Prova Finale (Progetto di Reti Logiche)

Prof. Fabio Salice - Anno 2020/2021

Paolo Longo (Codice Persona 10677668 - Matricola 911983)

Indice

1 Introduzione

1.1 Scopo del progetto

Lo scopo del progetto consiste nell'implementare un componente hardware, descritto in VHDL, che possa leggere un'immagine da una memoria, generarne una versione "equalizzata" e riscriverla nella memoria.

Il metodo utilizzato è una semplificazione dell'**equalizzazione dell'istogramma**, ed è pensato per ricalibrare il contrasto di un'immagine.

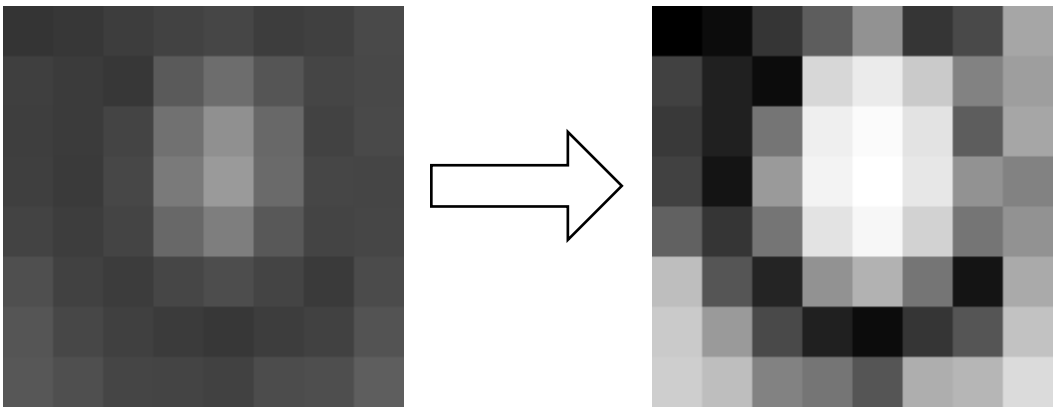


Immagine originale

Immagine equalizzata

1.2 Algoritmo di equalizzazione

L'algoritmo di equalizzazione del singolo pixel è così definito:

$$DELTA\_VALUE = MAX\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE$$
$$SHIFT\_LEVEL = (8 - FLOOR(LOG2(DELTA\_VALUE + 1)))$$
$$TEMP\_PIXEL = (CURRENT\_PIXEL\_VALUE - MIN\_PIXEL\_VALUE) \ll SHIFT\_LEVEL$$

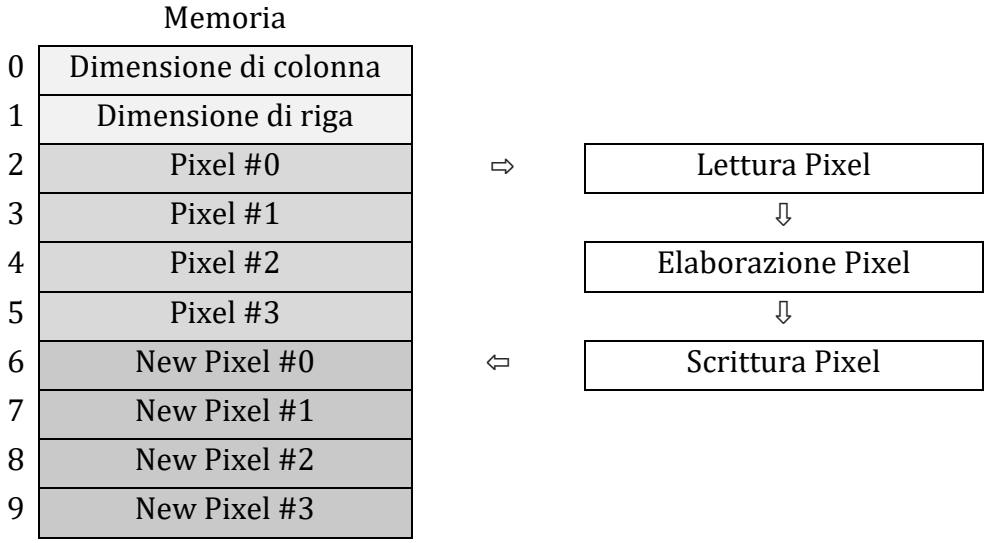
$$NEW\_PIXEL\_VALUE = MIN(255, TEMP\_PIXEL)$$

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 pixel equalizzato.

1.3 Struttura memoria

L'immagine, di dimensioni massime 128x128 pixel, dovrà essere letta da una memoria, equalizzata e riscritta nella stessa. Le celle della memoria, indirizzate al byte, saranno suddivise in questo modo:

- Le celle *(0)* e *(1)* conterranno rispettivamente la dimensione di colonna e di riga dell'immagine.
- Le celle comprese tra *(2)* e *(n_pixels + 1)* conterranno i pixel dell'immagine da equalizzare.
- Le celle comprese tra *(n_pixel + 2)* e *(2*n_pixel + 1)* dovranno contenere i pixel equalizzati. Questi dovranno avere il medesimo ordine dei pixel originali.



Il progetto non comprende la progettazione della memoria, quella fornita segue tipologia e protocollo *"Single-Port Block RAM Write-First Mode"* (Documentazione Xilinx).

1.4 Interfaccia componente

Il componente da descrivere ha un'interfaccia così definita:

```
entity project_reti_logiche is
  port (
    -- Inputs.
    i_clk     : in std_logic;
    i_rst     : in std_logic;
    i_start   : in std_logic;
    i_data    : in std_logic_vector(7 downto 0);
    -- Outputs.
    o_done    : out std_logic;
    o_en      : out std_logic;
    o_we      : out std_logic;
    o_data    : out std_logic_vector(7 downto 0;
    o_address : out std_logic_vector(15 downto 0)
  );
end project_reti_logiche;
```

In particolare:

i_clk Il segnale di CLOCK in ingresso.

i_start Il segnale di START per avviare il processo di equalizzazione.

i_rst Il segnale di RESET che inizializza la macchina pronta per ricevere il primo segnale di START.

i_data Il segnale *(vettore)* che arriva dalla memoria in seguito ad una richiesta di lettura.

o_done Il segnale di uscita che comunica la fine dell'elaborazione.

o_en Il segnale di ENABLE da dover mandare alla memoria per poter comunicare *(sia in lettura che in scrittura).*

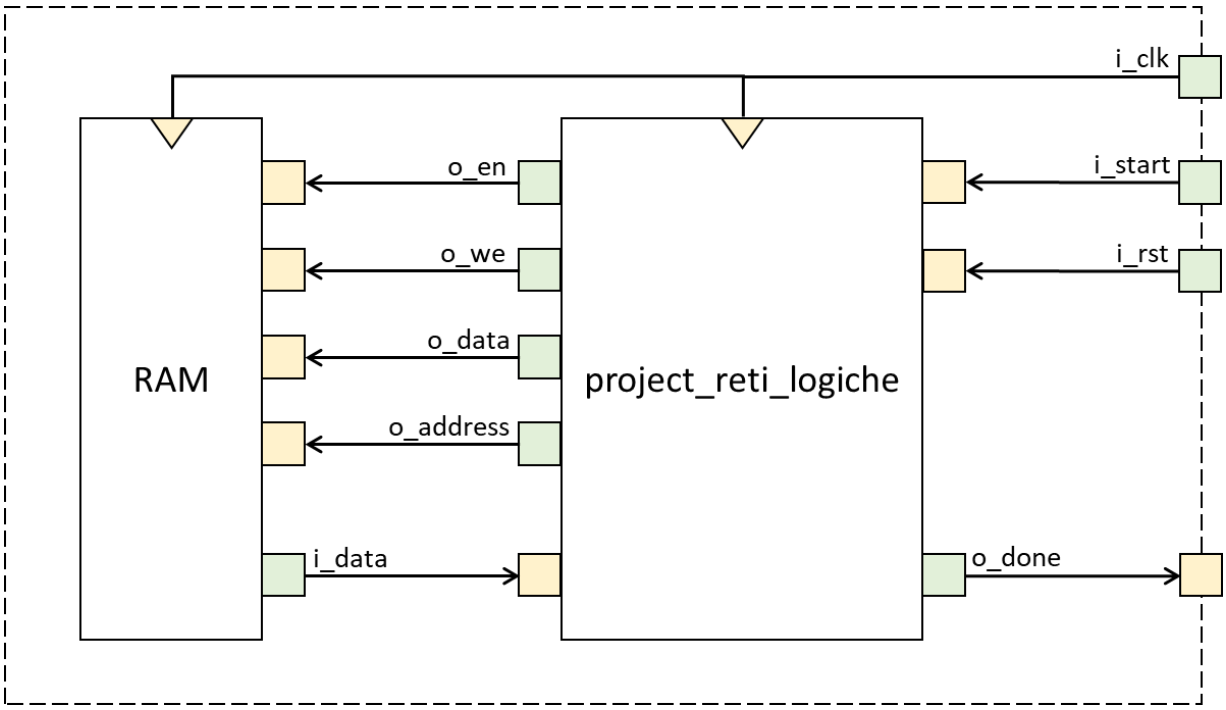
o_we Il segnale di WRITE ENABLE da dover mandare a 1 alla memoria per poterci scrivere. Per leggere da memoria esso deve essere 0.

o_data Il segnale *(vettore)* di uscita dal componente verso la memoria.

o_address Il segnale *(vettore)* di uscita che manda l'indirizzo alla memoria.

1.5 Panoramica e utilizzo

Il componente è così collegato alla memoria. I segnali **i_start** e **o_done** permettono di interagire con il processo di equalizzazione.



Il modulo partirà nella elaborazione quando il segnale **i_start** in ingresso verrà portato a 1 e terminerà portando il segnale **o_done** a 1. Per iniziare un'altra elaborazione, **i_start** dovrà essere riportato a 0 e si dovrà attendere che **o_done** venga riportato a 0.

Prima di poter utilizzare il modulo è necessario resettarlo tramite il segnale **i_rst**, successive elaborazioni oltre alla prima non richiederanno questo passaggio.

2 Architettura

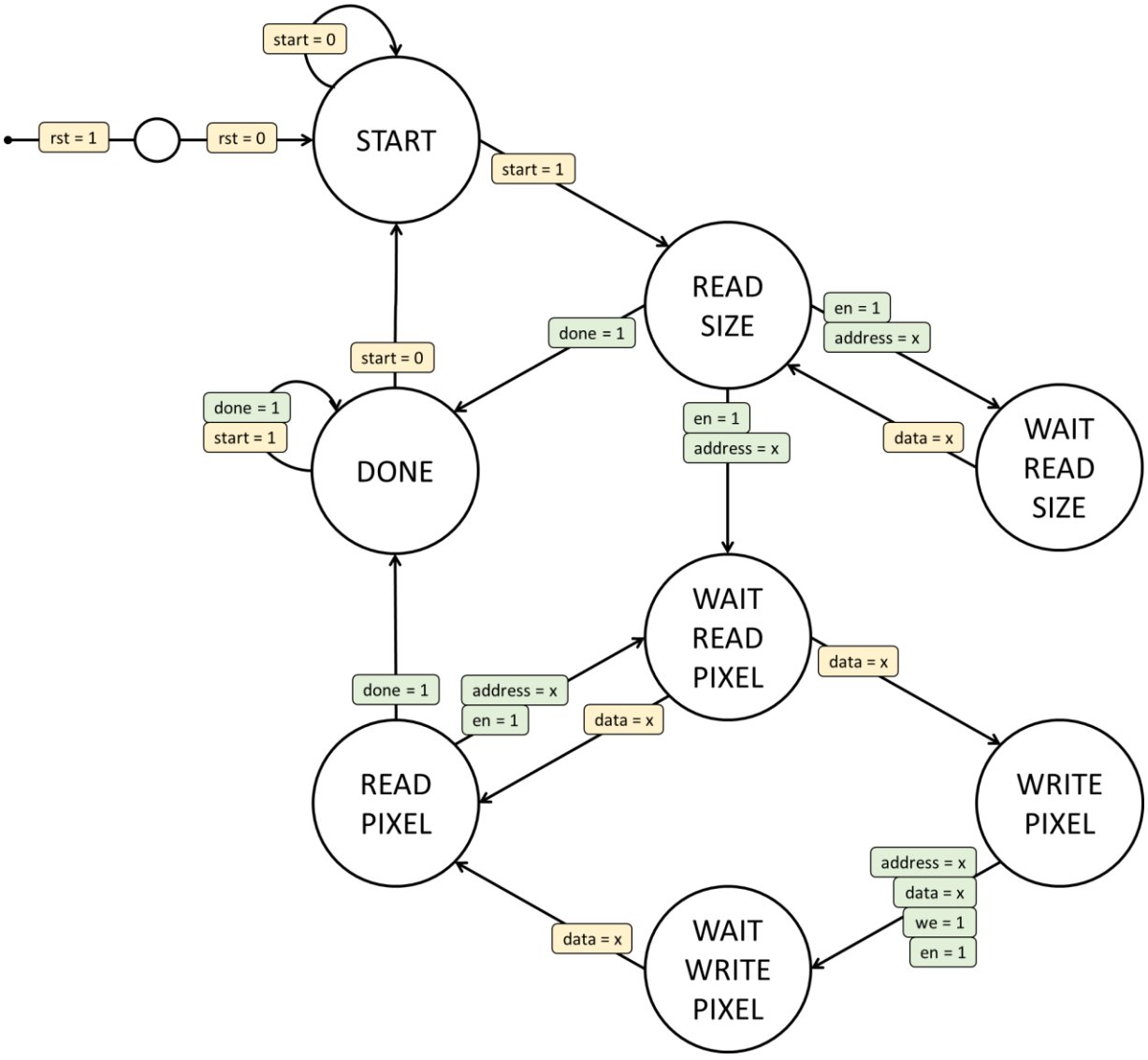
Si è deciso di approcciare il problema con una macchina a stati finiti. Trattandosi di un problema di complessità ridotta si è optato per una soluzione monomodulare con un singolo process che elabora i vari stati.

2.1 Funzionamento

La macchina dovrà leggere le dimensioni, eseguire un primo ciclo di lettura per determinare max_pixel_value e min_pixel_value, e infine eseguire un secondo ciclo nel quale ogni pixel verrà equalizzato e scritto in memoria.

2.2 Stati della macchina

La macchina è composta da 8 stati. Nel seguente grafo sono mostrati i segnali di output (verdi) e di input (gialli). Questi vengono riportati solo quando il loro valore è significativo per il funzionamento del modulo. Se un segnale di output non viene riportato in una transizione si sottintende uguale a 0.



Di seguito è fornita una breve descrizione per ogni stato.

2.2.1 START

Stato iniziale in cui si attende il segnale di **i_start**. In caso venga alzato il segnale **i_rst** si torna in questo stato.

2.2.2 READ_SIZE

Stato in cui viene richiesta la dimensione di **colonna** e successivamente la dimensione di **riga**. Questo viene fatto ritornando in questo stato una seconda volta, in quanto non è possibile effettuare queste operazioni in simultanea.

2.2.3 WAIT_READ_SIZE

Stato in cui si attende la risposta dalla memoria in seguito alla richiesta di una dimensione (colonne e righe).

2.2.4 READ_PIXEL

Stato in cui vengono elaborati i valori dei pixel richiesti per definire max_pixel_value e min_pixel_value. Qui viene anche elaborato l'indirizzo del pixel da leggere e assegnato a **o_address**.

2.2.5 WAIT_READ_PIXEL

Stato in cui si attende la risposta dalla memoria in seguito alla richiesta di lettura di un pixel.

2.2.6 WRITE_PIXEL

Stato in cui viene elaborato il nuovo valore del pixel corrente e il suo indirizzo in memoria. Questi vengono assegnati rispettivamente in **o_data** e **o_address**.

2.2.7 WAIT_WRITE_PIXEL

Stato in cui si attende la risposta dalla memoria in seguito alla richiesta di scrittura di un pixel.

2.2.8 DONE

Stato in cui si attende che **i_start** venga abbassato per poter abbassare **o_done** e tornare nello stato START.

2.3 Registri interni

Lo stato interno del componente è definito dai seguenti registri.

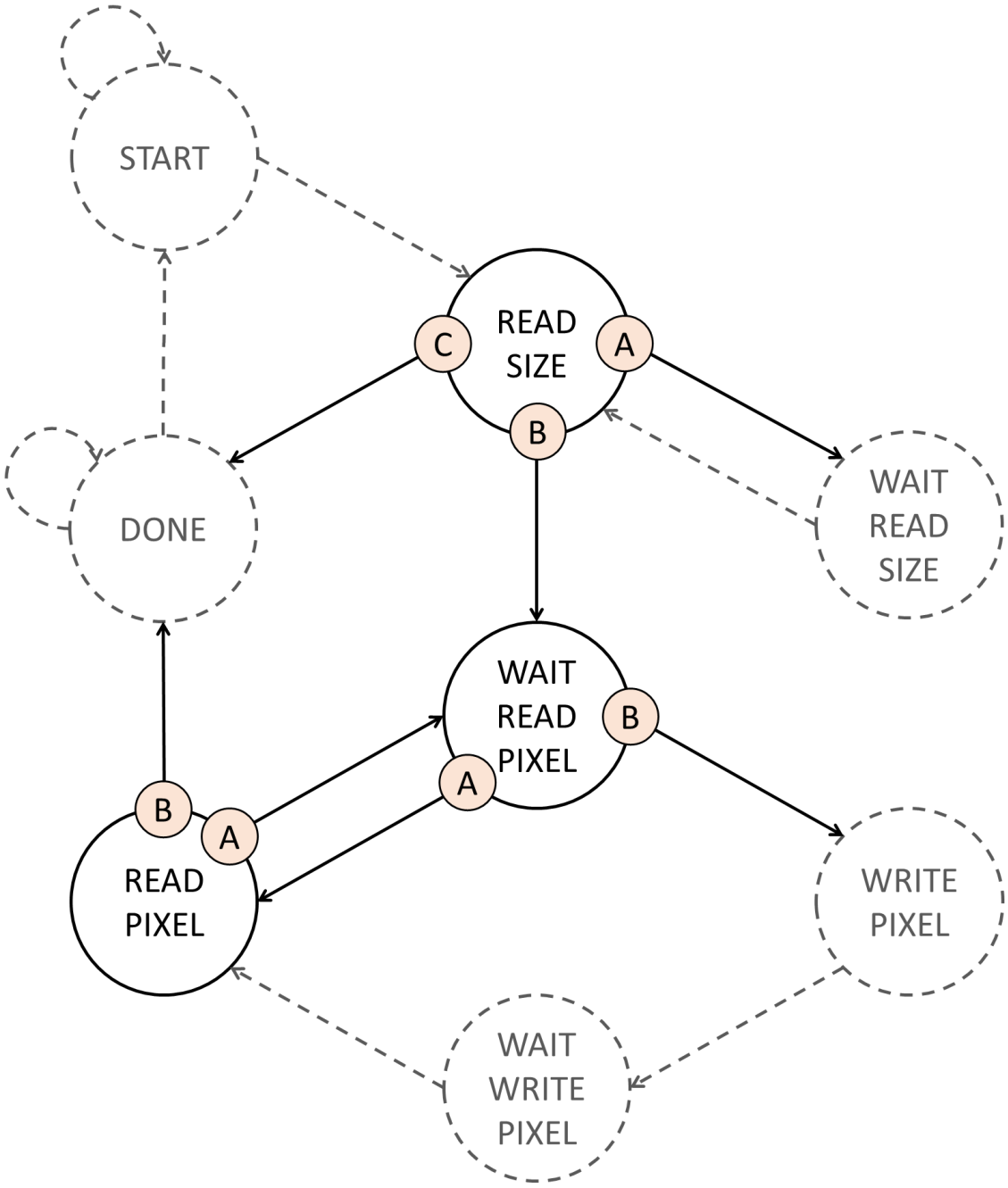
```
-- State flag registers.
signal has_column_size  : boolean := false;
signal has_row_size     : boolean := false;
signal has_pixels_range : boolean := false;

-- Pixels range registers.
signal MAX_PIXEL_VALUE : unsigned(7 downto 0) := (others => '0');
signal MIN_PIXEL_VALUE : unsigned(7 downto 0) := (others => '1');

-- Process registers.
signal last_pixel_address : std_logic_vector(15 downto 0) := (others => '0');
signal current_pixel      : std_logic_vector(15 downto 0) := (others => '0');
```

In particolare:

has_column_size	Registro che indica se la dimensione di colonna è stata letta.
has_row_size	Registro che indica se la dimensiona di riga è stata letta.
has_pixel_range	Registro che indica se max_pixel_value e min_pixel_value sono stati calcolati.
MAX_PIXEL_VALUE	Registro che indica il valore massimo dei pixel dell'immagine.
MIN_PIXEL_VALUE	Registro che indica il valore minimo dei pixel dell'immagine.
last_pixel_address	Registro che indica l'indirizzo dell'ultimo pixel dell'immagine (= 1 se non sono presenti pixel).
current_pixel	Registro che indica il numero posizionale del pixel in elaborazione (1 per primo pixel, 2 per il secondo…).

2.4 Effetto registri flags e controlli interni

Nel seguente grafo vengono evidenziati gli stati le cui trasformazioni dipendono da flags e controlli interni. Nelle immagine successive vengono illustrate nel dettaglio, tramite **pseudo-codice**, le modalità con le quali lo stato successivo viene scelto in questi casi.



2.4.1 **Decisioni** READ_SIZE

Vengono richieste dalla memoria le dimensioni di riga e di colonna, e si va in attesa nello stato WAIT_READ_SIZE. Se non sono presenti pixel il processo termina con lo stato DONE, altrimenti si inizia il processo di equalizzazione. Viene eseguito il **pre-read** *(vedi conclusioni 4.1)* del primo pixel e si va in attesa nello stato WAIT_READ_PIXEL.

```
-- state: READ_SIZE.
if not (has_column_size and has_row_size) then
  -- Option A.
  state <= WAIT_READ_SIZE;
else if not (last_pixel_address = "0000000000000001") then
  -- Option B.
  state <= WAIT_READ_PIXEL;
else
  -- Option C.
  state <= DONE;
end if;
```

2.4.2 **Decisioni** WAIT_READ_PIXEL

Viene deciso se il pixel richiesto verrà usato per definire i valori max_pixel_value e min_pixel_value tramite lo stato READ_PIXEL, o se il pixel dovrà essere equalizzato e riscritto in memoria tramite lo stato WRITE_PIXEL.

```
-- state: WAIT_READ_PIXEL.
if not (has_pixels_range) then
  -- Option A.
  state <= READ_PIXEL;
else
  -- Option B.
  state <= WRITE_PIXEL;
end if;
```

2.4.3 **Decisioni** READ_PIXEL

Vengono richiesti pixel, andando in attesa tramite lo stato WAIT_READ_PIXEL, fin quando entrambi i cicli di lettura non sono ultimati. Una volta terminati il processo finisce tramite lo stato DONE.

```
-- state: READ_PIXEL.
if not (current_pixel = last_pixel_address) then
  -- Option A.
  state <= WAIT_READ_PIXEL;
else if not has_pixels_range then
  -- Option A.
  state <= WAIT_READ_PIXEL;
else
  -- Option B.
  state <= DONE;
end if;
```

3 Risultati sperimentali

3.1 Sintesi

Il componente è correttamente sintetizzabile e implementabile dal tool con un totale di 197 LUT e 81 FF.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	...	LUT	FF
synth_1	constrs_1	synth_design Complete!							197	81
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	0	197	81

3.2 Simulazioni

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il TestBench di esempio, sono stati definiti altri 8 test, 3 dei quali spingono la simulazione verso i corner case. Di seguito sono riportati i test più significativi con una rappresentazione della memoria al termine di essi.

Le celle chiare sono state unicamente lette, quelle scure unicamente scritte e quelle non rappresentate non hanno interagito col modulo.

3.2.1 Test funzionamento

3.2.1.1 Test 1: Fornito dal docente

Test d'esempio fornito dal docente.

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

3.2.1.2 Test 2: Immagine 2x2

Test aggiuntivo per verificare il funzionamento del componente con un'immagine 2x2.

0	1	2	3	4	5	6	7	8	9
2	2	111	32	213	79	158	0	255	94

3.2.1.3 Test 3: Due equalizzazioni

Test che verifica la possibilità di equalizzare due immagini consecutivamente, e quindi che i registri interni vengano resettati correttamente.

0	1	2	3	4	5	6	7	8	9
2	2	111	32	213	79	158	0	255	94

3.2.2 Test corner case

3.2.2.1 Test 4: Senza pixel

Test che verifica il corretto funzionamento del componente quando la dimensione dell'immagine risulta uguale a 0.

0	1
0	0

3.2.2.2 Test 5: Full range dei pixel (255 - 0)

Test che verifica il corretto funzionamento del componente nel particolare scenario in cui l'algoritmo di equalizzazione non modifica i pixel.

0	1	2	3	4	5	6	7	8	9
2	2	255	50	100	0	255	50	100	0

3.2.2.3 Test 7: Uso del reset

Test che verifica il corretto funzionamento del componente nel particolare scenario in cui un processo viene interrotto dal segnale di reset e un secondo viene avviato.

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

4 Conclusioni

4.1 Scelte progettuali

Si è deciso di implementare in un unico stato, READ_PIXEL, il processo di ricerca dei valori max_pixel_value e min_pixel_value, e il calcolo dell'indirizzo del pixel da leggere. Questo permettere di risparmiare 2 cicli di clock per ogni pixel durante l'intero processo rispetto alla soluzione con 2 stati separati.

Questa decisione introduce un problema nello stato READ_PIXEL, che ora deve processare i pixel che esso stesso deve richiedere. Per risolverlo si è deciso di effettuare una **pre-read** del primo pixel nello stato precedente, READ_SIZE. Questo ha consentito di mantenere un'architettura semplice e di non introdurre nuovi registri.

Infine, si è deciso di utilizzare un set di registri per memorizzare l'indirizzo in cui leggere, e di **non** memorizzare quello in cui scrivere. Esso viene ricavato dalla somma del precedente indirizzo ad un offset statico. Questo permette di avere un architettura più chiara e di risparmiare sull'utilizzo di qualche registro (l'eventuale memorizzazione dell'indirizzo necessiterebbe altre componenti per poterlo incrementare di 1 per ogni ciclo di scrittura).