Progetto di Reti Logiche

Prof. William Fornaciari

2020 – 2021

Peizhou Chen (Cod. Persona: 10685981 – Matricola: 907212)

Andrea Chizzola (Cod. Persona: 10614212 – Matricola: 910249)

Indice

1. Introduzione *3*

2. Specifiche *3*

2.1 Inizio della Codifica *4*

3. Scelte Progettuali *4*

4. Testing e Risultati Sperimentali *7*

4.1 Test Benches *7*

4.2 Report di Sintesi *10*

5. Conclusioni *11*

1. Introduzione

L'obiettivo del progetto è di realizzare un componente hardware che permetta di equalizzare l'istogramma di un'immagine, al fine di incrementarne il contrasto quando l'intervallo dei valori di intensità risultano particolarmente vicini. Tale modello riceve come input le dimensioni e i pixel di un'immagine e produce in output i pixel dell'immagine equalizzata.

2. Specifiche

L'algoritmo di equalizzazione da utilizzare, applicabile ad immagini in scala di grigi a 256 livelli, prevede che ogni pixel venga trasformato come segue:

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* rappresentano, rispettivamente, il valore massimo e minimo dei pixel dell'immagine mentre *CURRENT_PIXEL_VALUE* è il valore del pixel da trasformare e *NEW_PIXEL_VALUE* è il valore equalizzato del pixel letto.

L'immagine da elaborare dovrà essere letta sequenzialmente da una memoria con indirizzamento al Byte, in cui ad ogni Byte, ad eccezione dei primi due, corrisponde un pixel. Nei primi due indirizzi (0 ed 1) sono invece presenti, rispettivamente, la dimensione di colonna e la dimensione di riga dell'immagine che può avere una dimensione massima di 128 x 128 pixel. Sia le dimensioni dell'immagine che i relativi pixel hanno quindi una dimensione di 8 bit.

Per chiarezza, è presentato di seguito un breve esempio di funzionamento dell'algoritmo applicato al primo pixel di un'immagine 2 x 2 (Fig. 1).

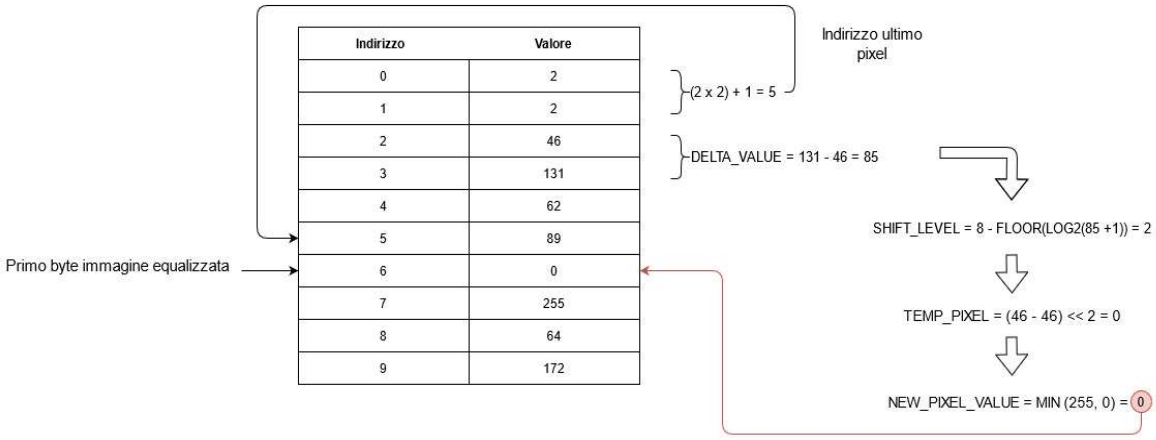

Fig. 1

2.1 Inizio della Codifica

Il modulo da implementare comincerà l'elaborazione una volta che il segnale START in ingresso sarà portato alto. Quest'ultimo rimarrà alto fino a che il segnale di DONE non verrà alzato, una volta scritto il risultato in memoria. Il segnale di DONE dovrà inoltre rimanere alto fino a che il segnale di START non sarà stato abbassato. Solo dopo che anche il segnale di DONE sarà stato abbassato, il segnale di START potrà essere nuovamente alzato, dando così inizio alla codifica di una nuova immagine.

Il modulo è stato quindi progettato affinché, come da specifica, la prima codifica sia sempre preceduta dal reset del modulo, mentre, nel caso di una codifica di più immagini consecutive, non sarà necessario attendere il segnale di reset tra un'elaborazione e quella successiva.

3. Scelte Progettuali

Per affrontare l'implementazione dell'algoritmo sopracitato, è stato deciso di cominciare dalla progettazione e schematizzazione di un Datapath (Fig. 2) per rappresentare il flusso dei dati, da quando vengono letti al momento in cui viene scritto in memoria il risultato dell'equalizzazione applicata a ciascuno dei pixel.

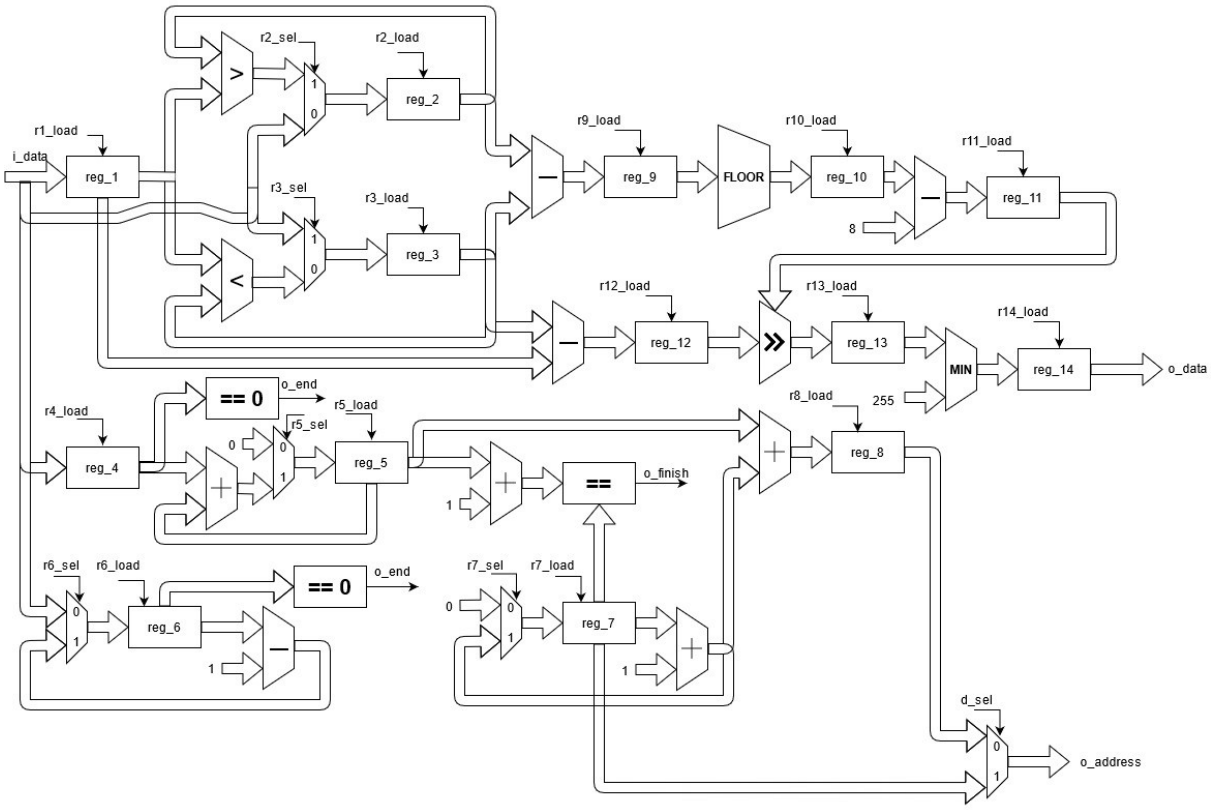

Fig. 2

Al fine di pilotare il Datapath e stabilire, ad ogni ciclo di clock, quali operazioni dovranno essere svolte è stata quindi progettata una macchina a stati finiti (Fig. 3). In particolare, è stato scelto di utilizzare una macchina di Moore in cui però le uscite, che in questo caso specificano il valore assunto dai diversi segnali del Datapath, sono riportate al di sotto di ciascuno stato e non al loro interno.

Per raggiungere tale obiettivo, all'interno dell'architettura della FSM, è stato quindi istanziato un componente che si occupa della corretta implementazione del Datapath.

Il modulo è stato implementato per svolgere una duplice scansione della memoria, prima di scrivervi il risultato finale. La prima scansione inizia con la lettura dei primi due valori in memoria corrispondenti alla dimensione di colonna e di riga dell'immagine, che sono inseriti rispettivamente all'interno del registro 4 e del registro 6 (stati S3 – S5). Prima di continuare con la scansione della memoria, viene quindi eseguita la moltiplicazione tra i due valori precedentemente letti, realizzata evitando l'utilizzo di un moltiplicatore. Viene infatti sommato il valore contenuto nel registro 4 a se stesso per un numero di volte pari al numero iterazioni necessarie per portare il valore presente nel registro 6, inizialmente decrementato di 1 prima della prima somma e poi decrementato ad ogni ciclo di clock, a 0, attivando così il segnale o _end (stati S6 – S7).

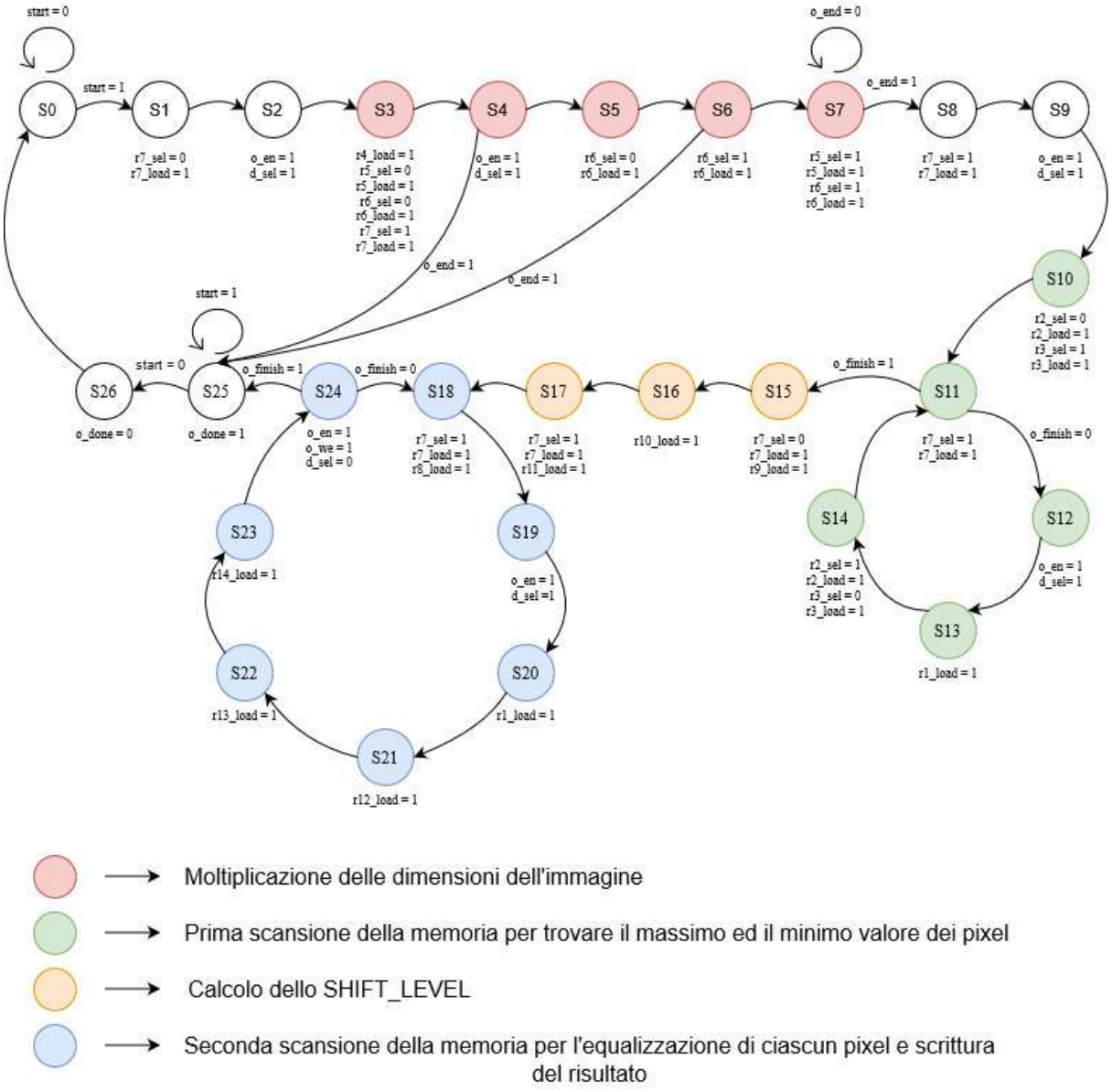

Fig. 3

Il risultato così trovato, corrispondente all'indirizzo in cui si trova l'ultimo pixel da equalizzare, verrà quindi memorizzato all'interno del registro 5. Questa prima fase potrebbe tuttavia terminare prima se uno dei valori moltiplicati fosse pari a 0, ed in tal caso, come mostrato in Fig. 3, dagli stati S4 o S6 verrebbe alzato il segnale di o_end. Trattandosi infatti di un'immagine "vuota", la procedura terminerebbe anticipatamente non essendoci alcun pixel da equalizzare.

La prima scansione può così continuare alla ricerca del valore massimo e di quello minimo tra i pixel dell'immagine, i quali verranno memorizzati, rispettivamente, all'interno del registro 2 e nel registro 3. Per capire quando fermare tale ricerca viene quindi utilizzato il valore della moltiplicazione precedentemente calcolato, in modo che il registro 7, un contatore che si occupa di generare gli indirizzi successivi da cui

leggere la memoria, possa capire quando l'immagine è terminata ed alzare il segnale o_finish (stati S10 – S11).

Conclusa quindi la prima scansione, il registro 7 verrà azzerato prima di poter partire con la seconda scansione e i valori presenti nei registri 2 e 3 saranno usati per calcolare il valore dello SHIFT_LEVEL, che sarà posto all'interno del registro 11. Per il calcolo di questo valore, come specificato nell'algoritmo, è necessaria anche l'applicazione dell'operatore FLOOR, implementato tramite opportuni controlli a soglia.

Una volta incrementato di due unità il valore del registro 7 appena azzerato, in modo che al suo interno sia presente l'indirizzo del primo pixel, può partire la seconda scansione della memoria (stati S18 – S24), con lo scopo di equalizzare ciascun pixel dell'immagine. Al pixel corrente, memorizzato nel registro 1, sarà quindi sottratto il valore del pixel minimo precedentemente calcolato ed il risultato di tale differenza, opportunamente shiftato di SHIFT_LEVEL, se minore di 255 sarà memorizzato all'interno del registro 14. Prima di poter essere scritto in memoria, è però necessario che il registro 8 calcoli l'indirizzo di memoria nel quale scrivere il nuovo valore equalizzato. Questo indirizzo è calcolato sommando al valore appena prodotto dal registro 7 quello corrispondente all'indirizzo dell'ultimo pixel dell'immagine, ancora presente nel registro 5. Tramite quindi l'utilizzo di un multiplexer, il segnale d_sel, che finora è sempre rimasto alto, sarà abbassato per portare in uscita il valore dell'indirizzo di scrittura.

Una volta rialzato il segnale o_finish, terminerà così la computazione. Gli stati S25 ed S26 si occupano quindi di preparare il modulo per l'eventuale inizio di una nuova codifica, prima di tornare allo stato iniziale.

4. Testing e Risultati Sperimentali

4.1 Test Benches

Sono riportati di seguito i risultati di alcuni dei test bench realizzati per testare casi limite e di particolare interesse ai fini del buon funzionamento del modulo progettato:

Immagine 1x1:

Testa il corretto funzionamento del modulo quando l'immagine risulta composta da un singolo pixel, verificando anche la corretta gestione del caso in cui il valore massimo e minimo dei pixel coincidano.

- behavioral:

```
Failure:  Simulation Ended! TEST PASSATO
Time: 782500 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB5/1_pixel.vhd
$finish called at time : 782500 ps : File "C:/Users/chenp/OneDrive/Desktop/TB5/1_pixel.vhd" Line 110
xsim: Time (s): cpu = 00:00:22 ; elapsed = 00:00:15 . Memory (MB): peak = 1004.570 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:00:30 ; elapsed = 00:00:49 . Memory (MB): peak = 1004.570 ; gain = 0.000
```

- post synthesis:

```
Failure:  Simulation Ended! TEST PASSATO
Time: 782600 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB5/1_pixel.vhd
$finish called at time : 782600 ps : File "C:/Users/chenp/OneDrive/Desktop/TB5/1_pixel.vhd" Line 110
xsim: Time (s): cpu = 00:00:24 ; elapsed = 00:00:16 . Memory (MB): peak = 1899.512 ; gain = 42.656
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_func_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:02:11 ; elapsed = 00:02:16 . Memory (MB): peak = 1899.512 ; gain = 894.941
```

Immagine 128x128:

Testa il corretto funzionamento del modulo nel caso in cui l'immagine da equalizzare abbia la dimensione massima consentita.

- behavioral:

```
Failure: Simulation Ended! TEST PASSATO
Time: 2705882500 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB5/Test_128_128.vhd
$finish called at time : 2705882500 ps : File "C:/Users/chenp/OneDrive/Desktop/TB5/Test_128_128.vhd" Line 32874
run: Time (s): cpu = 00:00:15 ; elapsed = 00:00:08 . Memory (MB): peak = 1017.480 ; gain = 0.000
xsim: Time (s): cpu = 00:00:37 ; elapsed = 00:00:21 . Memory (MB): peak = 1017.480 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:01:17 ; elapsed = 00:07:05 . Memory (MB): peak = 1017.480 ; gain = 0.000
```

- post synthesis:

```
Failure: Simulation Ended! TEST PASSATO
Time: 2705882600 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB5/Test_128_128.vhd
$finish called at time : 2705882600 ps : File "C:/Users/chenp/OneDrive/Desktop/TB5/Test_128_128.vhd" Line 32874
run: Time (s): cpu = 00:00:07 ; elapsed = 00:00:51 . Memory (MB): peak = 1903.125 ; gain = 9.543
xsim: Time (s): cpu = 00:00:31 ; elapsed = 00:01:04 . Memory (MB): peak = 1903.125 ; gain = 27.570
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_func_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:02:50 ; elapsed = 00:09:59 . Memory (MB): peak = 1903.125 ; gain = 885.645
```

Immagine 0 pixel:

Testa il corretto funzionamento del modulo nel caso in cui l'immagine sia "vuota". In particolare, viene verificata la corretta attivazione del segnale o_end che permette di evitare l'inizio della prima scansione dell'immagine in quanto priva di pixel da equalizzare.

- behavioral:

```
Failure: Simulation Ended! TEST PASSATO
Time: 5875 ns  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB2/ZeroPixel.vhd
$finish called at time : 5875 ns : File "C:/Users/chenp/OneDrive/Desktop/TB2/ZeroPixel.vhd" Line 107
xsim: Time (s): cpu = 00:00:21 ; elapsed = 00:00:13 . Memory (MB): peak = 1005.059 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:00:33 ; elapsed = 00:00:43 . Memory (MB): peak = 1005.059 ; gain = 0.000
```

- post synthesis:

```
Failure: Simulation Ended! TEST PASSATO
Time: 5875100 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB2/ZeroPixel.vhd
$finish called at time : 5875100 ps : File "C:/Users/chenp/OneDrive/Desktop/TB2/ZeroPixel.vhd" Line 107
xsim: Time (s): cpu = 00:00:14 ; elapsed = 00:00:08 . Memory (MB): peak = 1899.320 ; gain = 42.520
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_func_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:00:35 ; elapsed = 00:01:00 . Memory (MB): peak = 1899.320 ; gain = 426.539
```

Immagini consecutive:

Testa il corretto funzionamento del modulo nel caso di più esecuzioni consecutive. Viene verificato che il modulo non abbia bisogno di attendere il segnale di reset tra un'elaborazione e quella successiva, rispettando quindi protocollo di inizio della codifica discusso precedentemente.

-behavioral:

```
Failure: Simulation Ended! TEST PASSATO
Time: 1157072500 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB4/threeConsecutiveRunsAndResetOnFirstRun
$finish called at time : 1157072500 ps : File "C:/Users/chenp/OneDrive/Desktop/TB4/threeConsecutiveRunsAndResetOnFirstRun.vhd" Line 14130
run: Time (s): cpu = 00:00:15 ; elapsed = 00:00:08 . Memory (MB): peak = 1015.152 ; gain = 0.000
xsim: Time (s): cpu = 00:00:45 ; elapsed = 00:00:27 . Memory (MB): peak = 1015.152 ; gain = 0.000
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:00:56 ; elapsed = 00:00:53 . Memory (MB): peak = 1015.152 ; gain = 0.000
```

-post synthesis:

```
Failure: Simulation Ended! TEST PASSATO
Time: 1157072600 ps  Iteration: 0  Process: /project_tb/test  File: C:/Users/chenp/OneDrive/Desktop/TB4/threeConsecutiveRunsAndResetOnFirstRun.
$finish called at time : 1157072600 ps : File "C:/Users/chenp/OneDrive/Desktop/TB4/threeConsecutiveRunsAndResetOnFirstRun.vhd" Line 14130
run: Time (s): cpu = 00:00:12 ; elapsed = 00:00:44 . Memory (MB): peak = 1822.324 ; gain = 2.820
xsim: Time (s): cpu = 00:00:35 ; elapsed = 00:01:00 . Memory (MB): peak = 1822.504 ; gain = 12.496
INFO: [USF-XSim-96] XSim completed. Design snapshot 'project_tb_func_synth' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 10000000ns
launch_simulation: Time (s): cpu = 00:01:38 ; elapsed = 00:06:39 . Memory (MB): peak = 1822.504 ; gain = 484.254
```

4.2 Report di Sintesi

Timing Report

La seguente immagine mostra i risultati temporali dell'implementazione. In particolare, il ritardo di propagazione (Data Path Delay) risulta pari a 4,590 ns, un valore sufficientemente inferiore ai 100 ns richiesti dalla specifica.

```
Timing Report

Slack (MET) :             95.028ns  (required time - arrival time)
  Source:                 FSM_onehot_cur_state_reg[3]/C
                            (rising edge-triggered cell FDCE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Destination:            FSM_onehot_cur_state_reg[0]/CE
                            (rising edge-triggered cell FDPE clocked by clock  {rise@0.000ns fall@5.000ns period=100.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            100.000ns  (clock rise@100.000ns - clock rise@0.000ns)
  Data Path Delay:        4.590ns  (logic 1.247ns (27.168%)  route 3.343ns (72.832%))
```

Utilization Report

Dal punto di vista dell'area, la sintesi riporta il seguente utilizzo dei componenti:

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  197 |     0 |    134600 |  0.15 |
|   LUT as Logic          |  197 |     0 |    134600 |  0.15 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |  164 |     0 |    269200 |  0.06 |
|   Register as Flip Flop |  164 |     0 |    269200 |  0.06 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

È stata posta particolare attenzione, durante la scrittura del codice, al fine di evitare la presenza di Latch.

5. Conclusioni

Nella realizzazione del modulo, sono state compiute alcune scelte di ottimizzazione al fine di ridurne il più possibile il cammino critico ed aumentarne l'efficienza.

In particolare, è stato verificato che il componente funzioni correttamente anche per valori molto inferiori ai 100ns pari alla frequenza del clock, garantendo di conseguenza un migliore margine di sicurezza.

Inoltre, è stato scelto di sfruttare, durante la seconda scansione della memoria, l'indirizzo di lettura generato dal registro 7 per calcolare allo stesso tempo anche l'indirizzo di scrittura del pixel equalizzato, evitando quindi di dover aspettare un ulteriore ciclo di clock. Per distinguere l'indirizzo corretto, di lettura o di scrittura, è stato quindi utilizzato un apposito multiplexer.

Infine, considerando l'onerosità dell'operazione di moltiplicazione, è stato ritenuto più opportuno implementarla, in occasione del calcolo dell'indirizzo dell'ultimo pixel dell'immagine, tramite l'uso di somme successive regolate da un contatore.