1.5 Protocolli di funzionamento

Sono qui presentate le regole che definiscono i valori da applicare ai segnali durante la computazione.

1.5.1 Protocollo di inizio e fine computazione

Quando il segnale di ingresso `i_start` viene portato a `'1'`, il componente progettato inizia la computazione spostandosi nello stato `GET` in cui inizia a richiedere dati alla memoria. Il segnale `i_start` rimarrà alto fino a quando `o_done` non verrà portato alto.
Dopo aver scritto l'intero risultato in memoria, il componente alza a `'1'` il segnale di `o_done`, per segnalare la fine dell'elaborazione. Il segnale `o_done` rimane alto fino a quando il segnale di `i_start` non viene riportato a `'0'`; dopodichè anche `o_done` viene abbassato. Nell'intervallo di tempo durante cui `i_start='0'` e `o_done='1'` non può essere inviato nessun nuovo segnale di `START`.

1.5.2 Protocollo di accesso alla memoria

Durante la computazione si accede in memoria all'indirizzo indicato in `o_address` quando `o_en='1'`.
In particolare:

- Se `o_we='0'` si legge dalla memoria il dato contenuto nel segnale `i_data`
- Se `o_we='1'` si scrive in memoria il dato contenuto nel segnale `o_data`

2 Architettura

Il componente è stato progettato tramite una macchina a stati finiti (FSM) di Mealy, in cui il valore delle uscite è funzione sia dello stato che degli ingressi.

2.1 FSM

La macchina si compone di 10 stati. Di seguito se ne fornisce una sintetica descrizione comportamentale.

2.1.1 `IDLE` state

Stato iniziale e di default della macchina, in cui si attende il segnale di `i_start` e a cui si ritorna in caso di ricezione di un segnale di `i_rst`.

2.1.2 `GET` state

Stato in cui si attende la risposta della memoria in seguito alla richiesta di un dato. Lo stato prossimo è contenuto in `state_return`.

2.1.3 `COL` state

Stato in cui si legge il contenuto della prima cella di memoria (numero di colonne dell'immagine da trasformare) e lo si assegna a `col_count`.

2.1.4 `ROW` state

Stato in cui si legge il contenuto della seconda cella di memoria (numero di righe dell'immagine da trasformare) e si calcola `max_address`, l'indirizzo in cui è memorizzato il valore dell'ultimo pixel.

2.1.5 `MIN_MAX` state

Stato in cui vengono calcolati i valori del minimo e del massimo pixel presenti nell'immagine. Ciò avviene scorrendo la memoria dall'indirizzo 2 fino all'indirizzo `max_address` calcolato in `ROW`, nel caso pessimo (si veda il paragrafo 5).
Il valore del minimo e del massimo sono poi assegnati rispettivamente ai signals `min` e `max`.

2.1.6 `ARG` state

Stato in cui viene calcolato il valore `max-min+1` e viene assegnato al signal `delta_value`.

2.1.7 `LOG` state

Stato in cui viene calcolato lo `SHIFT_LEVEL` il cui valore viene assegnato al signal `shift_value`.

2.1.8 `UPDATE_R` state

Stato in cui vengono letti uno alla volta i valori originali dei pixel, che vengono poi modificati e assegnati alla variabile `temp_pixel_shift` in attesa di essere scritti in memoria in `UPDATE_W`.

2.1.9 `UPDATE_W` state

Stato in cui vengono scritti in memoria (uno per volta) i valori dei pixel dell'immagine trasformata.
Al termine della computazione, prima che avvenga la transizione verso `DONE`, tutti i segnali e le uscite vengono settati ai valori di default.

2.1.10 `DONE` state

Stato per la terminazione di un'istanza di computazione che riporta la FSM allo stato di `IDLE` rendendo il modulo pronto a ricevere una nuova immagine.

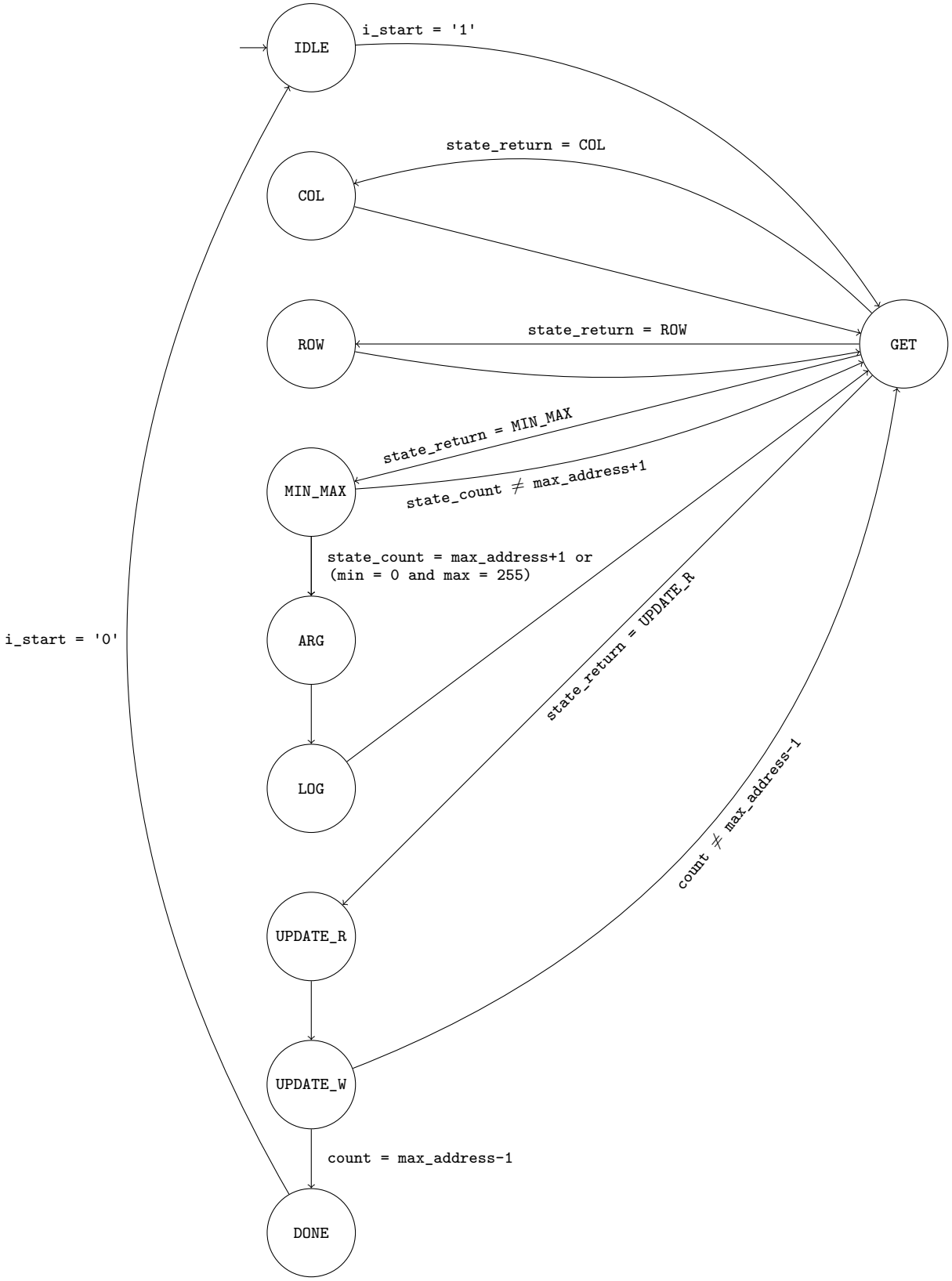


Figura 3: FSM

In ogni stato è presente una transizione verso lo stato `IDLE` in caso di ricezione del segnale di `RESET` (`i_rst='1'`), non rappresentata nel diagramma per questioni di leggibilità.

2.2 Uscite di default

In certi casi, il valore di alcuni segnali di uscita è indifferente. In particolare:

- se `i_start='0'`: tutte le uscite del componente non sono rilevanti
- se `o_en='1'` e `o_we='0'`: l'uscita `o_data` non è rilevante

Quando le uscite non sono rilevanti assumono un valore di default.
Di seguito sono riportati i valori di default assegnati alle uscite e le motivazioni delle scelte fatte.

Nome	**Default**	**Motivazione**
`o_address`	others ⇒'0'	L'indirizzo 0 è il primo indirizzo di lettura
`o_en`	'0'	Fino a quando `i_start='0'` non si accede alla memoria
`o_we`	'0'	Il primo accesso alla memoria è in lettura
`o_data`	others ⇒'0'	Scelta arbitraria
`o_done`	'0'	All'inizio dell'elaborazione vale '0'

2.3 Variables, signals e valori di default

Sono qui riportati i signals e le variables utilizzati con i rispettivi valori di default.

Nome	**Uso**	**Default**	**Motivazione**
`min`	Contiene il minimo valore	255	Qualunque valore è ≤ 255
`max`	Contiene il massimo valore	0	Qualunque valore è ≥ 0
`count`	Helper per leggere i pixel	2	I pixel iniziano dall'indirizzo 2
`max_address`	Contiene indirizzo fine lettura	2	I pixel iniziano dall'indirizzo 2
`delta_value`	Contiene `DELTA_VALUE` + 1	others ⇒'0'	Scelta arbitraria
`shift_level`	Contiene `SHIFT_LEVEL`	0	0 ⇒ istogramma già equalizzato
`state_return`	Contiene il `next_state` di `GET`	`IDLE`	Scelta arbitraria
`current_state`	Contiene lo stato attuale	`IDLE`	Scelta arbitraria
`col_count`	Contiene il numero di colonne	0	Scelta arbitraria
`o_address_reg`	Lettura dell'uscita `o_address`	others ⇒'0'	Scelta arbitraria
`temp_pixel`	Contiene `i_data-min`	others ⇒'0'	Scelta arbitraria
`temp_pixel_shift`	Contiene `temp_pixel` shiftato	others ⇒'0'	Scelta arbitraria

2.4 Scelte implementative

Nella realizzazione del componente come FSM si è deciso di utilizzare due soli processi: `state_reg` e `delta_lambda`.

- Il process `state_reg` definisce dei registri parallelo-parallelo dotati di reset asincrono che vengono scritti e letti in corrispondenza dei fronti di salita del clock (`i_clk`), aggiornando il valore dello stato e il valore dell'uscita;
- Il process `delta_lambda` rappresenta la FSM e determina lo stato successivo e i valori delle uscite analizzando lo stato presente e gli ingressi.

2.5 Scelte di progettazione

Per l'implementazione del componente si utilizza la seguente FPGA: xc7a200tfbg484-1, appartenente alla famiglia Artix 7 prodotta dall'azienda Xilinx. Ha un'altezza di 1mm e occupa una superficie di 23x23mm^2. A seconda della versione cambia il range di temperature di lavoro possibili.
Il suo speed grade è -1, dunque la frequenza di lavoro interna massima è inferiore alle frequenze delle altre FPGA appartenenti alla medesima famiglia. In ogni caso, è sufficiente a garantire un periodo di clock di 100ns, richiesto dalla specifica.

3 Sintesi

In questa sezione sono raccolte le informazioni di rilievo riguardanti la fase di sintesi.

3.1 Area occupata

Nel *Report Utilization* è riportato il numero di LUT e FF inferiti durante la sintesi.

Risorsa	**Utilizzo**	**Disponibilità**	**Utilizzo %**
Look Up Table	248	133800	0,18%
Flip Flop	100	267600	0,03%

Osservazioni:

- Entrambe le percentuali di utilizzo risultano essere molto minori del 100%, dunque il modulo implementato occupa solo una piccola parte della FPGA;
- Il codice VHDL è stato scritto in modo da evitare l'inferenza di latch, col preciso scopo di rendere l'intero componente sincronizzato sul fronte di salita del clock. Sono così evitati gli effetti di propagazioni indesiderate dei segnali;
- I 100 FF inferiti sono tutti di tipo D e sono così suddivisi:
  - 10 sono di tipo **FDPE** (dotati di **preset** asincrono) e vengono inferiti per i bit di quei segnali che vengono inizializzati a '1'. In particolare, rientrano in questa categoria tutti i bit relativi a `min` (inizializzato a 255="11111111") e al bit in seconda posizione di `max_address` e `count` (inizializzati a 2="0...010");
  - 90, i restanti, sono di tipo **FDCE** (dotati di **clear** asincrono) e corrispondono ai bit dei segnali inizializzati a '0'.

3.2 Codifica degli stati

Il *Synthesis Report* di Vivado contiene, tra le varie informazioni, il numero di bit utilizzati per ciascun registro. In particolare, per il registro contenente lo stato attuale (`curr_state_reg`) sono stati usati 4 bit e una codifica binaria naturale, più efficiente dal punto di vista spaziale di quella one-hot.

3.3 Report di timing

La specifica richiede che il componente sintetizzato funzioni correttamente con un periodo di clock $T_{clock} \leq 100$ns. Per verificare la conformità a tale requisito, dopo aver sintetizzato il componente, si è aggiunto un vincolo temporale a `i_clock` che setta il periodo a 100ns e il Duty Cycle al 50%.
I risultati sono riassunti nel *Design Timing Summary* riportato di seguito:

Setup		Hold	
Worst Negative Slack (WNS):	91.302 ns	Worst Hold Slack (WHS):	0.170 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:	166	Total Number of Endpoints:	166

All user specified timing constraints are met.

Figura 4: *Design Timing Summary* con $T_{clock} = 100$ns

In esso troviamo indicati i valori del Worst Negative Slack, ovvero dello Slack calcolato nel caso peggiore. Ricordiamo che in generale Slack = RAT - AAT, dove

- RAT rappresenta il tempo di arrivo del segnale richiesto affinchè il vincolo venga rispettato
- AAT rappresenta il tempo di arrivo effettivo

Nel nostro caso, il WNS è positivo sia per il tempo di setup che per quello di hold, dunque il vincolo stimato in fase di sintesi è effettivamente rispettato.
Per comprendere il margine temporale rimanente, e quindi la massima frequenza di clock ammissibile, sono stati sottoposti vincoli più stringenti al periodo di clock. Si può affermare che il componente funziona correttamente fino a $T_{clock} = 9$ ns. Infatti con tale vincolo si ricava la tabella in Figura 5 dove il WNS rimane positivo.

Setup		Hold	
Worst Negative Slack (WNS):	0.302 ns	Worst Hold Slack (WHS):	0.170 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:	166	Total Number of Endpoints:	166

All user specified timing constraints are met.

Figura 5: *Design Timing Summary* con $T_{clock} = 9$ ns

3.4 Warning post synthesis

I warning problematici, tra cui quelli riguardanti l'inferenza indesiderata di latch, sono stati risolti.
Gli unici warning presenti si trovano nel report DRC, ma non creano problemi in quanto segnalano la necessità di specificare vincoli fisici per le porte di I/O al fine di generare un corretto bitstream (aspetto non richiesto dalle specifiche).

4 Simulazioni

4.1 Obiettivo

Allo scopo di verificare il corretto funzionamento del componente sintetizzato, il codice è stato testato dapprima con il test bench fornito insieme alle specifiche e in seguito con test realizzati ad hoc aventi i seguenti obiettivi:

1. Verificare i casi limite
2. Verificare il corretto funzionamento dei segnali
3. Coprire il massimo numero di cammini

4.2 Test

4.2.1 Casi limite

Per testare i casi limite il componente è stato sottoposto ai seguenti casi di test:

1. **Immagine 1 x 1 pixel**: la dimensione dell'immagine è minima.
   - Behavioral Simulation: 1 950 ns
   - Post Syntesis Functional Simulation: 1 950,1 ns

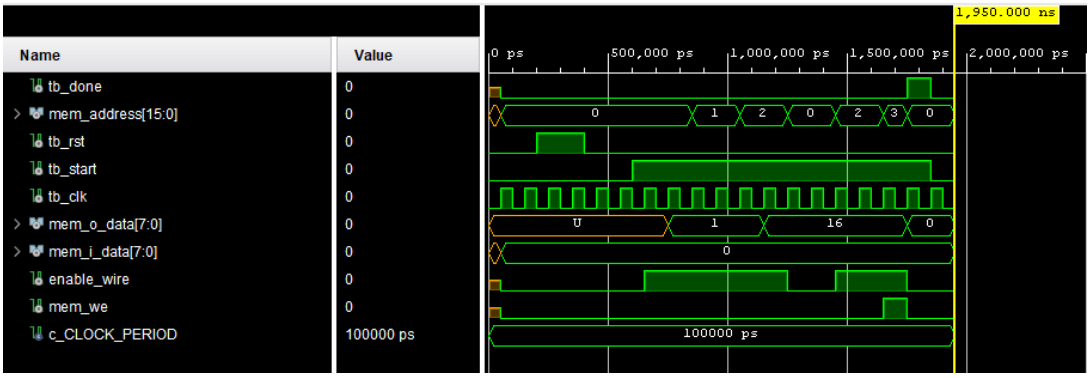


Figura 6: Simulazione con immagine 1 x 1 pixel

2. **Immagine 128 x 128 pixel**: la dimensione dell'immagine è massima.
   - Behavioral Simulation: 8 193 450 ns
   - Post Syntesis Functional Simulation: 8 193 450,1 ns
3. **Immagine 2 x 2 pixel con `DELTA_VALUE` = 0**: l'immagine contiene un solo valore di intensità. Il risultato sarà un'immagine nera composta solo da bit 0.
   - Behavioral Simulation: 3 450 ns
   - Post Syntesis Functional Simulation: 3 450,1 ns
4. **Immagine 2 x 2 pixel con `DELTA_VALUE` = 255**: l'immagine contiene sia il massimo che il minimo valore di intensità (0 e 255). L'immagine in uscita rimane invariata.
   - Behavioral Simulation: 3 450 ns
   - Post Syntesis Functional Simulation: 3 450,1 ns

4.2.2 Funzionamento dei segnali

Per la verifica del corretto funzionamento dei segnali, il componente è stato sottoposto ai seguenti casi di test:

1. **Reset Asincrono**: invia un segnale di `RESET` asincrono durante l'elaborazione, come mostrato in Figura 7.
   Il test è superato in quanto la computazione riprende da `IDLE` con i valori di default una volta che `i_rst` viene abbassato e il risultato finale risulta corretto

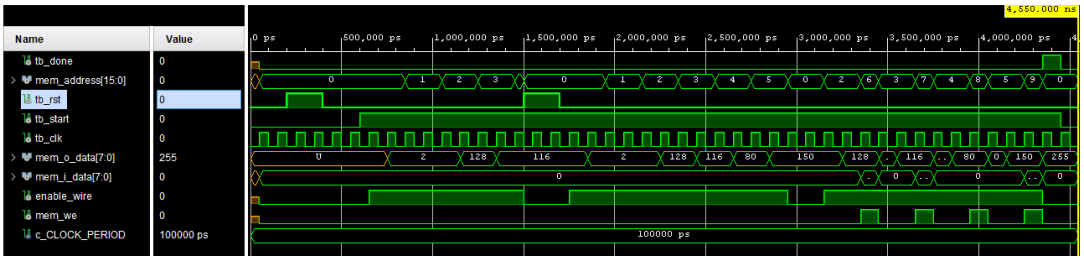


Figura 7: Simulazione con immagine 2 x 2 con segnale di `RESET`

2. **Computazione Multipla**: vengono fornite in ingresso due immagini diverse all'interno della stessa computazione senza l'attivazione di `i_rst` per passare da una all'altra. Il risultato della computazione è corretto.

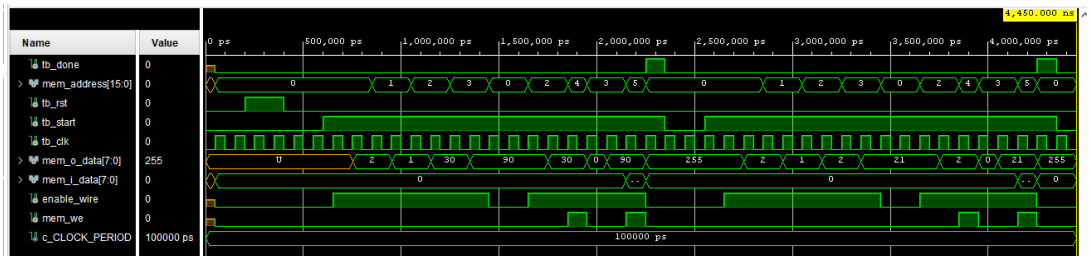


Figura 8: Simulazione con due immagini inviate in successione

4.2.3 Test casuali

Per aumentare la probabilità di trovare eventuali errori, il componente è stato sottoposto a numerosi test bench casuali generati utilizzando un applicativo scritto in C.
La correttezza di tali simulazioni è stata poi verificata tramite la lettura del file di log.

4.3 Osservazioni

Il componente supera correttamente tutti i test proposti in precedenza sia a livello *Behavioral* che *Post-Synthesis*. Quest'ultimo tipo di simulazione è stato essenziale per verificare che eventuali ottimizzazioni applicate dal tool in fase di sintesi non influenzino il corretto funzionamento del componente.

5 Efficienza e ottimizzazioni

In fase di progettazione è stata data importanza allo sviluppo di un dispositivo non solo funzionale, ma anche performante.
Non essendo l'ottimizzazione spaziale particolarmente significativa visto l'esiguo numero di risorse occupate (vedi paragrafo 3.1), si è focalizzata l'attenzione sul miglioramento delle performance temporali.
Il punto di partenza sono state le seguenti osservazioni:

- Il tempo di computazione, all'aumentare dei pixel, aumenta a causa di due percorsi critici:
  1. Ricerca del minimo e del massimo in `MIN_MAX`
  2. Elaborazione dei pixel in ingresso e conseguente scrittura in memoria in `UPDATE_R` e `UPDATE_W`
- Essendo necessario leggere i valori di tutti i pixel memorizzati in memoria almeno una volta per elaborarli e poi riscriverli, la complessità asintotica della computazione non può sicuramente scendere sotto l'$O(N)$, dove N è il numero di pixel dell'immagine fornita in ingresso;
- Essendo i valori dei pixel su 8 bit, il minimo e il massimo valore che essi possono assumere sono rispettivamente 0 e 255. Di conseguenza, l'individuazione di entrambi i valori nello stato di `MIN_MAX` rende superflua la lettura dei restanti valori.

Si è quindi deciso di aggiungere una transizione di uscita dallo stato `MIN_MAX` verso `ARG`, lo stato successivo, quando si verifica `min_reg=0` e `max_reg=255`. Il miglioramento apportato da tale ottimizzazione è ben visibile confrontando i tempi di simulazione dei due casi di test proposti. Entrambe le immagini sono 128 x 128 pixel e hanno `min=0` e `max=255`, ma differiscono per l'indirizzo di memoria in cui sono localizzati il minimo e il massimo.

	Cella min	**Cella max**	**Tempo**
Test 1	2	3	4917250ns
Test 2	16384	16385	8193450ns

Nel test 1, grazie all'ottimizzazione, si esce da `MIN_MAX` alla terza iterazione, mentre nel test 2 si eseguono 16384 iterazioni. Il tempo di computazione nel caso in cui viene sfruttata a pieno l'ottimizzazione è quasi dimezzato rispetto all'altro. Infatti, viene risparmiato un tempo pari a t = 2*(16381*T_{clock}) = 3276200ns (con T_{clock} = 100 ns). Il risultato ottenuto è coerente con il ragionamento fatto precedentemente in quanto per leggere ciascuna cella il componente ha bisogno di due cicli di clock (uno in stato di `GET` e uno in `MIN_MAX`).

6 Conclusioni

Il componente descritto nelle specifiche è stato sintetizzato con successo e tutte le simulazioni effettuate (sia *Behavioral* che *Functional Post-Synthesis*) terminano con esito positivo. Si può dunque affermare che il componente prodotto è in grado di interfacciarsi con una memoria avente indirizzamento al byte sfruttando il protocollo descritto nelle specifiche ed è in grado di applicare l'algoritmo di equalizzazione dell'istogramma dato a qualunque immagine di dimensione compresa fra 1x1 pixel e 128x128 pixel.
In altre parole, il componente realizzato soddisfa le specifiche.
Eventuali migliorie vanno nella direzione di ridurre il numero di LUT e FF inferiti dal tool di sintesi e nella riduzione del tempo necessario all'elaborazione di un'immagine.
In ogni caso, come detto in precedenza, la complessità temporale dell'algoritmo usato non potrà comunque mai scendere sotto $O(N)$, con N numero di pixel dell'immagine.