Prova Finale di Reti Logiche

Giovanni Demasi (Codice persona 10656704 - Matricola 906817)

Prof. William Fornaciari - A.A. 2020/2021

Contents

1 Introduzione

1.1 Scopo del progetto

Sia data la dimensione, in termini di colonne e righe, di una immagine e il valore di ogni singolo pixel che la compone: lo scopo del progetto è di implementare un componente hardware, descritto in VHDL, che effettui una equalizzazione dell'immagine secondo l'algoritmo semplificato riportato di seguito.

$$
\begin{aligned}
deltaValue &= maxPixelValue - minPixelValue \\
shiftLevel &= 8 - \lfloor(\log_2(deltaValue + 1))\rfloor \\
tempPixel &= (currentPixelValue - minPixelValue) << shift \\
newPixel &= min(255, tempPixel)
\end{aligned}
$$

1.2 Specifiche generali

I valori di colonna e di riga sono collocati, rispettivamente, in posizione zero e uno della memoria RAM. Il valore limite di questi due parametri è 128, dunque l'immagine in ingresso ha dimensione massima pari a $128 * 128$ pixel.

Sia $nCol$ il numero di colonne e $nRow$ il numero di righe dell'immagine. Dalla posizione numero 2 alla posizione numero $nCol * nRow + 1$ sono presenti, ognuno in una posizione, i valori dei singoli pixel che costituiscono l'immagine.

Di seguito troviamo una rappresentazione della memoria RAM contenente i valori che rappresentano una immagine di dimensione 4 prima del processo di equalizzazione.

2	2	46	131	62	89
0	1	2	3	4	5

Figure 1: Contenuto della memoria prima del processo di equalizzazione

Dopo aver equalizzato l'immagine, i pixel risultanti, ovvero quelli relativi all'immagine equalizzata, si troveranno a partire dalla posizione $nCol * nRow + 2$ fino alla posizione $2 * nCol * nRow + 1$. Troviamo qui di seguito il risultato dell'immagine sopra rappresentata.

2	2	46	131	62	89	0	255	64	172
0	1	2	3	4	5	6	7	8	9

Figure 2: Contenuto della memoria dopo il processo di equalizzazione

1.3 Esempio algoritmo

Illustriamo di seguito, in riferimento all'esempio precedente, un'applicazione dell'algoritmo di equalizzazione del singolo pixel.

Si supponga di aver trovato i valori di massimo e minimo pari a 131 e 46, rispettivamente. Possiamo dunque trovare il delta value, come differenza di quest'ultimi, che sarà pari a 85. Il logaritmo base due arrotondato per difetto di questo valore, più uno, è 6; si trova quindi uno shift value di 2. Prendiamo in esame il pixel avente, ad esempio, valore 89. Segue la sua rappresentazione in binario, con bit meno significativo a destra, in posizione 0.

0	1	0	1	1	0	0	1
7	6	5	4	3	2	1	0

Figure 3: Rappresentazione in binario del numero 89

A questo valore dobbiamo sottrarre il numero 46, ovvero il pixel minimo dell'immagine da equalizzare. Troviamo a questo punto il valore 43, cifra intermedia che ci consentirà di calcolare il temp Pixel.

0	0	1	0	1	0	1	1
7	6	5	4	3	2	1	0

Figure 4: Rappresentazione in binario del numero 43

Dopodiché dovremo shiftare a sinistra di due posizioni il numero binario sopra rappresentato, arrivando al seguente risultato.

0	0	1	0	1	0	1	1	0	0
9	8	7	6	5	4	3	2	1	0

Figure 5: Rappresentazione in binario del numero 172

Essendo questo valore minore di 255 siamo in grado, utilizzando l'ultima formula dell'algoritmo, di affermare che il valore che verrà scritto in output è il numero 172. Viceversa, se il risultato fosse stato maggiore o uguale a 255 avremmo scritto in memoria il valore 255.

1.4 Interfaccia del componente

Il componente da descrivere ha un'interfaccia così definita:

```
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 Testbench;
- `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 Testbench;
- `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 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` 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 memoria;

1.5 Dati e descrizione memoria

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

- L'indirizzo 0 è usato per memorizzare il numero di colonne dell'immagine, sia esso $nCol$;

- L'indirizzo 1 è usato per memorizzare il numero di righe dell'immagine, sia esso $nRow$;
- Gli indirizzi dal 2 al $nCol * nRow + 1$ sono usati per memorizzare i valori dei pixel dell'immagine originale (da un minimo di 0 a un massimo di 255);
- Gli indirizzi dal $nCol * nRow + 2$ al $2 * nCol * nRow + 1$ sono usati per scrivere il valore di uscita dei pixel dell'immagine equalizzata.

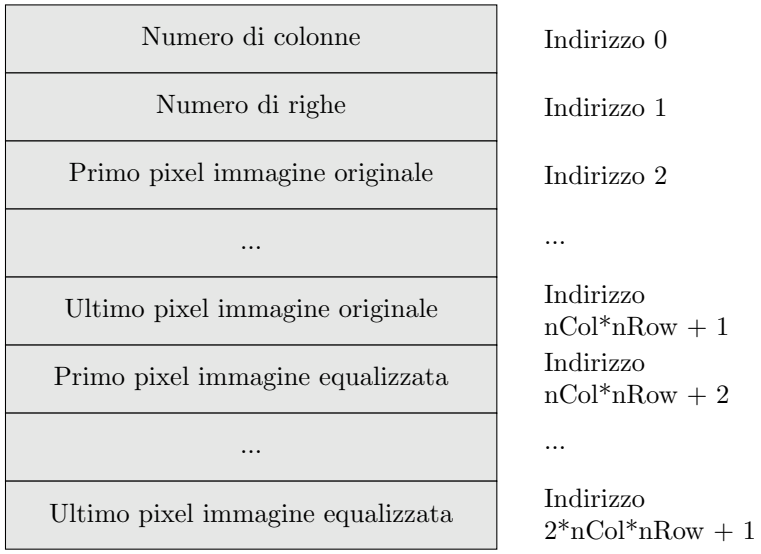

Figure 6: Rappresentazione indirizzi significativi della memoria

2 Design

Quando il segnale `i_start` in ingresso viene portato a 1, il componente sviluppato inizia l'elaborazione spostandosi dallo stato `IDLE` al primo stato della computazione. Una volta terminata la computazione, dopo aver scritto i risultati in memoria, il componente pone a 1 il segnale `o_done`. Il Testbench risponde portando a 0 `i_start` e, successivamente, il componente riporta a 0 `o_done` e ritorna nello stato `IDLE`, in attesa che il segnale `i_start` ritorni a 1.

Il componente dispone inoltre di un segnale `i_rst` che, insieme agli altri appena elencati, hanno portato alla definizione di una FSM, macchina a stati finiti con data path, che combina una normale FSM con tipici circuiti sequenziali.

Di seguito vi è sia la descrizione della FSM sia la descrizione della parte sequenziale della macchina, la quale permette di gestire i registri utilizzati.

2.1 Stati della macchina

La macchina costruita è composta da 14 stati. Di seguito è fornita una breve descrizione per ciascuno di questi.

2.1.1 `IDLE` state

Stato iniziale della macchina in cui si attende il segnale di `i_start`. Nel caso, in qualsiasi momento dell'elaborazione, venga alzato il segnale `i_rst` si torna in questo stato.

2.1.2 `GET_COL` state

Stato in cui viene richiesto il numero di colonne dell'immagine da equalizzare.

2.1.3 `GET_ROW` state

Stato in cui viene richiesto il numero di righe dell'immagine da equalizzare.

2.1.4 `COMP_DIM` state

Stato in cui viene calcolata, mediante un loop, la dimensione dell'immagine da equalizzare per somme successive. Si aumenta progressivamente, partendo da zero, la variabile `dimension` di una quantità pari al numero di righe, diminuendo di un'unità il numero di colonne. All'inizio dello stato viene valutata la condizione del numero di colonne uguale a zero (questo può accadere nel caso in cui si abbia immagine nulla in ingresso oppure al termine del calcolo della dimensione). Quando questa condizione è soddisfatta si passa allo stato successivo.

2.1.5 `CHECK_DIM` state

Stato in cui viene effettuato un controllo relativo al valore della dimensione. Nel caso in cui quest'ultimo sia pari a zero si salta allo stato finale, senza effettuare ulteriori azioni, viceversa si prosegue allo stato successivo.

2.1.6 `GET_FIRST` state

Stato in cui vengono inizializzate, con il primo valore della sequenza di pixel componenti l'immagine da equalizzare, le variabili relative al massimo e minimo parametro in ingresso.

2.1.7 `GET_MAXANDMIN` state

Stato in cui viene acquisito il valore del pixel nella posizione corrente, partendo da quella successiva rispetto a quella utilizzata nello stato precedente. Come prima cosa si effettua una valutazione relativa alla posizione: se tutti i pixel componenti l'immagine originale sono stati analizzati e dunque la sequenza in

ingresso si è esaurita, si passa allo stato `GET_SHIFT`, altrimenti si passa allo stato `CHECK_MAXANDMIN`.

2.1.8 `CHECK_MAXANDMIN` state

Stato in cui si valuta se il valore del current pixel, acquisito nello stato precedente, è rispettivamente maggiore o minore degli attuali valori di massimo e minimo. In tal caso si riassegna il valora al massimo o al minimo e si ritorna nello stato `GET_MAXANDMIN` per valutare il pixel successivo. Si procede in tale modo sino all'esaurimento dei pixel disponibili.

2.1.9 `GET_SHIFT` state

Stato in cui si calcola il valore di delta plus, ovvero il valore di delta, pari alla differenza tra massimo e minimo, più uno.

2.1.10 `COMP_SHIFT` state

Stato in cui, per mezzo di un priority encoder, si determina la posizione del MSB e il conseguente range di appartenenza del delta plus, potendo in questo modo riconoscere, e sottrarre dal valore 8, il risultato del logaritmo approssimato per difetto del parametro delta plus, cifra corrispondente allo shift level.

2.1.11 `GET_CURRENT` state

Stato in cui si riparte dall'inizio della sequenza di pixel in ingresso e, qualora questa non sia terminata, si acquisisce il valore del pixel in ingresso. Se la sequenza è terminata si passa all'ultimo stato per terminare la computazione, altrimenti si valutano le condizioni relative al current pixel, determinando il valore da scrivere in memoria.

2.1.12 `WRITE_VALUE` state

Stato in cui viene scritto in memoria il dato stabilito nello stato precedente. Si inizia dalla posizione di memoria successiva a quella corrispondente alla fine della sequenza di pixel componente l'immagine originale in ingresso.

2.1.13 `WAIT_DATA` state

Stato in cui si attende l'aggiornarsi del valore `i_data` che dovrà essere letto nello stato successivo, ovvero `GET_CURRENT`, dopo aver opportunamente incrementato la posizione di accesso alla memoria.

2.1.14 `END_STATE` state

Stato in cui viene dichiarato il termine della computazione, si attende che `i_start` diventi zero e si effettua un reset di tutti i parametri necessari ai valori di default, così da preparare il componente a una nuova computazione.

2.2 Scelte progettuali

Una scelta progettuale è stata quella di descrivere il componente per mezzo di due processi:

1. Il primo rappresenta la parte sequenziale della macchina e serve per gestire il Register Transfer e quindi come vengono manipolati i registri.

2. Il secondo rappresenta invece la FSM che analizza i segnali in ingresso e lo stato corrente al fine di determinare il prossimo stato della computazione.

L'algoritmo termina la sua elaborazione utilizzando come operazioni logiche solamente lo `SHIFT` per il calcolo del dato da scrivere in output e l'operatore di concatenazione `&` tra due registri. Nel codice sono stati utilizzati esclusivamente l'operatore addizione `(+)` e l'operatore sottrazione `(-)`.

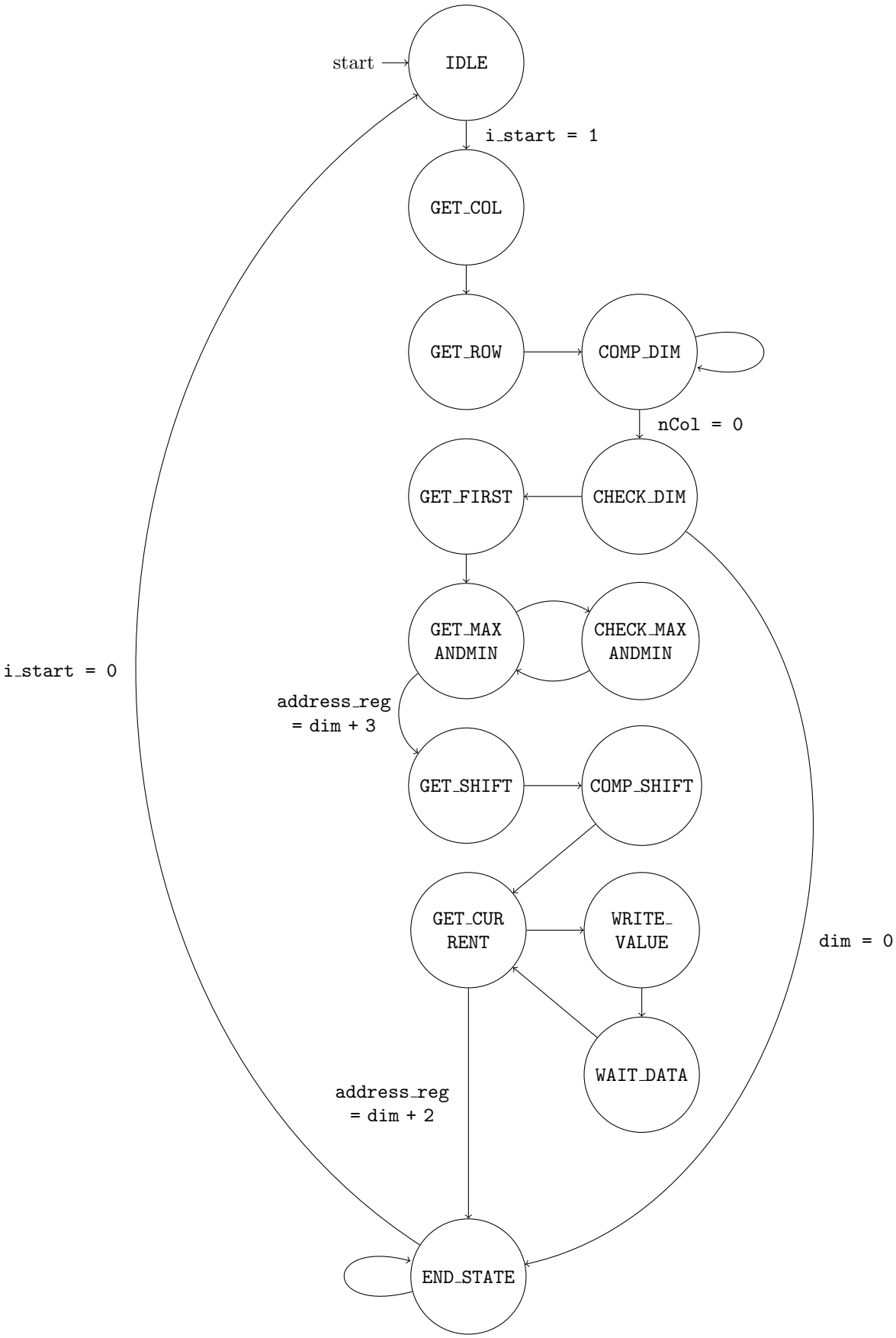

Figure 7: Macchina a stati con le principali condizioni di transizione

3 Risultati dei test

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il test bench di esempio, sono stati definiti altri 14 test (tra i quali anche quelli che spingono la simulazione verso i corner case) in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la FSM può effettuare durante la computazione.

Di seguito è fornita una breve descrizione dei test utilizzati e per alcuni viene anche mostrato l'effettivo corretto funzionamento grazie allo *screenshot* dell'andamento dei segnali durante la simulazione.

1. **Immagine in ingresso nulla**: l'immagine in ingresso ha dimensione nulla. Il test verifica che il componente non effettui alcuna azione e si porti direttamente allo stato finale.

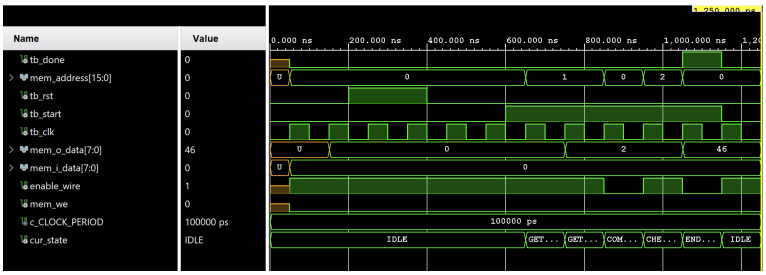

Figure 8: Simulazione con immagine di dimensione 0

2. **Tre immagini consecutive in ingresso**: il test verifica che il componente sia in grado di equalizzare più immagini consecutive, riportandosi nello stato `IDLE` alla fine di ogni equalizzazioni, in attesa di un successivo segnale di start.

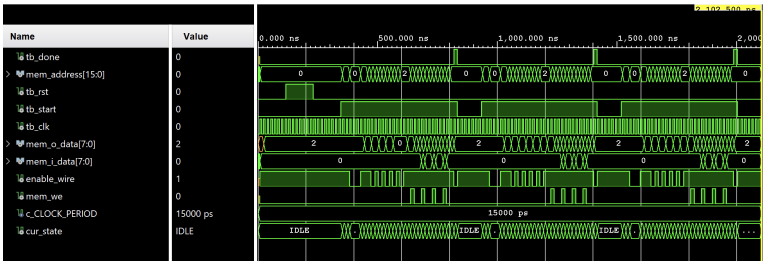

Figure 9: Simulazione con tre immagini consecutive

3. **Immagine in ingresso massima**: l'immagine in ingresso ha dimensione massima, ovvero $128 * 128$. Il test verifica che il componente sia in grado di calcolare la corretta dimensione dell'immagine, per somme successive, e di scrivere in output i valori dei pixel dell'immagine equalizzata.

4. **Immagine in ingresso con reset**: il test verifica che il componente, in caso il segnale di reset diventi 1 (anche in modo asincrono), si riporti nello stato di `IDLE` per ricominciare la computazione.

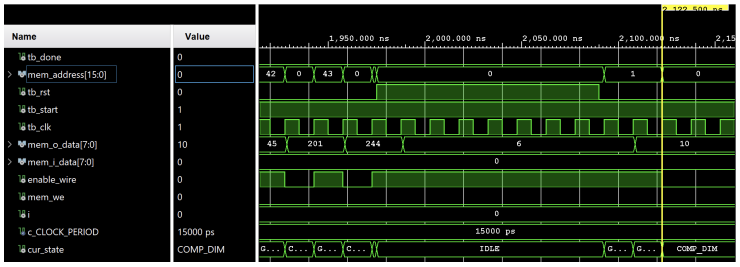

Figure 10: Simulazione di un test con reset

5. **Immagine in ingresso con shift selezionato**: i test, in questo caso 9, ognuno per ogni valore possibile dello shift, verificano che il componente si comporti correttamente in ogni caso. Tra questi vi sono presenti anche il caso dell'immagine full range con pixel compresi tra 0 e 255 inclusi, e il caso dell'immagine monocromatica, con pixel tutti uguali..
6. **Immagine in ingresso con massimo e minimo agli estremi**: il test verifica che ogni posizione della sequenza, inclusa la prima e l'ultima, venga considerata per il definirsi dei valori di massimo e di minimo.

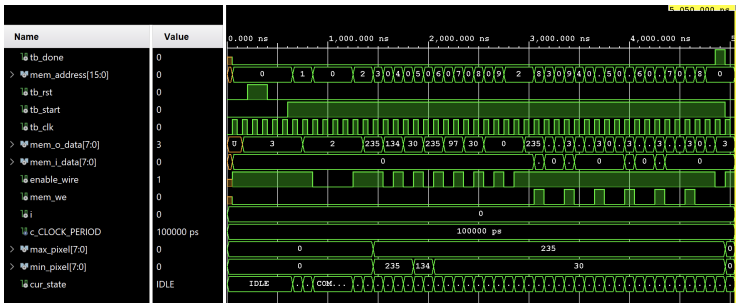

Figure 11: Simulazione con massimo e minimo in prima e ultima posizione

Di seguito è possibile vedere un confronto tra i tempi di simulazione dei due corner case che portano la macchina verso la più breve e la più lunga simulazione:

- **1350ns** - tempo di simulazione (*Behavioral*) con immagine in ingresso di dimensione nulla
- **8206650ns** - tempo di simulazione (*Behavioral*) con immagine in ingresso di dimensione massima

Oltre ai test mirati precedentemente descritti, sono stati simulati anche numerosi test random (più di 20.000) per testare ulteriormente il componente.

4 Conclusioni

4.1 Risultati della sintesi

Il componente sintetizzato supera correttamente tutti i test specificati nelle 2 simulazioni: *Behavioral* e *Post-Synthesis Functional*. I report di sintesi hanno rivelato una quantità di *register as Latch* pari a 0 e un *Data Path Delay* di 5,838ns, inoltre il processo di sintesi si è concluso senza *errors or warnings*. Seguono *screenshots* dei risultati del report di sintesi.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	212	0	134600	0.16
LUT as Logic	212	0	134600	0.16
LUT as Memory	0	0	46200	0.00
Slice Registers	125	0	269200	0.05
Register as Flip Flop	125	0	269200	0.05
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Figure 12: Slice Logic

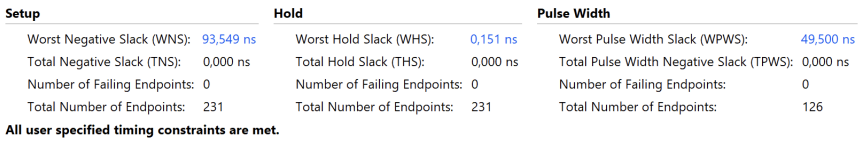

Figure 13: Design Timing Summary

Questi risultati sono stati utili al fine di poter concludere che il componente, avendo superato tutti i test effettuati e non avendo riscontrato alcun errore nella fase di sintesi e di post-sintesi, funziona correttamente ed è in grado di equalizzare un'immagine come da richiesta.