Prova Finale - Progetto di Reti Logiche

Prof. Gianluca Palermo – Anno Accademico 2020/21

Autori:
Valeria Pantè (Codice Persona 10712755 – Matricola 938047)
Alberto Panzanini (Codice Persona 10629004 – Matricola 912390)

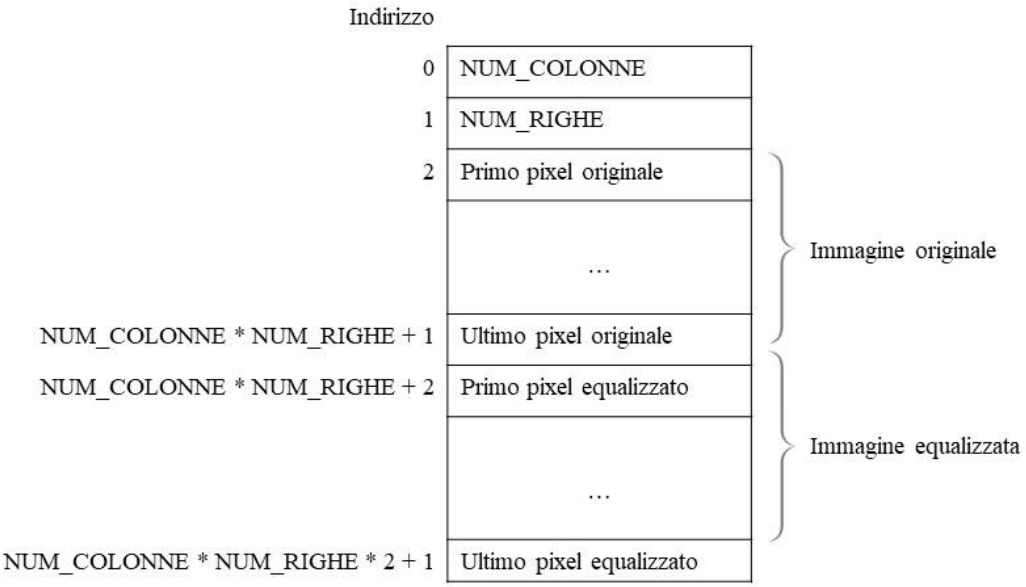
Indice

1 Introduzione

Si vuole progettare un componente che equalizzi un'immagine in ingresso secondo una versione semplificata del metodo di equalizzazione dell'istogramma, un metodo pensato per ricalibrare il contrasto di un'immagine quando l'intervallo dei valori di intensità sono molto vicini, effettuandone una distribuzione su tutto l'intervallo di intensità al fine di incrementare il contrasto.

1.1 Specifiche

Il modulo da implementare deve leggere l'immagine da elaborare da una memoria in cui è memorizzata sequenzialmente e riga per riga. Ogni byte corrisponde ad un pixel dell'immagine. La dimensione dell'immagine è definita da 2 byte, memorizzati a partire dall'indirizzo 0. Il byte all'indirizzo 0 si riferisce alla dimensione di colonna; il byte all'indirizzo 1 si riferisce alla dimensione di riga. L'immagine è memorizzata a partire dall'indirizzo 2 e in byte contigui. Quindi il byte all'indirizzo 2 è il primo pixel della prima riga dell'immagine. La dimensione massima dell'immagine è 128x128 pixel. L'immagine equalizzata viene scritta in memoria immediatamente dopo l'immagine originale, quindi partendo dalla posizione 2+(NUM_COLONNE*NUM_RIGHE).



L'algoritmo di equalizzazione sarà applicato solo ad immagini in scala di grigi a 256 livelli e deve trasformare ogni suo pixel nel modo seguente:

```
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)
```

Dove `MAX_PIXEL_VALUE` e `MIN_PIXEL_VALUE`, sono il massimo e minimo valore dei pixel dell'immagine, `CURRENT_PIXEL_VALUE` è il valore del pixel da trasformare e `NEW_PIXEL_VALUE` è il valore del nuovo pixel.

Nel caso in cui nell'immagine originale fossero presenti sia un pixel con il valore 0 che un pixel con il valore 255, l'immagine equalizzata rimane invariata, quindi il componente si limita a ricopiare i pixel dell'immagine originale nei rispettivi indirizzi dei pixel dell'immagine equalizzata.

Un altro caso particolare è quello in cui i valori di tutti i pixel dell'immagine sono uguali. In questo caso i valori di tutti i pixel dell'immagine equalizzata saranno uguali a 0. Infatti, `CURRENT_PIXEL_VALUE` sarà sempre uguale a `MIN_PIXEL_VALUE` e di conseguenza `TEMP_PIXEL` assumerà valore 0, qualsiasi sia il valore di `SHIFT_LEVEL`, il quale comunque sarà uguale a 8 essendo il `DELTA_VALUE` uguale a 0 (`MAX_PIXEL_VALUE` e `MIN_PIXEL_VALUE` coincidono).

Esempio 1 – Immagine con valori casuali

Indirizzo	Valore	
0	2	//byte numero colonne
1	3	//byte numero righe
2	76	//primo byte immagine originale
3	131	
4	109	
5	89	
6	46	
7	121	//ultimo byte immagine originale
8	120	//primo byte immagine equalizzata
9	255	
10	252	
11	172	
12	0	
13	124	//ultimo byte immagine equalizzata

Esempio 2 – Immagine con un pixel uguale a 0 e uno a 255

Indirizzo	Valore	
0	2	//byte numero colonne
1	3	//byte numero righe
2	30	//primo byte immagine originale
3	25	
4	0	
5	130	
6	255	
7	197	//ultimo byte immagine originale
8	30	//primo byte immagine equalizzata
9	25	
10	0	
11	130	
12	255	
13	197	//ultimo byte immagine equalizzata

Esempio 3 – Immagine con tutti i pixel uguali

Indirizzo	Valore	
0	2	//byte numero colonne
1	2	//byte numero righe
2	13	//primo byte immagine originale
3	13	
4	13	
5	13	//ultimo byte immagine originale
6	0	//primo byte immagine equalizzata
7	0	
8	0	
9	0	//ultimo byte immagine equalizzata

1.2 Interfaccia del componente

Il componente ha 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;
```

In particolare:

● `i_clk` è il segnale di CLOCK in ingresso, generato dal Test Bench;

● `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 comunica 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 (=1) per poterci scrivere. Per leggere da memoria esso deve essere 0;

● `o_data` è il segnale (vettore) di uscita dal componente verso la memoria.

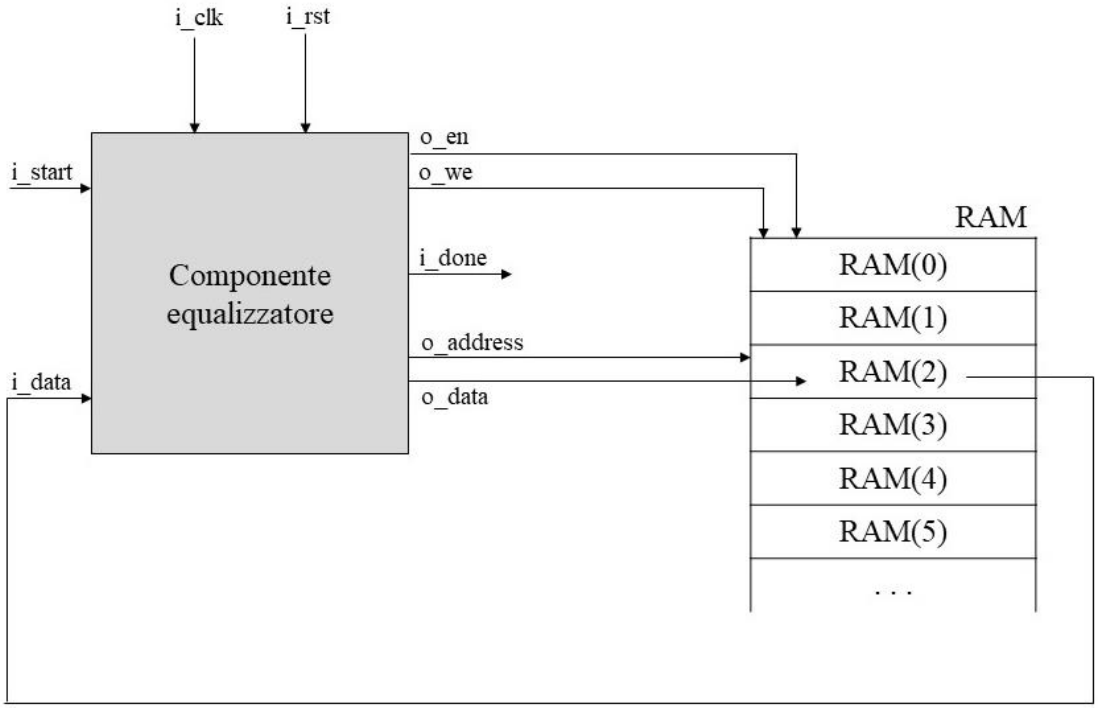
1.3 Memoria

La RAM con cui si interfaccia il componente è composta da 65536 (2^{16}) celle, ognuna contente un byte (8 bit). Da specifica, l'immagine da equalizzare può avere dimensioni massime pari a 128x128, segue che il componente accederà solo agli indirizzi compresi tra l'indirizzo 0 e l'indirizzo 32769 (al massimo), mentre non verrà mai richiesto l'accesso a nessuno degli indirizzi successivi.

La RAM fa parte dei Test Bench e presenta la seguente interfaccia:

```
entity rams_sp_wf is
     port(
          clk  : in std_logic;
          we   : in std_logic;
          en   : in std_logic;
          addr : in std_logic_vector(15 downto 0);
          di   : in std_logic_vector(7 downto 0);
          do   : out std_logic_vector(7 downto 0)
     );
end rams_sp_wf;
```

I segnali in ingresso `we`, `en`, `addr` e `di` corrispondono rispettivamente ai segnali di output `o_we`, `o_en`, `o_addr` e `o_data` del componente equalizzatore. Il segnale di output `do` corrisponde invece al segnale di input `i_data` del componente equalizzatore.



2 Scelte progettuali

Per la realizzazione del componente sono state pensate due possibili soluzioni:

1. Memorizzare l'immagine originale durante la sua lettura in un array di std_logic_vector da 8 bit, in modo da leggere l'immagine originale una sola volta, equalizzarla sovrascrivendo i valori dell'array e salvare i valori modificati nella memoria all'indirizzo corretto; l'array deve essere di dimensioni 128x128 per poter memorizzare anche l'immagine più grande che può essere data in input;
2. Leggere l'immagine originale, calcolare i parametri necessari all'equalizzazione e per ogni pixel leggere il valore originale, equalizzarlo e scriverlo nel corrispettivo indirizzo del pixel equalizzato.

Nel primo caso viene minimizzato il tempo di esecuzione dell'equalizzazione, tuttavia l'area occupata dal componente risulta essere considerevole. Nel secondo caso, invece, non è necessario memorizzare l'immagine, quindi si riduce notevolmente l'area richiesta, ma il tempo di esecuzione aumenta dovendo leggere l'immagine originale due volte.

Si è optato per la seconda soluzione soprattutto perché la presenza di un array di dimensioni 128x128 rende molto onerosa la sintesi.

Il componente è stato realizzato con un unico modulo contenente due processi:

- Processo "*standard*": permette di aggiornare i valori memorizzati e di resettare il componente nel caso di reset;
- Processo "*activity*": esegue l'effettiva equalizzazione.

Sono stati definiti diversi segnali ausiliari, come per il massimo e per il minimo, per memorizzare valori importanti per l'esecuzione. Inoltre, sono stati utilizzati due segnali ausiliari 'letto0' e 'letto255' che notificano il riscontro rispettivamente di un pixel con valore 0 e uno con valore 255. Ciò permette di saltare la fase di equalizzazione dei pixel per scrivere direttamente in memoria i pixel originali per l'immagine equalizzata, quindi di evitare calcoli inutili come quello dello shift_level.

Infine, è stata definita una macchina a stati composta da 6 stati (OFF, READ, CALCOLO, SHIFT, WRITE, DONE), di seguito descritti.

Stato OFF

È lo stato iniziale, in cui si aspetta che il segnale `i_start` venga impostato a '1', nel qual caso si imposta il prossimo valore di `o_en` a '1' e si passa allo stato READ.

Stato READ

Nello stato READ vengono letti inizialmente il valore di colonne e righe, rispettivamente nel primo e nel secondo indirizzo, dopodiché si passa a leggere il valore dei pixel per trovare massimo e minimo. Si resta in questo stato fino a quando non vengono letti tutti i pixel, oppure non vengono letti uno 0 e un 255, oppure ancora se righe o colonne sono nulle: nel primo caso si passa allo stato CALCOLO per poter calcolare lo `shift_level`, nel secondo si passa allo stato SHIFT, nel terzo si passa direttamente allo stato DONE.

Stato CALCOLO

In questo stato viene calcolato il valore dello `shift_level`. Lo stato successivo è sempre quello di SHIFT.

Stato SHIFT

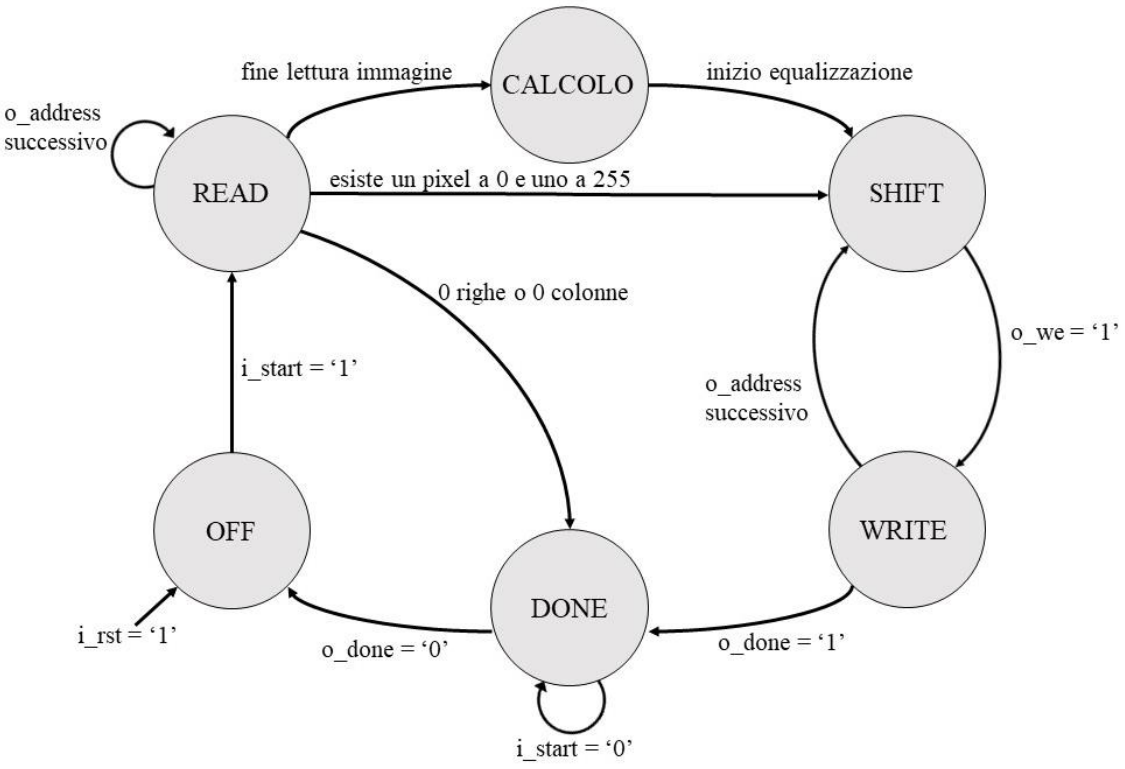
Viene calcolato il nuovo valore del pixel. Viene impostato il prossimo valore di `o_we` a '1' in modo da permettere la scrittura nello stato successivo WRITE.

Stato WRITE

Nello stato WRITE viene memorizzato il valore del nuovo pixel. Se è stata memorizzata l'intera immagine equalizzata, allora viene impostato il prossimo valore di `o_done` a '1' e si passa allo stato DONE; altrimenti sia aggiorna il prossimo indirizzo, si disabilita `o_we` e si passa allo stato SHIFT per equalizzare il pixel successivo.

Stato DONE

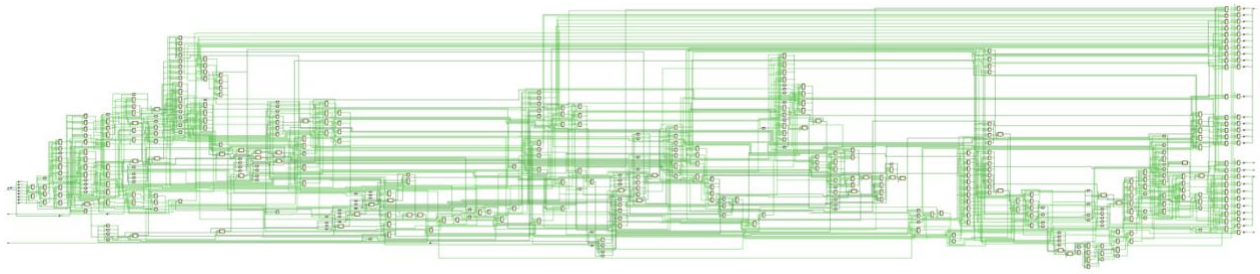
In questo stato si attende che il segnale `i_start` in input venga impostato a '0' in modo da porre conseguentemente a '0' il segnale di `o_done`. Fatto ciò, si ritorna allo stato di OFF.



Si fa notare che quando il segnale di reset viene impostato alto il componente tornerà sempre nello stato iniziale OFF, indipendentemente dallo stato in cui ci si trova.

3 Risultati sintesi

Date le scelte progettuali compiute, il componente risultante presenta dei tempi di esecuzione fortemente dipendenti dall'input, nonostante ciò, l'occupazione di area non è eccessiva come la soluzione alternativa che era stata considerata e descritta precedentemente. Di seguito viene riportata la rete logica sintetizzata.



3.1 Report di sintesi

Utilizzando la FPGA suggerita (xc7a200tfbg484-1), abbiamo ottenuto i seguenti risultati dall'analisi di utilizzo effettuata da Vivado dopo la sintesi.

Slice Logic:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	288	0	134600	0.21
LUT as Logic	288	0	134600	0.21
LUT as Memory	0	0	46200	0.00
Slice Registers	84	0	269200	0.03
Register as Flip Flop	84	0	269200	0.03
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	38
3	LUT1	17
4	LUT2	74
5	LUT3	22
6	LUT4	84
7	LUT5	43
8	LUT6	101
9	FDRE	76
10	FDSE	8
11	IBUF	11
12	OBUF	27

Detailed RTL Component Info :

+---Adders :

2 Input	17 Bit	Adders := 1
2 Input	16 Bit	Adders := 1
3 Input	16 Bit	Adders := 1
3 Input	8 Bit	Adders := 2

+---Registers :

31 Bit	Registers := 1
16 Bit	Registers := 2
8 Bit	Registers := 7
3 Bit	Registers := 1
1 Bit	Registers := 5

+---Muxes :

2 Input	16 Bit	Muxes := 6
7 Input	8 Bit	Muxes := 3
2 Input	8 Bit	Muxes := 2
2 Input	3 Bit	Muxes := 3
4 Input	3 Bit	Muxes := 1
7 Input	1 Bit	Muxes := 9
2 Input	1 Bit	Muxes := 12
3 Input	1 Bit	Muxes := 1
9 Input	1 Bit	Muxes := 1

Report Instance Areas:

	Instance	Module	Cells
1	top		502

I registri sintetizzati sono stati realizzati tramite l'utilizzo di 84 Flip-Flop di tipo D; essi includono:

- Registro da 31 bit: shift_level;
- Registri da 16 bit: indirizzo corrente, indirizzo in output;
- Registri da 8 bit: righe, colonne, massimo, minimo, dato in output;
- Registro da 3 bit: stato;
- Registri da 1 bit: segnale di enable, segnale di write enable, segnale di done, segnale per la lettura del valore 0, segnale per la lettura del valore 255.

Infine, si riporta il report di timing che permette di analizzare quale sia il percorso critico:

```
Timing Report

Slack:                    inf
  Source:                 curr_colonne_reg[3]/C
                            (rising edge-triggered cell FDRE)
  Destination:            curr_addr_reg[10]/D
  Path Group:             (none)
  Path Type:              Max at Slow Process Corner
  Data Path Delay:        9.492ns  (logic 4.033ns (42.488%)  route 5.459ns (57.512%))
  Logic Levels:           12  (CARRY4=5 FDRE=1 LUT2=1 LUT4=2 LUT5=1 LUT6=2)

    Location             Delay type                Incr(ns)  Path(ns)    Netlist Resource(s)
  -------------------------------------------------------------------    -------------------
                         FDRE                         0.000     0.000 r  curr_colonne_reg[3]/C
                         FDRE (Prop_fdre_C_Q)         0.459     0.459 r  curr_colonne_reg[3]/Q
                         net (fo=16, unplaced)        1.019     1.478    curr_colonne_reg_n_0_[3]
                         LUT6 (Prop_lut6_I0_O)        0.295     1.773 r  curr_addr[0]_i_30/O
                         net (fo=1, unplaced)         0.473     2.246    curr_addr[0]_i_30_n_0
                         CARRY4 (Prop_carry4_DI[3]_CO[3])
                                                      0.385     2.631 r  curr_addr_reg[0]_i_27/CO[3]
                         net (fo=1, unplaced)         0.009     2.640    curr_addr_reg[0]_i_27_n_0
                         CARRY4 (Prop_carry4_CI_O[1])
                                                      0.348     2.988 r  curr_addr_reg[11]_i_45/O[1]
                         net (fo=2, unplaced)         0.622     3.610    curr_addr_reg[11]_i_45_n_6
                         LUT4 (Prop_lut4_I3_O)        0.332     3.942 r  curr_addr[11]_i_44/O
                         net (fo=2, unplaced)         0.460     4.402    curr_addr[11]_i_44_n_0
                         LUT6 (Prop_lut6_I3_O)        0.124     4.526 r  curr_addr[11]_i_30/O
                         net (fo=2, unplaced)         0.650     5.176    curr_addr[11]_i_30_n_0
                         CARRY4 (Prop_carry4_DI[1]_CO[3])
                                                      0.507     5.683 r  curr_addr_reg[11]_i_26/CO[3]
                         net (fo=1, unplaced)         0.000     5.683    curr_addr_reg[11]_i_26_n_0
                         CARRY4 (Prop_carry4_CI_O[3])
                                                      0.329     6.012 r  curr_addr_reg[15]_i_19/O[3]
                         net (fo=5, unplaced)         0.646     6.658    curr_addr_reg[15]_i_19_n_4
                         LUT2 (Prop_lut2_I0_O)        0.307     6.965 r  curr_addr[0]_i_6/O
                         net (fo=1, unplaced)         0.000     6.965    curr_addr[0]_i_6_n_0
                         CARRY4 (Prop_carry4_S[1]_CO[1])
                                                      0.491     7.456 r  curr_addr_reg[0]_i_2/CO[1]
                         net (fo=17, unplaced)        0.667     8.123    curr_addr_reg[0]_i_2_n_2
                         LUT4 (Prop_lut4_I2_O)        0.332     8.455 r  curr_addr[10]_i_2/O
                         net (fo=2, unplaced)         0.913     9.368    curr_addr[10]_i_2_n_0
                         LUT5 (Prop_lut5_I0_O)        0.124     9.492 r  curr_addr[10]_i_1/O
                         net (fo=1, unplaced)         0.000     9.492    curr_addr[10]_i_1_n_0
                         FDRE                                       r  curr_addr_reg[10]/D
  -------------------------------------------------------------------    -------------------
```

Con la FPGA suggerita il percorso critico necessita di 9,492 ns, di cui meno della metà è effettivamente utilizzato per la logica del componente.

4 Test del componente

Al fine di verificare il corretto funzionamento del componente, sono stati effettuati numerosi test ottenuti grazie ad un generatore da noi realizzato in linguaggio di programmazione C. Tale strumento è stato sviluppato anche in considerazione del fatto che sarebbe stato proibitivo scrivere test per immagini con un numero di pixel pari a 128x128.

Come da specifica tutti i test sono stati condotti con un periodo di clock di 100 ns, tuttavia si fa notare che il componente funziona correttamente anche per periodi più brevi. In particolare, il limite inferiore risulta essere 4,1 ns a causa del ritardo dovuto all'interazione con la memoria (impostato a 2 ns, come da documentazione della *"Single-Port Block RAM Write-First Model"* utilizzata). Più in generale, per l'interazione con una memoria del tipo fornito, il periodo di clock per il funzionamento corretto del componente deve essere maggiore del doppio del ritardo di interazione con la memoria.

I test a cui è stato sottoposto il componente sono stati pensati sia in base alla specifica fornita che all'implementazione particolare che abbiamo realizzato.

Per iniziare, è stata verificata la possibilità di eseguire l'equalizzazione successiva di più immagini senza la necessità di fornire alcun segnale di reset intermedio. Tali immagini presentavano dimensioni tra loro diverse: sia per numero di righe/colonne che per area di pixel. Questi parametri, in immagini successive, erano talvolta più grandi, talvolta più piccoli. In questo modo è stato possibile controllare che il variare delle dimensioni delle immagini equalizzate successivamente non influenzasse in qualche modo il corretto funzionamento.

Al componente sono poi state sottoposte immagini di tutte le possibili forme: quadrate, rettangolari (con più righe che colonne e viceversa) e vettori riga/colonna. Anche in questo caso, come ci aspettavamo, il componente ha reagito positivamente equalizzando in modo corretto tutti questi tipi d'immagine, anche i casi limite in cui una o entrambe le dimensioni sono 1 e/o 128. Questo risultato è in linea con le nostre aspettative in quanto il componente interagisce con la memoria in modo "lineare", leggendo celle successive, indipendentemente dal numero di righe e colonne dell'immagine (e quindi dalla sua forma).

Abbiamo poi ritenuto importante testare il componente in base ai valori assunti dai pixel. Ci siamo dunque soffermati sull'analisi del comportamento in base alla posizione, sia reciproca che all'interno dell'immagine, dei pixel contenenti il valore massimo e minimo. Sono state testate tutte le combinazioni possibili: prima il massimo e poi il minimo e viceversa, uno all'inizio dell'immagine e l'altro alla fine (inizio e fine: considerando l'immagine sequenzialmente per come scritta in memoria), entrambi come primi o ultimi valori possibili e posizionati in modo casuale internamente all'immagine. In tutti i casi il componente si è rivelato funzionare correttamente.

Data la nostra scelta di gestire a parte il caso in cui in una stessa immagine fossero presenti i due valori estremi che possono assumere i pixel, abbiamo ritenuto d'interesse fare qualche verifica sul funzionamento effettivo del meccanismo implementato. Per fare ciò abbiamo usato immagini con pixel che assumono valore minimo uguale a 0 e massimo inferiore a 255 e altre che avessero come valore massimo 255 e minimo superiore a 0. In entrambi i casi il componente ha funzionato senza errori: questo prova la corretta gestione dei segnali interni che notificano la lettura dei due valori estremi assumibili dai pixel.

Data la scelta strutturale di calcolare direttamente il valore `shift_level` con una cascata di "if", abbiamo ritenuto utile, anche se non strettamente necessario, sollecitare tutti gli intervalli su cui tale

logica si basa (prestando particolare attenzione ai valori estremi per ciascuno di essi). Anche questo controllo ha avuto esito positivo: il componente si è dimostrato funzionare correttamente per qualsiasi valore assunto dai pixel dell'immagine, in qualsiasi combinazione e posizione.

I test descritti fino ad ora sono stati eseguiti sia nel caso in cui al componente venisse fornito un solo reset, quello iniziale, sia nel caso in cui ve ne fossero altri successivi.

Analizzando la specifica abbiamo compreso che si possono verificare solo due tipi di reset: tra l'equalizzazione di un'immagine e quella successiva mentre il componente è in attesa della nuova (vedi caso 1 in *Figura 1*) oppure durante l'equalizzazione di un'immagine con conseguente annullamento dell'esecuzione di tale operazione. In quest'ultimo caso abbiamo ritenuto possibile procedere richiedendo nuovamente l'equalizzazione dell'immagine per cui era stata interrotta, ponendo ad '1' il segnale di start, oppure cambiare immagine e svolgere l'elaborazione di questa.

Abbiamo proceduto testando la reattività al reset durante l'esecuzione sia della fase di lettura (vedi caso 2 in *Figura 1*) dell'immagine sia della fase di scrittura in memoria (vedi caso 3 in *Figura 1*). In entrambi i casi il componente ha reagito in modo adeguato: la correttezza delle esecuzioni successive non ha in alcun modo risentito di tali reset. Lo stesso è stato anche per l'altro tipo di reset. Questi casi sono stati testati sia singolarmente sia in modo combinato, senza che si verificasse alcun errore o comportamento indesiderato.

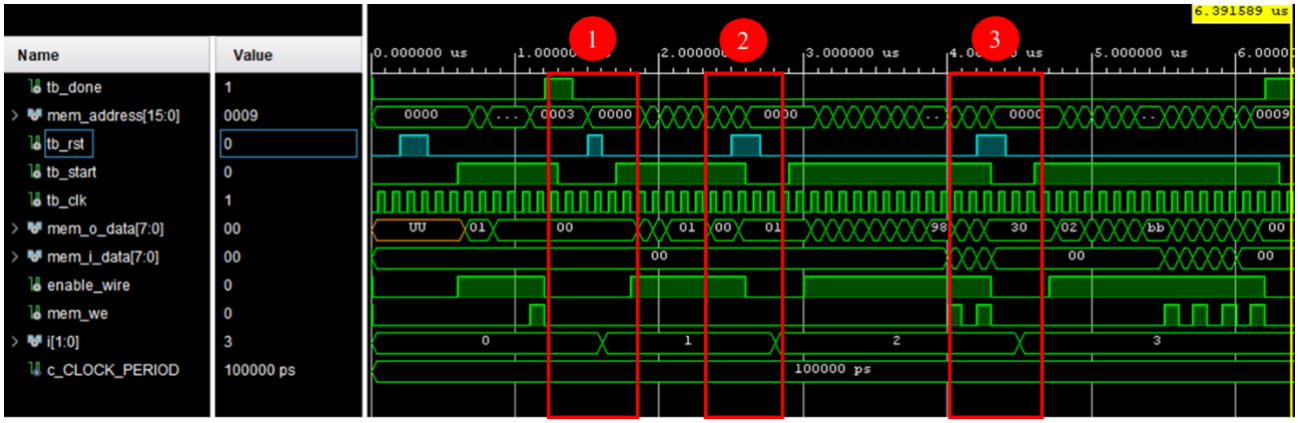


Figura 1 - Vari casi di reset: 1) reset tra equalizzazione di due immagini; 2) reset durante la fase di lettura di un'immagine; 3) reset durante la fase di scrittura di un'immagine.

Alla luce dei risultati ottenuti da tutti questi test, ed altri con un gran numero di immagini successive, possiamo affermare che il componente funziona in modo corretto per tutta la casistica descritta nella specifica.

A questo punto abbiamo voluto eseguire una serie di altri test atti alla validazione del componente come robusto: è stato nostro interesse ispezionare il suo comportamento anche per condizioni che non dovrebbero verificarsi, controllando che questo fosse in qualche modo "ragionevole". Questi test erano stati pensati inizialmente solo per comprendere i limiti di funzionamento del componente ma la positività dei risultati ci ha spinto a voler studiare più ampiamente le sue potenzialità.

Grazie alla dimensione della RAM con cui si interfaccia il componente (descritta precedentemente), abbiamo pensato di testarlo inserendo immagini con dimensioni superiori a 128x128 pixel. Ad esempio, sono state sottoposte al componente immagini con almeno una delle due dimensioni o entrambe pari a 181 pixel. Anche in questo caso il risultato si è dimostrato corretto. Ciò prova che il

limite per le dimensioni delle immagini equalizzabili è stabilito dalla dimensione della memoria (perché devono essere in essa contenibili) e non dal componente in sé.

Studiando il segnale di reset abbiamo compreso che si sarebbero potuti verificare due casi "illeciti", ossia non compatibili con la corretta interazione con il componente; entrambi si possono verificare durante l'equalizzazione dell'immagine. Il primo consiste nel passaggio del segnale di reset da basso ad alto a basso, senza modificare gli altri segnali in ingresso (vedi *Figura 2*). In questo caso il componente si è dimostrato operare correttamente. Inoltre, questo evento determina nel funzionamento un "restart" dell'equalizzazione, senza creare alcuno squilibrio né nell'esecuzione corrente né nelle successive. L'altro caso non differisce dal precedente per il movimento del segnale di reset ma mentre questo è alto viene cambiata l'immagine da equalizzare. Nemmeno questo avvenimento influenza il corretto funzionamento del componente che procede comunque con l'elaborazione della nuova immagine. Questi due tipi illeciti di reset sono stati testati anche in modo combinato con quelli leciti e ciò non ha comportato alcun malfunzionamento: l'esecuzione è sempre andata a buon fine.

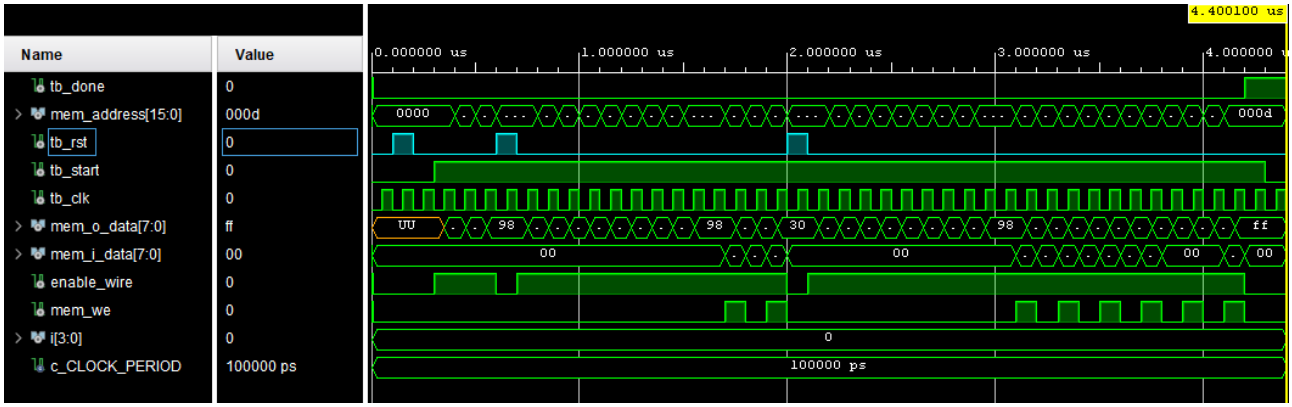


Figura 2 - Cambio del solo segnale di reset, prima durante la fase di lettura e dopo durante la fase di scrittura.

Ci siamo poi domandati se il componente avrebbe comunque funzionato anche nel caso in cui il segnale di start fosse variato durante l'equalizzazione (caso che non dovrebbe verificarsi), così lo abbiamo provato. Data la sensibilità del componente, come ci aspettavamo, anche in quest'ultimo caso non ci sono stati problemi di alcun tipo.

Avendo assunto come comportamento corretto la possibilità di equalizzare una sola volta l'immagine, è stato nostro interesse controllare che fosse possibile ripetere quest'operazione sulla medesima ponendo ad '1' il segnale di start dopo che quello di done viene impostato a '0'. Questa procedura è stata ripetuta in condizioni tra loro diverse: il componente ha comunque continuato a funzionare in modo adeguato.

Abbiamo pensato di testare il componente anche con immagini 0x0. Anche se tale caso non viene presentato nella specifica e non corrisponde ad alcun tipo sensato di immagine, potrebbe comunque presentarsi, ad esempio, se le immagini passate al componente fossero scritte da un programma malfunzionante. Il risultato dei test è stato ancora una volta positivo: non ci sono stati problemi con l'esecuzione.

5 Conclusione

Riassumendo è stato progettato e implementato un componente equalizzatore funzionante sia in pre-sintesi che in post-sintesi, infatti tutti i test descritti sono stati passati sia in Behavioral Simulation che in Post-Synthesis Functional Simulation.

I tempi di elaborazione di una singola immagine sono variabili e strettamente dipendenti dalle sue dimensioni: per equalizzare l'immagine più piccola (1x1), con un clock di 100 ns, sono necessari 600 ns, mentre la più grande (128x128) necessita di circa 3,3 ms.

In conclusione, il componente che abbiamo progettato si è dimostrato essere non solo correttamente funzionante quando usato come da specifica ma anche in una serie di altri casi che potrebbero, per particolari fenomeni fisici o utilizzo scorretto, verificarsi.